ABSTRACT

TSHIBANGU, NYUNYI MARCUS. Study and Analysis of Energy-Efficient DRAM-Cache with Unconventional Row-Buffer Size. (Under the direction of Dr. Paul Franzon).

The development of 3DIC technology has given rise to a new generation of fast and dense memories. These memories consist of multiple cell layers stacked on top of each other using interconnect in both X, Y, and Z dimensions. The vertical interconnect in this Die-stacking technology can help reduce the size, increase the speed, and reduce the energy simultaneously. These advantages have given researchers the motivation to incorporate the stacked DRAM as last level cache (LLC) or DRAM-cache. It is the goal of architects to have a fast and large memory on-chip, but the current SRAM cache technology provides a limitation. Therefore, a DRAM-cache is an alternative to achieving both high density, speed, and proximity to CPU. Studies to date demonstrating this used a 2KB row size and up. If the industry adopts a widespread use of DRAM for cache, stacked DRAMs need organization suitable not only for better performance but also for energy efficiency. In this work, we have studied and analyzed a range of DRAM cache with row buffer size ranging from 256B to 4 KB, and we determined that a DRAM cache mapped in unconventional 512B row buffer stacked memory is a better choice. Such stacked DRAM was set up to provide two independent ports with access to 32 independent banks each. A smaller row size significantly cut off the energy per access at row buffer miss. An 8-way single-set single-row and energy-efficient mapping (SSSR-LE-8) configuration achieved 36, 40, and 74 % improvement in overall energy consumption compared to 3 other configurations mapped in 2KB row buffer memory. This low-energy mapping at the same time also increased the performance by 16, 18, and 41%. The performance is determined in term of execution time and is mostly accurate since GEM5

simulator was modified to account for variable hit latency due to row buffer hit status. The energy consumption within the DRAM cache is also a new feature added to the tool capability.

Study and Analysis of Energy-Efficient DRAM-Cache with Unconventional Row-Buffer Size

by
Nyunyi Marcus Tshibangu

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Electrical Engineering

Raleigh, North Carolina

2016

APPROVED BY:

_____	_____
Dr. Eric Rotenberg	Dr. W. Rhett Davis
_____	_____
Dr. Bryan Floyd	Dr. Gregory Buckner

_____
Dr. Paul Franzon
Committee Chair

DEDICATION

To my Father, Karl Mukeba, Thank you for inspiring me to pursue higher education during your short and precious journey here on Earth. I owe you this degree.

Wish you were here to reap what you sowed.

BIOGRAPHY

Nyunyi Marcus Tshibangu was born in the city of Mbuji-Mayi, Democratic Republic of Congo, as the last of eight children. Four years after he graduated from high school, he moved to the US in 2002 to pursue high education. In 2006 he graduated from the University of North Carolina at Charlotte (UNCC) with a degree in Electrical Engineering. Upon graduation, he joined the Timken Corporation as Control's engineer providing support in Automation for high volume manufacturing of package bearings from 2006 to 2009. Earlier 2008, he joined NC State University to pursue his Masters in Electrical Engineering with a focus on Electronic Circuit and System (ECS). Upon completion of Masters at the end of 2009, he joined ESAB Corporation as R&D Hardware and Software Engineer for the development of electronic control system used for the design and fabrication of automatic welding and cutting machines, based on plasma, oxy-fuel and laser technology.

In spring 2012, He came back to NC State to continue with his PhD in Electrical and Computer Engineering. Given his industrial exposure to control system, he planned to join the control and mechatronics research area but this quickly changed when in summer of 2012, he was offered an important role in "3D implementation of Heterogeneous processor", a project sponsored by Intel Corporation under the direction of Dr. Paul Franzon. This completely changed his research interest to memory system and computer architecture. Being a big proponent of "learning by doing", he joined a low-power CPU group at Intel Corporation for multiple graduate internships to reinforce his exposure to computer design and architecture.

ACKNOWLEDGMENTS

First I would like to Thank God for his gift of salvation and countless blessings in my life. From a little boy born in the heart of Africa to finishing a PhD and playing important roles in the American society, I do not take anything for granted. From putting me in contact with the greatest and supportive wife (Priscas Mbaya) and having an amazing daughter (Kalubi Tshibangu) and son (Nyunyi M. Tshibangu), as well as my unborn child, he set me for success as the support and motivation they are giving me is unprecedented and I am forever grateful.

Second, I would like to thank my Father (Karl Mukeba) and my Mother (Celine Mpunga) for their amazing love and support. They gave me a deep understanding of the importance of education and my whole life I have done everything in my power to reach the highest level of education possible and make them proud. To all my brothers and Sisters, in the US and abroad, thank you for your love and support and this degree is my way of loudly express that.

Finally, I would like to thank Dr. Paul Franzon first, for endorsing my admission in the ECE department and second for accepting to take me as his student and providing me with guidance, constructive criticism, financial and technical support to complete my work. He allowed me to overstay my journey at Intel Corporation and gain additional industrial experience and a huge opportunity to advance my career with the company. To Dr. Davis and Dr. Rotenberg for allowing me to become part of this great IntelH3 project and providing feedback that has contributed to my success. In particular, Dr. Rotenberg for igniting my interest in computer architecture and for his comprehensive and detailed feedback that contributed to making me a better researcher and Dr. Davis for his open-mind and motivating

conversations beyond academic. To Dr. Floyd and Buckner thank you for serving on my committee.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1. Motivation and Objectives

3DIC allows the design of fast and high-density memory. It gives research opportunity to study the utilization of DRAM for the on-chip cache. The high-density memories built on vertical stacks can solve the size limitation of SRAM technology. The vertical interconnect can help reduce the size, increase speed and bandwidth, and reduce energy per access. This research work investigates the implementation of such stacked DRAM into low-energy and high-performance last-level cache or DRAM-cache for multicore processors. Recently, few studies have been conducted to use stacked DRAM as last level cache. There are two motivations for this. First, these 3DIC DRAM allows I/O busses as large as 128-bit which increases bandwidth compared to regular DDR DRAM with only 16-bit in data transfer. Second, the large size compared to any last level cache in modern processors can offset the average access latency by significantly reducing capacity miss.

In these recent studies, the focus is put mostly on performance for obvious reasons, but very few studies focus on performance while optimizing energy consumption in these DRAM caches. In this work, we will put our focus on energy consumption while keeping a competitive level of return in term of CPU execution time. Even though 3DIC technology can reduce the energy per access on DRAM, the total number of accesses and the size of data can significantly make a difference in the overall energy consumption. Most recent studies have simply considered the use of large row size DRAMs. Either the conventional 2 KB row [1] [2] [3] [4],

a large row of 4KB [7], and an extra-large row of 8KB [24]. In this research, we are studying the implementation of DRAM-cache configuration using unconventional row size optimized for better energy consumption

and increased performance. To do this, we cannot just be limited to large pages used in traditional DDR memories. In contrast, we are proposing the utilization of stacked memory with a small page. The primary motivation to this is the fact that a smaller row size reduces the energy per access during row operation such as the activation and precharge. This energy reduction is also noticed at read and write when a row is already open.

1.2. Overview of chapters

Before we get into our main research objectives, we took the time to gather all the necessary information that can make our study accessible and understandable by a vast majority of readers, not just the experts in the field. Therefore, we are giving a good overview of memory technology and hierarchy, computer performance, 3DIC technology and its challenges. Below is an outline of each chapter.

Chapter 2 is an overview of memory technologies used in industry today and their applications in computer design and performance. We put an emphasis on memory hierarchy using these type of memories. Since our research is in large part studying cache, it is in our best interest to mention cache organization and optimization as implemented in traditional computer design.

Chapter 3 is a general overview of Three-dimensional Integrated circuits (3DIC). Because it is a focus of this dissertation, it is suitable to dedicate time in literature to review the implementation of this technology. We are focusing on advantages and disadvantages of this technology including the most known challenges such as thermal issues that have slowed down the widespread implementation in commercial products.

Chapter 4 is the state-of-arts literature on stacked DRAM applications on which we have based our study and motivations. We are summarizing several proposed configurations and analyzing their advantages and disadvantages that have led to our recommended configurations and methodology for improvement.

Chapter 5 is the central part of our dissertation that focuses on our proposal and detailed study of its implementation. We have studied and analyzed a baseline configuration suitable to reach our objective of energy saving and demonstrate the advantages of using a stacked DRAM with a smaller page.

Chapter 6 lays out the simulation and methodology used to test our proposed configurations as well as a comparison to configurations mapped in larger row buffer memory. We have used CACTI3DD [40] to estimate the energy and timing model for all settings, and we described our modification made to GEM5 [41] simulator to extract CPU time and power numbers that are 3D specifics. This simulator originally evaluates cache hierarchy designed in 2D SRAM technology. Using stacked DRAM for cache requires implementation of its particular behavior to model non-fixed latencies as characterized in 2D cache templates.

In chapter 7 we conducted a survey of different row buffer sizes to complete our study of low-energy DRAM cache. The goal is to find a better organization of DRAM for energy-efficient last level of cache.

Chapter 8 focuses on our tape-out that led to the fabrication of DRAM-cache controller chip to control Tezzaron DiRAM4 3D stacked memory. This study includes the controller architecture, micro-architecture, pre-silicon design, and validation. It also describes many challenges encountered as well as solution needed for the success of the project. Chapter 9 is our concluding remark and future work.

We also added a comprehensive appendix section to give a reader a better understanding of our methodology and a possibility to duplicate work and results. In Appendix A, we provide in detail changes that were made to the simulator and in Appendix B, a copy of our RTL written for fabrication tape-out.

CHAPTER 2

Memory System in Modern Computers

2.1. Memory Technologies and Classifications

This section defines the different classification of memory and their impacts in modern computers. The demand in computer performance requires a lot of memories at both core and system-level. At the core level, more than half the transistors are dedicated to memory in the form of cache [8]. Particularly it is the case at system-level where off-chip memory is needed to keep up with computation bandwidth requirement. Therefore, different memory types are required to fit the system requirement in cost, density, and speed.

There are three main classifications of semiconductor memory: read-only memories (ROM) also characterized as non-volatile, read-write memories (RWM) also called volatile, and Non-volatile Read-write memories (NVM). The data in ROM is hard-wired and can only be read but not written, and its integrity does not depend on the existence of power source. Most use of these types of memories in modern computers is for firmware or initialization of information during system boot-up. On the other side, RWM can both be read and written during the process and requires the continuous existence of power source to keep the integrity of data.

The most popular type of RWM is Random Access Memory (RAM) and its two variants: SRAM and DRAM (static/dynamic Random access memories). Random refers to their abilities to read and write data in any order. It is a contrast to serial memories where the order of access is critical.

An example of these is FIFO (first-in-first-out) and LIFO (last-in-first-out) which used for stacks and queues. The two characteristics that make a major different in RAM are "*static*" and "*dynamic*". Static refers to its ability to retain data with continuous power source only. While in dynamic, the existence of permanent power supply is not enough. To maintain data integrity, periodic read and write operations are required. It is known as *refresh.*

The third and most emerging class of memories is the non-volatile read-write memories (NVM). Among these are EPROM (Erasable Programmable), EEPROM (Electrically Erasable Programmable), and FLASH memories. The latter is the fastest growing type of memories that serve to design mass storage as a better alternative to magnetic disks used for hard-drive. Solid-State-Drive (SSD) uses NAND flash memory as main storage unit [80] [81]. The storage cell used in almost all NVM is the floating-gate transistor which is an MOS device with an extra polysilicon stripe floating between the gate and channel hence the name floating-gate. The high voltage across the drain-source injects hot electron able to cross the first oxide and get trapped in the floating gate and changes the property of threshold voltage. This state corresponds to logic 1. Because the floating gate is insulated between a thick silicon oxide, charges can remain even after the removal of the power supply. Hence the name non-volatile.

Not all non-volatile memories use floating-gate as a storage device. Some use the magnetic property to store data instead of electrical property as used in MOS devices. An example of such memories is Magneto-resistive RAM (MRAM) or Spin-Transfer Torque RAM (STT-RAM). The storage device in MRAM is the magnetic tunnel junction (MTJ) which consists of two ferromagnetic layers whose directions determine the data stored. Recently, Everspin Technologies launched the first fully functional MRAM [83] [84]. Because of the nature of its storage element, the refresh as required in DRAM is not needed and this saves

energy and simplifies the design. But because it is still at an earlier stage, their 64Mbit density does not meet the current memory demand.

2.2. Random-Access Memories

Because we are studying the application of DRAM for cache purpose, it is important to elaborate more on these types of memories. RAM, as mentioned above, allows the system to read and write in random orders. The two kinds classified in this category depends on the storage methods. When data is stored in a latch, its value remains as long as the power is active and this type of storage or cell is called static and give rise to static RAM (SRAM). However, when data is stored in a capacitor, its value can change as the capacitor discharges with time and this type of storage or cell is called dynamic RAM (DRAM). SRAM is often a preferable on-chip memory for its speed, data integrity and mostly fabrication technology that is compatible with CMOS logic. But its limited density due to a large and costly cell size has led to consideration of DRAM for on-chip application. The widespread use of DRAM for on-chip applications is through embedded DRAM or e-DRAM [37] [38], and most recently the DRAM cache mapped in 3D memory is being studied for its ability to integrate logic chip and storage chips fabricated in separate technology. The latter is our primary interest in this work.

2.2.1. SRAM

SRAM cell consists of two back-to-back inverters forming a latch to preserve the integrity of data as shown in Figure 2.1. This basic CMOS device consists of six transistors. Four are used in the two inverters that form a latch, and the other two important NMOS are pass transistors used for read or write operations. The word line (WL), used as enable bit, activates these pass transistors. The bit lines value of BL= 0 and conversely BL_N = 1 writes a zero to the cell. If the cell were previously high, its value would change to low. The contrast is applied when writing the value 1. Since this device requires at least six transistors, SRAM density is low, and its cost per bit is high.

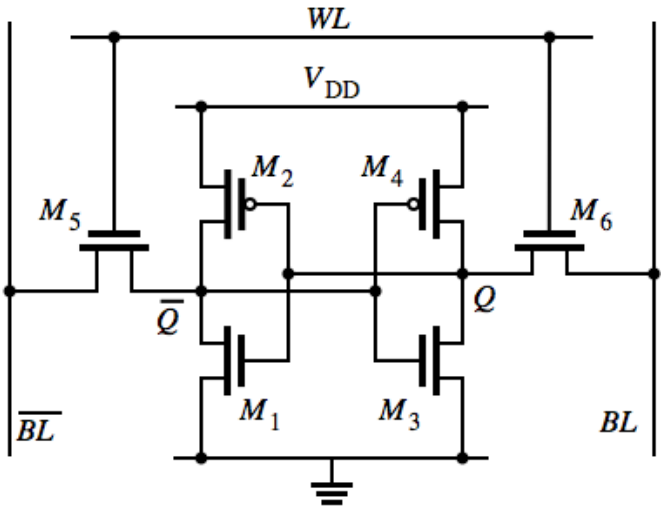


Figure 2.1. Basic SRAM Cell

The 6T back-to-back inverter SRAM cell shown here is the most basic device to keep density at an acceptable level. However, the noise margin in the 6T cell can cause the change of states during a read or write operation. Therefore, to improve read noise margin [RNM] and stability,

few improvements have been proposed regarding additional transistors. These complex cells can hold 7, 8, 9 and even 10 transistors [74] [75] [76]. The drawback of increased transistor count is the increase in cost and area. So some improvements are being made on the 6T to increase stability [77] and keep small area and low cost for mass production.

2.2.2. DRAM

A DRAM cell consists of a single transistor and a capacitor known as 1T1C cell as shown in figure 2.2. The transistor is used to charge and discharge a capacitor during a write or read operation. For a write operation, while the word line is raised high, the bit line value of 1 charges the capacitor, and the value of 0 discharges it. For a read operation, the bit line (BL) capacitor is pre-charged to a certain voltage (typically Vdd/2) before the WL is activated. When WL is high, the charges are redistributed between the two capacitors and there is a change in voltage on BL that determines the value of the cell. The sense amplifiers detect this voltage change and store the resultant value in built-in registers or cache called row buffer. At every read or write to the same column, data is accessed from the buffers rather than bit-cell itself. The reason for this is because during the read operation, the modification of cell charge destroys the data value; therefore, the value must be restored to keep its integrity.

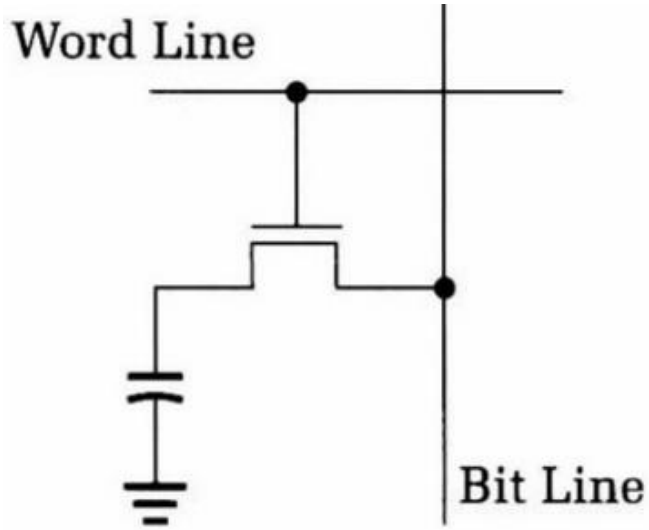


Figure 2.2. Basic 1T1C DRAM Cell

So instead of restoring the data right after read, most DRAM controllers wait until a different row access is required before writing back data stored in row buffers. It is called open-page policy as opposed to the closed-page policy, typically not frequent. Because data is stored in a capacitor, not only it is lost after a read operation and power source removal, but also when the capacitor discharges through leakage over time. Therefore, a periodic refresh is required to keep the integrity of data. This refresh is simply a row activation (ACT) followed by a write back of data from row buffer before row closing (PRE) (e.g. REF = ACT + PRE). Most memory controllers keep track of the discharge time with internal counters. Typically, depending on temperature, an interval of 8 ms to 64 ms is required to refresh active rows of the DRAM. Because the cell size is small, the density of a DRAM is high and the cost per bit is much lower compared to SRAM. Millions of these bit cells can be organized into an array of rows and column called banks.

A single DRAM chip can content many independent banks (e.g. 16) controlled by separate circuits (e.g. decoders, sense amps, drivers, buffers). The bank address selects a bank, and once active, the row address selects the row to activate. This operation is called row access strobe or RAS. During RAS operation, the word lines (WL) of a group of cells are driven high. The column address selects the group of bit lines to read or write. This operation is typically called column access strobe or CAS. Because of bandwidth requirement, column read or write is done in a consecutive range of address called burst mode, often locating 2, 4, or 8 different column addresses on a single CAS. To reduce access time, this is done on both rising and falling edge of the clock or double data rate (DDR). Even though 1T1C cell is the most popular and most economically sound, there are still some variations of this. A 2T1C cell consisting of an extra transistor [78] [79] had been proposed for embedded DRAM applications. This cell is called gain cell and is claimed to be logic compatible. But because DRAM is always being designed for low leakage and low cost, 1T1C still the most favorite bit cell for most applications.

2.3. Computer Performance

From 1986 until 2003, the frequency of CPU was increasing by 40% per year (e.g. 16.7 MHZ to 3.2 GHZ) as shown in figure 2.3 [9] and the Processor performance regarding memory requests per second also increased by 52% per year as shown in figure 2.4 [9]. But in the same period, the memory performance has been less than 7 % per year showing an increasing gap between the processor and DRAM. In the last decade, this aggressive improvement in CPU has seen a tremendous slow-down as the frequency increase and performance improvement has flattened out down to less than 1% per year [9]. This analysis shows that the CPU of a single processor has reached the maximum frequency that does not provide a noticeable increase in performance. Twenty years ago this was predicted through the idea of *memory wall* [12]. This idea states that as the off-chip memory access time diverges significantly from cache access time, the average memory access time will grow and therefore degrading the performance. As the speed of the memory determines the overall system performance, it has hit the wall and increasing the processor speed won't affect the program execution time.

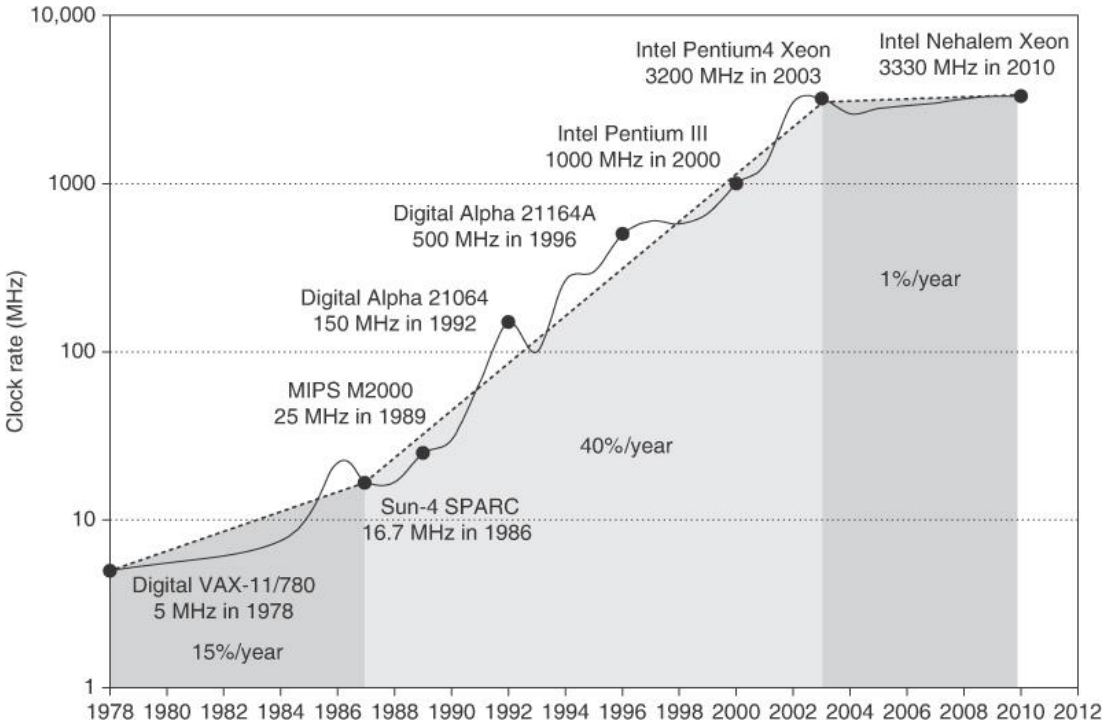


Figure 2.3. [9]. H & P Clock Frequency trend.
1986-2003 is the Golden-Age: 40%/year, 2003-present, frequency is flat: from 2005-2014, Intel Processor run between 3.2-3.3 GHZ.

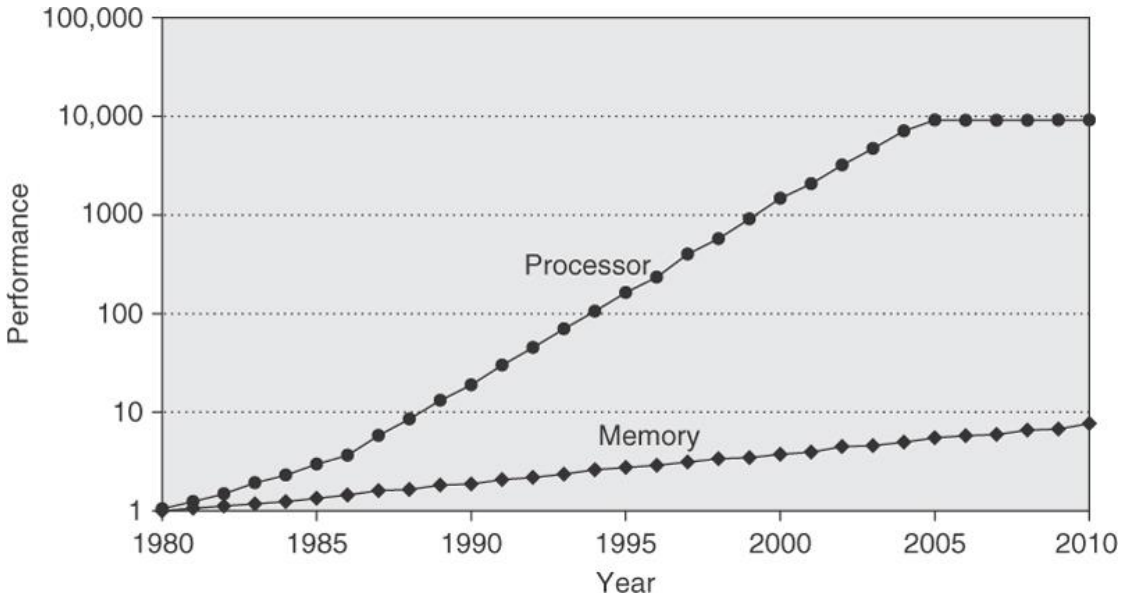


Figure 2.4. [9]. H & P Processor-Memory Gap.
CPU vs. Memory performance (Average memory requests/sec vs. memory accesses/sec). Golden-age: CPU was improving by 52%/year, while memory less than 7%/year. In the last decade, CPU performance has flattened out.

Another reason for flat frequency is the dependency of power and energy consumption on switching speed of transistors. As feature size decreases, the number of transistors that can fit on a single chip dramatically increases and dominates the power equation (fcv^2) despite a drop in voltage (v) and transistor capacitance (c). So continuing to increase frequency in a single processor has led to a *power wall* [9]. Because of the memory and power walls, modern techniques to optimize performance has focused on studying simultaneous computation by exploring different parallelism within a single processor: Instruction-level parallelism (ILP), and data-level parallelism (DLP) or through multiprocessors: Thread-level parallelism(TLP). These parallelism techniques are designed to hide memory latencies and reduce processor stalls giving a virtual ladder to climb over the wall [14]. Even though there is a noticeable performance improvement due to these multiple advancements in CPU computation techniques, there is also a net energy consumption increase due to their logic and hardware complexity. Therefore, a continuous research to improve memory hierarchy is required to break this persistent memory wall entirely and hopefully, reduce energy consumption while keeping an acceptable level of speed in processing.

2.4. Memory Hierarchy Overview

Traditionally, the memory system is designed to exploit the principles of spatial and temporal locality to reduce the length of the wall. By dividing it into different levels, a reduction of hit penalty can be achieved. Every increasing level (closer to CPU) is smaller and faster than its lower counterpart. But unfortunately these fast levels are more expensive as the cost per byte is a function of size and technology. Inside the CPU are found Register files or a set of flip-flops to contain data directly needed for computation and logic.

This level of memory (Level 0) is only a few hundreds of bytes (e.g. 500 bytes) and only within few hundreds of picosecond delay (e.g. 500 ps). The first on-chip two to three levels are implemented in SRAM. Level-1 cache (L1) is the closest structure to the CPU register file with a size of tens of kB (e.g. 32kB) and access speed of fewer than two cycles. Level-2 cache (L2) is relatively large within hundreds of KB (e.g. 256 kB) and access speed around ten cycles. Level-3 cache (L3) has a size in the magnitude of few MB (e.g. 4MB) and access speed around 20 cycles. Some systems have a fourth level L4 of on-chip memory, but this may cause performance reduction or diminishing return due to serialization of latency.

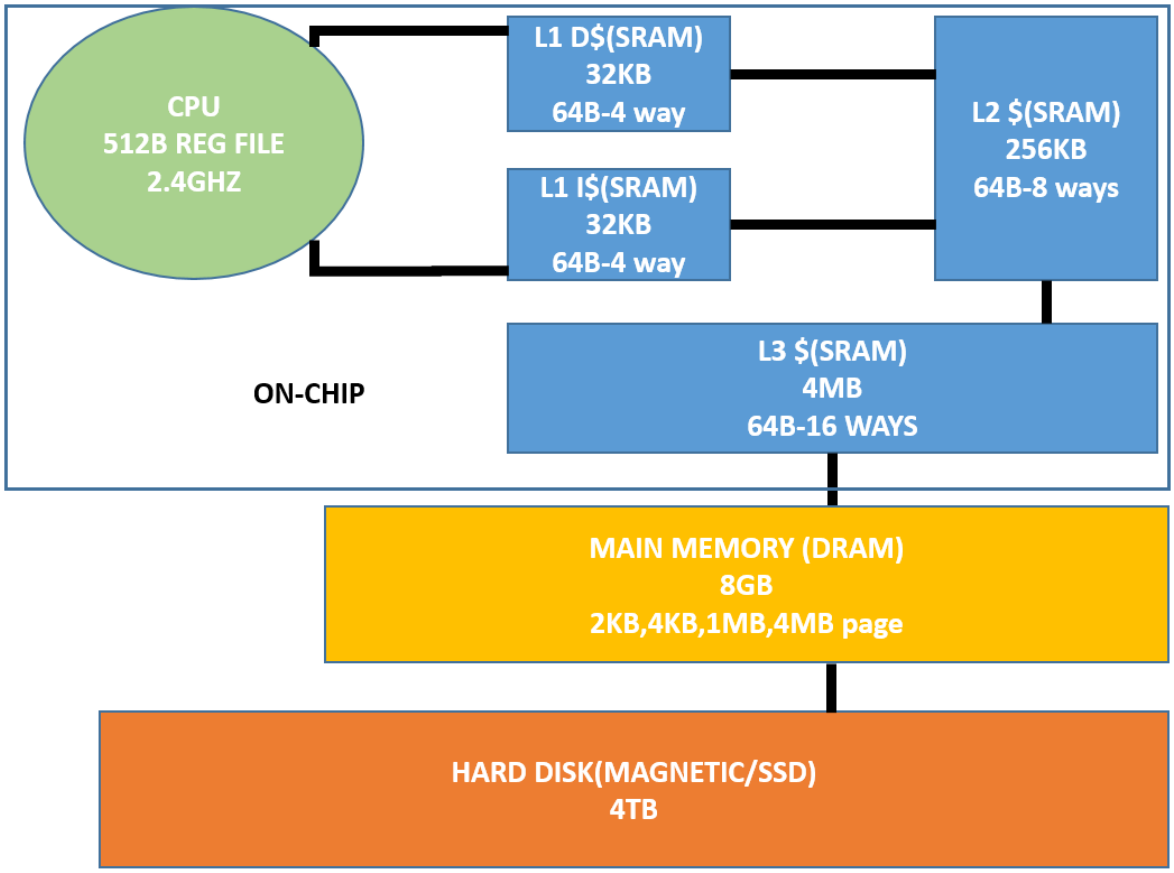


Figure 2.5. Example of Memory hierarchy

On system-level, there is the main memory which is an off-chip DRAM and contains most of the size of the entire memory hierarchy in GB level (e.g. 8 GB) and access speed between 50 to 100 ns. Because of its volatility, main memory or any other lower-level memories cannot store data outside program execution at the absence of power supply. Therefore, program files and other critical information need to be saved in permanent storage. There are two types of mass storage today. The first and the oldest is the hard drive (HDD) which uses a spinning magnetic metal plate to Read and write data. Because of moving mechanical part, the read/write latency is in order of ms. The second mass storage which is a potential replacement to HDD is the solid-state drive (SSD) that uses NAND flash memory as a storage unit. Because of lack of moving parts, SSD is faster and more reliable than HDD. Its high cost per gigabit (e.g. 4x) still prevents it from replacing HDD but with new Intel Corporation Micron Technology joint effort on 3D NAND technology [82], eventually, the cost will go down, and HDD may become as ancient as a floppy drive. Figure 2.5 gives a good visual representation of the current memory hierarchy for high-end computer processors.

This hierarchy is a way to provide the CPU with fast and close memory for its fast operation, but unfortunately, the closest and fastest memory is the smallest. Ideally, it is the goal of architects to have an infinite amount of memory closer to CPU and providing data at the CPU bandwidth. The continuous research in memory technology is to find the next generation of disruptive technology that can combine memory and storage in a single level of hierarchy. Intel Corporation and Micron Technology in their joint effort announced a new disruptive technology called 3D XPoint [87]. This technology, as they claimed will give rise to inexpensive (10X denser than DRAM), fast (1000X NAND Flash), and non-volatile memory [88] that can combine both memory and storage into a single level of hierarchy.

2.5. Cache Organizations

Cache is a small size memory organized into small *blocks* grouped into rows or sets. A typical block size is 16, 32 or 64B. Each block is larger than the operands size and is identified by its address in the memory. A tag is a portion of this address that identifies the block location in the memory. The number of blocks into a set determines the *associativity* or ways. An associativity of one is called direct mapping. Otherwise, it's a *set-associative*. On the other extreme, if all the blocks in the cache belong to a single set, it's referred to as *fully associative*. The following are the basic cache equations:

$$blockoffset = log_2(blocksize) \quad (2.1)$$

$$\#set = \frac{cachesize}{associativity\ X\ blocksize} \quad (2.2)$$

$$index = log_2(\#set) \quad (2.3)$$

$$\#tag\ bits = \#addr\ bits - index - block\ offset \quad (2.4)$$

When CPU requests data for computation, the level1 cache controller looks up its address in tag area. If there is a match, it's referred to as *cache hit* and data will be sent to CPU. Otherwise, it's a *cache miss*, and the request is redirected to next level in the hierarchy all the way to the main memory. The address is decoded using the above cache equations to calculate the size of each field. For example, from Figure 2.5, a 32 KB cache with a 64 B block and 4-way associative will be mapped to 128 sets. It will need 7-bit row decoder as the index; 6-bit multiplexer is the block offset to provide data required by the CPU.

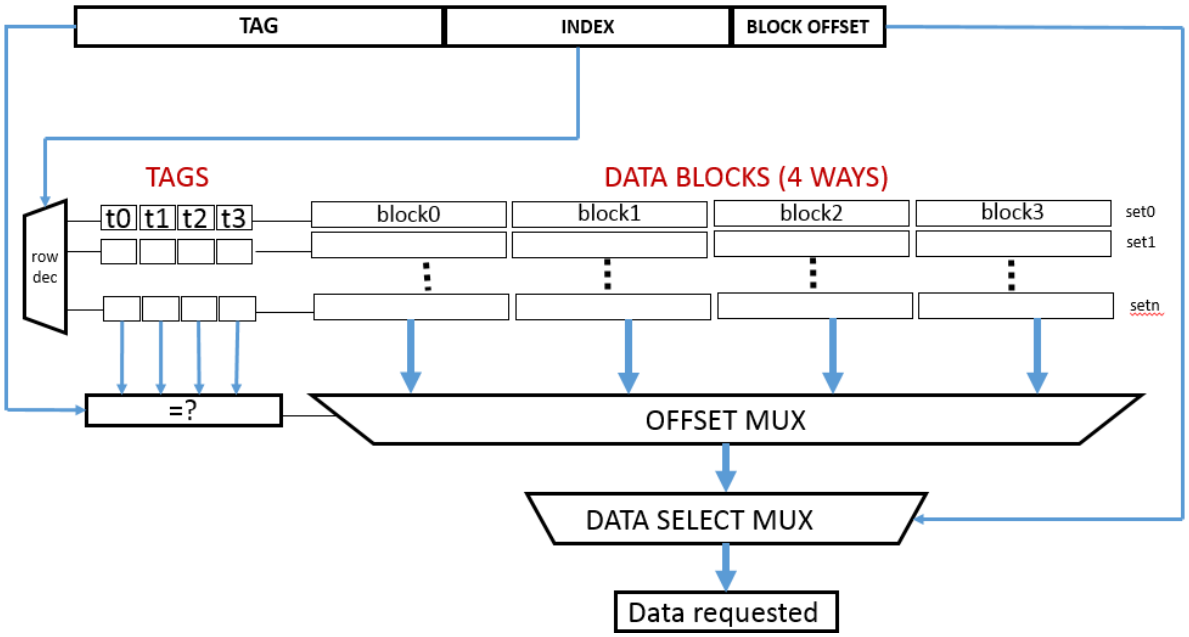


Figure 2.6. 4-way set associative cache

2.6. Cache Performance Optimizations

There are three factors used to determine cache performance: hit time, miss rate, and miss latency. All optimizations techniques used today are targeting a reduction of one or all these performance metrics. Hit time is the time it takes to access data when it is present in the cache. It is a function of configuration, circuit design, and technology. Miss rate is the ratio of the number of access misses to the total number of accesses. It is a function of cache configuration and program characteristic (e.g. memory intensive will have high miss rate). Miss latency is the time it takes to get data from next level of hierarchy into the cache after a miss. It is also a function of cache configuration and circuit optimization.

2.6.1. Reducing Miss Rate

Before talking about miss rate reduction, it's good to define different types of misses. *Compulsory* miss is the first reference to a memory block. *A capacity* miss occurs as the result of cache capacity or size and does not depend on cache configuration unlike *conflict* miss that depends on the number of blocks in a set (e.g. associativity). Conflict miss does not exist in a fully-associative cache. Few techniques are used to reduce miss rate. First, it can be accomplished by an increase in associativity, block size, or cache size. But there is a limitation beyond which there is a diminishing return. For instance, when increasing block size, spatial locality helps to bring neighboring data but beyond a certain threshold, data become polluted and increase miss penalty. Increasing block size reduces compulsory misses but can also worsen the other two type of misses in a small cache. Set associative cache has better miss rate than the direct mapping as it reduces conflict misses. Full-associative may have a slight

advantage over a 4-way associative of the same size but can dissipate much more energy and even increase hit latency. Therefore, in most cases, a 4-way associative cache is almost equivalent to having better miss performance than an extra-large set associative cache of the same size. Also, an increase in cache size reduces capacity misses but can increase hit penalty and chip area (e.g. high cost) and power. Therefore, to keep hit time down, smaller caches are a choice.

Second, *prefetching* can be used either in hardware or software. The purpose of prefetching is to predict which memory blocks may be needed and get them before the processor needs them. In hardware, additional logic block and registers are required to supplement cache. Prefetched instructions or data can be saved into cache or stream buffers. On a miss, the requested block is fetched and placed in the cache, and the next block is also fetched and placed in the stream buffer. So on the next demand, the stream buffer is checked for a match first before a request is sent to next level. In software prefetching, before the processor misses, software or compiler predicts the access and compute the address of an instruction and sends a request. Because the processor is not in immediate need of the instruction, a miss will not stall its execution. For such a miss, the controller can proceed in a way to overlap with normal execution. Like other techniques used above, prefetching can also cause cache pollution, meaning fetching useless data and possibly replacing the good ones, affecting the power negatively without a reducing in miss rate.

2.6.2. Reducing Miss Penalty

Additional cache level can be added to reduce miss penalty. It does two things. First, it keeps small cache closer to the processor to keep up with its speed. Second, it holds large caches in a low-level of hierarchy (e.g. L2, L3) to increase on-chip memory and avoid off-chip accesses. For instance, IBM Power 7 processor contains three levels of cache with 32MB last level in embedded DRAM [17]. With this large L3 cache, an implementation in SRAM technology would have required large area and cost. We will study in more depth throughout this document the advantage of large caches. Also adding write buffers can allow the CPU to continue normal execution while write operations are pending. It is understandable that write misses to level 2 cache do not contribute to overall miss penalty because write backs should not stall the CPU activities as this can be done in the background while executing other instructions. However, reads misses to L2 on the other side, whether they are part of load or store misses, they affect overall miss penalty.

CHAPTER 3

Three-Dimensional Integrated Circuits (3DIC)

3.1. Motivation

Integrated circuits (IC) are designed to keep up with Moore's law which predicted that every two years, the number of transistors will double [13]. Ever since the transistor size have been scaled down to reach high performance and low cost. But it is not without consequences. First, as the device size decrease to low sub-micro scale (e.g. 65,32,22,14 nm), the wire decreases in cross-section and becomes a very critical elements affecting not only the speed of the circuits but also the energy dissipation, power distribution, and reliability. The wire introduces different parasitics (resistive, capacitive, and inductive) which can be modeled as a lump or distributed elements. Equation 3.1 can estimate the propagation delay when considering only resistive and capacitive parasitics.

$$t_p = 0.38RC = 0.38rcl^2 \qquad (3.1)$$

Where R, C are respectively total distributed resistance and capacitance and r and c are respectively per unit length resistance and capacitance. So R=r*l and C=c*l. The square relationship of length justifies the rapidly increasing total propagation delay at next technology node. Global and semi-global wires mostly dominate this increase. In contrast, delays at metal 1 and intermediate wires have little influence as their lengths are usually also scaled down accordingly.

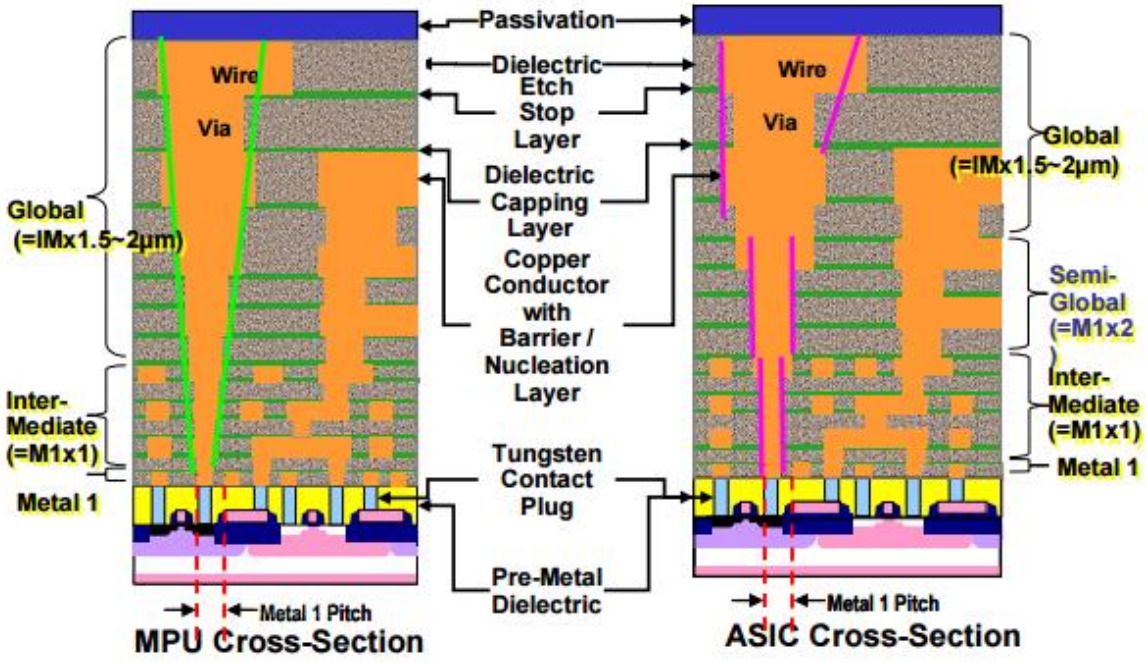


Figure 3.1. ITRS Cross-section of wire scaling.
From low metal level to global wiring, Metal 1 and intermediate wires have low-pitch and high congestion increasing capacitance and hence power consumption [5]

However, at this low-level metallization, the capacitance is dominant due to congestion and low pitch wire as shown in Figure 3.1. This growing capacitance also increases power consumption according to CV^2f relation. At semi-global and global levels, insertion of repeaters to reduce delay increases area and hence contribute to power consumption increase. According to ITRS, for process technology less than 22nm, roughness in via sidewall barrier, and copper surface will all negatively affect electron scattering in copper lines and cause an increase in resistivity [5]. Figure 3.2 shows that the resistivity of Cu will continue to increase for deep submicron technology.

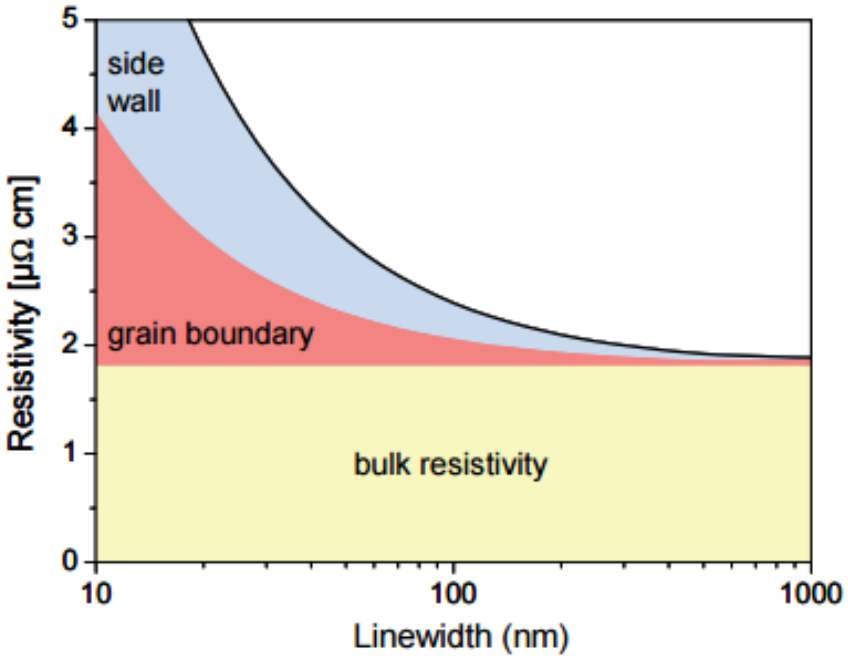


Figure 3.2. ITRS roadmap for copper (Cu) Resistivity [5]

Second, the demand for performance requires more devices to be confined while the chip increases in size to accommodate for more functionalities. As a result, the total interconnecting wire also grows in length. These two elements increase the wire resistance and hence the RC propagation delay. Therefore, a significant portion of power consumption of the chip is attributed to interconnects [6]. Also, the growing demand for functionality and high productivity has given rise to system-on-chip (SoC). SoC allows circuits of disparate technology and mixed-signals such as embedded DRAM, analog, low-power and high-performance logic blocks to be combined in a single chip. This combination causes numerous issues including complexity in design tools, interference noise caused by RF signals and high-

frequency digital circuits. All these challenges mentioned above are the motivations for the development of three-dimensional integrated circuits or 3DIC [6] [11] [18] [19] [50]. It refers to vertical interconnection of different circuit blocks into many layers. From equation 3.1, resistance (R) and capacitance (C) are increasing with technology scaling. The only way to slow down the growth of delay is to reduce length. As mentioned above, the 2D SoC demand does not help with this either. So this question can be answered by 3DIC.

3.2. Advantages

3DIC technology has the benefit of reducing global interconnection length. It allows for the reduction of parasitic and hence reducing energy consumption [49]. The average reduction of length is by a factor of $\sqrt{N_{tier}}$ [11]. Where N_{tier} represents the total number of layers. Given the linear relationship between power and wire length, the average power consumption would assume to have the same reduction factor. Because of the vertical stacking, the increase in volumetric density allows the incremental density of circuit elements (e.g. cell, MOS devices). It makes it possible to increase the capacity in the same amount of surface area. This advantage is evident in the design of stacked memories [25] [31] [43] [44]. Many memory cells can fit in a small area making memory attractive to various architectural innovations. In this dissertation, we will study the application of stacked DRAMs for cache purpose. We will explore their advantages over standard 2D memories on bandwidth and energy consumption.

Another advantage of 3DIC is the heterogeneity. It is the co-existence of blocks fabricated with different technology processes into a single chip. Unlike SoC, these blocks can be placed in different layers. It helps achieve better circuit performance and better noise control

between analog and high-performance logic circuits. When designs for various purposes (e.g. analog/digital/memory) are fabricated separately in their best appropriate technologies, there is better yield, optimization, and performance for each. Even on a digital portion, 3DIC can also allow a multicore processor to contain small and large cores [36] which can perform at different frequencies. It can also integrate cores fabricated in various transistor sizes [48] (e.g. 65 nm and 32 nm) into single chip but different layers.

3.3. 3DIC Integration Process

3DIC has different integration processes. Among these are the vertical interconnect through bond wires, and the most interesting is the stacking with through-Silicon Via (TSV). There are four different methods for creating 3DIC chips [50]. Among those methods, there are **chip stacking**, **transistor stacking**, **die-on-wafer,** and **wafer-on-wafer**. *Chip stacking* refers to connecting standalone chips vertically with wire bonding and flip-chip to reduces the area overhead. *Transistor stacking* refers to placing multiple levels of MOS devices on a single substrate. It is the most invasive integration used to contribute to reducing subthreshold leakage in nano-scale CMOS devices [51] [52]. But the thermal challenges are the main limitations because any high temperature during process destroys the devices and metals. In *die-on-wafer*, dies are bonded on top of the un-diced wafer using organic glues or metal bonding. The interconnect can occur on die edge, face or through silicon. *Wafer-on-wafer* allows bonding of un-diced wafers through the substrate. Its alignment accuracy is better than die-on-wafer and reduces cost. Both die-to-wafer and wafer-to-wafer can use direct bonding technology [63] [64] [65]. This technology consists of steps such as chemical mechanical polishing (CMP) applied at room temperature to reduce surface roughness. Also, it uses a high-temperature heat

treatment process to strengthen the bonding between the surfaces. Die-on-wafer and wafer-on-wafer can use through-silicon Via (TSV) for interconnect. There are three different orientations for stacking: face-to-face, face-to-back, and back-to-back. Face and back are referred respectively as connecting from the top metal or the substrate (e.g. below metal 1).

The figure 3.3 (a) (b) (c) shows the 3 topologies. Face-to-face is mostly used to connect logic tiers using metal bonding such as Cu-to-Cu thermo-compression process. Such process is used by Tezzaron's FaStack [53]. Because this process requires putting chips through harsh conditions of pressure and high temperature, there are other low-temperature and low-pressure [55] [56] thermo-compression. There are also non-thermo-compression methods that use room temperature to produce high-yield throughput [54]. Thermal compression can also apply to other metal bonding such as Cu-SnAg through a process called self-assembly. This process uses a high-precision alignment of chip-to-wafer or wafer-to-wafer [61] [62]. Face-to-back uses TSV to connect logic layers to memory for instance. In back-to-back, TSVs have to go through two substrates increasing its length and parasitic, therefore making it the least preferred technique.

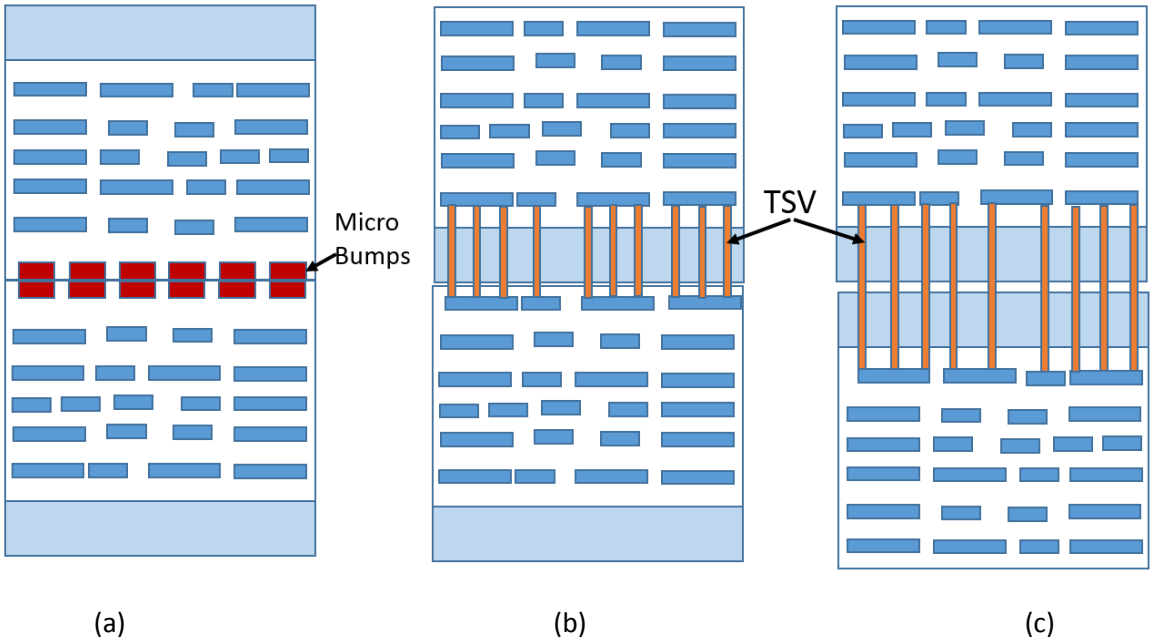


Figure 3.3. 3D stack orientation.
(a) face-to-face, (b) face-to-back, and (c) back-to-back

3.3.1. TSV

Through-silicon via (TSV) is the interconnect of choice for fabrication of 3DIC chips. The difference between regular via and TSV is that the former is an inter-metal layer within a die, but the latter is the pathway between top-metals of separate dies. It is a cylindrical tungsten or copper filled via that has full electrical characteristics such as resistance, capacitance, and inductance. These features depend on physical specs and material characteristics such as length, cross-sectional area and metal conductivity [57] [58]. It is important to study the electrical characteristics of TSV because they affect the performance of 3DIC. TSV is now very popular because it helps to reduce the interconnect length in semiconductor chips by replacing wire bonds and reducing parasitic and noise. Figure 3.3 shows the use of TSVs within different topologies.

3.3.2. 3D vs. 2.5D

The most known three-dimensional fabrication processes are full 3D stack and 2.5D using silicon interposer. The former helps connect multiple chips using TSV that pass through active elements. Whereas the latter only allow TSV to go through a passive platform called interposer. It means that there are no TSV through the active chips [66]. Both processes avoid the use of long and slow bond wires that limit interconnect density. Full 3D process relies on an intensive use of TSV to connect active elements within a die as shown in figure 3.4. It creates numerous issues including thermal dissipation challenges as described in next section. Although there is various research in the area of the heat problem, a full production of 3D chips is not yet a mainstream concept.

However, the non-ideal alternative to full 3D but more realistic is the 2.5D process. It helps avoid the issue created by TSV into active dies. In this process, dies fabricated in different process and technologies can be glued together on a passive silicon interposer which in turn serve as connections to package using TSV as shown in figure 3.5. Because all TSVs are on a passive silicon substrate, the thermal issues can be avoided by applying well know heat dissipation technics used in standard IC design. Chips are flipped and connected to the interposer with micro-bumps. The interposer, in turn, is connected to package substrate using TSV. The most popular type of interposer is the silicon interposer [67] [68]. Recently there was research conducted on glass interposer that utilizes through-glass vias (TGV) [69]. TGVs are similar to TSVs, but they are crossing a glass substrate instead of silicon. The motivation behind this is that glass is an excellent insulator that provides low signal loss. It can increase performance and reduce the size of the system [70].

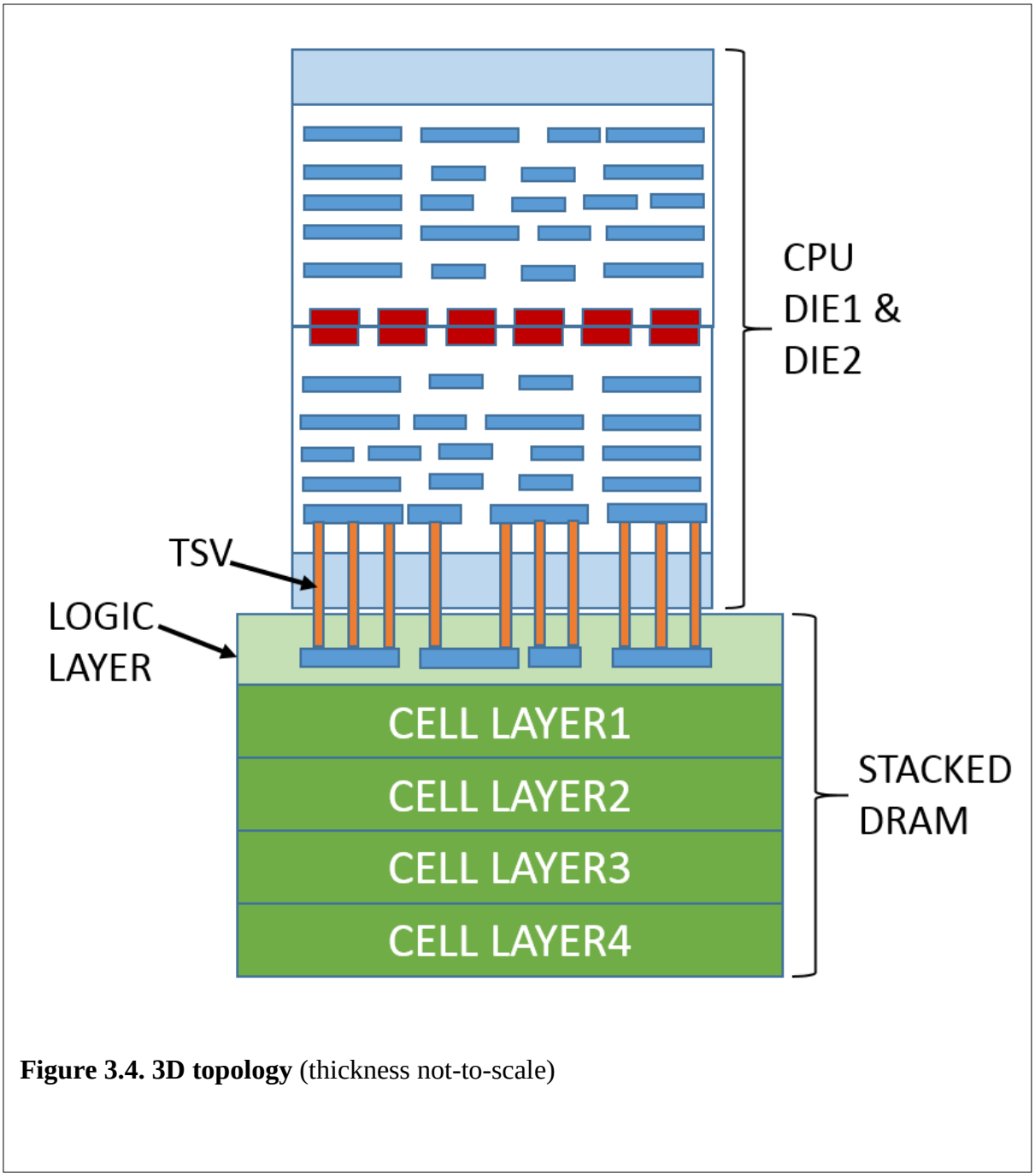


Figure 3.4. 3D topology (thickness not-to-scale)

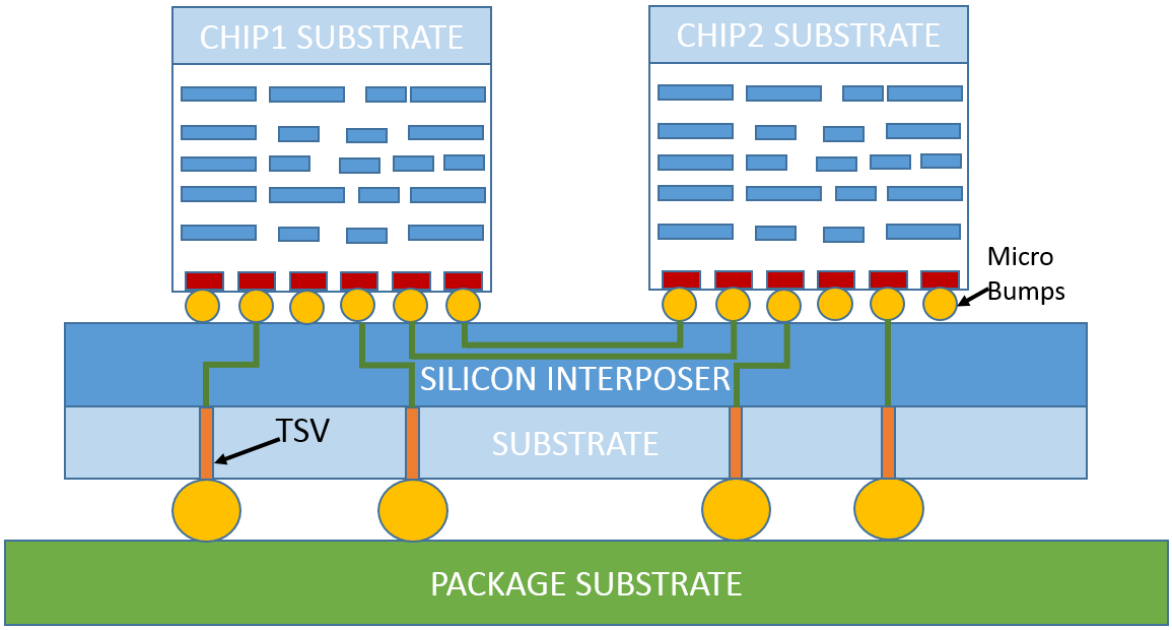


Figure 3.5. 2.5D topology

3.3.3. Die-Stacked DRAM

Die-stacked DRAMs are a type of memories that combines multiple layers of memory cells using vertical interconnects. These dense cell layers are fabricated in density-aware DRAM NMOS processes and sit on top of CMOS logic layer. This bottom layer contains multiple peripheral circuits such as sense amps, decoders, row buffer cache, drivers, receivers, and various custom logic and optimization circuits. Unlike 2D DRAM and embedded DRAM [37], there is a distinct advantage to separate the two technologies for better performance. As logic layer when designed in CMOS technology, the performance is much higher than when fabricated in NMOS DRAM process. Conversely, when ato performance-aware CMOS technology integrates embedded DRAM, there is a loss of density and yield [38].

3D memories with TSVs are more of reality than just research concept. In recent years few prototypes had been fabricated and tested promising a wide-spread commercial use of 3D technology soon. In 2010 Samsung published their first fabricated stacked DRAM [25]. It is 11 x 9 mm 8 Gb memory made in 50 nm technology. Each of the four layers represents a separate rank that consists of 8 banks each. About 300 TSVs with 80 um pitch and measuring 30 um in diameter were used to connect these layers. Their post-silicon results show a reduction in standby and active power by 50 % and 25% respectively compared to a conventional quad-die package. In 2011, they also presented a fabricated DRAM that stack two layers of 512 DQs 1Gb wide I/O DRAM for a total of 2 Gb memory [31]. Both micro-bumps and TSVs connect the 2 RAMs. Each TSV is 7.5 um diameter and about 48 fF of capacitance.

In 2012, Micron published papers for their new stacked DRAM called Hybrid Memory Cube (HMC) [43] [44]. This 1 Gb memory built on 68 mm2 surface consists of 4 cell layers partitioned into a total of 128 banks. These cell layers are stacked on top of a logic layer that

contains all peripherals to controls all memory operations. For interlayer interconnect, they used approximately 1866 TSVs separated by 60um pitch. They approximated its power consumption at ~10pJ/bit. In 2014, they released a 2GB HMC in a 31 x 31 mm package [85]. Today The Hybrid Memory Cube Consortium (HMCC) is a group of technology leaders including developers, manufacturers, and earlier adopters who enable HMC technology. Its goal is to develop a standard protocol for integration of HMC technology into a wide variety of applications. The latest and recent revision of HMCC specs leverages 18.5 x 22 mm to 34 x 34 mm packages with a density of 4GB and 256 banks to 8GB and 512 banks partitioned vertically. Each package comes with 4 to 8 cell layers [86].

3.4. Thermal Challenges in 3DIC

Despite the many advantages of 3DIC over 2D technology, vertical stacking has its challenges. 3DICs produce high heat density than conventional ICs causing thermal issues due to relative positions of circuit layers. The thermal problem in 3DIC is a well-researched topic because heat dissipation is a tremendous challenge for the widespread production of this technology. Active circuits elements sandwiched within each other prevent heat removal methods from working efficiently. Even in 2D chip, there is already thermal issues caused by transistor switching as the frequency increase. But in this case, the heat is conducted to heat sink through the substrate and dissipated with different cooling methodologies. During 3D stacking, only one layer is closer to heat-sink leaving other heat dissipating layers far away. Therefore, elaborate heat removal methods are required to prevent performance reduction in 3DIC, which can even negate the above-given advantages.

Studies in the recent years have proposed thermal architectures. To reduce the temperature of the isolated layer and heat-sink layer, Zhang et al. [10] proposed a design consisting of an air gap and a thermal bridge. They claimed a reduction of temperature from 75 C to 36 C in the memory layer, and down to 60C in processor layer compared to conventional air-cooling. C. Santos et al. [71] proposed a potential cost-effective thermal management method by placing graphite-based heat spreaders either on top of exposed stack or between the stacks. They claimed a temperature reduction down to 45 C. Matsumoto et al. [72] also proposed the used of graphite-based material in dual-side cooling. The conventional cooling is used from the top of the stack while a graphite sheet is placed at the bottom of the

stack to ensure cooling from the bottom side of the chip. Although many types of research are underway, thermal management in 3DIC is still a huge challenge.

CHAPTER 4

State-of-arts in Stacked Memory Applications

4.1. Overview

Few ideas can be explored to use the stacked DRAM efficiently for latency and energy optimization. There are few ways to utilize a stacked DRAM. A first way is to use it simply as a traditional main memory but with DRAM stacked on top of processor chip [30] [32] [33] [34] [35]. A second way is to use it as "on-chip memory" as an extension of off-chip memory to avoid tagging. It requires the operating system to manage physical address between the two memory regions. The OS can analyze page usage and determine which ones need to be mapped in on-chip portion of memory. Remapping pages is not a straightforward operation that requires software development. Software development can take a substantial amount of time making this solution a hardware-software issue. The third way is to consider the stacked DRAM as a shared last level of cache [1] [2] [3] [4] [7] [20] [28]. Some proposed a combination of SRAM and DRAM cache into a hybrid last level cache to take advantage of SRAM's low latency and DRAM's high capacity [21] [26] [27]. In this scheme, the system will determine where to place an incoming allocated block.

The first thought is to maintain the separate SRAM for tag [22] [23], but this may require a large SRAM even bigger than most L3 cache in modern processors. Such a large SRAM can be slow and power hungry even to the point of negating the speed and energy saving provided by 3D stacking. Zhao et al. [46] proposed the use of combined on-chip 8-bit partial tag with the sectorial cache to reduce on-die tag space to 1.5 MB and maintain low miss

rate. But this works better for small DRAM cache such as 64MB as they used. For large DRAM cache, this tag overhead will rapidly increase. So the most efficient approach is to map the tag within the DRAM to eliminate the separate SRAM. In the next section, we will explore different recent proposals of mapping tag within DRAM and some techniques that can reduce hit latency and miss rate.

There are recent publications that advocated for the use of large row buffer (4KB or 8KB) for page-based mapping [23] [24]. But we will only analyze in-depth the different DRAM cache block-based mappings that take advantage of the conventional 2 KB row size DRAM. With up to 32 blocks in 2 KB row, some authors have used the entire row as a single set providing an extra-large associativity to take advantage of spatial locality and reduce conflict miss. Others have mapped multiple sets inside a single row. Amongst those, some have made every block to represent a set making it direct mapped, but others have separated small and equal groups of blocks into multiple sets with reduced associativity. Each mapping style has its advantages and drawbacks as will be analyzed throughout this document. In the next few sections, we will explore these proposals that target different goal. Miss rate and Hit latency are the two parameters that all of them are targeting to improve performance. In section 4.2, we explore proposals that map tag in DRAM. Among them, some focus on reducing miss rate, some on hit latency, and others on both. In section 4.3, we explore proposals that map tag in SRAM or use both SRAM and DRAM as last level cache. In section 4.4, we will talk about the page-based cache.

4.2. Tag mapped in DRAM

4.2.1. Single Set Single Row High Associative (SSSR-HA)

Loh and Hill [1] [2] proposed a tag-in-DRAM cache where three blocks contain tags, and 29 blocks include data all within a single row as captured in figure 4.1 (b). This compound mapping avoids opening a new row in case of a hit as row-buffer hit is guaranteed. Each block consists of 64B for the total page or row of 2KB. The motivation behind this scheme is to avoid a large external SRAM for tag store as captured in figure 4.1 (a). For instance, a 256MB DRAM will require about 24MB of tag space which is already larger than last level cache of most processors. Considering the nature of SRAM, such a large tag structure will create extra overhead in area and also add additional latency to overall access time. Combining tags and data within the DRAM eliminates this cost although it reduces the cache capacity (e.g. 232MB with 24MB dedicated for tags). Therefore in this work, we will consider tag-in-DRAM to reserve a large tag per block to account for future scalability.

This configuration is used as a baseline by many DRAM cache publications as it is one of the first proposed tag-in-DRAM comsbination. With this scheme, three 64B memory transfers will be performed for tags (*RD1*) when using 48-bit per tag and one 64B transfer for data (*RD2*) adding up to 4 transfers per request. After RD2, data is ready to be returned to the requestor. Tag update (WR) also requires additional three transfers that increase energy consumption. The high associativity of 29 ways has the advantage of reducing miss rate, but the serialization of tags (e.g. three separate tag blocks) increases hit latency.

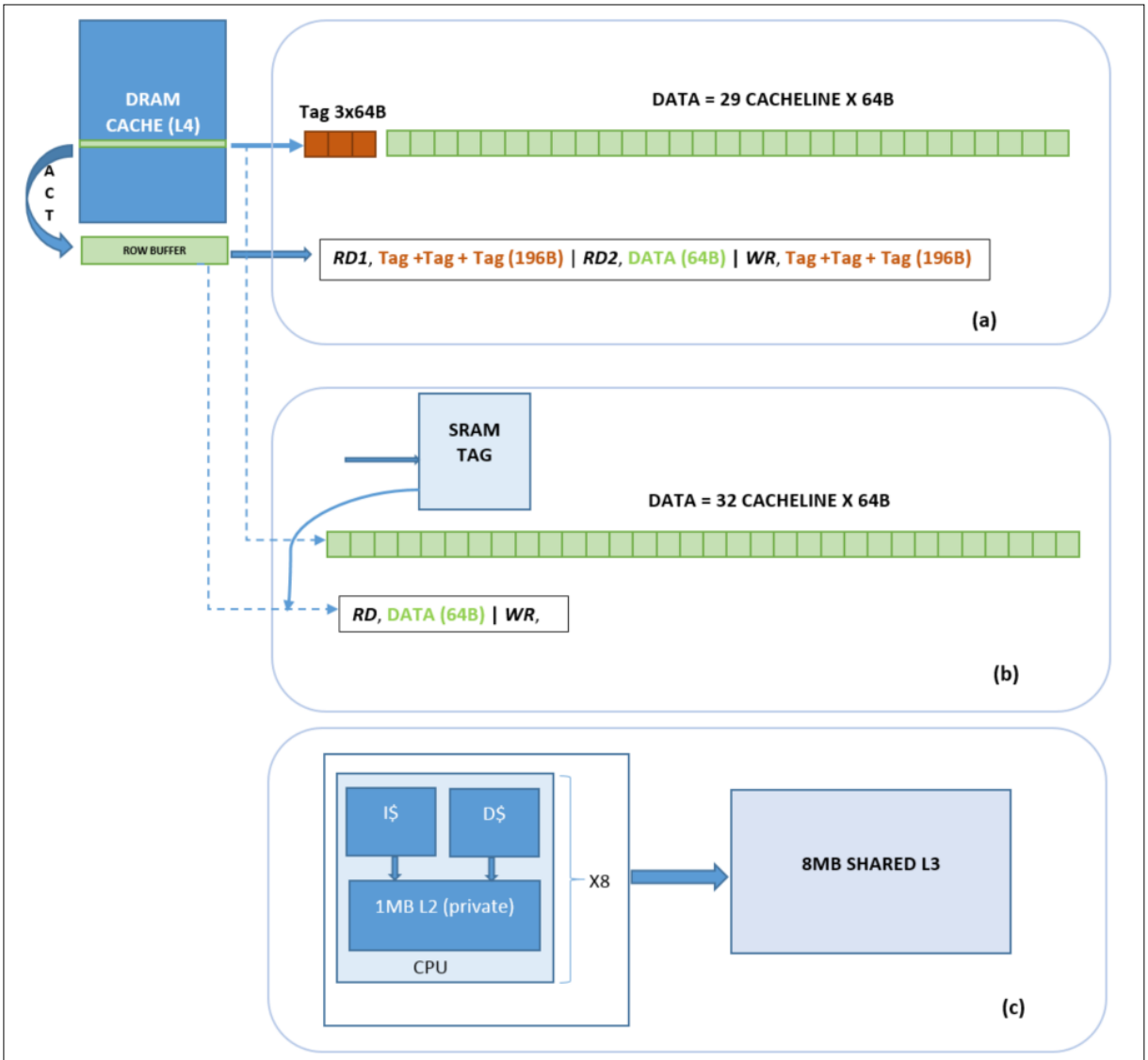


Figure 4.1. SSSR-HA: The Loh-Hill configuration.
(a) **Loh-Hill configuration**:
Tag + data are mapped in DRAM for a compound access. Three access for tag and one access for data after activation (ACT). Data area is reduced by the amount of tag needed (e.g. three tag blocks for every 29 data block).
(b) **Impracticable SRAM tag**: This maximizes data area by using a separate SRAM for tag. The size of tag area becomes unsustainable as the cache increases (e.g. 24MB tag for 256 MB DRAM cache).
(c) **Baseline memory hierarchy**: 8 CPUs with private L2 all served by a shared SRAM L3.

To avoid these many tag transfers during a cache miss, they also proposed a miss predictor implemented in a data structure called Missmap. The intended purpose of Missmap is to avoid DRAM cache access on a cache miss and therefore removing few cycles from miss penalty. By predicting if the request will be a miss, tag access will not be a prerequisite to the main memory request. They based the idea of missmap on the purpose of tag lookup: the first use of tag access is to determine the existence of data in the cache. The second goal is to identify the physical location of the data in the cache if it is a hit. They implemented the missmap to fulfill the first purpose of data residency while tag located in DRAM fulfills the second purpose of data location. At first, it makes missmap behave like a tag, but the big difference is that tag bits are replaced by a single bit.

The bit is asserted when a new tag is allocated and cleared when existing tag is evicted. Using one single bit allows for many tags to be tracked outside the DRAM cache. But because a cache as large as 256MB can contain many millions of cache lines, this missmap can quickly reach the order of megabytes. Their proposed improvement involves an additional large SRAM for their experiment setup. They suggested using a portion of L3 cache for missmap space (e.g. 2MB on 8MB L3) as captured in figure 4.2. It increases overall access latency because each request still selooks up a large L3 whether it's a miss of hit. It is important to mention that this missmap is simply a predictor, meaning that there can be a bad prediction. Either false positive (e.g. predict data present in DRAM cache, but it is not) or false negative (e.g. predict data is not present, but it is).

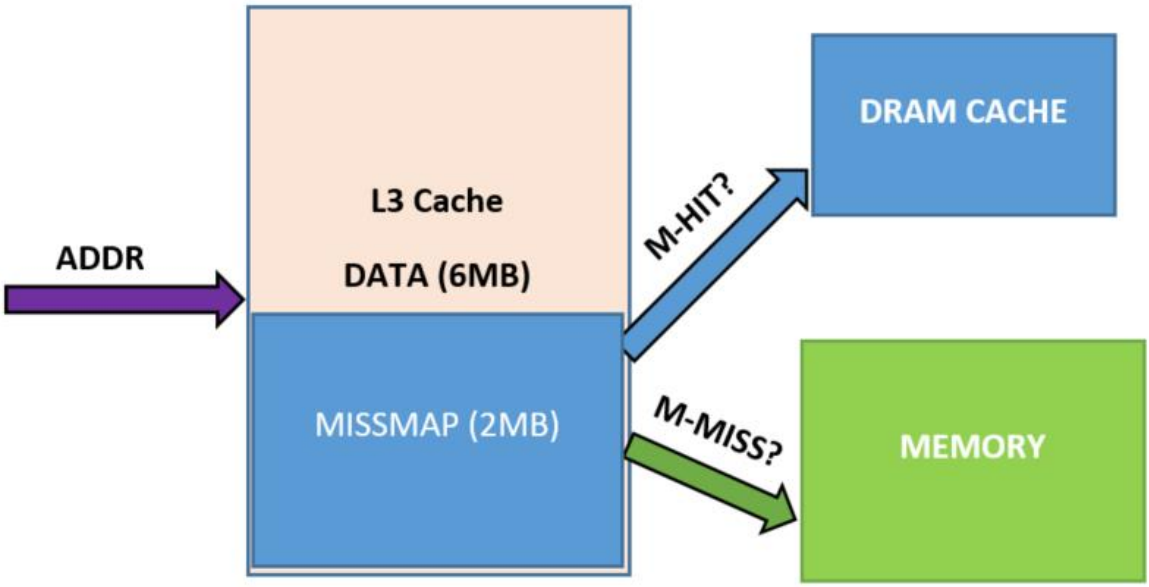


Figure 4.2. Loh-Hill DRAM cache access using MissMap. A portion of L3 is used to store tag residency states. If new tag is allocated MissMap bit is asserted. If tag is evicted, MissMap bit is cleared. Each L3 miss will go through missMap lookup to predict its presence in DRAM cache. If hit, access will be sent to DRAM cache. If Miss, access will be redirected to memory

The false positive cannot cause harm to program execution. But it can directly cause a performance degradation due to unnecessary DRAM cache tag look-up. In contrast, the false negative can cause damage to program execution. The off-chip data return can overwrite dirty block without proper writeback. To remedy this issue as well as the size overhead, they proposed a small predictor similar to a branch predictor. This predictor has a self-balancing dispatch that can speculatively send a request to off-chip regardless of the correctness of decisions [39]. They claimed to have observed performance improvement due to this predictor, but we are not implementing any predictor in our methodology.

According to their simulation setup, their proposed compound-access as depicted in figure 4.1 (a) combined with missmap as described in figure 4.2 produced a speed-up of 1.55 on average compared to baseline memory hierarchy as outlined in figure 4.1 (c). The impractical SRAM-tag of figure 4.1(b) provided better improvement at 1.65 speedup due to the fact that 100% DRAM space is used for data area as all the tag space in mapped in external large SRAM. This comparison is for evaluation purpose only to show that tag-in-DRAM can reach within 90% performance of ideal case while remaining practical and cost efficient.

One more thing to mention is that even though the missmap may speed up memory access on a DRAM cache miss, the tag access cannot be entirely bypassed as they claimed. Because besides the two tag lookup purposes mentioned above, evicted block need to be determined and also if dirty, the write back is to be scheduled. Therefore, the tag access will still occur although outside the memory access critical path. But energy consumption within DRAM cache is not avoided because of miss predictor. It is the additional reason why we are not implementing any predictor in our methodology as we are also focusing on the energy consumption within the DRAM cache, not just performance.

4.2.2. Multiple Set Single Row Direct map (MSSR-DM)

Qureshi and Loh [3] asserted that a separate tag and data access as proposed by SSSR-HA is justifiable for conventional SRAM cache design because Tag and data areas are physically separate structures. But this is not optimal for stacked DRAM given that tag and data are co-located in the same structure. Therefore, they proposed an alloy-cache which is a structure that compounds tag and data into a single entity called TAD (e.g. Tag and Data) as shown in figure 4.3. One TAD is considered to be a separate set turning one DRAM row into a group of direct-mapped sets. One access will fetch a single TAD which contains both tag and data. Doing this can significantly reduce hit latency because tag serialization is eliminated.

In addition, because of this direct-mapped organization, many consecutive sets are located within a single row and therefore, increases row-buffer hit. A TAD consists of 64B of data and 8B of tag for a total of 72B per access. One drawback with this scheme is the use of only a direct mapping which can increase miss rate due to conflict. But this increased conflict miss can be offset by an increased Row-buffer hit due to the proximity of sets within a row. To realize this scheme, a DRAM with the burst-5 mode is needed. For a 128-bit bus size, 80 B will be fetched, but it only needs 72 B. The extra 8 B will always waste energy. In addition, these lost 8 B will be located either at the beginning of the block or at the end of the block depending on whether the block number is even or odd to avoid unaligned accesses.

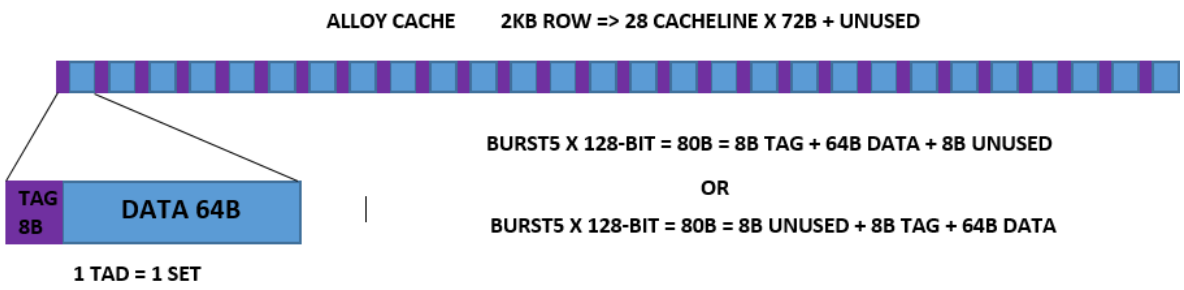


Figure 4.3. MSSR-DM: The alloy-cache. 28 sets of 72B are mapped within one single row for a direct-map configuration. Using burst-5 mode, the first 8B can either be unused or used as tag to maintain aligned access. 1 TAD consisted of 64B data + 8B tag.

Their first target of optimization would be the hit latency rather than hit rate. They argued that design proposed in the past that targeted hit rate were not suited for overall optimization even though the increased associativity also increases the hit rate. So to reach their primary target of hit latency reduction or latency-optimized design, they proposed this alloy-cache access to reduce hit time. If it is a cache miss, tag and data both are still transferred which still incur some penalty. So they proposed some type of memory access predictor to compensate for this. They asserted that their predictor was so straightforward and small enough to only add one cycle delay but yet still get performance within 2% of a perfect predictor as laid out by missmap in Loh-Hill proposal.

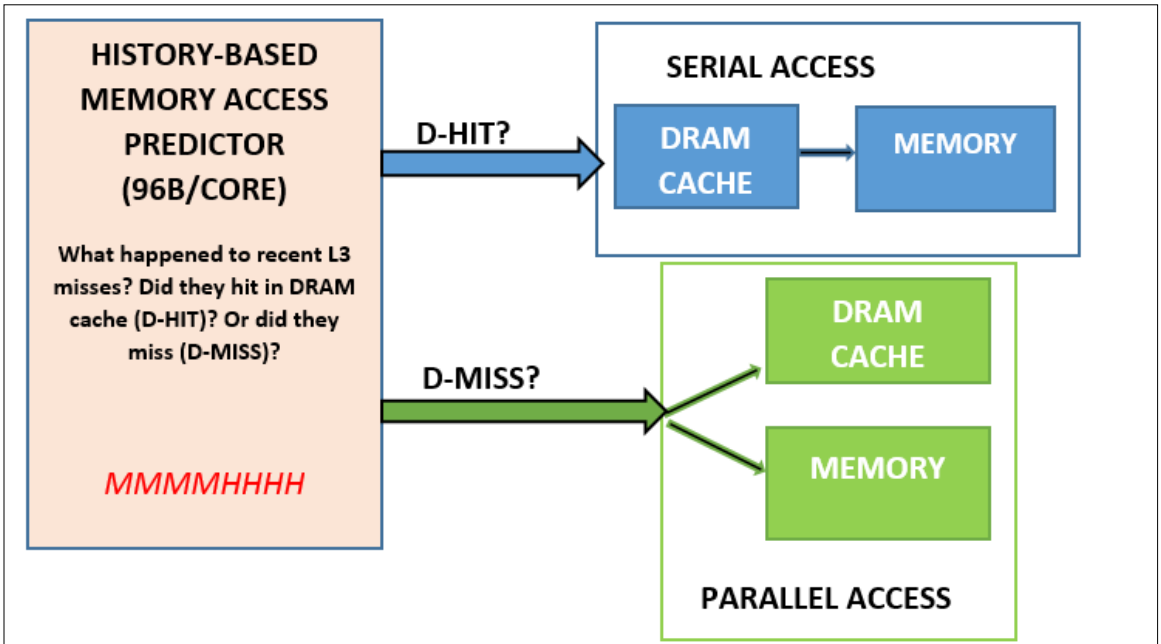


Figure 4.4: Memory Access Predictor (MAP): keep tracks of recent L3 misses. If predicted HIT in DRAM cache, Serial access is chosen. If predicted MISS, Parallel access is selected. Notice that MAP is a separate structure that keeps track of L3 miss history

Unlike Missmap, their proposed predictor only requires 96 bytes per core. They implemented this predictor by dynamically switching between serial access and parallel access models. The serial access refers to memory request *after* the cache miss is determined. The parallel model allows for memory request *while the* cache is being looked up as shown in figure 4.4. The serial access, as traditionally implemented in many cache design, prevent unnecessary memory accesses when the cache access is a hit. On the other hand, parallel models can reduce memory access time by bypassing the cache lookup when it is a miss. But when it is a cache hit, the unnecessary memory access wastes bandwidth and energy consumption. It also makes it an inefficient model if used without any smart predictor. With their dynamic switching and simple predictor, they claimed to have increased performance by selecting serial model when

L3 access is predicted to be a hit. If it is predicted to be a miss, parallel access model is selected. When using a 256MB of DRAM cache, they claimed an improvement of 25% over Loh-Hill proposal and 9% over SRAM-tag design. They attribute most of this improvement to their latency optimized design and predictor.

4.2.3. Multiple Set Single Row Set Associative (MSSR-SA)

Hameed at al. [4] argued that DRAM cache design goal should address a combination of both miss rate and hit latency improvement. Like MSSR-DM, they also mapped multiple sets in a single row but they used high associativity instead of a direct mapping. Each set has 7 cache blocks with its own tag block. Each tag consists of 6B for a total of 42B. Figure 4.5 describes their mapping.

They trying to achieve a reduction in *hit latency* by reducing the number of tag block to eliminate excessive serialization as seen in SSSR-HA. Instead of 3 blocks of tag to cover all 29 cache lines, they only used a single block to cover one-quarter of the total cache lines for a total of 4 sets. Because each set is a 7-way associative, they can reach their second goal of reduction of *miss rate*. In addition, since the 4 sets are located in a single row, this locality increases the row buffer hit rate.

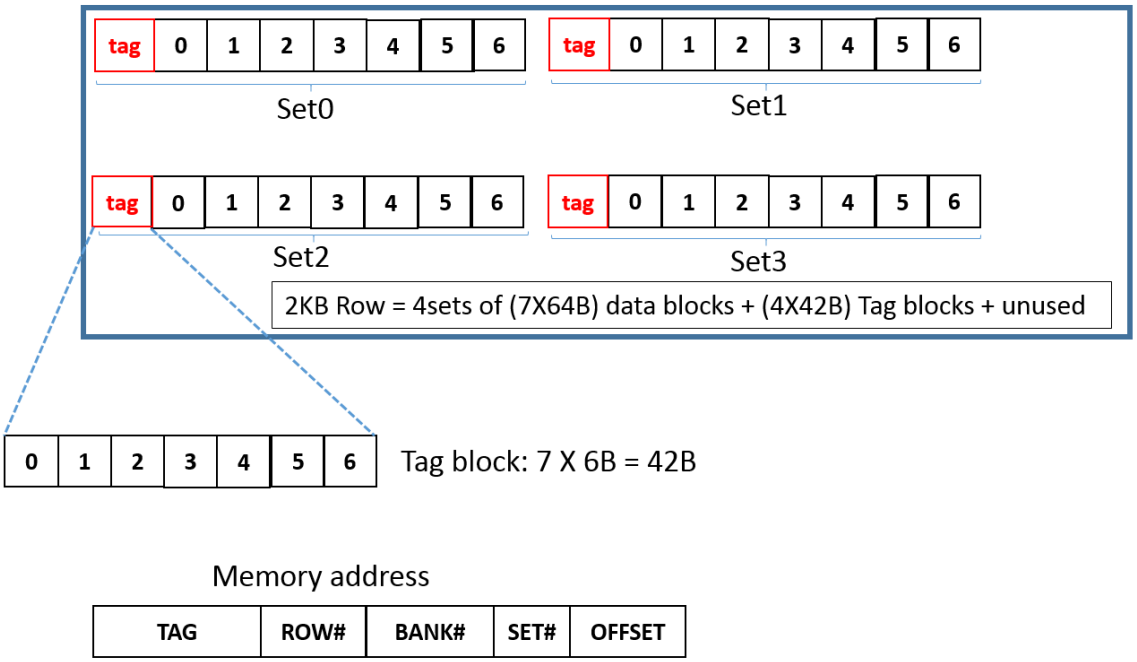


Figure 4.5. Hameed et al. MSSR-SA configuration based on 2KB row buffer [4]

4.2.4. Multiple Set Single Row Set Associative with 4K row buffer

4.2.4.1. MSSR-8S-7

Hameed at al [7] also proposed another configuration based on 4KB row buffer DRAM. Similar to their 2K row buffer design, their primary goal was to improve hit latency and miss rate. There are two variations to their design as shown in figure 4.6 and 4.7. First, they proposed a configuration similar to [4] by increasing the number of sets within a single row from 4 to 8 sets of 7-way associative. The main difference between this proposed configuration and the previous 2K version is the upgrade on the predictor. In the previous proposal, they used the missmap similar to Loh's. In this one, they used a DRAM Tag Cache structure (DTC) and its insertion policy to predict tag hit or miss. Again to simulate this configuration, we would normalize all the size to 256MB of pure data by adding 8 extra blocks for tags to the 4KB row to obtain 8 sets of 8 ways.

4.2.4.2. MSSR-2S-29

Second, they proposed a configuration similar to Loh's [1] [2] with high associativity of 29-way and 2 sets in a single row as shown in figure 4.7. They claimed that a high associativity will improve miss rate while set proximity will increase row buffer hit rate. The main reason why they proposed their configuration in [4] was to eliminate tag serialization and reduce hit latency. That is why it's hard to understand why they chose to increase tag serialization and significantly increase hit rate. The improvement in miss rate together with the increase in row buffer hit will not compensate the negative effect of tag serialization on hit rate. But most importantly, the increase of row buffer size from 2KB to 4KB will worsen the energy consumption per access at row buffer miss and even read and write access. Unless their

claimed advantages will supersede all these adverse effects, it does not look like this configuration will help improve performance and mostly not energy consumption.

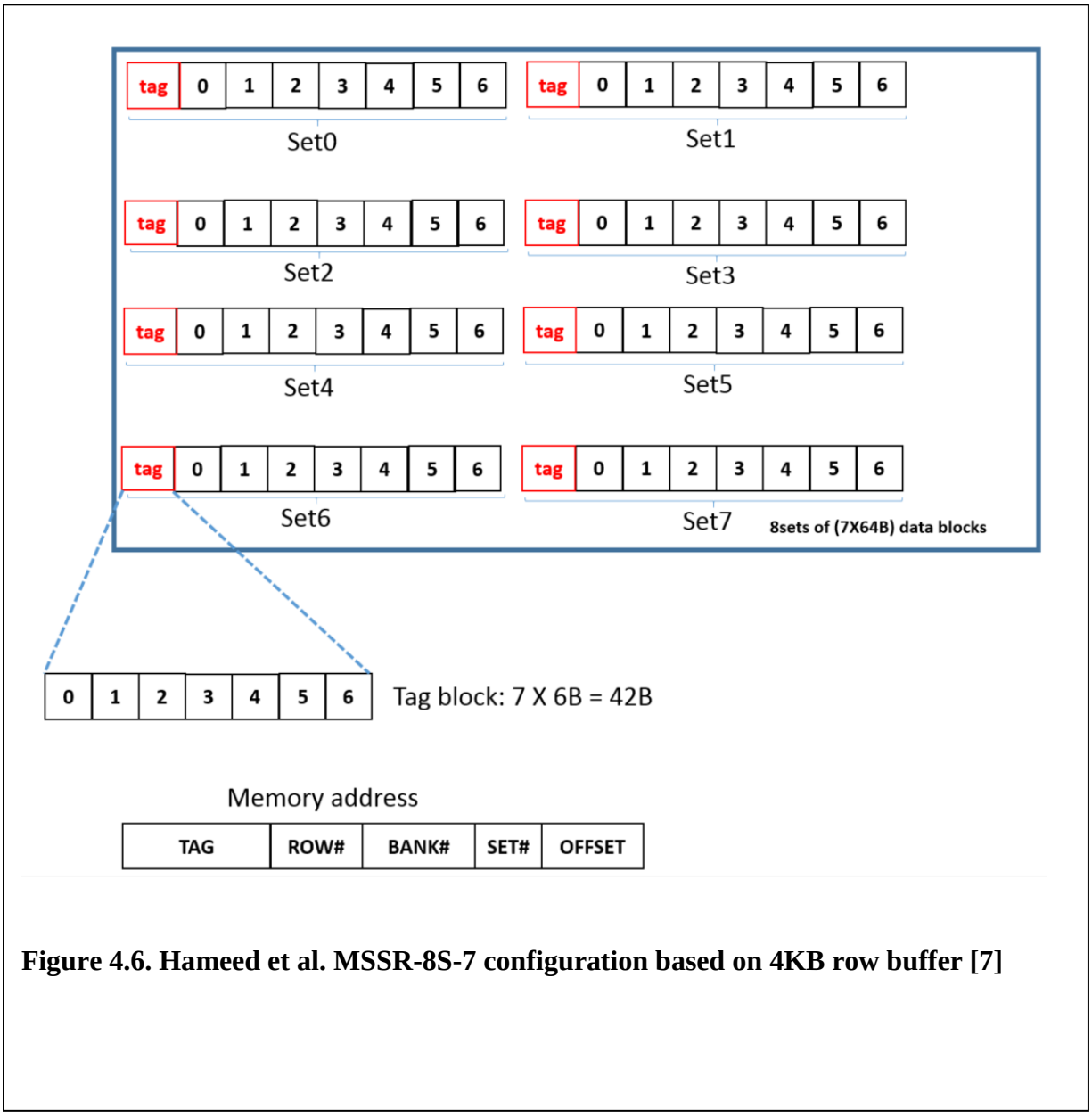


Figure 4.6. Hameed et al. MSSR-8S-7 configuration based on 4KB row buffer [7]

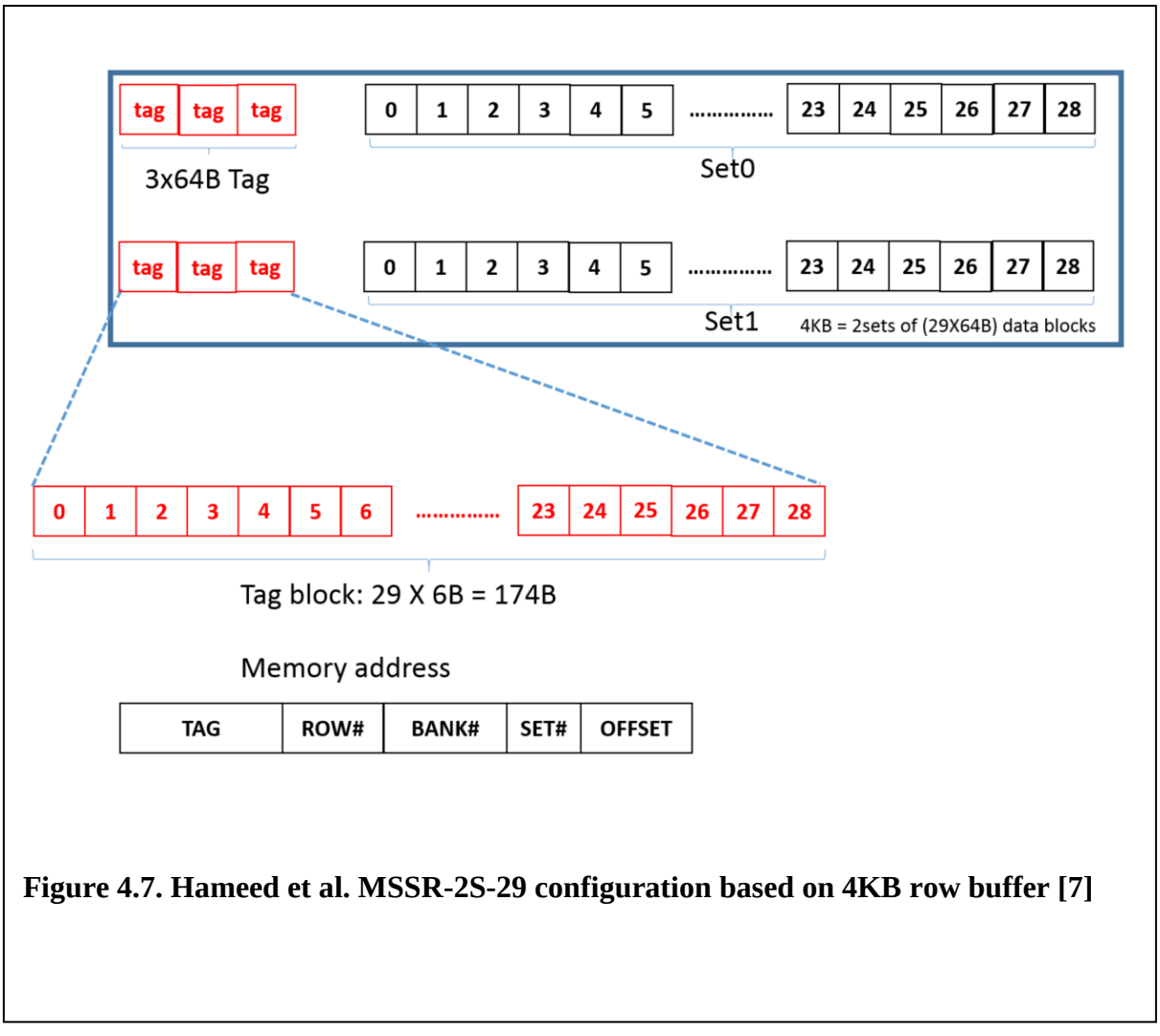


Figure 4.7. Hameed et al. MSSR-2S-29 configuration based on 4KB row buffer [7]

4.3. Tag-in-SRAM consideration

4.3.1. Tag Cache

The biggest challenge in designing DRAM cache is deciding where to map the large tag. One of the most popular solutions is to map it inside DRAM for its density and to avoid a large SRAM cache. However, in recent research, there was a proposal to keep tag outside DRAM to reduce access latency. Huang and Nagarajan [22] assessed that a tag-in-SRAM is still attractive if we consider a standard memory address size. For instance, a CPU with 40-bit address size with high associativity will only require 23-bit for a 256MB DRAM cache (17-bit tag per block plus 5 status bits), not the 6 to 8 Byte assumed by others [1][2][3][4][7]. In this case, only 11.5 MB of total area will be needed. Even though the size is less than half of the predicted tag-in-DRAM area, its physical dimension and cost are still higher due to the nature of SRAM technology. Therefore, instead of using SRAM to map tags, they proposed a hybrid method where full tag (but reduced size) is maintained in DRAM and a small number of tags is cached in SRAM structure called Aggressive Tag cache or ATCache. They only cache the tags of recently accessed sets to maintain temporal locality. In the same fashion, they prefetch tags of adjacent sets upon evidence of spatial locality to avoid cache pollution. Notice that they are proposing a tag prefetch similar to data prefetch. They assumed a DRAM configuration similar to [1] [2] [4] where 4 sets of 15-way are mapped in a 4KB row buffer DRAM. As 4 tags of adjacent sets are prefetched upon miss, a total of 4 tag blocks can be prefetched. If these tags are not used, there is tag pollution from fetching useless data. This prefetch may not incur the full latency of tag serialization because it may occur in background outside access critical

path. However, there is a considerable amount of energy consumed. This along with the use of a large 4KB row buffer makes this proposal energy-inefficient and does not fit our goal.

4.3.2. Hybrid SRAM/DRAM Cache

Always in order to find a better way to store tag, Inoue et al. [27] proposed a last level L2 hybrid cache consisting of a 2MB SRAM and 32MB DRAM cache. This adaptive configuration allows for two modes of operations. On the first operation mode, SRAM is used for cache while DRAM is disabled. For second operation mode, DRAM is used as cache and SRAM is used as its tag area (e.g. both units active). The choice of operation modes depends upon the type of workload to be executed. The second operation mode is, of course, appropriate for memory-intensive execution. If there is no need for large memory footprint, the first mode of operation can be selected to take advantage of fast access of SRAM. The authors assume that the mode of operation can be chosen before execution. Overall this proposal works for a small DRAM cache size such as 32MB as they assumed. But for a large DRAM cache, it involves implementing a broad tag array to determine the data location. For instance, a 64MB DRAM cache requires 5MB SRAM tag area which already begin to cost in term of area overhead and even latency.

Hameed et al. [21] [26] also proposed a hybrid SRAM/DRAM cache where both SRAM and DRAM are used as L3 last level cache at the same time. The main challenge is to decide where the incoming cache block will be saved. So they proposed an adaptive DRAM placement policy (ADP) that determines at runtime where to place the block. It can be either in both SRAM and DRAM or in SRAM only. The goal of this policy is to reduce the number of fill requests from applications that do not reuse them in the future. They called them

thrashing applications. Such application delay cache requests from non-thrashing applications through inter-core interference. On L3-SRAM miss, a request is sent to Missmap to predict whether the block is located in L3-DRAM. If so, the incoming data from L3-DRAM is used to fill both high-level caches from requester CPU and L3-SRAM. It increases the chance of a hit if same data is requested in the near future (e.g. temporal locality). If it is a Missmap miss, the block is not in L3-DRAM, and control is forwarded to main memory. The incoming data from main memory will be filled in L3-SRAM and in L3-DRAM only when a particular enable bit is 1. If not, only L3-SRAM will be filled.

The enable bit logic is done by some sophisticated algorithm that accounts for thrashing behavior of incoming requests. For more detail about how enable bit is set and reset see [26]. They maintained a complementary replacement policy between the two last-level caches (LLC). To invalidate data from all private caches, both an eviction from one LLC cache (e.g. L3-SRAM) and a miss from the complementary cache (e.g. L3-DRAM) must occur. In summary, the authors believe that not all incoming data should be filled in both L3-SRAM and L3-DRAM cache. If unnecessary L3-DRAM cache accesses are avoided, there is better performance.

4.4. Page-based cache

Jevdjic et al. proposed a footprint cache [23] and unison-cache [24] that both use cache line at the granularity of pages instead of conventional block size. In the block-based cache, cache lines are in order of few bytes up to 64B for most design. This is already larger than the CPU operation registers of 32 or 64 bits. The reason of fetching 64B is to increase spatial locality by pre-fetching neighbor data and hope that the processor will also use them. In Unison-cache they advocated the use of DRAM with extra-large row buffer of 8KB. Then, they mapped 2 sets of 4-way associative per DRAM row. Each way is a 1KB page. For footprint cache, they used the entire 2KB page as a single block. By doing this, they reduced the amount of tag space required (e.g. from 32 tags to 1 tag) and make tag-in-SRAM attractive. This limits the access to DRAM cache only for data by eliminating tag access altogether and reduce hit latency.

Another advantage of page-based cache is to increase hit rate by maximizing spatial locality. But an excessive prefetching of data will definitely create cache pollution where extra fetched data will never be used until the entire block is evicted. This is a waste of bandwidth and mostly a waste of energy. To remedy to some of these issues, they proposed a footprint predictor that predicts “all” the blocks that will be later used and only fetch those blocks. Any under-prediction or over-prediction will either cause an excessive miss or excessive energy consumption. So the predictor accuracy at 100% is the key component of their proposal. Such prediction rate is only ideal and is not easily achieved. Given our primary goal in this research to find energy efficient design, we do not explore in detail page-based configurations.

4.5. Chapter Summary

In this chapter, we have explored many state-of-arts proposals striving to find a better DRAM cache design for increased performance. We have looked at (1) block-based cache versus page-based cache, (2) tag mapped in DRAM versus tag mapped in SRAM or combination of both, and (3) hybrid SRAM/DRAM as both last-level cache (LLC) versus DRAM cache as the only LLC. Based on all the above options, we have made the following assessments: (i) we have estimated that page-based cache does not support our quest for energy-efficient DRAM cache and have adopted block-based cache for our proposal. (ii) We have also estimated that tag mapped in SRAM is not feasible for large cache due to excess area requirement. And finally (iii) we have adopted DRAM as last level cache for further evaluation.

We have explored in detail three proposals that match our assessment (i), (ii), and (iii). They are either focused on reducing miss rate or reducing hit latency. SSSR-HA proposed by Loh-Hill [1] [2] achieved miss rate reduction by using the entire memory row as single set. Many cache blocks can be obtained from this for a high associativity. But doing this also increased the tag area per set. It led to tag serialization as three blocks are fetched from row buffer. This configuration has the worst hit latency that is not entirely covered by the reduction in miss rate.

MSSR-DM proposed by Qureshi-Loh [3] achieved hit latency reduction by eliminating tag serialization and de-optimized miss rate with the direct mapping. Unlike SSSR-HA, they mapped multiple sets in a single row. They combined tag and data in one block called TAD so that one access of burst-5 mode fetched both tag and data and hence reduce significantly hit

access time. One drawback is the increased miss rate due to the direct map but this is covered by reduced hit latency and increased row-buffer hit.

MSSR-SA proposed by Hameed et al [4] [7] is a combination of both reduced hit latency and miss rate. They mapped multiple sets in a single row for a set associative to maintain a good miss rate. Since many sets are mapped to one row, the tag area needed by each set is reduced to one cache block. It reduces tag serialization and hit latency. It also increases row buffer hit rate.

All proposals have one thing in common: the use of large row buffer. The energy consumption of opening, closing, and reading from the buffer is higher in all cases. Our goal in this research is to find an energy efficient configuration that also keeps a good level of performance.

CHAPTER 5

Energy-Efficient DRAM-Cache with Unconventional Row-Buffer

5.1. Overview

The traditional 2 KB row can hold 32 blocks of 64B each. This large number of blocks allows for a variety of set mappings. As we studied in chapter 4, these mappings are designed to maximize associativity for a conflict miss reduction, or to minimize hit latency. Overall improvements in performance depend on which element the mapping is focusing. However, the one common fact among all set mapping in 2 KB, 4KB or 8KB regimes is that they consume a high amount of energy per access during row buffer miss. We have analyzed different configurations mapped in 2KB row buffer such as high associative (SSSR-HA), direct map (MSSR-DM), or multiple set associative (MSSR-SA). For a 2 KB row buffer memory for instance, at row activation and precharge, all 16384 bits are copied from RAM cells to the row buffer or vice versa consuming the same amount of energy regardless of the mapping policy used for cache. Therefore, a reduction in row size can be viewed as a better way to ensure low consumption on energy in these structures. In this chapter, we will study a possible low-energy configuration based on 512B row buffer size to demonstrate our concept. Then in rest of chapters, we will explore the remaining low-energy row buffer sizes.

5.2. Low Energy Stacked DRAM consideration

To account for energy efficiency, we believe that reducing the row buffer size of DRAM can provide a net advantage in energy consumption. In this section, we are describing a possible 3D DRAM memory that can fit our purpose. To get to that, we have to recall first how data is read from and written to a DRAM. Each RAM is divided into small partitions called banks. Each bank is divided into rows, and each row contains a different number of bits depending on the architecture. At a read request, the entire row of data is copied to an internal cache called row buffer. From the row buffer, the requested amount of data is fetched into a burst of 2, 4, or 8. The energy consumption depends on the size of the row. Therefore, by reducing the size of the row down to 0.5 KB, not only the energy of activation and closing a row is reduced but also read or write operation energy is reduced. The overall power consumption in a DRAM cache, however, is also affected by the percentage of row buffer hit, but we will determine which one has more impact on energy than the other.

A stacked DRAM that can fit this goal can look like the structure in figure 5.1. This 4-layer DRAM contains two ports providing a bus size of 128-bit for high bandwidth. Each port is independently connected to 32 banks. There is not internal circuitry linking the ports operation. Each bank contains 8192 rows of 4 Kbit as shown in figure 5.2. With only 8 x 64B blocks per row, this seems to reduce flexibility that 32 blocks from a conventional 16384-bit row can provide, as we have studied in Chapter 4. Our task in the rest of this dissertation is to explore different possibilities that can come from this reduced row size and how much energy saving can result from it.

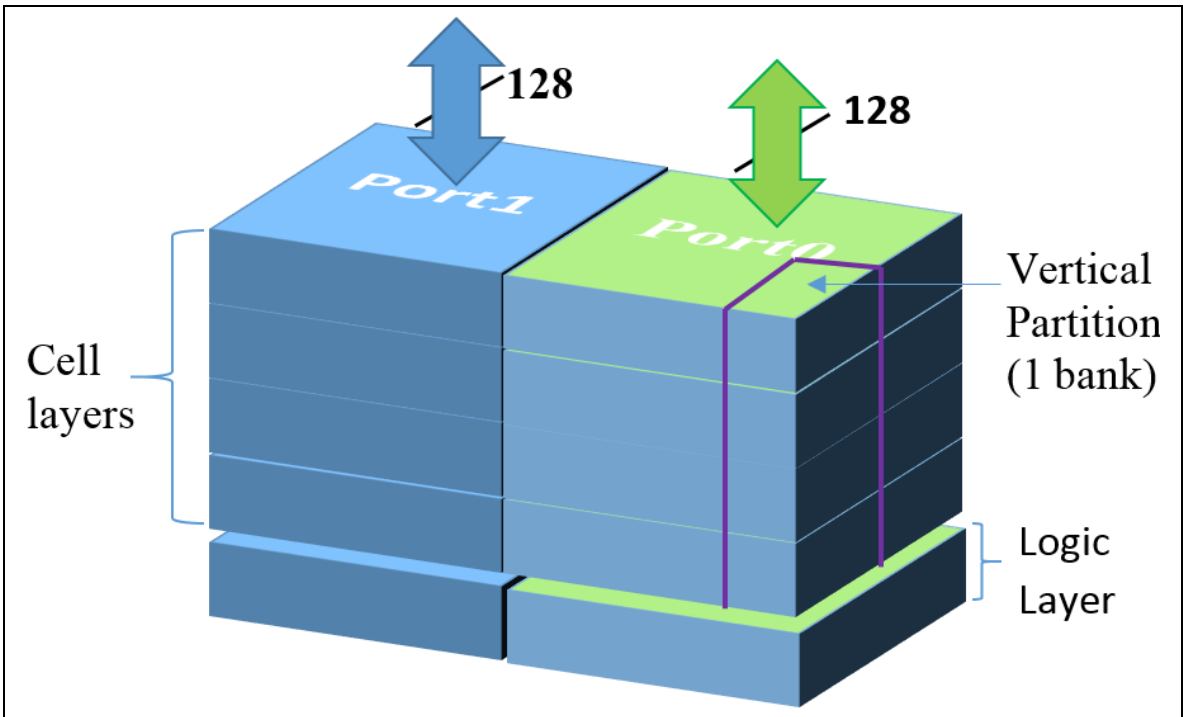


Figure 5.1. A possible Stacked DRAM for low-energy DRAM cache.
Each bank is a vertical partition for a total of 32 banks connected to a separate port. There are 4 cell layers and 1 logic layer per port for a total of 2 ports

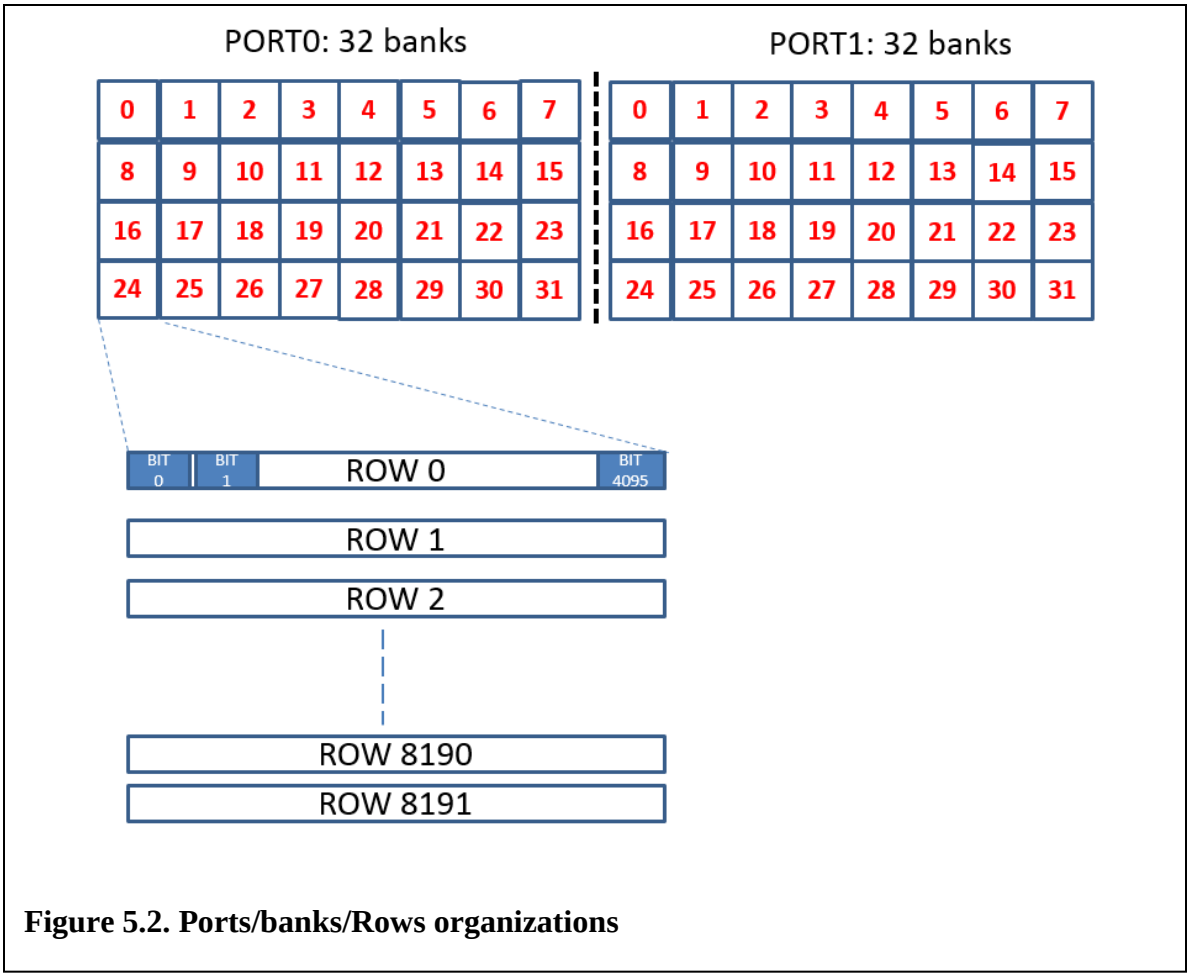


Figure 5.2. Ports/banks/Rows organizations

We need to consider performance aspect and how it relates to this reduction of energy consumption. Because in computer design, performance is an important part and most often the primary focus of conception. Since low power consumption and performance are mutually exclusive concepts, high performing computers are also large energy consumers. Now that we are focusing on energy reduction, can we still achieve an acceptable performance level?

We are using CACTI3DD [40] to estimate the timing and power numbers that this low-size row DRAM can provide. We will compare it to the conventional 2KB row size memory to make an initial conclusion about the use of small size row in our design of DRAM cache. We assumed a stacked DRAM with 4 cell layers and 1 logic layer for all row-buffer sizes. Table 5.1 shows different resulting parameters for our desired memory against traditional 2KB memory.

Table 5.1. Timing and Energy parameters of 2KB vs. 0.5 KB row buffer stacked memory

Timing Parameters (ns)				
Memory type by row buffer size	**tRP**	**tRCD**	**tCL**	**tRAS**
2KB Row: single port 2Gb, 128-bit port, 32bank	6.4627	9.5592	7.133	12.42
0.5KB row: 2 port (1Gb/port) ,128-bit ,32bank, burst 4	2.8006	4.0757	6.098	5.7436
Improvement	**57%**	**57%**	**15%**	**54%**

Energy Parameters (nJ)				
Memory type by row buffer size	**e_ACT**	**e_READ**	**e_WRITE**	**e_PRE**
2KB Row: single port 2Gb, 128-bit port, 32bank	0.4736	2.0618	2.062	0.4197
0.5KB row: 2 port (1Gb/port) ,128-bit ,32bank, burst 4	0.3469	1.3669	1.3671	0.3268
Improvement	**27%**	**34%**	**34%**	**22%**

From this table, we can observe that Activation-to-read time (tRCD) and Precharge time (tRP) are all 57% improved. It takes less than half the time of traditional 2KB row-buffer memory to open or close a 0.5 KB row buffer memory. Also once the row is open, there is another 15% reduction time to read or write to the 64B block in a burst of 4. The same improvement also holds for energy as 27 and 22% reductions are observed in activation and precharge respectively. Even more saving of 34% in read or write operations is possible. Therefore, our initial assumption is valid, but, the total CPU performance is subject to many other parameters such as miss rate and row-buffer hit rate. So this does not conclude about final performance. However, it is enough to encourage us to adopt a small size row buffer for DRAM cache purpose.

5.3. Low Energy Configuration Mapping

The observation from section 5.1 gives us the reason to start looking at different configurations that can result from this small row-buffer size memory. The 3D DRAM as described in figure 5.1 and 5.2 has two port connected to the 128-bit bus and a row buffer of 512 B for data. It gives us two choices. One of the choices can be a highly associative configuration where all two ports are used in parallel, and both provide data at each cache access. Another choice can be a relatively low associative configuration where only one port is active at a time.

We have considered using all 512B for data only and add a small portion to the row for the tag which increases row buffer to 576 B. With this choice, each row of each port will provide 8x64B for data plus 1x64B for tag area. With this 576 B row, a 288MB of total DRAM

will be needed to achieve a 256MB DRAM-cache. Having all configurations with same data size will eliminate size discrepancy from performance and only account for important features. To be fair to all other DRAM cache arrangements, we will normalize the size and emphasize the essential element during simulations.

We suggest two single-set single-row (SSSR) configurations in the small row buffer regime. Our first configuration is a 16-way associative (SSSR-16) that uses a row from each port in parallel to achieve high associativity. Our second configuration uses a row from either of the ports depending on the address. It is a low-energy 8-way associative (SSSR-LE-8). It constitutes a trade-off between low miss rate and low-energy.

5.3.1. Single Set Single Row 16-way Configuration (SSSR-16)

The two ports are combined to provide a 16 blocks set (e.g. 8 data blocks from each). The 9th block from each of the two port is used to serve as the tag. Two 64B tag blocks will provide 64-bit tag to each of the 16 blocks as shown in figure 5.3. 3-bit will help for valid, dirty, and replacement policy. There is remaining 61-bit for the tag. Some of these bits can be utilized for various reliability such as Error-Correction Code as applied in [47].

Up to 85-bit of external physical memory can be supported by such a system. A high-performance server with increasing memory capacity such as 3.4 GHz Intel Xeon Processor E7-8893 [15] with 1536 GB memory extension only needs a 40-bit address. So under this configuration, servers with billions of TB memory (e.g. 85-bit physical address extension) can be supported. With this fast advancement in technology, it may not be too long in the future before we can see servers with that much memory capacity used for everyday computations. If this happens, then it is still highly scalable to address such storage need. A total of 1 KB row

will be opened at row buffer miss, but still takes over half the precharge and activation time of a conventional 2 KB row. Even though two tag blocks serve a single set, the tag serialization as observed in SSSR-HA is eliminated by the parallelism created by the dual port. However, concerning energy consumption, these two tag blocks access increases energy. It raises the question whether this arrangement is suitable for low-energy configuration as we target in this research.

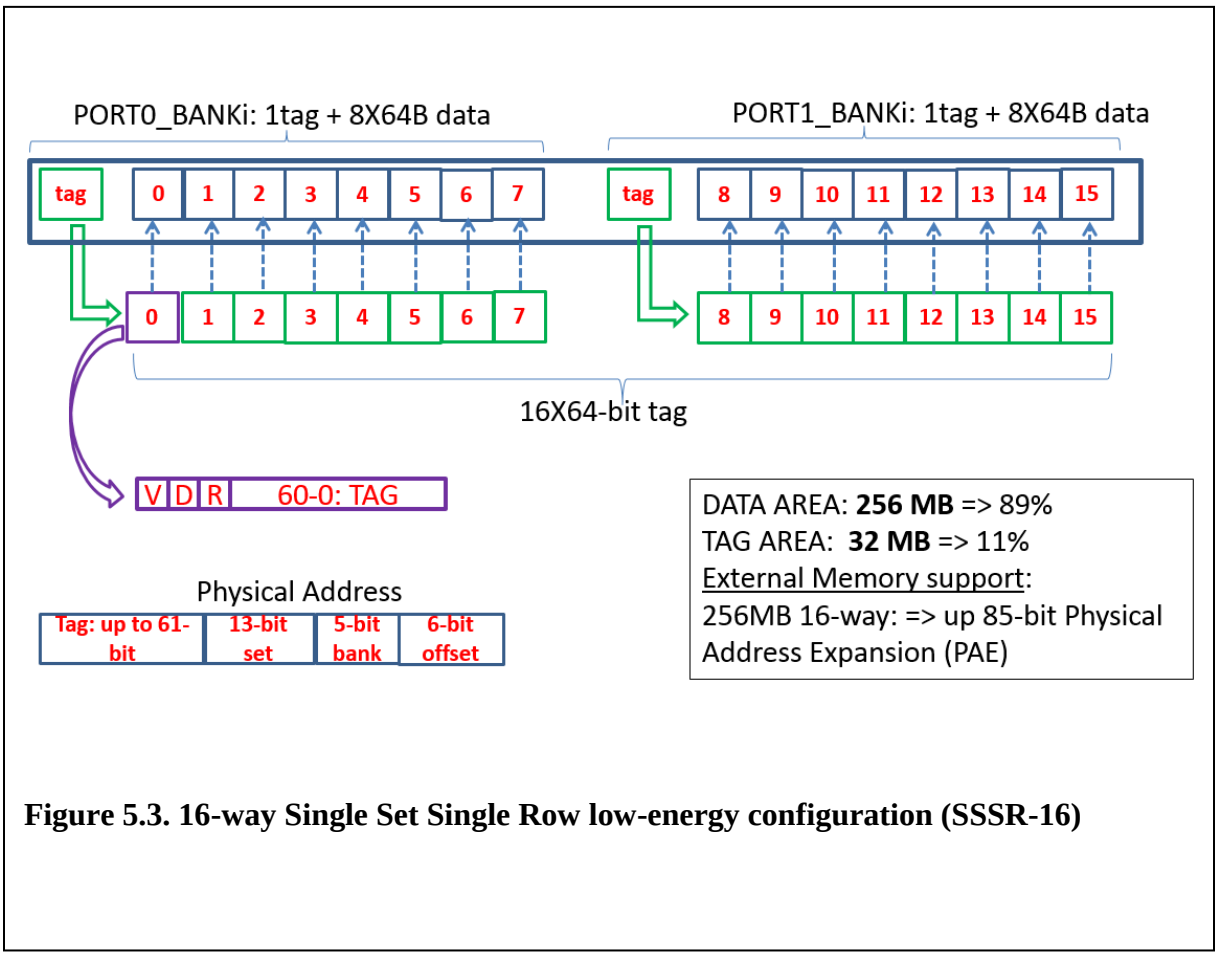
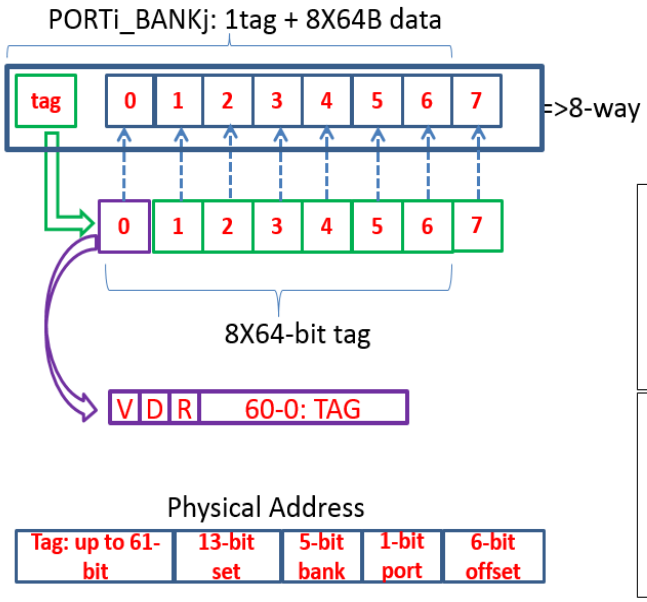


Figure 5.3. 16-way Single Set Single Row low-energy configuration (SSSR-16)

5.3.2. 8-way Single Set Single Row Low Energy Configuration (SSSR-LE-8)

The second configuration is an energy-efficient configuration where only one port provides data. The selected port depends on the requesting address. Only 0.5 KB row of data becomes available at row buffer miss leading to the 8-way configuration as described in figure 5.4. This configuration can support a system of up to 86-bit of external physical memory. Compared to our 16-way configuration, it has a low associativity, but its ability to keep the energy level low by avoiding double tag block utilization is the biggest advantage. The goal in the subsequent sections of this paper is to study the performance deterioration due to reduced associativity compared to SSSR-16. But keeping in mind that the overall energy consumption of this configuration will be the lowest, we will speak more regarding the trade-off between energy and performance due to miss rate. But since they both have the same size for data, 8-way and 16-way may not make a large difference in miss rate.



PORT1 is extension of Port0
PORT0: BANK 0 -31
PORT1: BANK 32 -63 (bank renaming)
=> 64 banks (5-bit bank# + 1-bit port#)

DATA AREA: **256 MB** => 89%
TAG AREA: **32 MB** => 11%
External Memory support:
256MB 8-way: => up 86-bit Physical Address Expansion (PAE)

Figure 5.4. 8-way Single Set Single Row configuration (SSSR-LE-8)

5.4. Rationale for Small Size Row buffer leading to Low Energy-efficient Configuration

There are two unique characteristics to DRAM-cache that are not applicable to SRAM caches. First, Tag and data are all mapped in the same memory rows which creates two different accesses to read both. Second, Row buffer hit (RBH) or row buffer miss (RBM) turn the energy per access and hit latency into variables, unlike SRAM, which has these values almost fixed. It is important to mention that row buffer hit is not to be confused with cache hit because there is no correlation between them. When data is requested from an already opened row, it is a row-buffer hit (RBH), and the contrast is a row-buffer miss (RBM), but this can still be a cache hit.

There are two ways to reduce energy within DRAM-cache. First, is the increase in Row buffer hit rate (RBHR) to reduce precharge and page activations. The second way is to focus on reducing energy during row buffer miss. However, increasing row buffer hit alone is inefficient as less than 20% RBHR [4] on average can be achieved leaving about 80% of access for row buffer miss rate (RBMR). In this work, we are working on reducing energy consumption by the 80% of access even without a noticeable RBHR. To better picture this, we consider two scenarios.

5.4.1. Scenario 1: DRAM Cache Hit

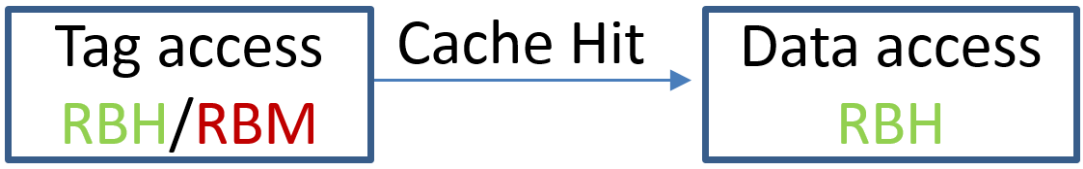


Figure 5.5. DRAM cache hit vs. RBH/RBM

In scenario 1, when the upper-level SRAM cache requests data, the first step is the DRAM cache tag lookup. It is very understandable that this may be a row buffer hit (data exists in currently opened row) if the controller uses open page policy. Or it can be a row buffer miss (data exists in closed row). During DRAM cache access, if it is a row-buffer miss, the previous row is closed, and a new row is activated. The row buffer holds the new row content, and tag block is read and leaves the row opened. Because it is a cache hit, the second step which is the data access is immediate, and therefore it's a row buffer hit by default. It is the advantage of using compound access and open page policy for cache controller.

5.4.2. Scenario 2: DRAM Cache Miss

In this scenario, the tag access can either be RBH or RBM. Because it is a cache miss, data access can occur at a later time equivalent to off-chip latency at best since it has to go through many other steps.

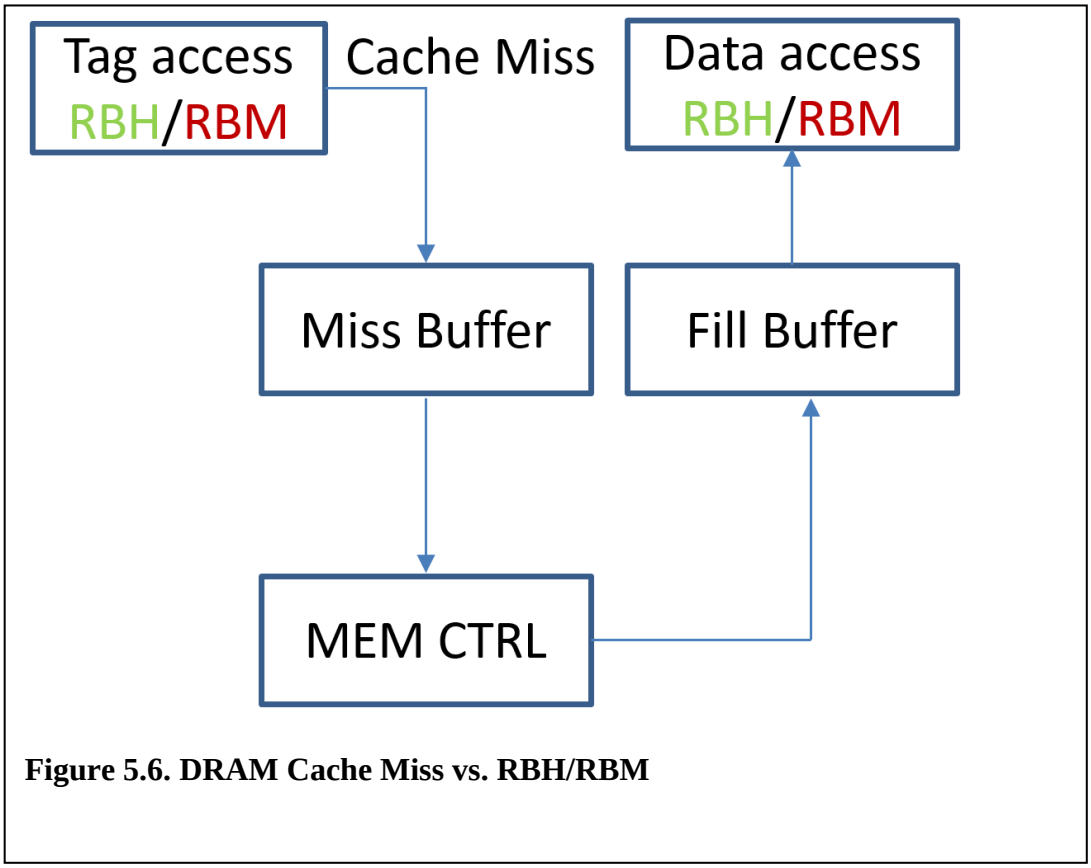


Figure 5.6. DRAM Cache Miss vs. RBH/RBM

Figure 5.6 shows the steps it takes to request data from main memory. So other requests outside the missed row will continue to proceed and mostly will close the row. It does not guarantee that the return data access will also be a row buffer hit. But more likely, it can be a row buffer miss. In this case, if both tag and data accesses are RBM, double ACT and PRE determine the energy of one request. Even if data fill can happen in the background and discounted from access latency critical path, for energy calculation, this other RBM is counted and has an impact on the overall energy. So in these cases, an energy-efficient configuration as we propose here is needed to cut the consumption of double accesses.

5.5. Estimation of Hit latency and Energy per access

Our reduced row buffer gives rise to two configurations with their advantages. On one side we have a high associativity, and on the other hand, we have a reduced associativity but lowest energy per access. In this section, we will estimate the worst hit latency and Energy per access using simple equations.

5.5.1. Estimation Equations

Equation (5.1) represents hit latency at default condition of row buffer miss (RBM).

$$Hit\text{-}time\text{-}RBM = t\text{-}tagCAS\text{-}updt + t\text{-}PRE + t\text{-}RCD + t\text{-}tagCAS + t\text{-}dataCAS \quad (5.1)$$

Where t-PRE is precharge time, t-RCD is row opening to CAS delay, t-tagCAS is tag read access and lookup time, t-dataCAS is the data access time, and t-tagCAS-updt is tag write access to update tag of the previous row before a new one is opened. We have added the later to our equation since tag update is not done in the DRAM row buffer but only done at a local copy of tag register for energy efficiency. Tag update is done only once at RBM to propagate the up-to-date tag values. We will give more details of this on our controller design in chapter 8.

Energy per access can use the same equation format as in equation (5.2). All the suffixes in the equation have the same meaning as in equation (5.1) except they apply to energy.

$$Energy\text{-}RBM = E\text{-}PRE + E\text{-}ACT + E\text{-}tagCAS + E\text{-}dataCAS + E\text{-}tagCAS\text{-}updt \quad (5.2)$$

5.5.2. Assumption used for latency and Energy parameters

To normalize the size of previous configurations to our own, we fixed the data size to 256MB and add the size of tag area to match their specifications. The total DRAM size depends on each configuration features. It has two advantages. First, it eliminates the impact of cache size differential from the overall performance. Second, it helps avoid the case where The number of the set is non-power of two. For the configurations that have multiple sets in a row, a non-power of two can generate unexpected results from the simulator. With this assumption, all the configurations will have the number of sets equal to a power of 2 for simulation purpose. For instance, SSSR-HA's associativity will be increased from 29 to 32, 7 to 8 for MSSR-SA, and the number of sets in MSSR-DM will increase from 28 to 32. We are using timing and energy numbers from CACTI3DD [40]. It is a 3D memory simulation that models coarse-grain rank-level partitioning as applied to Samsung 3D DRAM [25]. It also models fine-grains partition as it applies to Hybrid Memory Cube [43] including ITRS aggressive TSV predictions. We are inclined to use this simulator because its accuracy is within 99% of CAS latency compared to Samsung 3D memory [25] data sheet. Additionally, it is used as a baseline for other 3D memory architecture and simulator [42] [45]. Table 5.2 is a summary of total memory required to make 256 MB cache along with 3Ts and 3Es numbers for all configurations.

Table 5.2. Summary of Total DRAM Size, 3Ts, and 3Es

	Total Size(MB)	Row Size(B)	tCAS-tRCD-tRP (ns)	eCAS-eACT-ePRE (nJ)
SSSR-HA	280	2240	7.11-9.81-6.93	2.13-0.46-0.40
MSSR-SA	288	2304	7.26-10.40-7.14	2.17-0.46-0.41
MSSR-DM	320	2560	7.58-11.52-8.02	2.79-0.55-0.49
SSSR-LE	288	576	5.72-4.05-2.64	1.29-0.33-0.31

What we call 3Ts are standard memory timing parameters used by many DRAM manufacturer to determine the performance of DRAM memory. The smaller they are, the faster the memory will run. *tCAS* is the time between sending the requested address to the memory and the read of the first bit of data from the DRAM with an already opened row. *tRCD* is the Row to Column Address Delay; it is the delay between the opening of a new row to sending columns address. Finally, *tRP* is the precharge time or time it takes to close an existing row before an attempt to open the next one. Most often these values are given in nanoseconds. To determine the values in CPU clock cycles, we can multiply by the rated clock cycle time. In this work, we assume that our CPU is operating at 3.2 GHZ.

We adopt the same terminology to define our 3Es as the energy parameters that set the consumption for the three primary DRAM operations (ACT/PRE/CAS). From Table 5.2, tCAS-tRCD-tRP is just 5.72-4.05-2.64 ns for our proposed DRAM with 576B row size and 128-bit bus in burst-4 mode. A total row size of 576 B is equal to 4608 bits that include 4096 bits for data and an extra 512-bits for tag. As far as energy is concerned, ACT/PRE cost 0.33nJ/0.31nJ, and CAS cost 1.4nJ for 64B block.

We can already notice a difference the size of row buffer makes on time and energy number, but we are yet to evaluate its impact on overall performance.

5.5.3. Latency and Energy per access

We use Table 5.2 and equation (1) and (2) to compute the values of hit latency and energy per access for our configuration as well as the selected configurations from the previous studies. From Table 5.3, our configurations show a hit latency evaluated to 29.5 ns at the worst case of row buffer miss.

Table 5.3. Summary of DRAM cache Hit latency and Energy per access

	RBM		RBH	
	Hit latency (ns)	Energy (nJ)	Hit latency (ns)	Energy (nJ)
SSSR-HA	76	15.8	9.13	2.13
MSSR-SA	43.1	7.37	9.13	2.17
MSSR-DM	38	6.62	19	5.58
SSSR-16	27.6	7.75	7.6	1.29
SSSR-LE-8	27.6	4.52	7.6	1.29

For our SSSR-LE-8, there is already an improvement of a minimum of 27% compared to configurations mapped in 2 KB row buffer memory. For energy, there is a minimum of 32% improvement. Again this is just per access basis. Our SSSR-16 is not energy efficient as we predicted but its timing is as good as our low-energy configuration SSSR-LE-8. The overall performance is subject to additional criteria such as miss rate which can depend on associativity.

But in general, for a large cache capacity, it's hard to predict how much performance improvement can result from increased associativity especially when it is above 4. We can expect a huge difference in performance between the last level 8MB L3 SRAM and a last level 200 MB DRAM cache, but we cannot expect a significant difference in performance between two DRAM caches of same capacity but different associativity. For a single-thread, the performance will be almost the same, but the difference can start to become apparent on a multi-core system running a huge amount of programs simultaneously on multi-thread or multi-programmed workloads. Because in this case, the capacity difference and associativity can play a role to service multiple requests for all the threads or programs. Only simulations can give us a better idea of the difference.

5.5.4. Pros and Cons of Alloy-cache (TAD)

Among all the mappings provided above, the MSSR-DM stands out different than most because it combines tag and data in a single cache block called TAD. All other mappings with separate tags in this paper (including our mappings) are considered set associative. MSSR-DM being unique has its advantages and flaws. The main advantage is the reduction of hit latency as there is not separate tag access. For flaw, first, in the case of cache miss, all the set associative mappings will only spend energy of tag access as data is located in separate blocks. In contrast, the TAD will incur the penalty of tag and data access even on a miss.

Second, in the case of row buffer hit (RBH), hit latency can further be reduced for some set associative mappings. A local register file (up to 2 KB for 32 banks) can be used to save previous tag blocks from all the banks row buffers avoiding DRAM access for tag lookup or update. This option does not apply to MSRR-DM where TAD does not allow for separate tag

search or update. Since tags are spread throughout the row, an update is required after every use. But keep in mind that because tag is inside data, any tag update required the entire 80B TAD to be written under burst-5 mode. It's the reason why the hit latency and energy per access at RBH for this configuration are not significantly lowered that the RBM values.

5.5.5. Additional advantages of our low-energy configuration SSSR-LE-8

As we mentioned above, a grouping of all tag in a separate block(s) for set associative configurations, gives an ability for the controller to maintain a local tag registers. It can be used to avoid tag update on DRAM row buffer at every access. It gives our low-energy configuration another advantage over other set associative configurations analyzed in Chapter 4. First, we assume that each of 32 banks in the port maintains its separate row buffer. To compute row buffer hit rate at each access, we use the port, bank, and row numbers extracted from the incoming address (see Appendix A). Also, our controller must maintain a separate 64B tag register per bank for a total of 2KB per port or 4KB register file for all two ports.

To do the same for SSSR-HA configuration, it requires three tag registers for a total of 6KB register file. For MSSR-SA configuration, four tag registers per bank must be maintained for a total of 8KB register file. We can see that the size of the register file is increasing for other configurations. In additional, the goal of MSSR-SA configuration was to avoid tag serialization by separating sets within the row. Only one block tags each set. Therefore, it should update the tag block at each access to avoid updating four tag blocks at the same time and avoid serialization. But for simplicity, we assumed that only 1 out of 4 tag blocks per row is to be updated at row buffer miss although not realistic.

Therefore, the values from Table 5.3 are the unrealistically best estimate for this configuration. The most practical energy per access for row buffer hit will be higher to account for tag update.

CHAPTER 6

Simulations and Results

6.1. Assumptions

Loi and Benini [20] asserted that the use of multi-ported and multi-banked L2 as shared last level of cache has the advantages of increased utilization, fast inter-core communication, and reduced aggregate footprint to avoid undesired replication of lines. Also, it reduces the complexity in coherence protocol and cost in implementation of multi-core architectures designed for low-power such as embedded applications. It is a valid point that can encourage a second look at the complexity of excessively increased level of cache hierarchy. Also since our multi-banked DRAM cache is large and fast, it can significantly reduce average latency for memory accesses, while minimizing the bandwidth toward off-chip memory. Therefore, it can also be used as the last level L2 cache. The following are the assumptions we used so as to arrive at a fair comparison. First, in agreement with [20], we combined L2 and L3 caches into a compound DRAM cache so that our hierarchy is simply CPU→L1→shared-DRAM-cache→MEM. We estimate that with a large and fast on-chip DRAM cache, the performance improvement due to excessive level of the hierarchy will not outperform the cost and complexity in design but in other cases, it may increase the overall memory access latency. Also, we want to stress DRAM cache with unfiltered misses that could've been blocked by intermediate levels of hierarchy to test the effectiveness of our low-energy DRAM cache. 2) We normalized the sizes of all our experiments to remove the cache capacity differential as contributing criteria for performance. 3) We assume that tag update is done only at row buffer

miss for all set associative configurations even for MSSR-HA. For MSSR-DM, tag update is done at every access due to TAD.

6.2. Methodology

6.2.1. Workloads

Table 6.1. Workload Simulated

Expriment	8 CPUs multiprogrammed workload assignments
8mcf	*mcf x 8*
8bzip2	*bzip2 x 8*
8Libquantum	*libquantumx 8*
8gcc	*gcc x 8*
8sjeng	*sjeng x 8*
8cactusADM	*cactusADM x 8*
8omnetpp	*omnetpp x 8*
8povray	*povray x 8*
8soplex	*soplex x 8*
8comb1	*mcf,bzip2,libquantum,hmmer,gcc,sjeng,cactusADM,omnetpp*
8comb2	*(bzip2,libquantum,sjeng,cactusADM) x 2*
8comb3	*(bzip2,libquantum,omnetpp,povray) x 2*
8comb4	*(mcf,gcc,sjeng,catusADM) x 2*
8comb5	*(sjeng,catusADM) x 4*
8comb6	*(hmmer,gcc,omnetpp,povray) x 2*
8comb7	*(mcf,bzip2,gcc,omnetpp) x 2*
8comb8	*(mcf,bzip2) x 4*
CPU specs	Timing CPU, 3.2 GHZ
Cache specs	L1 I/D$(32KB): 64B 4-way, access time: 1ns
DRAM-Cache	1.6 GHZ, 64B block, other specs see table 1 for each config
Off-chip MEM	8GB DDR3, 13.75-13.75-13.75 (ns), 800MHZ
	1 and 2-channel (for mcf 16GB), 2-ranks/channel

For our methodology, we run spec2006 benchmarks [89] using GEM5 simulator [41] to evaluate all the mappings. All experiments are run on multi-programmed workloads using eight cores where each workload runs a minimum of 2 billion instructions. In some experiments, we assigned the same benchmark to all the CPUs to separate memory intensive from computation intensive. On others, we feed different combinations of benchmarks to each CPU as shown in Table 6.1. Doing this is necessary to obtain the whole picture by combining memory intensive and computational intensive benchmark.

6.2.2. Simulator Setup

In this work, we will use total execution time instead of average DRAM cache access latency to measure performance and compare among all the mapping. Because DRAM-cache has its unique characteristics such as Row buffer hit and combined tag and data in the same memory structure, we cannot treat it the same way as SRAM cache. There are four different variations of latency. First in case of row buffer miss, there are two variations: RBM with cache hit (RBM_CH) is the full access latency which accounts for both tag access and data access while RBM with cache miss (RBM_CH) does not account for data access because it is a miss. Second, in the case of row buffer hit, given the fact that there is no ACT or PRE, there is no tag access neither because previous tag registers are assumed to be updated locally. Therefore, RBH with cache hit (RBH_CH) only accounts for data access and lookup latency while RBH with cache miss (RBH_CM) only account for lookup latency. Distinguishing between tag access and lookup latency is important. The former refers to fetching tag blocks from DRAM cache into tag registers while the latter refers to checking the incoming address for cache hit

or miss from tag registers. Table 5.2 only shows RBM_CH and RBM_CH. The other two variables are calculated accordingly.

To ensure that the execution time is accurate, we have modified GEM5 [41] to reflect variable hit latency due to the above variations of row buffer hit or miss. It allows us at the same time to capture the Row Buffer Hit Rate (RBHR) of all the configurations. Also, we have added a capability to the tool to calculate energy used within DRAM cache. Again this also accounts for energy per access due to Row buffer hit or miss. We are running over 2 billion instructions per core as we set the simulation to halt only when "all" cores have completed at least 2 Billion instructions each. Doing this will keep the memory pipeline busy throughout the entire execution. We also use an off-chip memory that has a more accurate DDR3 model instead of simple memory with fixed delay. This simulator was designed to model SRAM cache where latency is almost constant for all the conditions. It is not true for DRAM cache as we can see above four variations of latency can result from different conditions of row buffer hit or miss. To model this, we divided hit latency into many other latencies reflecting row buffer hit state. These are the different latency:

We replace the original hit_latency initialization statement from:

```
hit_latency = Param.Cycles(0,"Hit latency");
```

to:

```
RBH_Latency_CH = Param.Cycles(0,"row-buffer hit latency with cache hit");
RBH_Latency_CM = Param.Cycles(0,"row-buffer hit latency with cache miss");
RBM_Latency_CH = Param.Cycles(0,"row-buffer miss latency with cachehit");
RBM_Latency_CM = Param.Cycles(0,"row-buffer miss latency withcachemiss");
RBM_Latency_FILL = Param.Cycles(0,"row-buffer miss latency for Fill, no
Tag access needed");
```

We have assumed 32 banks per port for a total of 2 port. At every DRAM cache request, first we break the address into bank, row, and port as shown below:

```
port_num = (addr_in & 0x40)>>6;
bank_num = (addr_in & 0xF80)>>7;
row_num = (addr_in & 0x1FFF000)>>12;
```

The above is the address manipulation for our SSSR-LE-8 configuration. For more information on how every configurations are broken into port, bank, and row, see appendix A.

Second, we compare incoming address to previous address with rbh_ctrl function and return RBH status. The snippet of this function is as follow (for complete transcript see Appendix A):

```
bool rbh_ctrl (int addr_in)
{
…
if (bank_num==0)
   {
    if(row_num==pre_row_num0) {RBH1=true;}
    else {pre_row_num0 = row_num;RBH1=false;}
   }
else if (bank_num==1)
   {
    if(row_num==pre_row_num1) {RBH1=true; }
    else {pre_row_num1 = row_num;RBH1=false;}
   }
....
else if (bank_num==31)
   {
    if(row_num==pre_row_num31) {RBH1=true; }
    else {pre_row_num31 = row_num;RBH1=false;}
   }
…
Return RBH1
}
```

At every original request and response request, we call rbh_ctrl and accumulate the number of row buffer count, assign appropriate latency, and compute energy as:

```
isRBH = rbh_ctrl(input_addr);
if(isRBH)
      {
      rbhcount += 1;
      lookupLatency = RBHLatency_CH;
      forwardLatency = RBHLatency_CM;
      if(miss) rbhcount_CM += 1;
      else rbhcount_CH += 1;
      }
else  {
      lookupLatency = RBMLatency_CH;
      forwardLatency = RBMLatency_CM;
      if(miss) rbmcount_CM += 1;
      else rbmcount_CH += 1;
     }
...
if(isRBH) fillLatency = RBHLatency_CH;
else fillLatency = RBMLatency_FILL;
...
DRAMCache_Energy = (rbhcount_CM*RBHEnergy_CM + rbhcount_CH*RBHEnergy_CH
+ rbmcount_CM*RBMEnergy_CM + rbmcount_CH*RBMEnergy_CH);
```

In the above snippet, lookupLatency and forwardLatency are assigned appropriate values according to RBH status. According to original code, these two latencies point to the fixed hit_latency while fill latency and response latency all point to the same fixed response value as shown below:

lookupLatency(p->hit_latency),

forwardLatency(p->hit_latency),

fillLatency(p->response_latency),

responseLatency(p->response_latency),

It is also important to mention that for DRAM cache, fillLatency is not necessary the same as responseLatency because the former is a variable that depends on RBH status within DRAM cache while the latter does not because it is just a fixed return path. Data may reach the requesting cache or CPU before it is filled up in the DRAM cache.

6.3. Results and Analysis

6.3.1. Figure of Merit: Execution time & DRAM Cache energy

Figures 6.1 and 6.2 show the detailed simulation results on all experiments. The two primary performance measurements are total execution time and the total DRAM-cache energy. From Figure 6.1, we can see that our energy-efficient mapping SSSR-LE-8 consumes the lowest energy compared to configurations mapped in 2 KB row buffer DRAM cache. It improved energy consumption by **36%**, **40%**, and **74%** in relation to MSSR-DM, MSSR-SA, and SSSR-HA respectively while our SSSR-16 consumed even more energy compared to MSSSR-DM. Therefore, it does not fit our low-energy configuration requirement. There is also a noticeable improvement in performance. Both our SSSR-LE-8 and SSSR-16 reduced CPU time by **16%**, **18%**, and **41%** respectively compared to the three analyzed configurations as above.

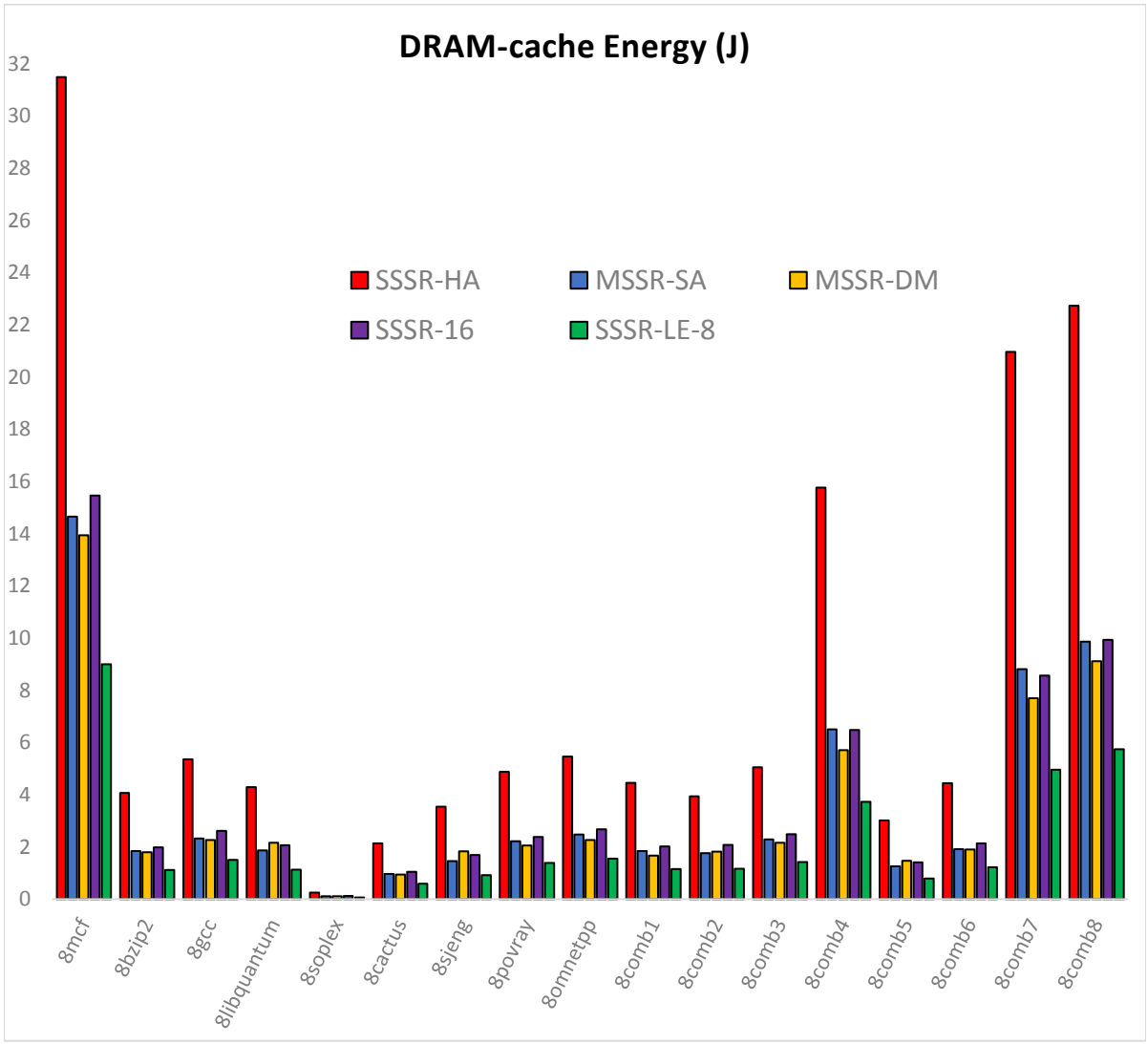


Figure 6.1. Total Energy per run

Our 16-way configuration, even though it did not fit energy requirement but its performance is made better by tag parallel access. The mapping SSSR-HA with 32-way cache, because of serialization of tags, its energy consumption and execution time are the highest. It goes to show that the hit latency has more impact over associativity. So all our two configurations offer improvement in timing performance compared to analyzed configurations. But only our SSSR-LE-8 meet the low-energy requirement. It goes out to answer the central question to know how much performance degradation we will obtain by reducing energy. The answer is that the performance is even increasing as Energy is reduced. We have accomplished our primary goal of energy reduction while also increasing our performance compared to our selected configurations mapped in 2 KB row buffer. We retain SSSR-LE-8 to be our baseline for low-energy configurations and dismiss SSSR-16.

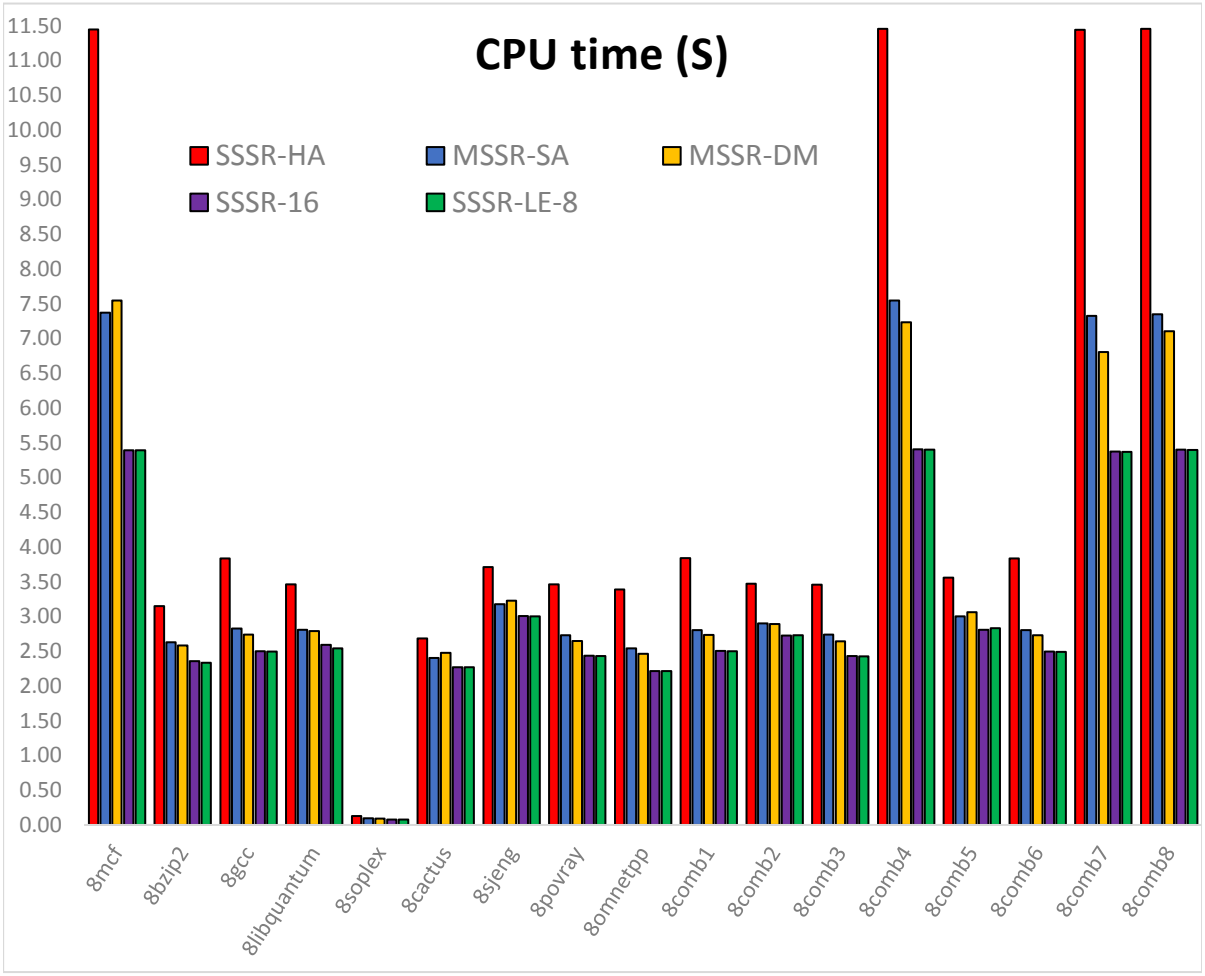


Figure 6.2. Total Execution Time per run

Table 6.2 is the summary of our most important performance metrics which represent the execution time and DRAM cache Energy. We are using an arithmetic average since these two quantities are aggregate numbers that can be summed and averaged.

Table 6.2. Average performance Summary

	SSSR-LE-8 IMPROVEMENT (%)		
	SSSR-HA	MSSR-SA	MSSR-DM
CPU time	41	18	16
Energy	74	40	36

	SSSR-LE-16 IMPROVEMENT (%)		
	SSSR-HA	MSSR-SA	MSSR-DM
CPU time	41	18	16
Energy	54	-5	-11

6.3.2. Other Metrics

In the previous section, we have analyzed our most important figures of merit. But in this section, we will examine other contributing factors. We have mentioned in earlier chapters that the overall performance also depends on miss rate (MR) and row-buffer hit rate (RBHR) beside hit latency that we have already thoroughly analyzed.

6.3.2.1. Miss Rate

Figure 6.3 shows the miss rate of all our individual workloads. From this, we can notice that MSSR-DM has high miss rate as expected due to direct mapping. Noticeably it is observed for benchmarks such as mcf, bzip2, cactus, as well as several other workload combinations. The values of MR are spread out within a broad range. From as little as 0.25 % to as high as 52 %. It explains why the arithmetic mean or average is only within 10% for most of the configurations as seen in figure 6.4. So it is better to look at the entire picture rather than just average values. All other configurations are within 2% difference except for MSSR-DM that is at 15% higher than the lowest.

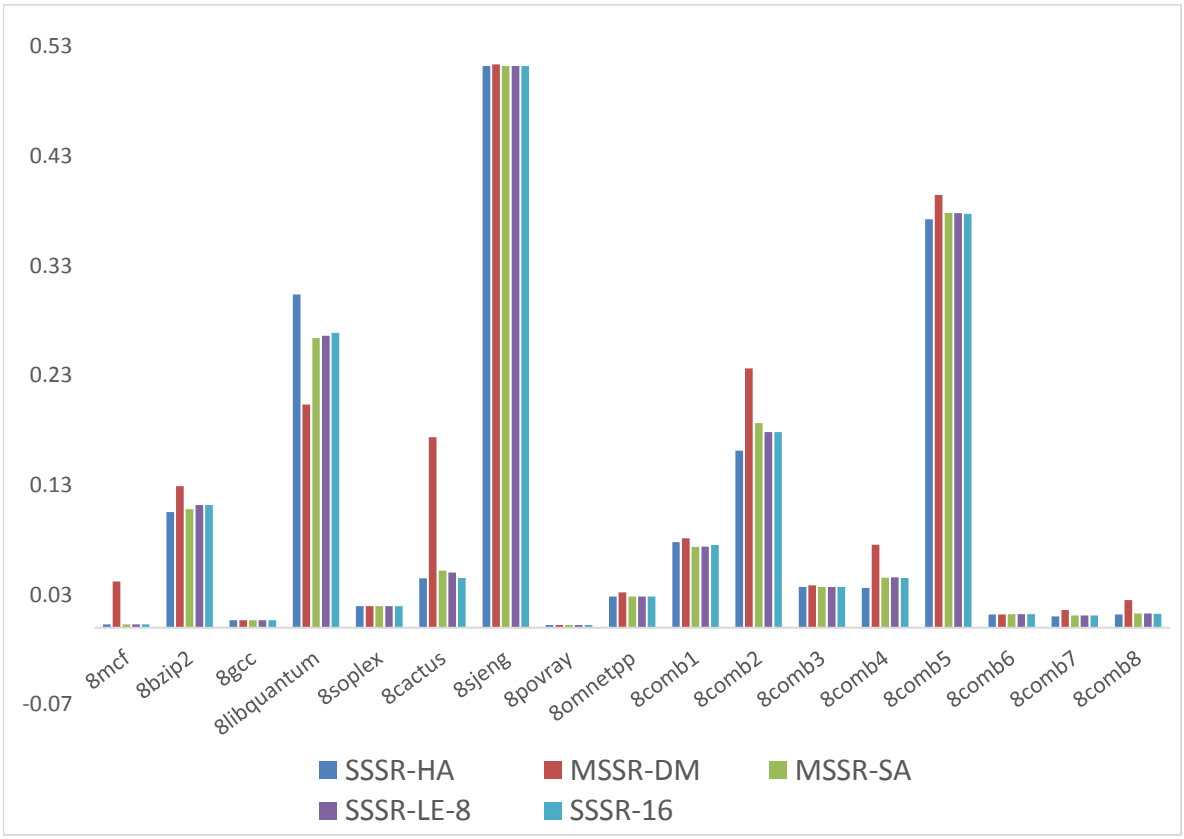


Figure 6.3. Detailed Miss Rate per run

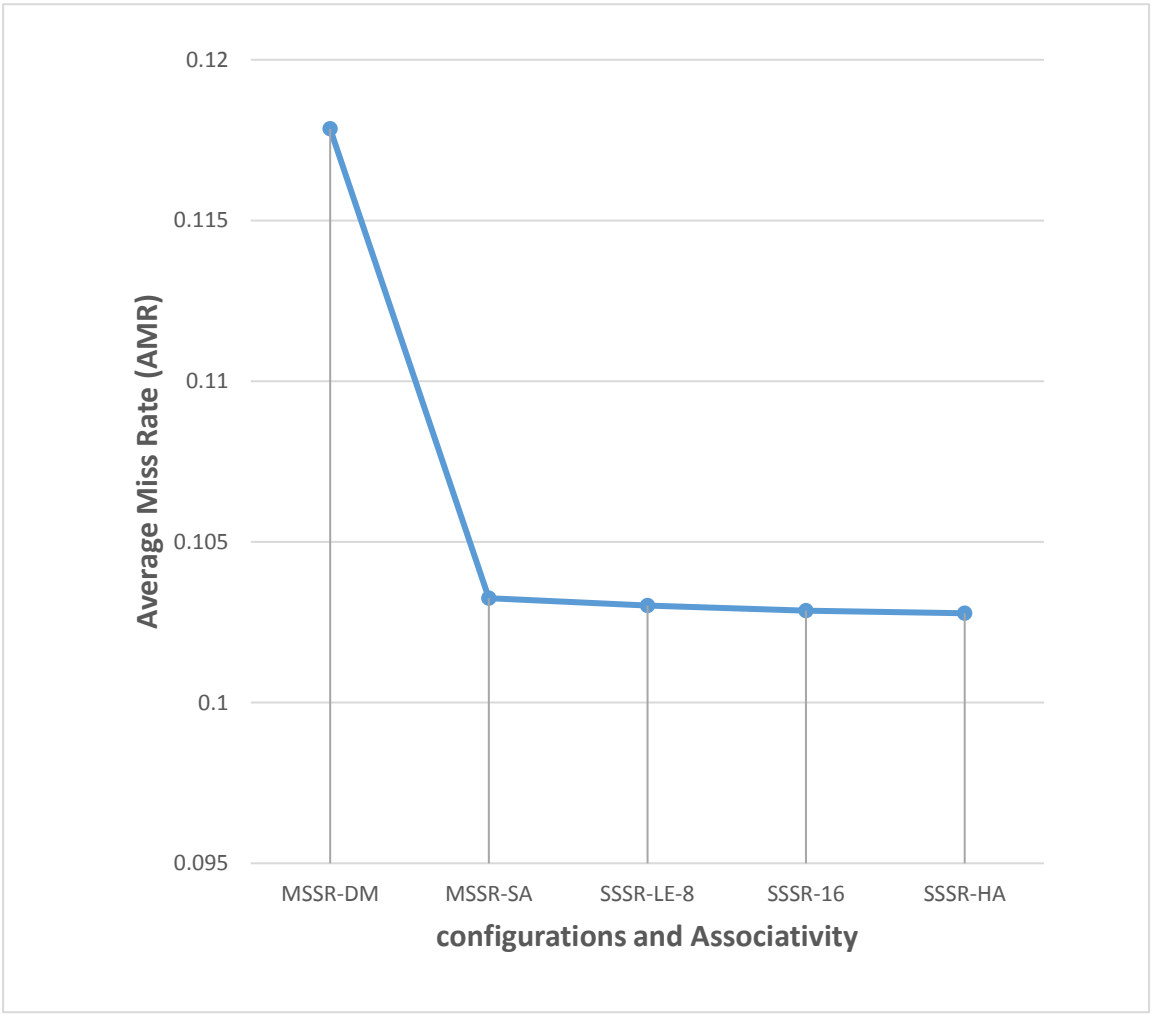


Figure 6.4. Average Miss Rate

In summary, all other configurations besides the direct map have very similar miss rate. It proves that any associativity beyond eight does not contribute much to miss rate reduction and therefore to overall performance. The authors of MSSR-DM were correct to try to sacrifice MR for less hit latency because as we can see from results, as the size of cache increases, the impact of associativity on miss rate is getting diminished. This assessment can be reinforced by observing the performance of the SRAM-based cache. As cache size increases, an associativity of 4 has almost the same impact on miss rate than a fully associative cache.

6.3.2.2. Row Buffer Hit Rate

The second important metric is the row buffer hit rate. From figure 6.5, we can observe that MSSR-DM has the highest RBHR among all configurations including our own. The values are also spread out from as little as 1% to as high as 25% to an average of 7.4%. Despite their high Miss Rate, a relatively increased RBHR helped compensate for it and put them on the map for a better performance than the other two analyzed configurations. MSSR-SA also had a goal to increase RBHR, and it has a slightly higher rate but not as much as the authors claimed. Our configurations have less than 5% rate as expected since they didn't optimize for it.

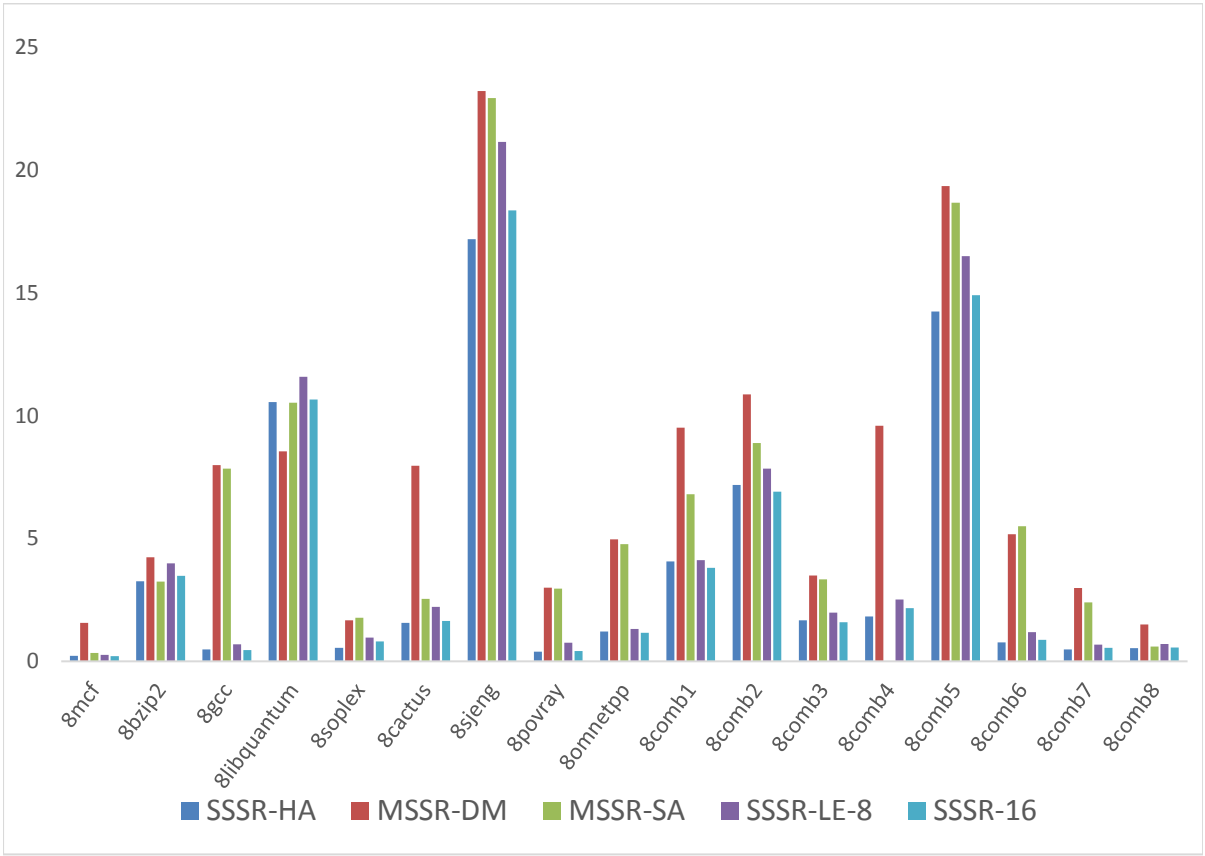


Figure 6.5. Row buffer hit rate

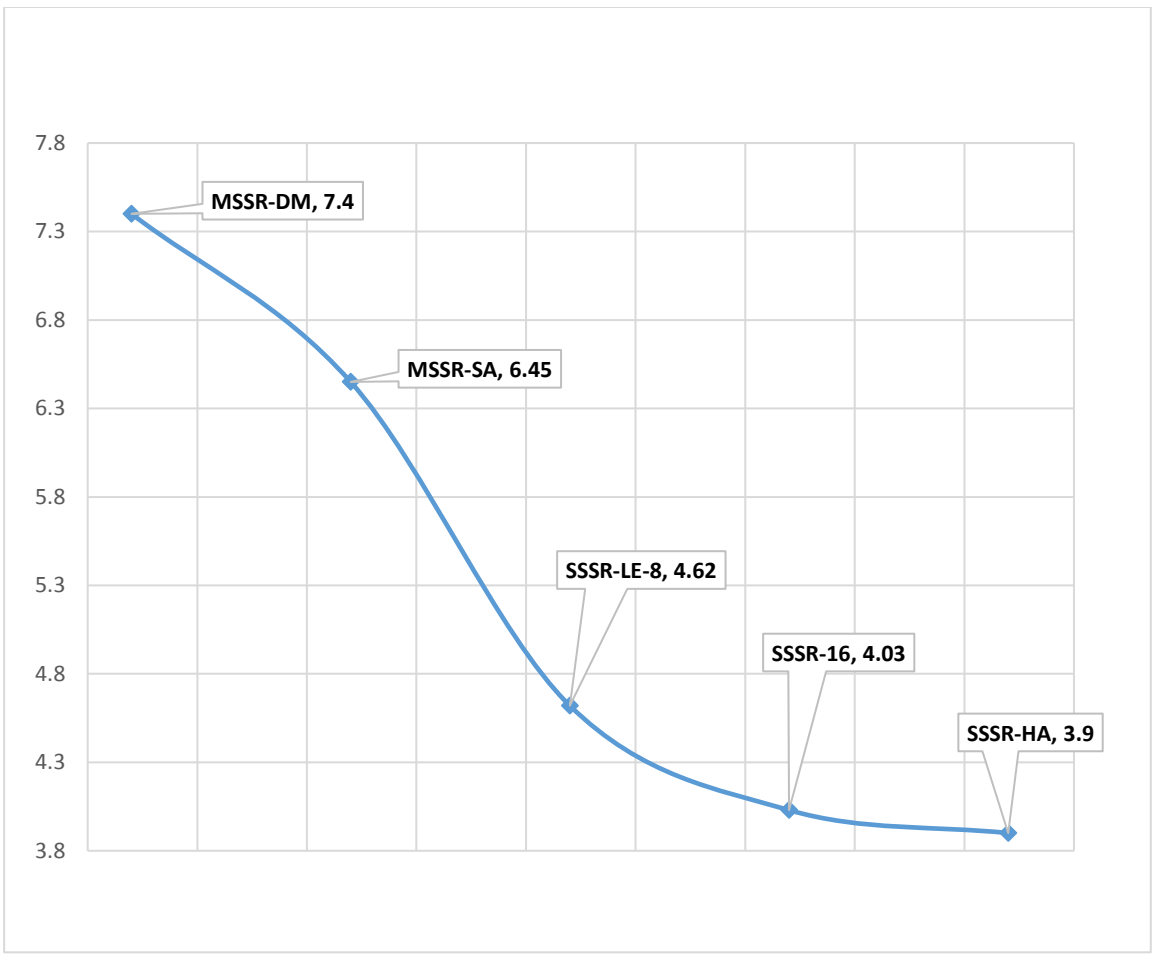


Figure 6.6. Average Row Buffer Hit Rate (RBHR)

In general, we can observe that RBHR is not an overwhelming factor to energy reduction. So this again answers another one of our questions to know whether an increase in RBHR was preferable than focusing on reducing energy at RBM, which happens in the majority of cases. It is coming to a conclusion that our focus on reducing the energy of RBM is more efficient that increasing RBHR alone.

6.4. Improvement from SRAM LLC

To complete our analysis, we have selected our full workload set to run with SRAM last level cache (LLC) of 8 MB 16-way associative. We assume the hit latency to be 24 cycles as most previous configurations have considered [1] [2] [3] [4]. We compare the result with our low-energy SSSR-LE-8. Figure 6.7 shows execution time for each of our workloads. For memory intensive workloads, improvement using our SSSE-LE-8 is almost 2X. For all the workloads, the average improvement is about 28%. Figure 6.8 shows relatively high miss rate for SRAM LLC as expected. We can see that MR for mcf, for instance, is very high about 66% because the cache size is only 8MB but when we compare to DRAM Cache design, the same MR is significantly reduced to less than 1%. It shows how a large DRAM cache can be useful especially for memory intensive benchmarks.

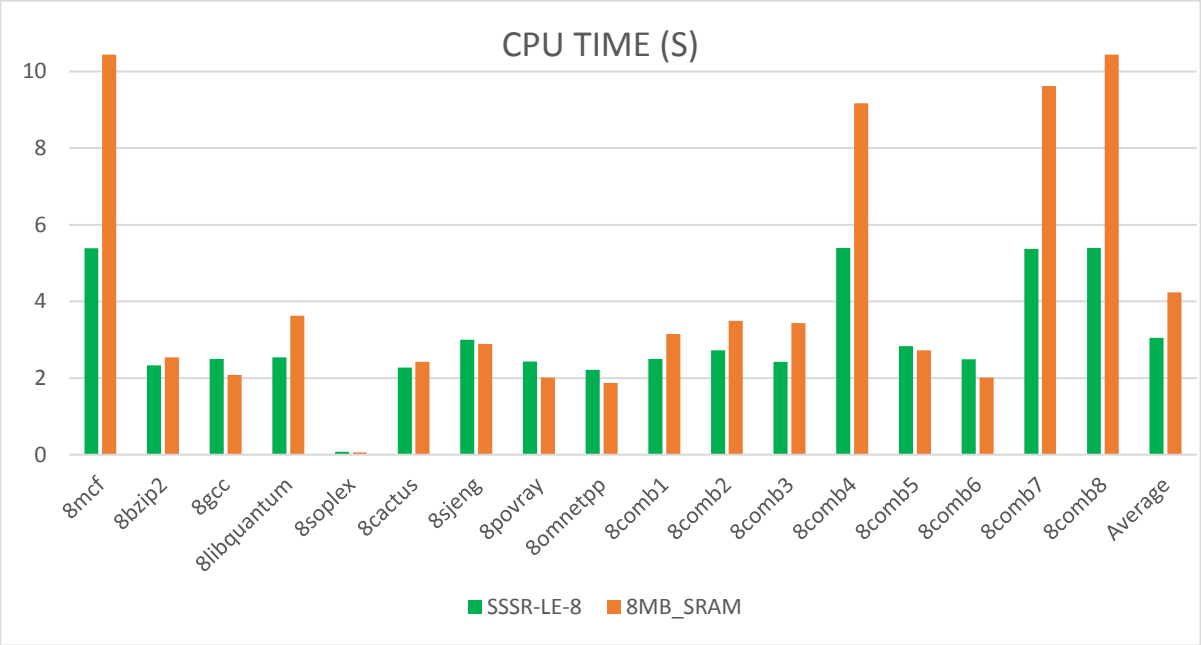


Figure 6.7. Total Execution Time per run for SRAM LLC

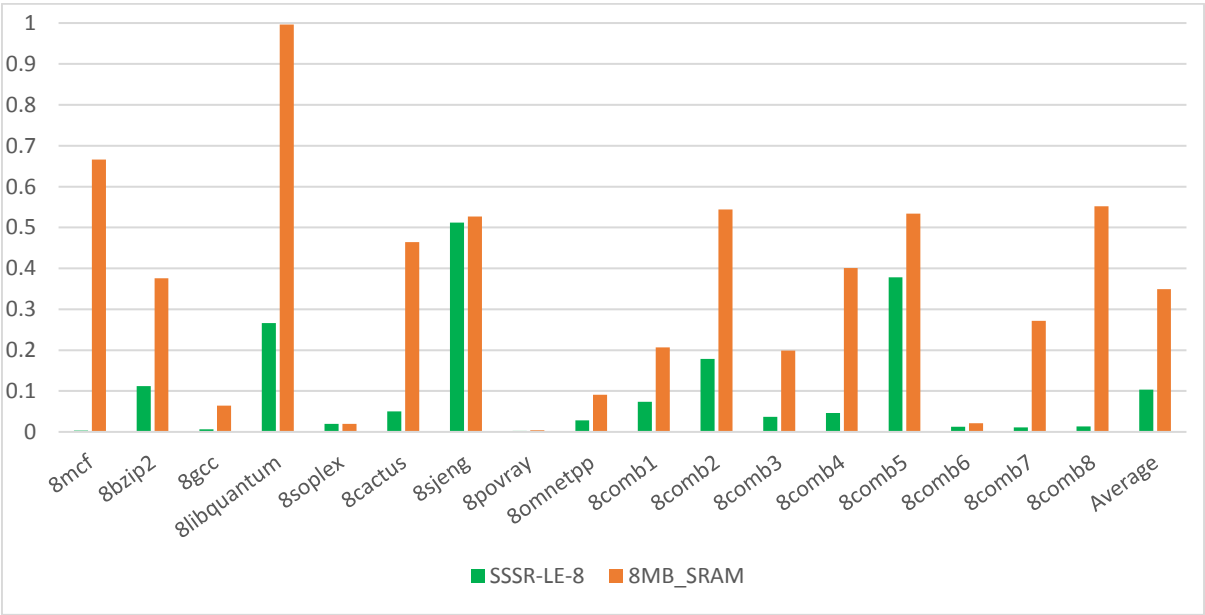


Figure 6.8. Detailed Miss Rate per run for SRAM LLC

6.5. Summary

From the above results, we were able to achieve an average of **36%**, **40 %, and 74%** improvement in energy within DRAM cache while keeping a noticeable **16%**, **18%**, and **41%** average timing performance compared to three other **configurations mapped in 2 KB row buffer DRAM ca**che. Because our SSSR-LE-8 proposal satisfies both energy and performance improvements, it is well fit for DRAM-cache application than other proposed configurations. We can conclude that building a DRAM-cache on memory with small row size is beneficial to reduce energy and latency per access. It provides more impact than just focusing on increasing row buffer hit rate and reducing miss rate while using large row buffer. We also saw that average miss rate is almost the same for all configurations. It shows that for a very large cache, the associativity has a slight impact on miss rate because the capacity miss is subtle. Both Miss rate and row buffer hit rate have less impact on overall performance and energy. Reducing hit latency and energy per access is much significant for the design of DRAM cache.

CHAPTER 7

Survey of Stacked DRAM specs for low-energy DRAM Cache

7.1. Overview

From chapter 5 and 6, we established a net energy consumption and performance improvement of our reduced row buffer DRAM Cache configuration over the large row buffer mappings. We have also established our low-energy single set 8-way (SSSR-LE-8) configuration mapped in 512 B row buffer memory as the baseline. It allowed us to prove the concept that using DRAM with small row buffer size is preferable than a large row size counterpart. However, we have not yet established the DRAM organization for lowest energy and high performance. So in this chapter, we will study a survey of stacked DRAM with row buffer ranging from 256B to 4KB and number of banks of 16, 32, 64 with a single or a dual port.

We picked 256 B to be the lowest because it gives us at least four blocks of 64 B as the typical cache line size. If we start with 128 B, it would be difficult to obtain a good percentage of cache data area. For instance, if we use one block for tag, this will be 50% of memory area dedicated to tag. But at least with 256B, there is at least 75% memory usage for data and then 80% if we add an extra block for tag. We have chosen two different design layouts for the stacked DRAM: a 1-port and a 2-ports. The design with 2-ports reduces the size of banks, and the idea is to make the ports completely independent and not to share any circuitry as shown

in Figure 5.1. Consequently, there may be a reduction of delay and energy per access. But because memory manufacturers may want to maintain good energy and delay, layout choices may increase the number of internal partitions. Enhancement of word and bit lines may affect aspect ratio and area efficiency. But CACTI3DD can compensate for this internally and give us a better estimation.

We will also try to use different techniques as studied in chapter 4 to find the most performing configuration within the selected row size. For instance, the increase in the number of sets within the row is a possibility to increase row buffer hit rate by sacrificing associativity. Given the small impact of associativity on large caches, we can try to map 2, 4, or 8 sets in a single row even though associativity decreases. We will determine the speedup and energy reduction using our SSSR-LE-8 as a baseline.

7.2. Methodology

Our goal is to identify the most effective row buffer size for stacked DRAM as well as the impact of the number of banks. We are focusing on 256MB memory since we used it for our full workload simulation. In the first step, we will evaluate the impact of row-buffer size and then the impact of the number of banks. These two criteria will be enough to interpolate conclusion of other larger memory sizes. Table 7.1 already has 20 different combinations. We are not going to analyze each of them to avoid an excessive number of tables and figures but instead, we are analyzing only the seven highlighted 32-banks combinations from Table 7.1.

Table 7.1: 256MB Stacked DRAM Combinations of specs

#ports	# banks	combination#
256B Row buffer		
2-port	16 Bank	1
	32 Banks	2
	64 Banks	3
512B Row buffer		
2-port	16 Bank	4
	32 Banks	5
	64 Banks	6
1KB Row buffer		
1-port	16 Bank	7
	32 Banks	8
	64 Banks	9
2-port	16 Bank	10
	32 Banks	11
	64 Banks	12
2KB Row buffer		
1-port	16 Bank	13
	32 Banks	14
	64 Banks	15
2-port	16 Bank	16
	32 Banks	17
4KB Row buffer		
1-port	16 Bank	18
	32 Banks	19
2-port	16 Bank	20

We are using a simple methodology based on elimination.

1) Use Cacti 3DD to estimate DRAM 3Ts and 3Es parameters and area of selected combinations from table 7.1. We assumed a stacked DRAM with 4 cell layers and 1 logic layer for all row-buffer sizes.
2) We use 3Ts and 3Es to compute DRAM cache hit latency and energy per access using equations from chapter 5.
3) We use Pareto optimality to determine the most efficient organization among the given choices. The lowest point on the pareto frontier is the best row buffer size given our choices.
4) We use the full methodology of Chapter 6 for a discussion of the possible configurations that can result from the ideal layout. Execution time and total energy are decisive factors. We will also collect Miss Rate (MR) and Row Buffer Hit Rate (RBHR) to complete analysis. However, since we already have an analysis of our two configurations (SSSR-L2-8, and SSSR-16), we do not need to run a full simulation on all of them; we can proceed with quick elimination and only run a full simulation on the best ones to avoid duplicate work.

7.3. Impact of Row Buffer Size on performance

7.3.1. Row Buffer size of 256B

Table 7.2. CACTI3DD specs for 256B row buffer memory with one port

Total size	256MB	Timing Parameters (ns)			
Port io	128	tRP	tRCD	tCL	tRAS
Burst	4	3.99146	3.98594	8.39612	7.28334
Row Size	256B				
#port	1				
#bank	32	Energy Parameters (nJ)			
Width	2.84806	e_ACT	e_READ	e_WRITE	e_PRE
Length	8.7149	0.374243	1.86226	1.8624	0.34318
Area(mm2)	12.4103	Area per port		12.4103	

Table 7.3. CACTI3DD specs for 256B row buffer memory with 2-port

Total size	256MB	Timing Parameters (ns)			
Port io	128	tRP	tRCD	tCL	tRAS
Burst	4	2.85692	3.49161	6.68696	5.45165
Row Size	256B				
#port	2				
#bank	32	Energy Parameters (nJ)			
Width	2.68664	e_ACT	e_READ	e_WRITE	e_PRE
Length	5.5550	0.335865	1.46805	1.46819	0.31589
Area(mm2)	14.9242	Area per port		7.4621	

From Table 7.2 and 7.3, it is not surprising that the read and write energy, as well as access delay of 1-port system, are higher than those of 2-port system. But the later has a slightly reduced area efficiency. For the same memory size, it takes 14.9 mm2 for 2-port rather than 12.4 mm2 used for a 1-port system. But the former is faster and energy efficient at all levels. Table 7.4 shows that worst-case hit latency is improved by 18% from 1-port to 2-port. The same goes for worst-case energy per access by 20%. Based on this, we will choose 2-port layout for 256B row buffer memory.

Table 7.4. Comparison of 1-port vs. 2-port on 256B row buffer

	RBM	
	Hit latency (ns)	Energy (nJ)
1-port	36.91576	6.304347
2-port	30.15941	5.056042
% diff	18	20

7.3.2. Row Buffer size of 512B

Table 7.5. CACTI3DD specs for 512B row buffer memory with 1-port

Total size	256MB	Timing Parameters (ns)			
Port io	128	**tRP**	**tRCD**	**tCL**	**tRAS**
Burst	4	3.34425	4.4117	6.96979	6.58051
Row Size	512B				
#port	1				
#bank	32	**Energy Parameters (nJ)**			
Width	4.33573	**e_ACT**	**e_READ**	**e_WRITE**	**e_PRE**
Length	2.8333	0.379708	1.5648	1.56494	0.35412
Area(mm2)	12.2844	Core Area		12.2844	

Table 7.6. CACTI3DD specs for 512B row buffer memory with 2-port

Total size	256MB	Timing Parameters (ns)			
Port io	128	**tRP**	**tRCD**	**tCL**	**tRAS**
Burst	4	2.80064	4.07571	6.09797	5.74362
Row Size	512B				
#port	2				
#bank	32	**Energy Parameters (nJ)**			
Width	4.08047	**e_ACT**	**e_READ**	**e_WRITE**	**e_PRE**
Length	4.0940	0.346934	1.36692	1.36706	0.32682
Area(mm2)	16.7056	Area per port		8.3528	

Tables 7.5 and 7.6 show the estimates for both 1-port and 2-port respectively. We can see that read or write energy is higher for 1-port compared to 2-port layout. Table 7.7 confirms that hit latency and energy per access for 2-port are better than those of 1-port by 11 and 12% respectively. We believe that for a small increase in area (e.g. 4x3 to 4x4), an improvement over 10% is worth it. It is enough to conclude that 2-port is also better for 512B row buffer memory.

Table 7.7. Comparison of 1-port vs. 2-port on 512B row buffer

	RBM	
	Hit latency (ns)	Energy (nJ)
1-port	32.41532	5.428369
2-port	28.92026	4.774658
% diff	11	12

Three possible configurations can come from this as shown in figure 7.1. (a) is our low-energy configuration SSSR-LE-8. The other two are the alternatives to increase RBH. They are the two potential candidates for full simulation run as we applied it in chapters 6. With 1x64B tag + 8x64B data, the total memory size including 256MB of data is 288MB with 89% of data area.

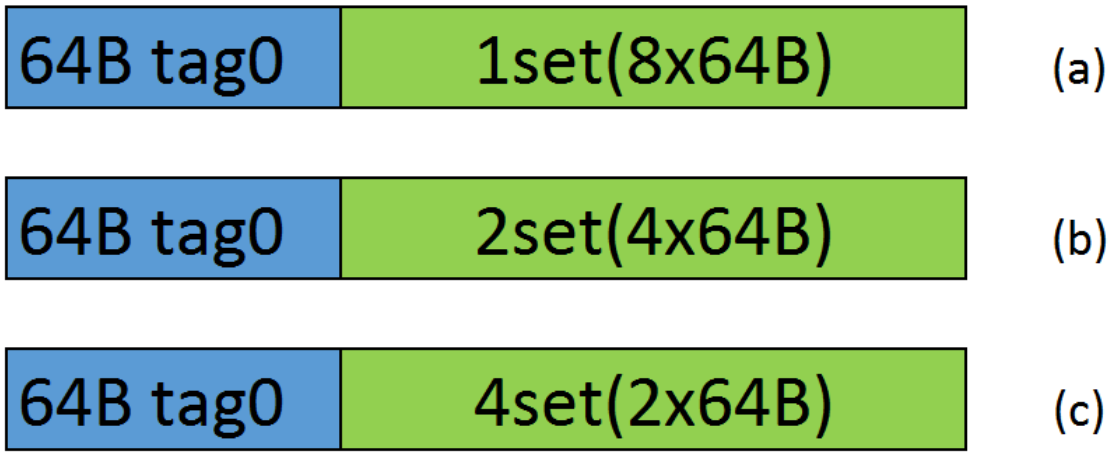


Figure 7.1. Possible configurations from 512B 2-port row buffer memory

7.3.3. Row Buffer size of 1KB

Table 7.8. CACTI3DD specs for 1KB row buffer memory with 1-port

Total size	256MB	Timing Parameters (ns)			
Port io	128	**tRP**	**tRCD**	**tCL**	**tRAS**
Burst	4	3.91931	5.85707	6.24765	7.9599
Row Size	1KB				
#port	1				
#bank	32	**Energy Parameters (nJ)**			
Width	6.67807	**e_ACT**	**e_READ**	**e_WRITE**	**e_PRE**
Length	2.2153	0.407381	1.57627	1.57642	0.376
Area(mm2)	14.7938	Area per port		14.7938	

Table 7.9. CACTI3DD specs for 1KB row buffer memory with 2-port

Total size	256MB	Timing Parameters (ns)			
Port io	128	**tRP**	**tRCD**	**tCL**	**tRAS**
Burst	4	3.63376	5.58098	5.89874	7.43751
Row Size	1KB				
#port	2				
#bank	32	**Energy Parameters (nJ)**			
Width	6.48096	**e_ACT**	**e_READ**	**e_WRITE**	**e_PRE**
Length	3.6445	0.37774	1.50963	1.50977	0.3487
Area(mm2)	23.6196	Area per port		11.8098	

Table 7.10. Comparison of 1-port vs. 2-port on 1KB row buffer

	RBM	
	Hit latency (ns)	Energy (nJ)
1-port	32.26933	5.512337
2-port	30.66096	5.255469
% diff	5	5

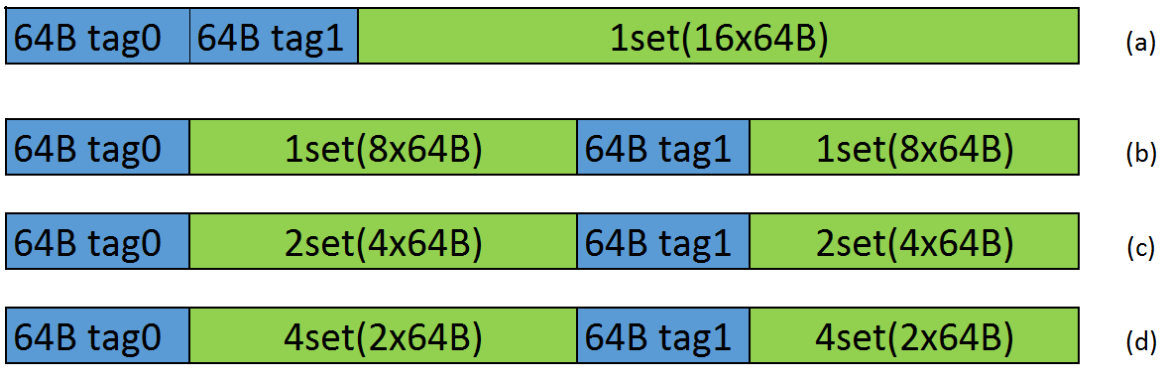


Figure 7.2. Possible configurations from 1 KB 2-port row buffer memory

Table 7.8 and 7.9 show very close values between 1-port and 2-port for 1 KB row buffer. The difference is within 5% as seen in Table 7.10. For only 5% improvement, the area is almost doubled. So we are choosing 1-port for its reduced area even though its performance is slightly lower. From that, Figure 7.2 shows four possible set associative configurations that can come out of this. They all use one port sequentially to avoid doubling precharge and activation energy. Figure 7.2(a) is the full 16-way associative where the two tag blocks are all used simultaneously. It is the worst case because tag serialization will increase both hit latency and energy per access. It is similar to the SSSR-16 configuration whose full simulation in chapter 6 revealed less competitive total energy within DRAM cache. However, Figure 7.2 (b), (c), and (d) help avoid tag serialization. Table 7.10 reflects their latency and energy values.

7.3.4. Row Buffer size of 2KB

Table 7.11. CACTI3DD specs for 2KB row buffer memory with 1-port

Total size	256MB	**Timing Parameters (ns)**			
Port io	128	**tRP**	**tRCD**	**tCL**	**tRAS**
Burst	4	6.46269	9.55919	7.13299	12.4198
Row Size	2KB				
#port	1				
#bank	32	**Energy Parameters (nJ)**			
Width	6.67807	**e_ACT**	**e_READ**	**e_WRITE**	**e_PRE**
Length	3.7783	0.473608	2.06182	2.06196	0.41974
Area(mm2)	25.2320	Area per port		25.2320	

Table 7.12. CACTI3DD specs for 2KB row buffer memory with 2-port

Total size	256MB	**Timing Parameters (ns)**			
Port io	128	**tRP**	**tRCD**	**tCL**	**tRAS**
Burst	4	6.31067	9.27791	6.95551	12.0232
Row Size	2KB				
#port	2				
#bank	32	**Energy Parameters (nJ)**			
Width	11.4129	**e_ACT**	**e_READ**	**e_WRITE**	**e_PRE**
Length	3.9881	0.444286	2.02793	2.02807	0.39245
Area(mm2)	45.5154	Area per port		22.7577	

Table 7.13. Comparison of 1-port vs. 2-port on 2KB row buffer

	RBM	
	Hit latency (ns)	Energy (nJ)
1-port	41.17085	7.078952
2-port	40.20511	6.920663
% diff	2	2

Like row buffer of 1KB, Table 7.11 and 7.12 also show very close values between 1-port and 2-port for 2 KB row buffer. The difference is only 2% within the margin of error as seen in Table 7.13. However, the area for 2-port is exceeding the improvement by far. For only 2% improvement, the area is almost doubled. So we are choosing 1-port for its reduced area as the performance difference is negligible. We are not going to analyze the possible configurations that can result from this 2KB row size because we have done enough analysis on it in chapter 4, 5, and 6.

7.3.5. Row Buffer size of 4KB

To complete our analysis, we are also studying a larger row buffer of 4KB as used by [7]. Like the case of 1KB and 2KB row size, as the row buffer increases, the multi-port layout provides area inefficiency for a negligible improvement in performance. So we assume the same for 4KB and only consider 1-port design. From Table 7.14, we can see that all the timing parameters have almost quadruple compared to 1KB row memory, and all energy parameters doubled. The total hit latency increased to 66 ns and energy/access to 10.5 nJ. These numbers characterize the MSSR-SA-S8 proposed in figure 4.6 as the authors are using a 4KB row buffer memory for their illustration.

Table 7.14. CACTI3DD specs for 4KB row buffer memory with 1-port

Total size	256MB	**Timing Parameters (ns)**			
Port io	128	**tRP**	**tRCD**	**tCL**	**tRAS**
Burst	4	14.3453	19.4736	9.56766	24.3528
Row Size	4KB				
#port	1				
#bank	32	**Energy Parameters (nJ)**			
Width	21.2692	**e_ACT**	**e_READ**	**e_WRITE**	**e_PRE**
Length	2.7535	0.607745	3.14073	3.14087	0.50724
Area(mm2)	58.5648	Area per port		58.5648	

7.3.6. Pareto Optimality for a choice of energy-efficient Stacked DRAM organization

Table 7.15. Comparison of row buffers from 256B to 4 KB

	RBM		
	Hit latency(ns)	Energy (nJ)	Area (mm2)
512B-16Bank	31.22	4.73	12.33
512B-32Bank	28.92	4.77	16.71
512B-64Bank	27.78	4.85	25.39
256B-16Bank	35.05	5.23	12.50
256B-32Bank	30.16	5.05	14.92
256B-64Bank	27.53	4.91	19.50
1KB-16Bank	31.35	4.96	14.82
1KB-32Bank	30.66	5.26	23.60
1KB-64Bank	30.77	5.73	41.18
2KB-16Bank	40.27	6.53	25.34
2KB-32Bank	40.21	6.92	45.50
4KB-16Bank	65.15	9.84	58.60

Table 7.15 provides a summary of hit latency and energy per access for the selected row buffer sizes of 256B, 512B, 1 KB, 2 KB, and 4 KB and number of banks of 16, 32, and 64. From this table we generated Figure 7.3 which represents the energy per access versus hit latency for given DRAM cache configuration mapped in our considered row buffer sizes. There are 12 points representing the following set: (16, 32, 64) banks of (256B, 512B, 1 KB) row buffer. (16, 32) banks of 2 KB row buffer, and 16 bank of 4 KB row buffer. The four lowest point A, B, C, and D consisting of (16, 32, 64) banks of 512 B row buffer and 64 bank of 256 B row buffer represent the *pareto frontier* or *pareto efficient choices*. The other 8 points (e.g. red color) representing the other combinations have their values scattered above the

efficiency line. The bigger the row buffer size, the further away it gets from the efficiency line. Based on this, we can narrow our choice on these 4 points and dismiss the other 8 points. Among these, point B represents the mid-point. If we go further left, energy is increasing. Conversely, if we move further to the right, the hit latency is increasing. We can conclude that this point B which is 512B row buffer of 32 bank per port is the better choice for stacked DRAM organization for efficient DRAM cache.

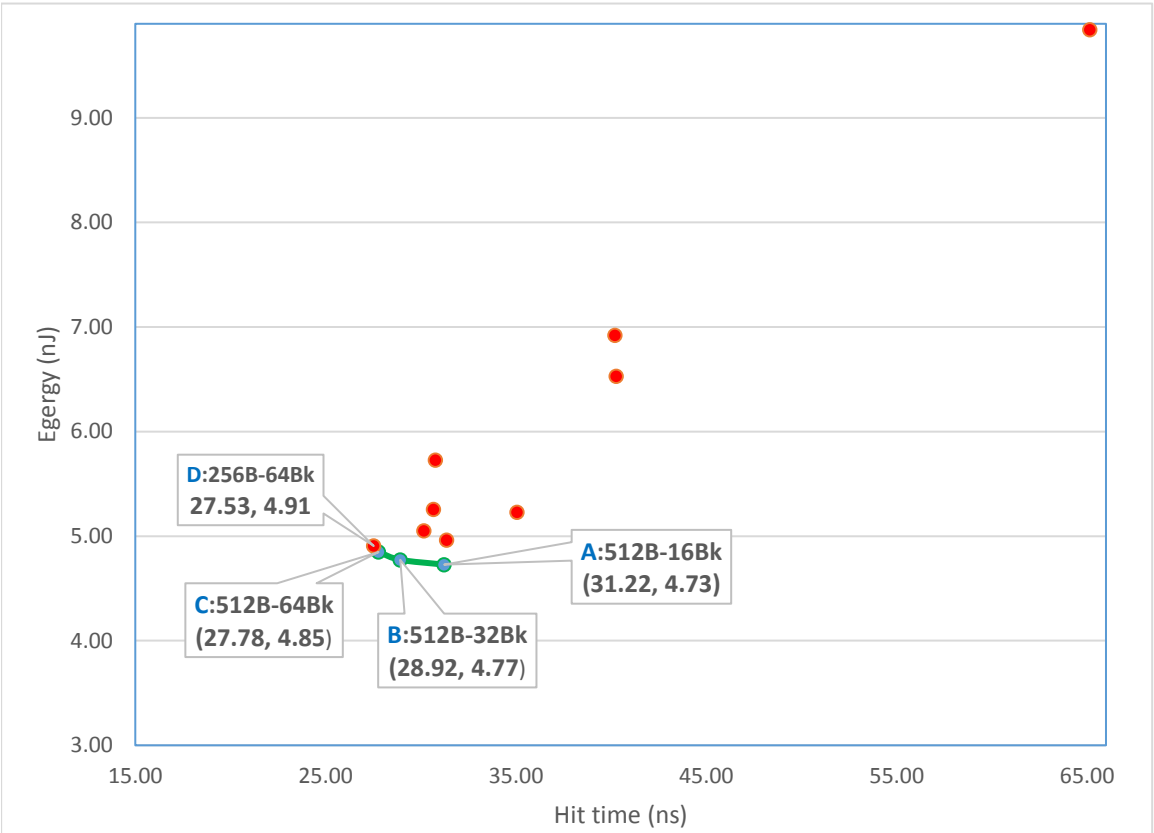


Figure 7.3. Pareto Optimality: Energy vs hit time for 256B, 512B, 1KB, 2KB, and 4KB row buffer with 16, 32, 64 banks

It is very clear to conclude that an adequate row buffer size for DRAM cache is a 512B based on figure 7.3. We understand that this conclusion is based on per access latency and energy only. To make a conclusion, we need total execution and energy after full simulations. We already have a baseline configuration (SSSR-LE-8) mapped in 512B row memory. This configuration had a net improvement over three other mapped in 2KB row. We can conclude without a doubt that it is a better configuration for DRAM cache application. However, there is still room for improvement when potential low energy Multiple set single row configurations in figure 7.2 (b) and (c) promise an increase in RBH. We are calling them **MSSR-LE-2S-4** and **MSSR-LE-4S-2** to represent 2-set 4-ways and 4-sets 2-ways respectively.

7.3.7. SRAM Consideration

In this chapter, we have been using 3D memory with four stacks to obtain a better configuration for cache applications. From Table 7.6, using CACTI3DD, a 4-layer 256MB stacked DRAM with 32-bank/port has an area of approximately 4 x 4 mm (16mm2). What would be the area overhead to design the same cache size of 256MB using SRAM technology in 2D? Is this even practical? To do this, we also used the same tool to estimate the area for 256MB SRAM cache with following parameters. Technology size: 32 nm, number of banks: 32. It provided a dimension of 33.1397 x 19.6 mm. Compared to 4 x 4 mm of stacked DRAM, it is 40x large and impractical. No modern CPU would be able to accommodate such cache on chip. Second, we want to estimate the SRAM cache size (e.g. Byte) that can fit in a 4 x 4 mm area. A cache of 3MB resulted with dimensions of 2.56 x 7.22 mm which is about 18 mm2 closer to 16 mm2 desired. So for the same footprint, we can fit a 4-layer 256MB stacked DRAM while only 3 MB of 2D SRAM can fit. We can see how stacked DRAM technology

helps to reduce the size of memories as area overhead. Although 3D technology can reduce area significantly, cost of 3DIC is still high due to stacking techniques and TSV fabrication. With more advance in manufacturing techniques, the cost can go down and justify the widespread adaptation of this technology.

7.4. Analysis of other 512B row-buffer size configurations

We would like to confirm our assessment on the other configurations resulting from different manipulations within our 512B row buffer size. We can map two sets of 4 blocks into a configuration, and we call it MSSR-LE-2S-4. Also, we can map four sets of 2 blocks into MSSR-LE-4S-2 to evaluate the effect on row buffer hit rate and miss rate as well as their impact on overall performance. We employ the full workload simulation of chapter 6 to collect total energy within DRAM cache and total execution time and compare them to our baseline low-energy configuration SSSR-LE-8. Figure 7.4 and 7.5 show both execution time and energy for all workloads. For performance, there is a remarkable difference on memory intensive workload: for instance, the 8-way configuration performs better on mcf than the two small associativity configuration whereas the 2-way performs the worst due to high miss rate. The same goes for energy. But in general, there is a slight difference in performance and energy between all three configurations. SSSR-LE-8 performs slightly better on average within 3% while MSSR-LE-4S-2 performs slightly better on average energy within less than 1%. This negligible difference is justified by variations in miss rate and row buffer hit rate. We expected the 8-way to have the lowest miss rate while the 2-way to have the highest row buffer hit rate and this is confirmed from Figure 7.6 and 7.7 respectively.

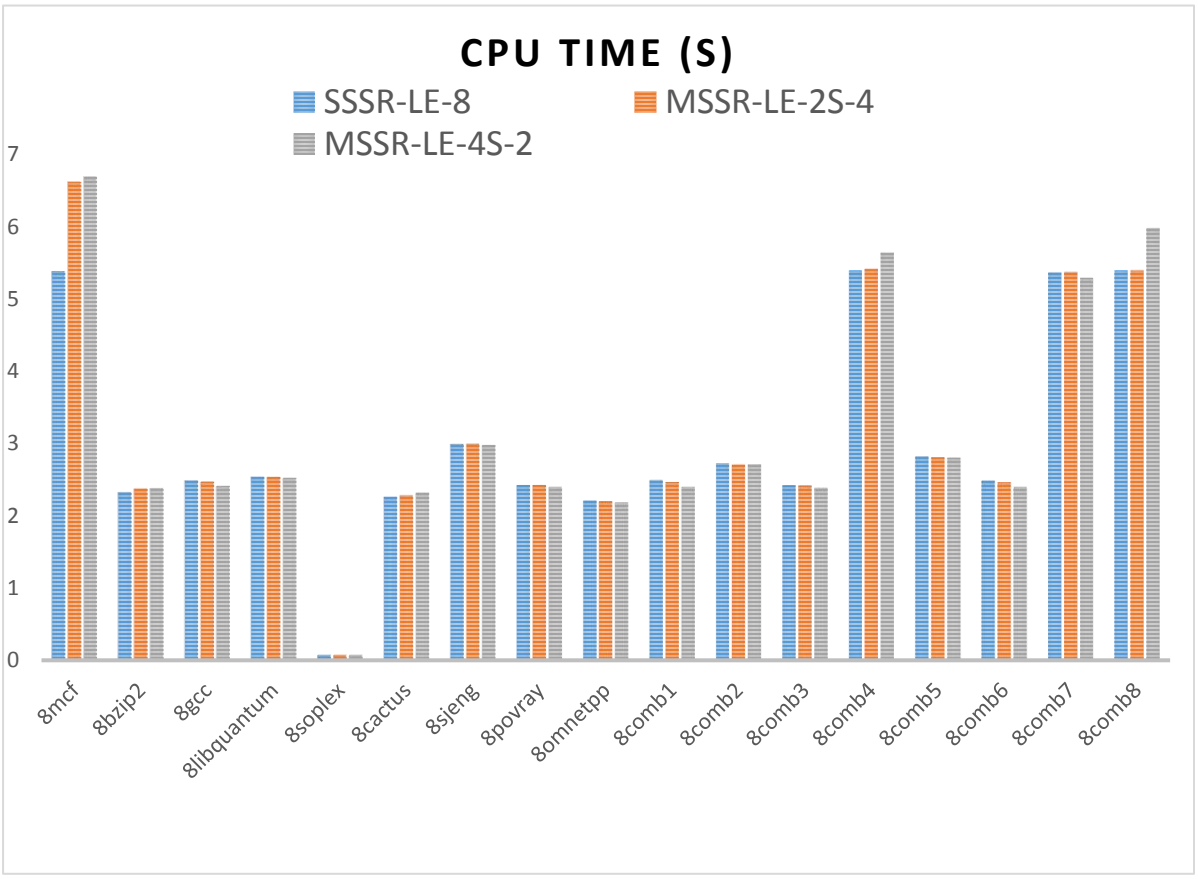


Figure 7.4. Total execution time for variable configurations within 512B row buffer

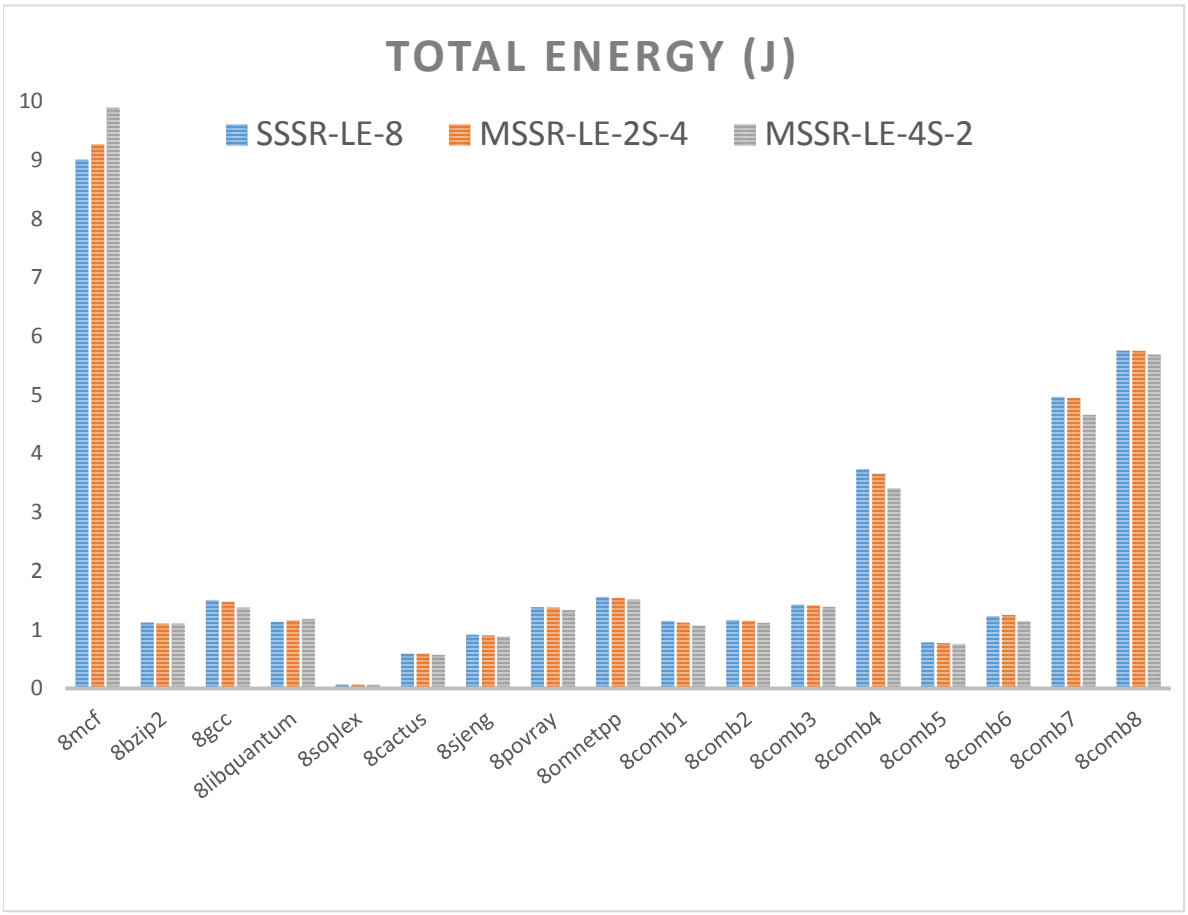


Figure 7.5. Total Energy for variable configurations within 512B row buffer

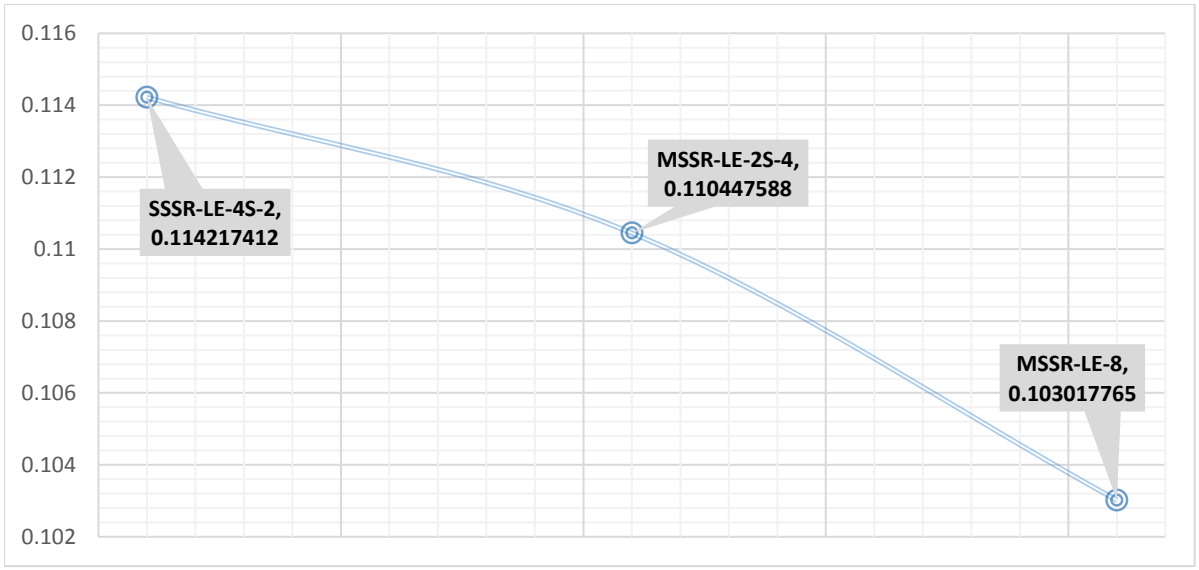


Figure 7.6. Average Miss Rate (AMR) for variable configs within 512B row buffer

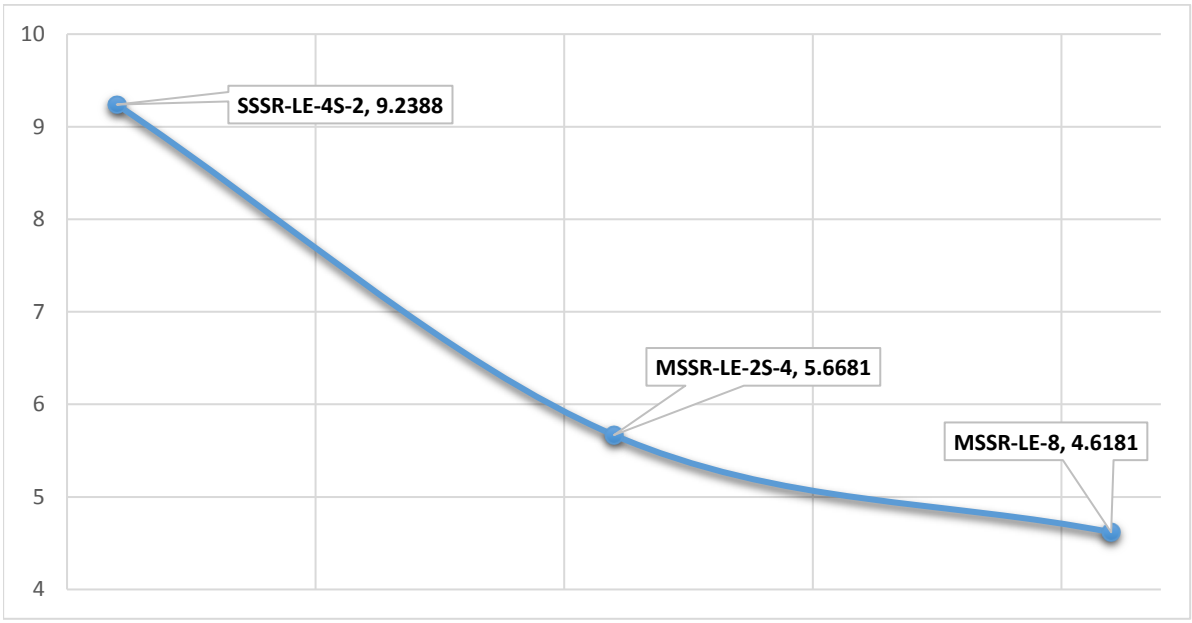


Figure 7.7. Average Row Buffer Hiss Rate (ARBHR) within 512B row buffer configs.

CHAPTER 8

Case Study: Design of Controller for a shared Last Level DRAM Cache Mapped in Tezzaron 3D stack Memory

8.1. Overview

In this research project, we proposed practical techniques and innovations to leverage upon the usage of 3D DRAM as the last level (LLC) of on-chip cache for multiprocessor. It provided both energy saving and speed improvement. This research effort was both an architectural study and a VLSI implementation of a complete DRAM cache controller. This design block is suitable for integration into heterogeneous multicore for fabrication in 65nm CMOS 3DIC technology. The cores used for this fabrication are architected through FabScalar [59] project at NC State. So in this Chapter, we are presenting the steps taken toward the implementation of a cache controller that lead to a microelectronic fabrication chip.

Throughout this research project, we have been partners with Tezzaron Semiconductor [53] who is the world leader in 3D-ICs. We obtained access to a series of their state-of-arts 3DIC DRAM's documentations noticeably the Octopus2 memory and DiRAM4 memory. We had started the design of DRAM cache controller for Tezzaron memory as outlined in our previous work [60] but the final fabrication tape-out was implemented for DiRAM4. We will give an overview of the design work we did to integrate Octopus2 and its adaptation for

DiRAM4. We will also present the architecture, logic design, and simulation platform leading to fabrication.

8.2. Tezzaron Memory specs

The first DRAM cache controller we tried to implement was for the 3-layer 256 MB Octopus 2 stacked DRAM. This memory has eight banks with 8-ports independently designed to support 128-bit read and write operations.

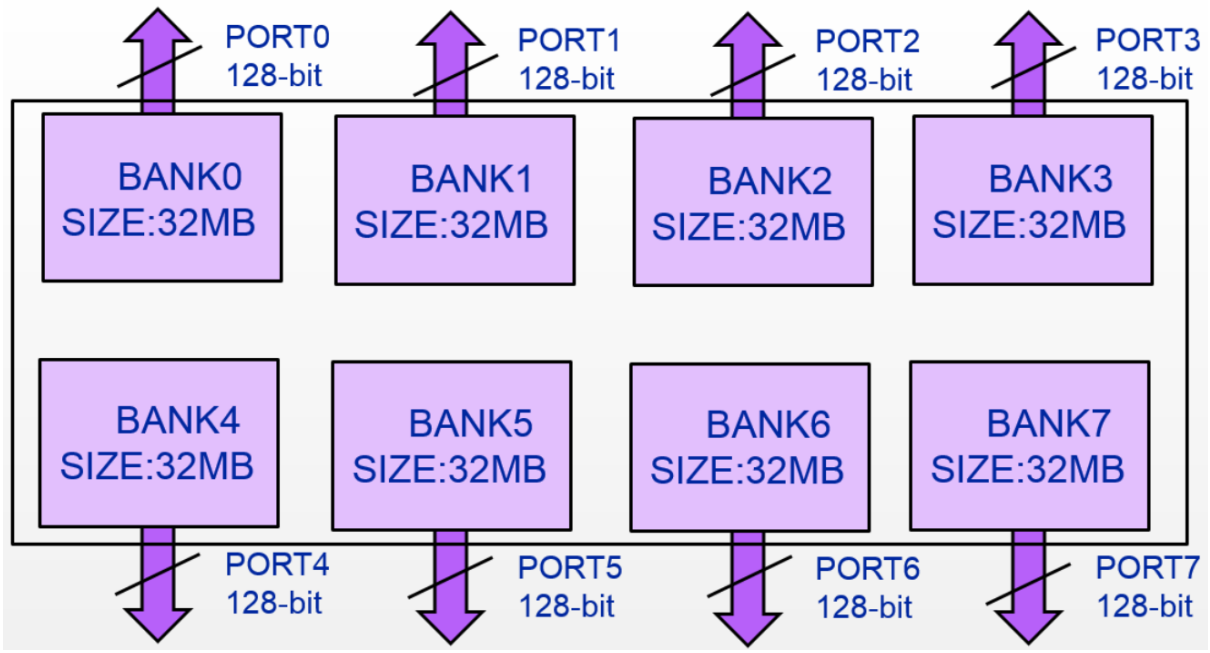


Figure 8.1. Tezzaron Octopus2 Bank specs

It was designed to run at a frequency up to 2 GHZ a fast data access through burst-4 and burst-8 mode. I was also assumed to consume only 3 pJ/bit. Figure 8.1 and 8.2 shows its banks and timing specs respectively. A remarkable 8-port was an interesting aspect of this

memory because we wanted to integrate it with open Sparc T2 [29] L2 cache controller. T2 cache L2 has eight banks, each with its independent controller the same way Octopus memory behaves. The diagram in Figure 8.2 represents the different timing delays and finite state machine (FSM). It defines sequence limitations of various memory operations such as precharge, Row Activation and column read and write operations. Our task was to implement this FSM into logic. It is a complex time diagram that requires careful design.

8.3. Micro-architecture

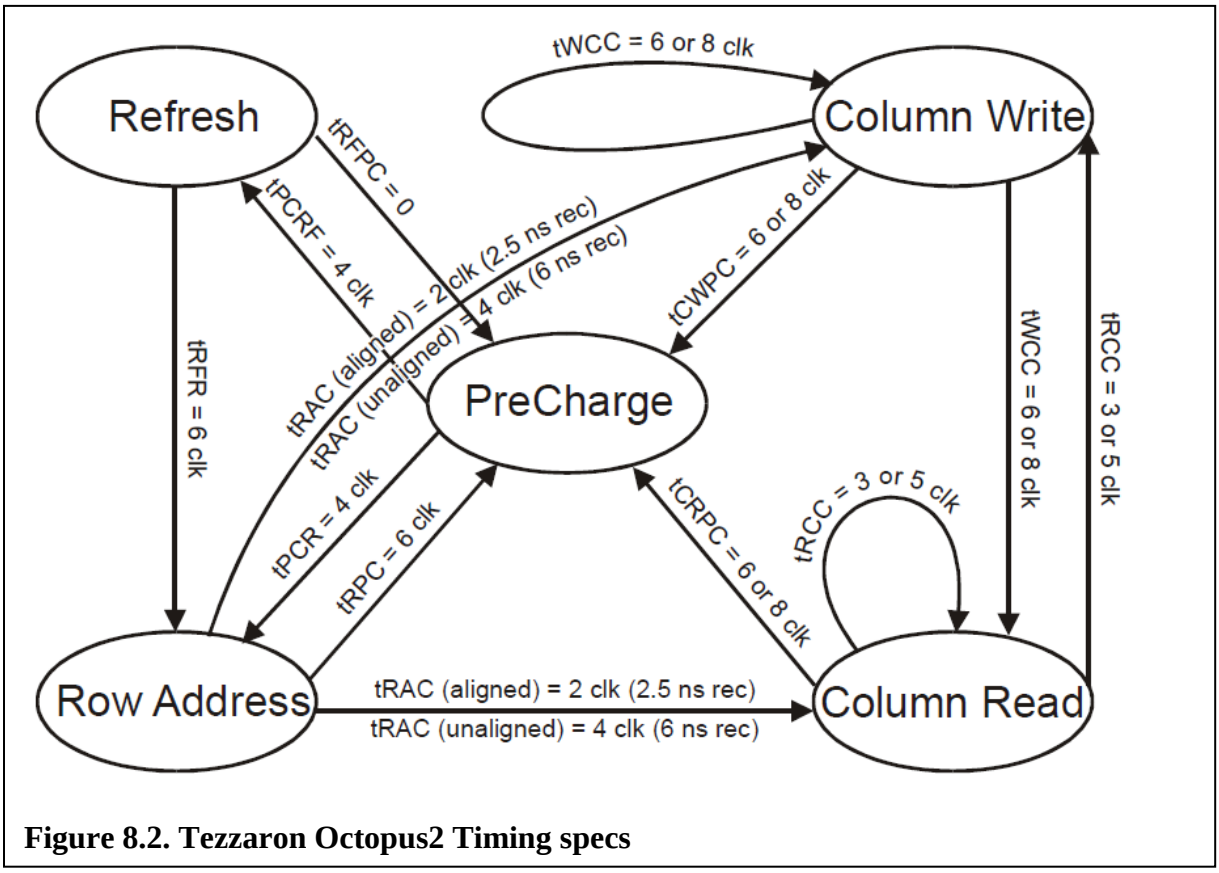


Figure 8.2. Tezzaron Octopus2 Timing specs

The micro-architecture of the overall cache controller was inspired by open-source SPARC T2 [29]. The novelty and main contribution are found in our DRAM-cache controller as highlighted in Figure 8.3. An implementation of a large DRAM as cache is not an industry standard. Therefore, by designing and testing it, we will demonstrate the concept of DRAM cache controller fabrication. Such a large cache of size can be very useful in server applications. A new RTL was written from scratch to implement all the building blocks from figure 8.3. The main parts are as follows:

The cache crossbar: is an 8x8 bus that connects the 8 cores to the 8 banks of the DRAM cache. The processor to cache crossbar (PCX) portion can process 8 requests from the CPU simultaneously. The cache to processor crossbar (CPX) can also process 8 data return from DRAM cache simultaneously. The priority of access is determined by the internal arbiters on the first come first serve basis.

Input Queue (IQ) and Output Queue (OQ): take in packets from CPUs to the controller and return data from the controller to the CPUs respectively.

Fill Buffer (FB): holds data returned from memory controller unit (MCU). But in our case, MCU along with other blocks in gray color (e.g. MB/WBB) were not implemented. We will use FPGA interface to preload data directly before program execution and therefore assume that all requests will be a hit. Therefore we will not implement miss buffer (MB).

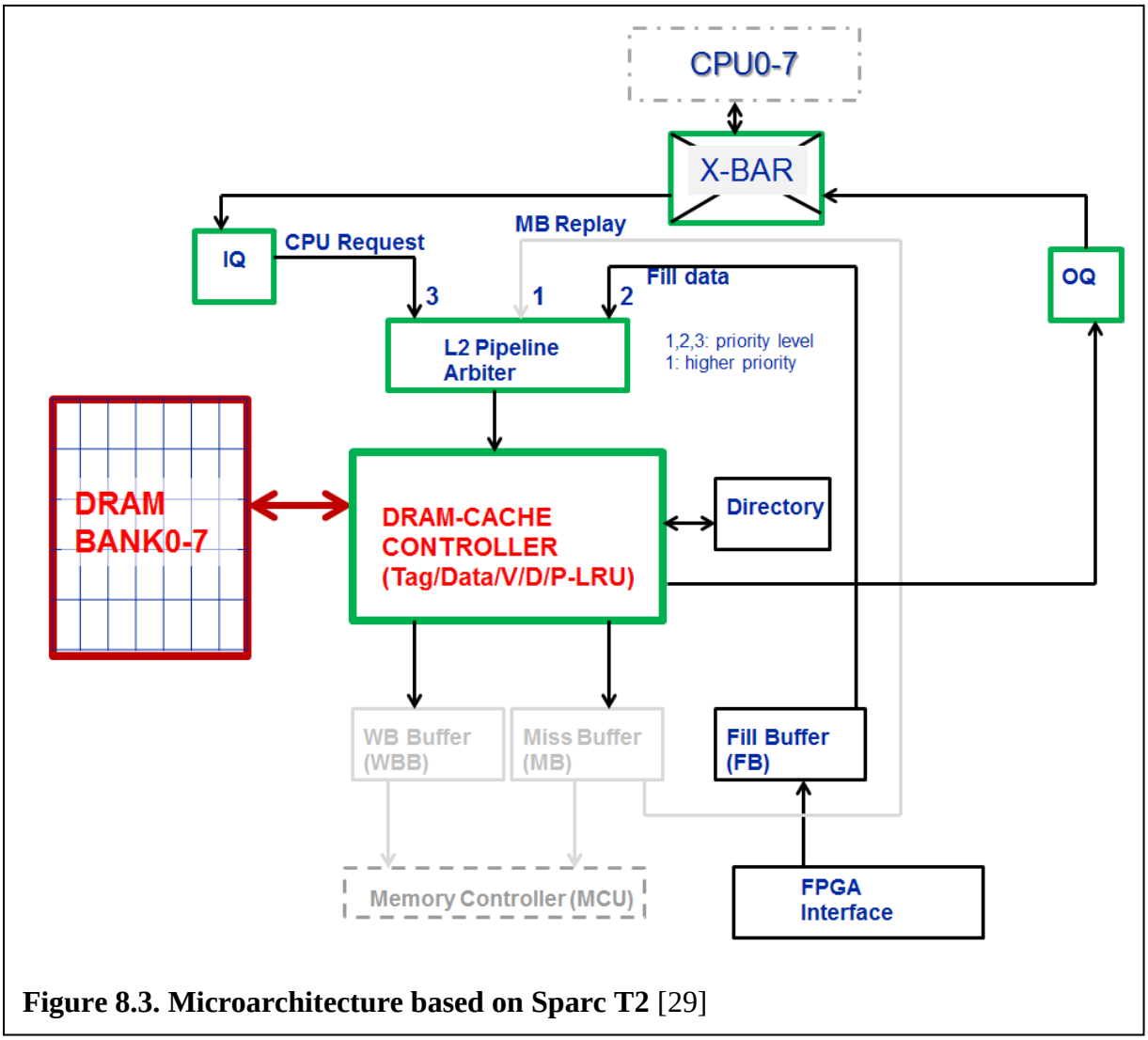


Figure 8.3. Microarchitecture based on Sparc T2 [29]

L2 Pipeline Arbiter: controls the priority of access to the controller between IQ, Fill buffer (FB), and Miss buffer (MB). But since MB was not implemented, only two elements were considered.

The DRAM-cache controller: This is the main engine used to control each DRAM bank. With eight banks, the controller was instantiated 8 times.

8.4. DRAM Cache Controller

Figure 8.4 shows the microarchitecture for DRAM cache controller for Octopus2 memory. The main blocks are *read data path*, *write data path* and *controller*.

The contro2ller will have both a read and a write data path to reflect the independent read and write ports of Octopus2 DRAM. We are using clocks with different phases to account for Tezzaron timing constraints.

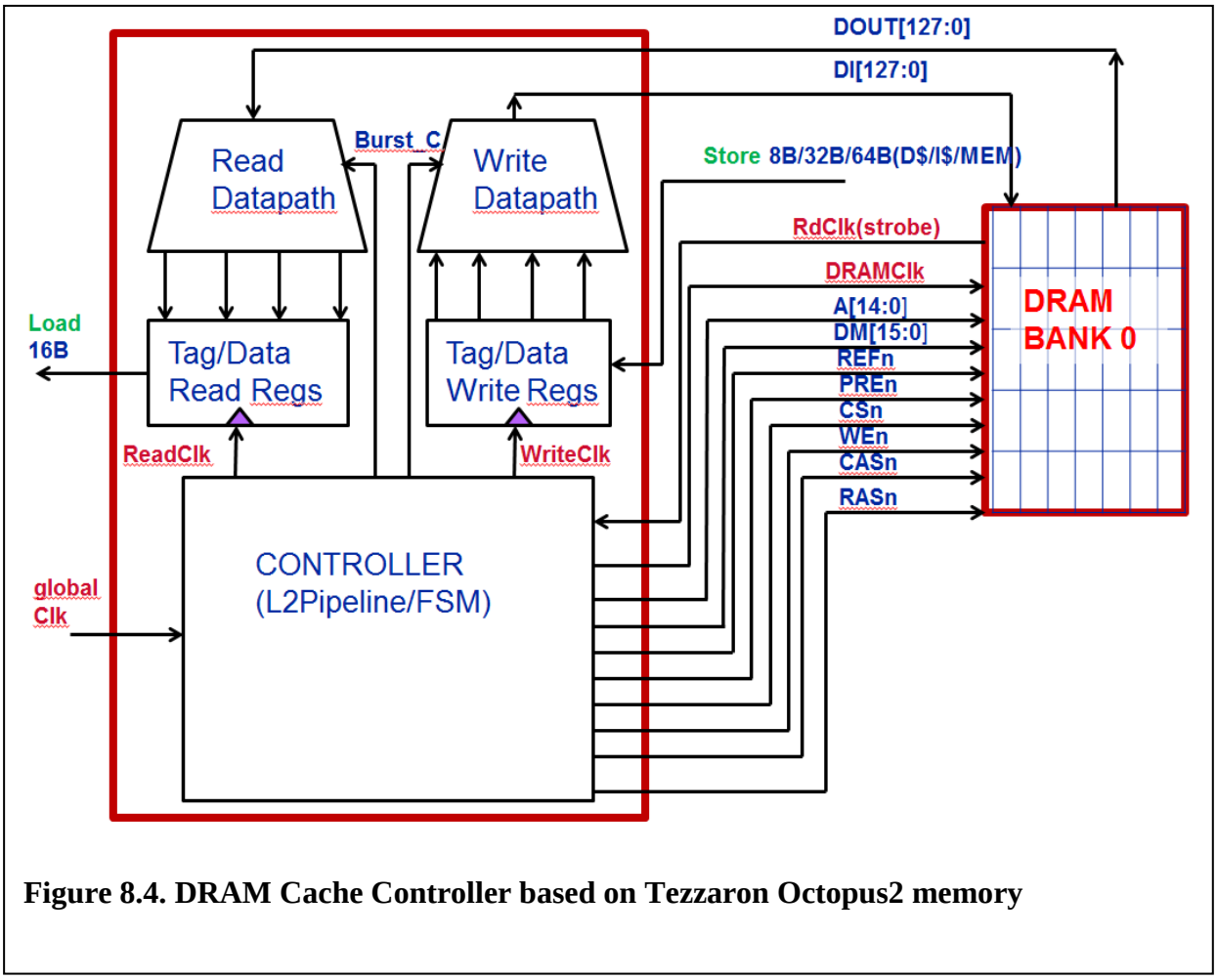


Figure 8.4. DRAM Cache Controller based on Tezzaron Octopus2 memory

8.4.1. Read Datapath

The Read data path is designed to process data coming from the memory. It takes in 128-bit in a burst of 4 and redistributes them into tag and data Read Registers (*Read Regs*) base on burst counter value as shown in figure 8.5.

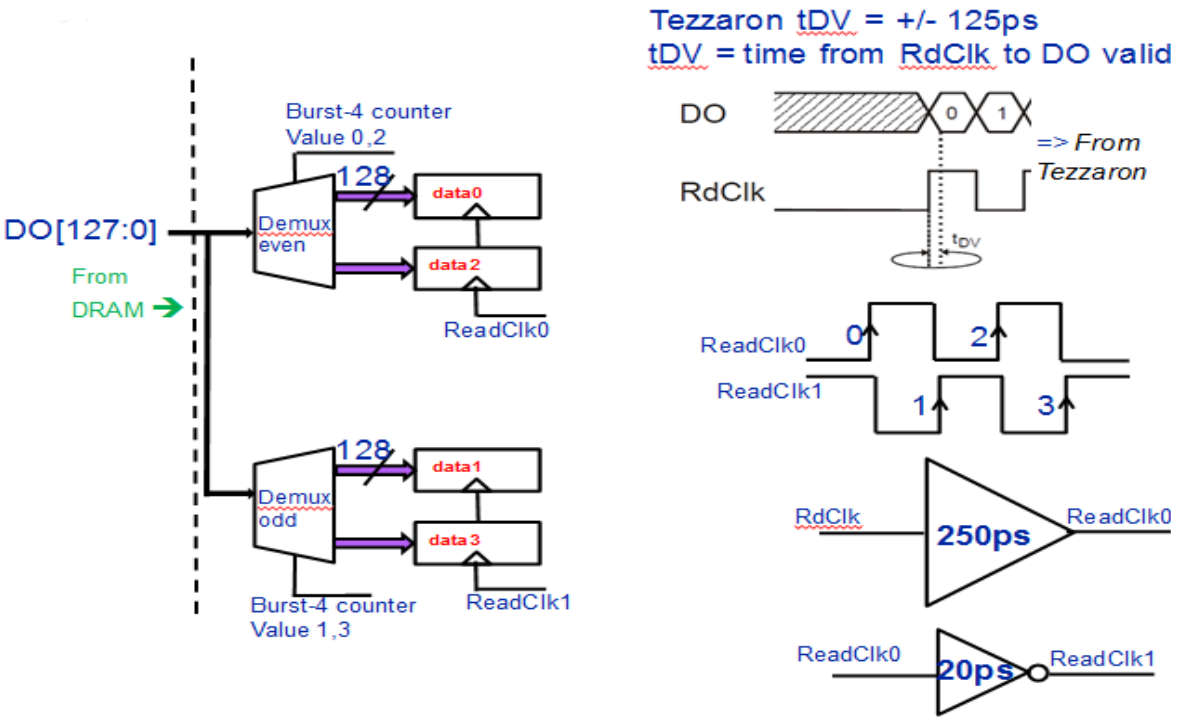


Figure 8.5. DRAM Cache Controller Read Datapath

Each one of the read registers is a 128 bit. Read clock (ReadClk0) is a delayed version of the data strobe (RdClk). It is used to clock the Read registers that have even numbers. ReadClk1 is the inverted version and used to clock read registers with odd numbers. The delaying mechanism is a chain of buffers with variable delays that account for variations of slow and fast corners. Four different clock edges corresponding to burst-4 mode help read data according to odd and even burst counter outputs.

8.4.2. Write Datapath

To process data going to the memory, we used the Write data path as shown in figure 8.6.

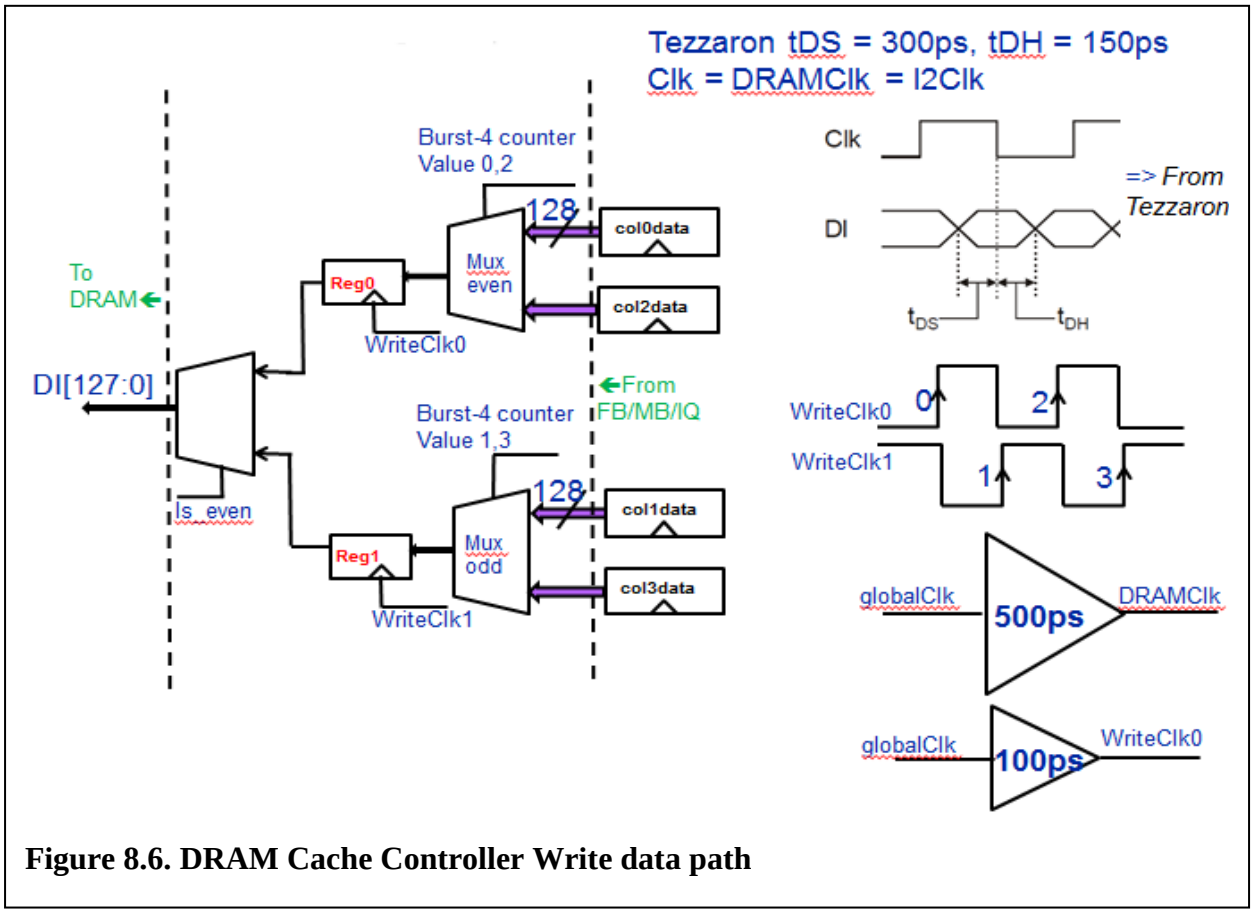


Figure 8.6. DRAM Cache Controller Write data path

It takes in 128-bit in a burst of 4 and redistributes them into 128-bit tag and data Write Registers (*Write Regs*) base on burst counter value. Write clock (WriteClk0) is delayed version of the external global clock (globalClk). It is used to clock the Write registers that have an even number. WriteClk1 is the inverted version and used to clock odd numbers of Write registers.

8.4.3. Controller

The Controller is the module that processes all the signals necessary to control the DRAM banks. Amongst those signals, there are DRAM commands (ACT, PRE, CAS, REF, WE, and CS), address busses (Row/columns address, and data mask (DM)), and clock redistributions. For instance, DRAM clock (DRAMClk) is a delayed version of the external global clock (globalClk) and is used as the memory clock. It is delayed to meet setup and hold time defined in Tezzaron Octopus data sheet.

All the command signals are asserted according to the values in the Status Register (SR). It is a 5-bit counter that defines the current and next steps in the pipeline. Figure 8.7 describes the Finite State Machine showing the Row-Buffer Miss (RBM) vs. Row-Buffer Hit (RBH) paths. RBH is the shortest path needed to reduce the steps significantly at each request. SR is reset to 0 once at startup to allow the first memory request to increment its value. Each value of SR defines the current step and the next step. For instance, when SR = 1, precharge in being performed, and retains its value until ACT or REF command is asserted. The timing diagram from Figure 8.2 defines all the legal transitions. The charts in Figure 8.7 describes the conditions from one step to another as well as the meanings of SR values.

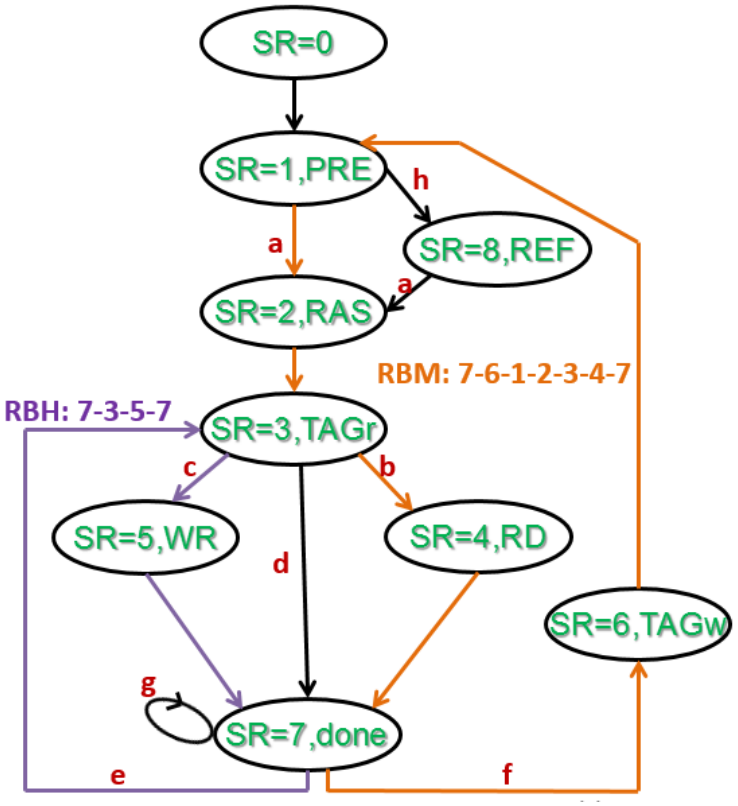


Figure 8.7. Controller

Figures 8.4 to 8.7 constitute the major parts of building block for DRAM Cache controller mapped in Tezzaron Octopus memory. The detailed implementation of this controller was captured in RTL and tested for hit time. Approximately 25 cycles elapsed before step 7 was reached using Tezzaron Octopus2 memory timing constraints. This hit latency was the worst case delay assuming the Row-buffer miss. Practical techniques were used to allow for multiple row buffer hit including a better controller and pre-load. In the next section, we will touch base on this notion that is 3D specific.

8.5. Preload: A 3D specific possibility

8.5.1. Motivation

Memory footprint of programs can be preloaded into the DRAM-cache. It can help to implement a system that can eliminate misses during program execution. In the traditional program execution, the benchmark memory footprint is loaded into the main memory. The first access to a given data goes across the entire memory hierarchy under a compulsory miss as the program executes. In this system, we can turn **Pre-load into a part of regular program execution** as follows:

i. Fill up DRAM-cache with the memory footprint data of benchmark at start-up. Traditionally the main memory is the only virtual infinite world to the CPU. With this technique, the DRAM-cache can also represent a virtually infinite world. The entire data set can be pre-loaded at program start-up and fit for some benchmark. Programs after that can execute without any access to off-chip main memory except at start-up.

During pre-load, there are virtually no misses as the only operation is to fill up tags and data.

ii. Use software to:

-compute total memory requirement of a benchmark

-sort checkpoint data addresses by DRAM-cache row number so that all blocks of that row will be written before moving to next row. Doing this will increase multiple row-buffer hit rate.

-decide on what type of pre-load will be performed as:

- ⇨ If DRAM-cache available memory is **greater than** program memory, proceed to Full Pre-load. This means DRAM-cache will be seen by CPU as the virtually infinite world so that no off-chip access will be performed during program execution. So **capacity + conflict miss latency = 0**. And **compulsory miss latency = pre-load latency**
- ⇨ If DRAM-cache available memory is **less than** benchmark memory but at least 20% of benchmark size, for example, proceed to Partial Pre-load of selected most frequently used data. So there will be off-chip access during program execution, but energy will still be saved. There is not yet a study of what data will be pre-loaded in this case as this also may require compiler designers to add some flexibilities to this effort.
- ⇨ Else proceed to the traditional memory allocation where no pre-load will be executed and data access goes through compulsory miss throughout the memory hierarchy.

8.5.2. Case Study: Rationale of Pre-load

The central question that may arise is: why pre-load? Why not just let the program run in a traditional way of compulsory misses through the entire memory hierarchy? The primary goal of pre-load is to **reinforce** the multiple row-buffer hits. It is an excellent way to load multiple blocks of data without opening the row multiple times. Consider benchmark such as Bzip2 with 184 MB of memory footprint. An analysis of its memory addresses shows that a cache of 23 blocks per set is required to avoid any conflict misses. With pre-load, 22 of these 23 blocks can be loaded under a single Row activation as shown in table 8.1.

Only the first block requires the full energy and latency (worst case) because the previous row had to be closed after tag update. The subsequent blocks are under row-buffer hit and therefore only require a fraction of energy and latency. In other terms, block #1 follows RBM path in figure 8.7 while blocks # 2 to 23 are forced to follow RBH path. Therefore, due to Pre-load only, on average there is **85% energy saving** to fill up every set of the cache. Given a DRAM-cache of over 240MB, the energy savings can be significant to the overall system. Without pre-load, misses through the entire memory hierarchy could require multiple row activation before filling every single block in one set.

Table 8.1: Bzip2 Preload

EFFECT ON ROW-BUFFER HIT		
Block#	E(nJ)	T(cycles)
1	4.486	25
2	0.512	10
3	0.512	10
4	0.512	10
5	0.512	10
6	0.512	10
7	0.512	10
8	0.512	10
9	0.512	10
10	0.512	10
11	0.512	10
12	0.512	10
13	0.512	10
14	0.512	10
15	0.512	10
16	0.512	10
17	0.512	10
18	0.512	10
19	0.512	10
20	0.512	10
21	0.512	10
22	0.512	10
23	0.512	10
Average	0.685	10.652

There is also 57 % access delay reduction. But it has a little impact during pre-load steps as compulsory misses still exist and require off-chip accesses for each block of data pre-loaded. Therefore, energy improvement is the only figure of merit to consider during pre-load. However, on the overall system improvement, the timing improvement is eminent as pre-loaded data will prevent off-chip accesses during program execution. Now this energy and time saving is significant only for single thread system with small workloads or any workload that can completely fit in DRAM cache. It is the reason why we have not used pre-load as the focus of our research. However, a partial pre-load can be a good research topic. Such research will be an extension of prefetching but in a larger scale.

8.6. Design Consideration from Octopus2 to DiRAM4

8.6.1. Difficulty of converting to DiRAM4

All the building blocks are based on Tezzaron Octopus memory but the final design was converted to match DiRAM4 specs. The main difference between the two memories is that DiRAM4 bus size was reduced from 128-bit to 32-bit and burst mode was changed from 4 and 8 to 2 and 8. The timing constraints became complicated because data go through two internal channels. These channels require two internal clocks: odd channel and even channel clocks simultaneously fetch data at a single data rate (SDR) of burst-8 for a total of (2x32bit) x8 = 512-bit per access. Two ACT or two PRE commands are needed within one clock cycle. To simplify our design and increase the chance of a working prototype for our tape-out, we considered only 1-channel and only send 1 ACT/PRE per request. It provided a total of 256-

bit per request. DiRAM4 has a row size of 512B, which provides 16x32B cache lines per row. One tag block of 32B serves 15 blocks of data for a 15-way set associative cache.

Another difficulty in designing cache controller for DiRAM4 was that it takes only 2-bit signal to control seven memory operations. It made clock phases and sequences very critical. The way to make this to work was to use the internal clocks so that row operations such as ACT/PRE/REF complete on rising edge. Data operations such as CAS for read and write complete on falling edge of the internal clock for a given channel. The micro-architecture in figure 8.3 remains the same. The significant changes appear in the DRAM Cache controller block of figure 8.4. Bus sizes were reduced from a 128-bit burst of 4 for Octopus2 memory to 32-bit burst of 8 for DiRAM4. Additionally, because of reduced I/O budget, only 18-bit DQs were used instead of 32-bit defined by the memory. Below is how we picked the 18-bit.

Each 32-bit fetched covers 2 tags of 16-bit each. [15:0] represent one tag and [31:16] the second tag. For 1st tag, only the first 6 bit [5:0] are used for tag storage and 3 last bits [15:13] are used for valid, dirty, and replacement (VDR). Bit [12:6] are ignored. For 2nd tag, the same is applied: [21:16] for tag storage, [31:29] for VDR and [28:22] unused. The read and write data paths of figure 8.5 and 8.6 remain the same except for the size of the bus. The number of internal registers was changed from 4X128-bit to 8X32-bit to accommodate for 15 tags of 16-bit each and 32B block size. Figure 8.8 shows a modified block diagram of DRAM cache controller for DiRAM4. This figure also includes the FPGA interface for the test bench. All the clocks will be internally processed within the FPGA to add necessary delays specified by DiRAM4 timing constraints.

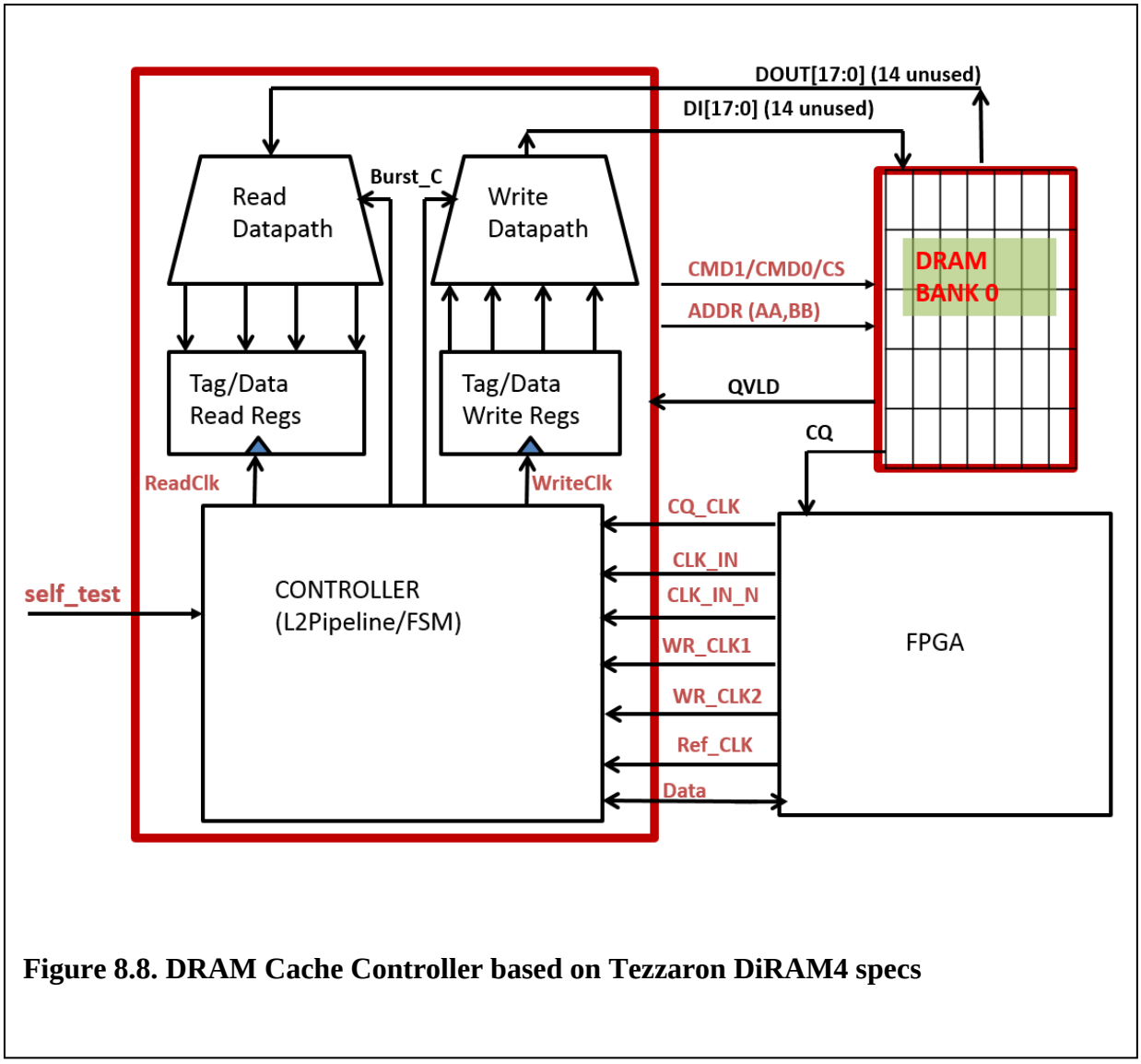


Figure 8.8. DRAM Cache Controller based on Tezzaron DiRAM4 specs

8.6.2. DiRAM4 Read and Write constraints

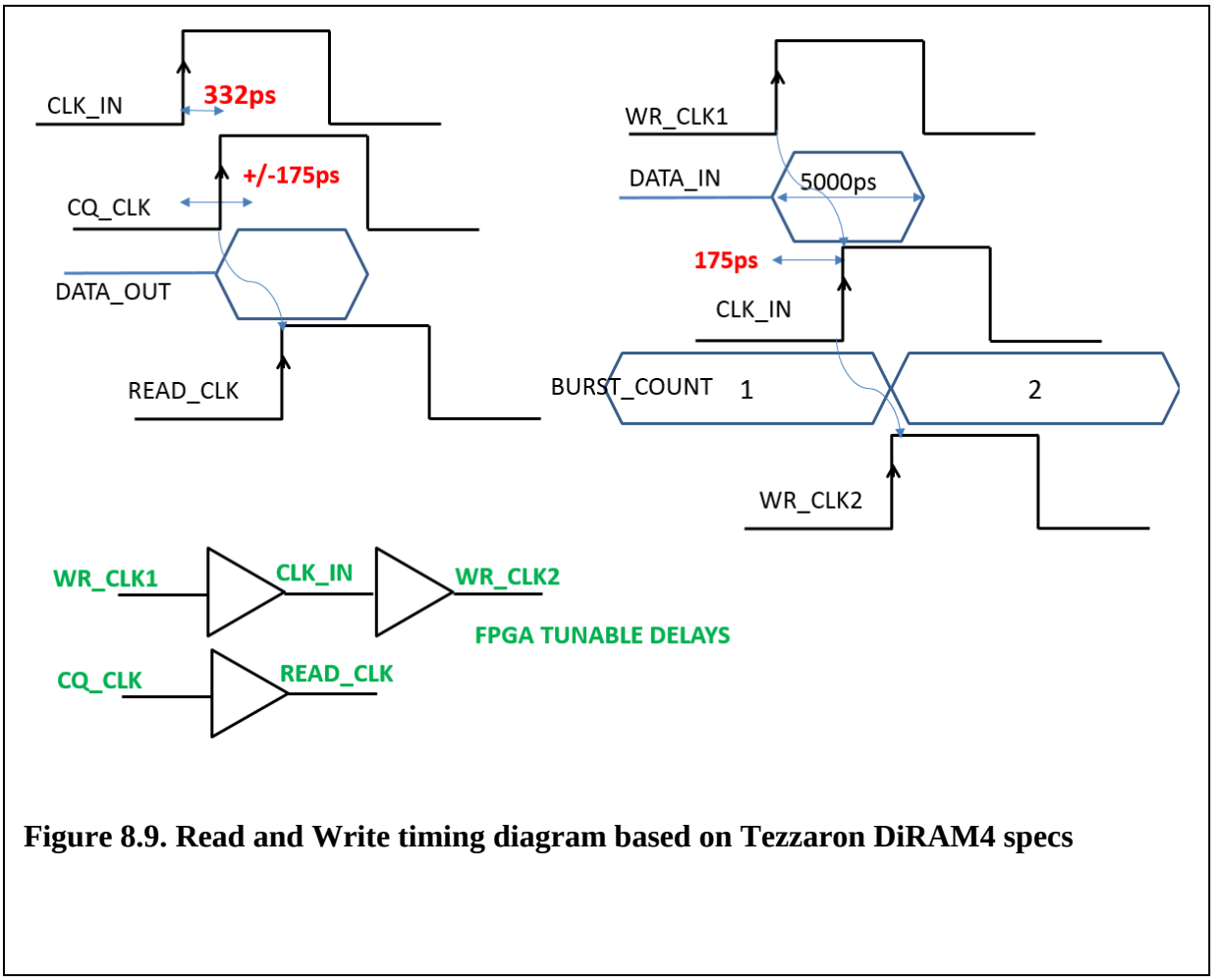


Figure 8.9. Read and Write timing diagram based on Tezzaron DiRAM4 specs

Figure 8.9 shows timing diagram with constraints for data read and write operations. For data read, for example, there is a delay of 332 ps from channel clock to output clock (tKCQ) and a +-175ps from output clock to data valid (tCQQV). READ_CLK, WR_CLK1, and WR_CLK2 can be generated by a fine-tuning mechanism within FPGA by applying series of buffers to meet these delay requirements. The RTL captured in Appendix B is the final code including all the necessary modules for complete tape-out.

CHAPTER 9

Conclusions and Future Work

9.1. Conclusion

In this work, we studied and analyzed a range of DRAM cache with row buffer size ranging from 256B to 4KB, and we determined that a DRAM cache mapped in unconventional 512B row buffer stacked memory was the better choice. Such stacked DRAM was set up to provide 2 independent ports with access to 32 independent banks each. A smaller row size significantly cut off the energy per access at row buffer miss. An 8-way single set single row mapping and low-energy (SSSR-LE-8) configuration resulting from such a memory improved energy by 36, 40, and 74 % compared to 3 other configurations mapped in 2KB row buffer memory. This low-energy mapping at the same time also increased the performance by 16, 18, and 41%. We also studied other configurations within 512B row buffer DRAM Cache by varying associativity number of sets per row. We concluded that there was only slight difference among them due to the variation of row buffer hit and miss rates. We modified GEM5 simulator to account for variable hit latency due to row buffer for accurate performance in term of execution time. We also added the energy model to the tool to estimate the DRAM cache energy consumption. We conclude our work with a case study to design the controller for DRAM cache mapped in two of Tezzaron stacked DRAM. This was a necessary hands-on exercise to acquire some insight on difficulty encountered when creating a system with complex time constraints.

9.2. Future Work: Heterogeneity

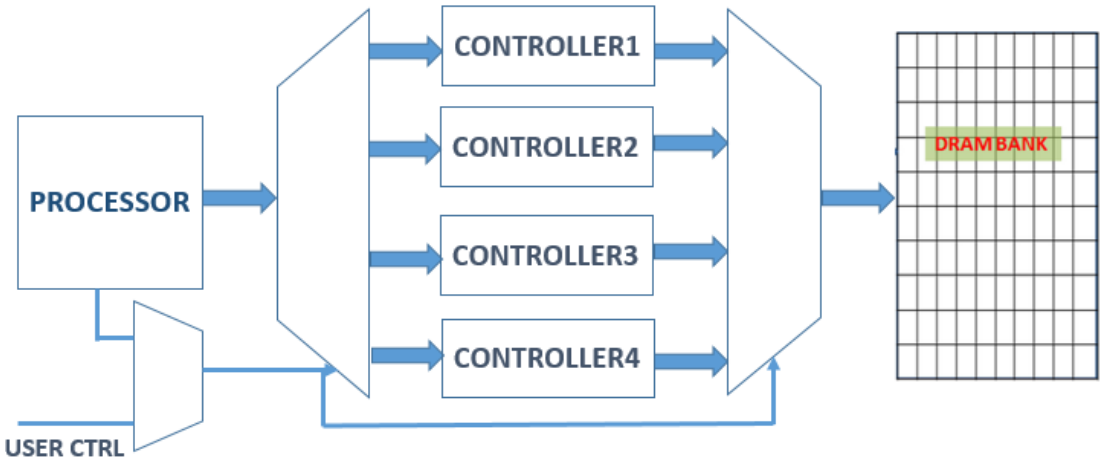


Figure 9.1. Heterogeneity in controller design

Instead of having a single configuration for all cases, we can provide the ability to pick a configuration that can best fit a specific situation to maximize performance. Figure 9.1 shows a multiple controller design that can help accomplish this. There are two ways to operate. 1) static behaviors: we can choose a configuration at startup based on user defined criteria or CPU profiling. 2) Dynamic behavior: heterogeneity is done during execution. A true adaptive algorithm can help achieve that. The question is to study how to switch control during execution and the advantages and drawbacks of this. It is a central question that constitutes a possible future work.

BIBLIOGRAPHY

[1] G. H. Loh and M. D. Hill. “Efficiently enabling conventional block sizes for very large die-stacked DRAM caches”. MICRO-44, 2011.

[2] G. H. Loh and M. D. Hill, “Supporting very large DRAM caches with compound-access scheduling and missmap”, IEEE Micro, vol. 32, no. 3, pp. 70–78, 2012.

[3] M. K. Qureshi and G. H. Loh, “Fundamental latency trade-offs in architecting DRAM caches” IEEE/ACM, 45th annual international symposium on microarchitecture, pp. 235-247 2012.

[4] Hameed, F.; Bauer, L.; Henkel, J., “Simultaneously Optimizing DRAM Cache Hit Latency and Miss Rate via Novel Set Mapping Policies”, Compilers, Architecture and Synthesis for Embedded Systems (CASES), 2013 International Conference.

[5] International Technology Roadmap for Semiconductors, 2011
http://www.itrs.net

[6] K. Banerjee, S. Souri, P. Kapur, and K. Saraswat, "3-D ICs: a novel chip design for improving deep-submicrometer interconnect performance and systems-on-chip integration," Proceedings of the IEEE, vol. 89, no. 5, pp. 602-633, May 2001

[7] Hameed, F.; Bauer, L.; Henkel, J., “Architecting On-Chip DRAM Cache for Simultaneous Miss Rate and Latency Reduction”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems Vol. 35 , Issue 4, pp. 651 – 664, March 2016

[8][546_book] J.M. Rabaey, A.Chandrakasan and B.Nikolic, “Digital Integrated Circuits, a design perspective”, Prentice-Hall, Upper Saddle River, NJ, 2003, pp. 624-630

[9][521_book] J.L. Hennessy and D.A. Patterson, “Computer Architecture: a Quantitative Approach”, Morgan Kaufmann, Waltham, MA, 2012, p. 3,24,55,73

[10][Y_zhang] Yang Zhang; Sarvey, T.E.; Bakir, M.S. "Thermal Challenges for Heterogeneous 3D ICs and Opportunities for Air Gap Thermal Isolation", 3D Systems Integration Conference (3DIC), 2014

[11]W.R. Davis , J. Wilson, S.Mick, J.Xu, H. Hua, C.Mineo, A. Sule, M.Steer, P. Franzon, “Demystifying 3DICs: the pros and cons of going vertical", IEEE design & test of computers, pp. 498-5107, Dec. 2005

[12] W. A. Wulf and S. A. McKee. “Hitting the memory wall: implications of the obvious”. SIGARCH Comput. Archit. News, 23(1):20–24, 1995

[13] G. Moore. Cramming more components into integrated circuit. Electronics, 38(8), 1965

[14] J. Mudigonda, H. M. Vin, R. Yavatkar, "Overcoming the memory wall in packet processing: hammers or ladders?" Proceedings of the 2005 ACM symposium on Architecture for networking and communications Systems, 2005

[15][Xeon] Intel Xeon Processor E-7 8893, http://ark.intel.com/products/75260/Intel-Xeon-Processor-E7-8893-v2-37_5M-Cache-3_40-GHz

[16] D. Wendel et al., "The implementation of POWER7: A highly parallel and scalable multi-core high-end server processor," in IEEE ISSCC Tech. Dig., San Francisco, CA, USA, 2010, pp. 102–103.

[17] R. X. Arroyo, R. J. Harrington, S. P. Hartman, and T. Nguyen, "IBM power7 systems," IBM J. Res. Develop., vol. 55, no. 3, pp. 220–232, 2011.

[18] K. Bernstein, P. Andry, J. Cann, P. Emma, D. Greenberg, W. Haensch, M. Ignatowski, S. Koester, J. Magerlein, R. Puri, and A. Young, "Interconnects in the third dimension: Design challenges for 3D ICs," in 44th ACM/IEEE Design Automation Conference, June 2007, pp. 562-567

[19] B. Black et al., "Die-stacking (3D) microarchitecture," in Proc. IEEE/ACM Int. Symp. Microarchit. (MICRO), Orlando, FL, USA, 2006, pp. 469–479.

[20] F. Hameed, L. Bauer, and J. Henkel, "Reducing inter-core cache contention with an adaptive bank mapping policy in DRAM cache," in Proc. IEEE Int. Conf. Hardw. Softw. Codesign Syst. Synth. (CODES+ISSS), Montreal, QC, Canada, 2013, pp. 1–8.

[21] F. Hameed, L. Bauer, and J. Henkel, "Reducing latency in an SRAM/DRAM cache hierarchy via a novel tag-cache architecture," in Proc. Design Autom. Conf. (DAC), San Francisco, CA, USA, 2014, pp. 1–6.

[22] C.-C. Huang and V. Nagarajan, "ATCache: Reducing DRAM cache latency via a small SRAM tag cache," in Proc. Int. Conf. Parallel Archit. Compilat. Tech. (PACT), San Francisco, CA, USA, 2014, pp. 51–60.

[23] D. Jevdjic, S. Volos, and B. Falsafi, "Die-stacked DRAM caches for servers: Hit ratio, latency, or bandwidth? have it all with footprint cache," in Proc. Int. Symp. Comput. Archit. (ISCA), Tel Aviv-Yafo, Israel, 2013, pp. 404–415.

[24] D. Jevdjic, G. H. Loh, C. Kaynak, and B. Falsafi, "Unison cache: A scalable and effective die-stacked DRAM cache," in Proc. IEEE/ACM Int. Symp. Microarchit. (MICRO), Cambridge, U.K., 2014, pp. 25–37.

[25] U. Kang et al., “8 Gb 3-D DDR3 DRAM using through-silicon-via technology,” IEEE J. Solid-State Circuits, vol. 45, no. 1, pp. 111–119, Jan. 2010.

[26] F. Hameed, L. Bauer, and J. Henkel, “Adaptive cache management for a combined SRAM and DRAM cache hierarchy for multi-cores,” in Proc. Design Autom. Test Europe (DATE), Dresden, Germany, 2013, pp. 77–82.

[27] K. Inoue, S. Hashiguchi, S. Ueno, N. Fukumoto, and K. Murakami, “3D implemented SRAM/DRAM hybrid cache architecture for high-performance and low power consumption,” in Proc. Int. Midwest Symp. Circuits Syst., 2011, pp. 1–4.

[28] G. H. Loh, “Extending the effectiveness of 3D-stacked DRAM caches with an adaptive multi-queue policy,” in Proc. IEEE/ACM Int. Symp. Microarchit. (MICRO), New York, NY, USA, 2009, pp. 201–212.

[29] Sun Microsystems, “OpenSPARC T2 processor megacell specification”, December 2007, revision A.

[30] G. H. Loh, “3D-stacked memory architectures for multi-core processors,” in Proc. Int. Symp. Comput. Archit. (ISCA), Beijing, China, 2008, pp. 453–464

[31] J.-S. Kim et al., “A 1.2V 12.8GB/s 2Gb Mobile Wide-I/O DRAM with 4x128 I/Os Using TSV-Based Stacking,” in ISSCC, 2011.

[32] T. Kgil, S. D’Souza, A. Saidi, N. Binkert, R. Dreslinski, T. Mudge, S. Reinhardt, and K. Flautner, “PicoServer: Using 3D Stacking Technology to Enable a Compact Energy Efficient Chip Multiprocessor,” in ASPLOS, 2006

[33] G. Loi, B. Agrawal, N. Srivastava, S. Lin, T. Sherwood, and K. Banerjee, “A Thermally-Aware Performance Analysis of Vertically Integrated (3-D) Processor-Memory Hierarchy,” in DAC, 2006.

[34] Y. H. Son, S. O, H. Yang, D. Jung, J. Ahn, J. Kim, J. Kim, and J. W. Lee, “Microbank: Architecting Through-silicon Interposer-based Main Memory Systems,” in SC, 2014.

[35] X. Dong, Y. Xie, N. Muralimanohar, N. Jouppi, and R. Kaufmann, “Simple but Effective Heterogeneous Main Memory with On-chip Memory Controller Support,” in SC, 2010

[36] Rotenberg, E. ; Dwiel, B.H. ; Forbes, E. ; Zhenqian Zhang ; Widialaksono, R. ; Basu Roy Chowdhury, R. ; Tshibangu, N. ; Lipa, S. ; Davis, W.R. ; Franzon, P.D. “Rationale for a 3D heterogeneous multi-core processor”
Computer Design (ICCD), 2013 IEEE 31st International Conference

[37] P.W. Diodato et al., “embedded DRAM: more than just a memory”, July 2000, IEEE communications Magazine, pp. 118-126.

[38] G. Birk, D.G. Elliott, B.E. Cockburn, “Design and Characterization of an embedded ASIC DRAM”, 1999 IEEE Canadian Conference on Electrical and Computer Engineering, pp. 427 – 432

[39] J. Sim, G. H. Loh, H. Kim, M. O’Connor, and M. Thottethodi, “A mostly-clean DRAM cache for effective hit speculation and selfbalancing dispatch,” in Proc. IEEE/ACM Int. Symp. Microarchit. (MICRO), Vancouver, BC, Canada, 2012, pp. 247–257.

[40] K. Chen, S. Li, N. Muralimanohar, J. H. Ahn, J. B. Brockman and N. P. Jouppi, "CACTI-3DD: Architecture-level Modeling for 3D Die-stacked DRAM Main Memory," in DATE, 2012

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

[42] Thakkar, Ishan G; Pasricha, Sudeep "3D-Wiz: A novel high bandwidth, optically interfaced 3D DRAM architecture with reduced random access time", Computer Design (ICCD), 2014 32nd IEEE International Conference on, pp 1 – 7.

[43] J. T. Pawlowski, "Hybrid Memory Cube (HMC)," in in proceedings of Hot Chips 23, 2011.

[44] J. Jeddeloh and B. Keeth, “Hybrid memory cube new dram architecture increases density and performance,” in VLSI Technology (VLSIT), 2012 Symposium on, June 2012, pp. 87 –88.

[45] M. J. Khurshid and M. Lipasti, “Data compression for thermal mitigation in the hybrid memory cube,” in Proc. IEEE 31st Int. Conf. Comput. Design, Oct. 2013, pp. 185–192.

[46] L. Zhao et al., ‘‘Exploring DRAM Cache Architectures for CMP Server Platforms, ‘‘ Proc. 25th Int’l Conf. Computer Design, IEEE CS, 2007, pp. 55-62.

[47] J. Sim, G. H. Loh, V. Sridharan, and M. O’Connor, “Resilient diestacked DRAM caches,” in Proc. Int. Symp. Comput. Archit. (ISCA), Tel Aviv-Yafo, Israel, 2013, pp. 416–427.

[48] S. Priyadarshi, N. Choudhary, B. Dwiel, A. Upreti, E. Rotenberg, R. Davis, and P. Franzon, "Hetero 2 3D integration: A scheme for optimizing efficiency/cost of chip multiprocessors," in International Symposium on Quality Electronic Design, March 2013

[49] S. Borkar, "3D integration for energy efficient system design," in 48th ACM/EDAC/IEEE
Design Automation Conference, June 2011, pp. 214-219

[50] R.S. Patti. Three-dimensional integrated circuits and the future of system-on-chip designs.
Proceedings of the IEEE, 94(6):1214{1224, June 2006

[51] Z. Chen, M. Johnson, L. Wei, and K. Roy, "Estimation of standby leakage power in CMOS circuits considering accurate modeling of transistor stacks," in Proc. Int. Symp. Low Power Electron, 1998, pp. 239–244.

[52] S. Mukhopadhyay and K. Roy, "Accurate modeling of transistor stacks to effectively reduce total standby leakage in nano-scale CMOS circuits," VLSI Symp. Tech. Dig., pp. 53–56, 2003.

[53] Tezzaron Semiconductor Corporation, "www.tezzaron.com"

[54] C. S. Tan and G. Y. Chong, "High throughput Cu-Cu bonding by nonthermo-compression method," in Proc. IEEE Electronic Components Technology Conference, Las Vegas, NV, USA, May 2013, pp. 1158–1164.

[55] S.Satoh, H. Fukushi, M.Esashi and S. Tanaka, "Low-temperature aluminum thermo-compression wafer bonding with tin antioxidation layer for hermetic sealing of MEMS," in 29th IEEE MEMS, January 2016, pp. 581-584

[56] A. K. Panigrahi, S. Bonam, T. Ghosh, S. R. Vanjari, and S. G. Singh. "Low temperature, low pressure CMOS compatible Cu-Cu thermo-compression bonding with Ti passivation for 3D IC integration," In 65th Electronic Components and Technology Conference (ECTC), IEEE 2015, pp. 2205-2210

[57] C. H. Cheng, A. S. Ergun, and B. T. Khuri-Yakub, "Electrical throughwafer interconnects with sub-pico farad parasitic capacitance," in *Proc.Microelectromech. Syst. Conf.*, Aug. 2001, pp. 18–21.

[58] G. Katti, M. Stucchi, K. De Meyer, and W. Dehaene, "Electrical Modeling and Characterization of Through Silicon Via for 3-D ICs," *IEEE Trans. Electron Dev.*, vol. 57, no. 1. pp. 256-262, Jan. 2010.

[59] Choudhary, N.K. ; Wadhavkar, S.V. ; Shah, T.A. ; Mayukh, H. ; Gandhi, J. ; Dwiel, B.H. ; Navada, S. ; Najaf-abadi, H.H. ; Rotenberg, E. "FabScalar: Composing synthesizable RTL designs of arbitrary cores within a canonical superscalar template", Computer Architecture (ISCA), 2011 38th Annual International Symposium

[60] N. M. Tshibangu, P. D. Franzon, E. Rotenberg, and W. R. Davis, “Design of controller for L2 cache mapped in Tezzaron stacked DRAM,” in Proc. IEEE Int. 3D System Integration Conference, San Francisco, CA, 2013, pp. 1–4.

[61] Fukushima, T.; Suzuki, T.; Hashiguchi, H.; Nagai, C.; Bea, J.; Hashimoto, H.; Murugesan, M.; Kang-Wook Lee; Tanaka, T.; Asami, K.; Kitamura, Y.; Koyanagi, M. "Transfer and non-transfer stacking technologies based on chip-to-wafer self-asembly for high-throughput and high-precision alignment and microbump bonding", *3D Systems Integration Conference (3DIC), 2015 International,* On page(s): TS7.4.1 - TS7.4.4

[62] T. Fukushima, J. Bea, M. Murugesan, H.-Y. Son, M.-S. Suh, K.-Y. Byun, N.-S. Kim, K.-W. Lee and M. Koyanagi, "3D Memory Chip Stacking by Multi-Layer Self-Assembly Technology"
Tech. Dig. IEEE Int. 3D System Integration Conference (3DIC), 2013 San Francisco, CA

[63] Z. Tang et al., “Low temperature direct bonding technology for wafer-scale integration and packaging”, 3rd annual IEEE NEMS, 2008, pp. 95-98

[64] P. Gueguen *et al.*, “Direct bonding: An innovative 3D interconnect”, ECTC Proceeding, 2010, pp. 878-883

[65] Y. Beillard, L.D. Cioccio, & AI, "Chip to wafer copper direct bonding electrical characterization and thermal cycling", 3DIC 2013

[66] V. Sundaram, Q. Chen, Y. Suzuki, G. Kumar, F. Liu, and R. Tummala, "Low-cost and low-loss 3D silicon interposer for high bandwidth logic-to-memory interconnections without TSV in the logic IC," Electronic Components and Technology Conference (ECTC), 2012 IEEE 62nd, May 2012, pp.292-297

[67] D. Velenis, M. Detalle, E. J. Marinissen, and E. Beyne, “Si Interposer Build-Up Options and
Impact on 3D System Cost”, in Proc. IEEE Int. 3D System Integration Conference, San Francisco, CA, 2013

[68] Y. Lamy, J.P. Colonna, G. Simon, P. Leduc, S. Cheramy and C. Laviron, “Which interconnects for which 3D applications? Status and perspectives”, in Proc. IEEE Int. 3D System Integration Conference, San Francisco, CA, 2013, pp. 1–4.

[69] O.Nukaga, T. Shioiri, S.Yamamoto and T.Suemasu, “Glass Interposer with High-density Three dimensional Structured TGV for 3D System Integration ", in Proc. IEEE Int. 3D System Integration Conference, San Francisco, CA, 2013.

[70] Youngwoo Kim, Jonghyun Cho, Kiyeong Kim, Heegon Kim, Srikrishna Sitaraman, Venky Sundaram, Rao Tummala and Joungho Kim, “Analysis and Optimization of a Power

Distribution Network in 2.5D IC with Glass Interposer", in Proc. IEEE Int. 3D System Integration Conference, Cork, Ireland, December 2014.

[71] C. Santos, P. Vivet, J-P. Colonna, P. Coudrain and R. Reis, "Thermal Performance of 3D ICs: Analysis and Alternatives", in Proc. IEEE Int. 3D System Integration Conference, Cork, Ireland, December 2014.

[72] K. Matsumoto, H. Mori, and Y. Orii, "Cooling from the bottom side (laminate (substrate) side) of a three-dimensional (3D) chip stack", in Proc. IEEE Int. 3D System Integration Conference, Cork, Ireland, December 2014.

[73] DesignWare Library - Datapath and Building Block IP, https://www.synopsys.com/dw/buildingblock.php

[74] Madiwalar, Basavaraj; Kariyappa, B. S. "Single bit-line 7T SRAM cell for low power and high SNM", Automation, Computing, Communication, Control and Compressed Sensing (iMac4s), 2013 International Multi-Conference on, On page(s): 223 – 228

[75] M.S. Sarker, M. Hossain, N. Hossain, M. Rasheduzzaman and M. Islam, "Area optimization in 8T SRAM cell for low power consumption", Electrical & Electronic Engineering (ICEEE), 2015 International Conference on, pp: 117 – 120

[76] P.N. Kran and N. Saxena, "Design and analysis of different types of SRAM cell topologies", International Conference on Electronics and Communication Systems (ICECS), pp. 1060-1065, February 2015

[77] Majumdar, Budhaditya; Basu, Sumana "Low power single bitline 6T SRAM cell with high read stability", Recent Trends in Information Systems (ReTIS), 2011 International Conference on, pp. 169 – 174

[78] K. Chun, W. Zhang, P. Jain, and C. H. Kim, "A 700 MHz 2T1C embedded DRAM macro in a generic logic process with no boosted supplies," in IEEE ISSCC Dig. Tech. Papers, 2011, pp. 506–507.

[79] K. C. Chun et al., "A 2T1C embedded DRAM macro with no boosted supplies featuring a 7T SRAM based repair and a cell storage monitor," IEEE J. Solid-State Circuits, vol. 47, no. 10, 2012, pp. 2517–2526.

[80] K. Parat, "NAND technology presentation", IEEE Hot Chips Symposium, Stanford University, CA, August 2013, pp. 1-18

[81] H. Q. Pon et al., "Reliability issues studied in solid-state drives," in Proc. IEEE 6th Int. Memory Workshop (IMW), May 2014, pp. 1–4

[82] K. Parat and C. Dennison, “A floating gate based 3D NAND technology with CMOS under array,” International Electron Devices Meeting (IEDM), Dec. 2015, pp. 3.3.1–3.3.4.

[83] N. Rizzo, et al., “A Fully Functional 64 Mb DDR3 ST-MRAM Built on 90 nm CMOS Technology,” IEEE Transactions on Magnetics, 49(7), July 2013.

[84] J. Janesky, et al., “Device performance in a fully functional 800MHz DDR3 spin torque magnetic random access memory,” in IMW, 2013.

[85] Micron Short-Reach HMC, https://www.micron.com/products/hybrid-memory-cube

[86] Hybrid Memory Cube Specification 2.1, October 2015, http://www.hybridmemorycube.org

[87] 3D XPoint Technology, https://www.micron.com/about/emerging-technologies/3d-xpoint-technology

[88] A Revolutionary Break-through in Memory Technology, http://www.intel.com/newsroom/kits/nvm/3dxpoint/pdfs/Launch_Keynote.pdf

[89] SPEC CPU2006 Benchmark Suite

[90] I. Loi and L. Benini, “A multi banked - Multi ported - Non blocking shared L2 cache for MPSoC platforms”, in DATE, 2014

APPENDICES

Appendix A

Simulator Tutorial

A.1. GEM5 Installation

Gem5 is an open-source simulator that requires many dependencies (e.g. compilers, libraries, etc…) to work properly. The following are the dependencies and their minimum versions: gcc 4.8, python 2.7.10, scons 0.98.1, swig 2.0.4, zlib, m4. For the complete list refer to *gem5.org*. There are many difficulties that may be encountered during installation especially if one doesn't have sudo privilege to linux server or just want to install on local home area. This guide may help minimize these difficulties. There are many online sources such as gnu.org, python.org, sourceforget.net where most of these open-source packages can be found. For the purpose of this tutorial, assuming that all the packages have been downloaded and extracted including the simulator itself.

1) Gcc installation: new version of gem5 requires gcc 4.6 or newer but preferably 4.8. Our linux servers (e.g. bunyip, Olympia) have gcc 4.4. Before installation of gcc, it requires the installation of 3 libraries: MPFR, GMP, and MPC. These library can be downloaded and installed individually but this creates many incompatibilities that makes successful installation hard to obtain. The easiest way to install these libraries is to use gcc built-in prerequisites script from source directory as:

   ./contrib/download_prerequisites

   Then the main compiler can be installed with:

   ./configure --pefix=/local/home/<user_name>/usr/local
   make
   make install

2) Python installation: version 2.7.10 or newer
   ./configure --enable-shared --prefix=/local/home/<user_name>/usr/local
   make
   make install
3) Scons installation:
   python setup.py install --prefix*=/local/home/<user_name>/usr/local*

4) Building Gem5 binary
scons build/<ISA_type>/gem5.opt
There are many ISA supported by gem5 including X86, ARM, ALPHA, etc...
Example: *scons build/ALPHA/gem5.opt*
gem5.opt is the name of final binary

To run: *build/ALPHA/gem5.opt <many many options>*
Because there are many and long options needed, a script may help automate the process and reduce time see section A.2.

A.2. GEM5 Execution

Assuming that all the above steps were successful and the simulator binary is working properly. Below is an example of simple script that can be used to run a multi-threaded workload of 8 bzip2 spec2006 benchmark. Assuming the name of script is samplescript.csh. From gem5 home directory run:

./samplescript.csh

Samplescript.csh content

#start: assuming SHELL type is BASH (e.g. Olympia server)

```
out_dir='m5out8cpu'
mkdir $out_dir
machine='olympia_'
email='<user_name>@ncsu.edu'
#8bzip2
bench='bzip2'
mkdir $out_dir/$bench;
build/ALPHA/gem5.opt  --outdir=$out_dir/$bench configs/example/se.py --cpu-type="timing" --cpu-
clock=3.2GHz --mem-size='8GB' --mem-channels=1 \
--cmd="cpu2006/bzip2/bzip2_alpha;cpu2006/bzip2/bzip2_alpha;
cpu2006/bzip2/bzip2_alpha;cpu2006/bzip2/bzip2_alpha;cpu2006/bzip2/bzip2_alpha;
cpu2006/bzip2/bzip2_alpha;cpu2006/bzip2/bzip2_alpha;cpu2006/bzip2/bzip2_alpha"  \
--options="cpu2006/bzip2/chicken.jpg;cpu2006/bzip2/chicken.jpg;
cpu2006/bzip2/chicken.jpg;cpu2006/bzip2/chicken.jpg;cpu2006/bzip2/chicken.jpg;
cpu2006/bzip2/chicken.jpg;cpu2006/bzip2/chicken.jpg;cpu2006/bzip2/chicken.jpg;" \
-n 8 --caches --l1i_assoc=4 --l1d_assoc=4 --l2cache --l2_size="256MB"  \
--l2_assoc=8 -I 2000000000 --output=$out_dir/$bench/log_sdout | tee $out_dir/$bench/log;
subject=$machine$out_dir$bench;mail -s $subject $email < message.txt
#end
```

The above script sends the user an email notification with content of message.txt when completed just for convenience. It is convenient also to combine all workloads in one script

to run sequentially or split them into multiple scripts that can run simultaneously depending on the host speed and memory capability.

A.3. Change made to GEM5

This section contains various change made to source file of the simulator. All added lines are highlighted in green color and red lines denote the removed or commented lines of code. Black means no change made. There are 3 areas that require change:

gem5/src/mem/cache: for all things related to cache. Most C++ file contains various class definition used for cache control. The main file that control all the cache implementation is ***cache_impl.hh***. This is where most changes are made. Different configurations require different logic for bank, row, and port number extraction. A.3.1 contains all the logic needed for the purpose.
gem5/src/sim: for simulation and statistic control
gem5/configs/common: contains python wrapper for various options

A.3.1. Bank/row/port extraction logic per configuration

State-of-art configuration based on 2K row buffer

SSSR-HA [1] [2]
32-way associative, 1port, 32 bank, 2^12 rows, 2KB row-buffer, 1 set per row
addr_in => |**xx-bit tag|12-bit row|5-bit bank|6-bit offset|**
bank_num = (addr_in & 0x7C0)>>6;
row_num = (addr_in & 0x7FF800)>>11;

MSSR-DM [3]
Direct map, 1port, 32 bank, 2^12 rows, 2KB row-buffer, 32 sets per row
Addr_in => |**xx-bit tag|12-bit row|5-bit bank|5-bit set |6-bit offset|**
bank_num = (addr_in & 0xF800)>>11;
row_num = (addr_in & 0xFFF0000)>>16;

MSSR-SA [4]
8-way associative, 1port, 32 bank, 2^12 rows, 2KB row-buffer, 4 sets per row
Addr_in => |**xx-bit tag|12-bit row|5-bit bank|2-bit set |6-bit offset|**
bank_num = (addr_in & 0x1F00)>>8;
row_num = (addr_in & 0x1FFE000)>>13;

Our configurations based on 512B row buffer

SSSR-16 our 16-way configuration
16-way associative, 2-port (parallel), 32 bank, 2^13 rows, 512B row-buffer, 1 set per row
Addr_in => |**xx-bit tag|13-bit row|5-bit bank|6-bit offset|**

bank_num = (addr_in & 0x7C0)>>6;
row_num = (addr_in & 0xFFF800)>>11;

SSSR-LE-8 our low-energy single-set configuration
8-way associative, 2-port (serial), 32 bank, 2^13 rows, 512B row-buffer, 1 set per row
Addr_in => **|xx-bit tag|13-bit row|5-bit bank|1-bit port |6-bit offset|**
port_num = (addr_in & 0x40)>>6;
bank_num = (addr_in & 0xF80)>>7;
row_num = (addr_in & 0x1FFF000)>>12;

MSSR-LE-2S-4 our low-energy multiple-set 4-way configuration
4-way associative, 2-port (serial), 32 bank, 2^13 rows, 512B row-buffer, 2 set per row
Addr_in => **|xx-bit tag|13-bit row|5-bit bank|1-bit set |1-bit port |6-bit offset|**
port_num = (addr_in & 0x40)>>6;
bank_num = (addr_in & 0x1F00)>>8;
row_num = (addr_in & 0x3FFE000)>>13;

MSSR-LE-4S-2 our low-energy multiple-set 2-way configuration
2-way associative, 2-port (serial), 32 bank, 2^13 rows, 512B row-buffer, 4 set per row
Addr_in => **|xx-bit tag|13-bit row|5-bit bank|2-bit set |1-bit port |6-bit offset|**
port_num = (addr_in & 0x40)>>6;
bank_num = (addr_in & 0x3E00)>>9;
row_num = (addr_in & 0x7FFC000)>>14;

To run each of these configurations, make change for bank, row and port in gem5/src/mem/cache/cache_impl.hh and recompile gem5.

A.3.2. Code modification

gem5/configs/common/Simulate.py

In the above command, -I option determine the maximum number of instruction each cpu can run. The simulator was modified to complete when "all" CPU run at least –I <maxinst>. On the default, each any CPU reach the number, all CPU quit and simulation is complete. This does not give picture. The following is change made to benchCheckpoints function:

```
def benchCheckpoints(options, maxtick, cptdir):
   #exit_event = m5.simulate(maxtick - m5.curTick())
   #exit_cause = exit_event.getCause()
   if options.maxinsts:
         count = 0
         while count < options.num_cpus: # to ensure all thread reaches max inst count
            exit_event = m5.simulate()
            exit_cause = exit_event.getCause()
            print 'Exiting core %d @ tick %i because %s' % (count,m5.curTick(), exit_cause)
            count += 1
            m5.stats.dump()    #dump stat when core completes max insts count simulation.
```

gem5/configs/common/Caches.py

This python wrapper contains cache options. Originally the simulator was design for SRAM cache where hit_latency is fixed. But for DRAM cache purpose, this latency depends on row buffer hit status. Below is the replacement of original options for L2 cache:

```
class L2Cache (BaseCache):
   assoc = 8
   #hit_latency = 10
   response_latency = 10
  #latency(cycles) and Energy(nJ) for SSSR-LE-8 configuration 8 way 256MB
   RBM_Latency_CH = 88
   RBM_Latency_CM = 66
   RBH_Latency_CH = 24
   RBH_Latency_CM = 2
   RBM_Latency_FILL = 68
   #Energy(nJ)
   RBM_Energy_CH =  4.520073  #this is in nJ
   RBM_Energy_CM =  3.228643
   RBH_Energy_CH =  1.29143
   RBH_Energy_CM =  0
```

To run other configurations including our state-of-arts, just change the values of latency and energy per access. (See chapter 5 and 6 on how these values are estimated)

gem5/src/mem/cache/base.cc

```
BaseCache::BaseCache(const Params *p)
```

```
    : MemObject(p),
      …
      responseLatency(p->response_latency),
      RBHLatency_CH(p->RBH_Latency_CH),
      RBHLatency_CM(p->RBH_Latency_CM),
      RBMLatency_CH(p->RBM_Latency_CH),
      RBMLatency_CM(p->RBM_Latency_CM),
      RBMLatency_FILL(p->RBM_Latency_FILL),
      //Energy (nJ)
      RBHEnergy_CH(p->RBH_Energy_CH),
      RBHEnergy_CM(p->RBH_Energy_CM),
      RBMEnergy_CH(p->RBM_Energy_CH),
      RBMEnergy_CM(p->RBM_Energy_CM),
      isDebugPrint_i(p->is_debug_print),
      isRBH(p->is_RBH),

      …
      system(p->system)
{
}
```

gem5/src/mem/cache/base.hh

```
Cycles lookupLatency; //const Cycles lookupLatency;
Cycles forwardLatency; //const Cycles forwardLatency;
Cycles fillLatency; //const Cycles fillLatency;

/*below are added to account for variable latency of DRAM cache*/
    const Cycles RBHLatency_CH; //row-buffer hit with cache hit:
    const Cycles RBMLatency_CH; //RBM cache hit:longest it account for data access
    const Cycles RBHLatency_CM; //RBM with cache miss:shortest because only tag compare cycle is applicable
    const Cycles RBMLatency_CM;
    const Cycles RBMLatency_FILL;
    const bool isDebugPrint_i;
    bool isRBH;//check is RBH
    //Energy(nJ)
    const float RBHEnergy_CH; //row-buffer hit with cache hit:
    const float RBMEnergy_CH; //row-buffer miss with cache hit:
    const float RBHEnergy_CM; //row-buffer hit with cache miss:
    const float RBMEnergy_CM;
```

gem5/src/mem/cache/BaseCache.py

```
class BaseCache(MemObject):
    type = 'BaseCache'
    cxx_header = "mem/cache/base.hh"
    …
    #hit_latency = Param.Cycles("Hit latency")
    response_latency = Param.Cycles ("Latency for the return path on a miss");
    RBH_Latency_CH = Param.Cycles (0,"row-buffer hit latency with cache hit");
```

```
  RBH_Latency_CM = Param.Cycles (0,"row-buffer hit latency with cache miss");
  RBM_Latency_CH = Param.Cycles (0,"row-buffer miss latency with cache hit");
  RBM_Latency_CM = Param.Cycles (0,"row-buffer miss latency with cache miss");
  RBM_Latency_FILL = Param.Cycles (0,"row-buffer miss latency with for Fill, no Tag access needed");
  #energy (nJ)
  RBH_Energy_CH = Param.Float (0,"row-buffer hit Energy with cache hit");
  RBH_Energy_CM = Param.Float (0,"row-buffer hit Energy with cache miss");
  RBM_Energy_CH = Param. Float (0,"row-buffer miss Energy with cache hit");
  RBM_Energy_CM = Param. Float (0,"row-buffer miss Energy with cache miss");
  is_RBH = Param.Bool (True,"is the cache access row buffer hit?");
  is_debug_print = Param.Bool (False, "allow print statement for various parameters")
```

…

gem5/src/mem/cache/cache_impl.hh

Major changes are made here:

```
#include "mem/cache/glob_var.hh"
#include <iostream>
using namespace std;

int input_addr;
uint64_t RBHcount = 0; int* rbhcountPtr; uint64_t rbhcount; bool isRBH = false;
bool miss; uint64_t instr_count = 0; int bank_num; int row_num;
//port0
int pre_row_num0=10000, pre_row_num1=10000, pre_row_num2=10000, pre_row_num3=10000,
pre_row_num4=10000, pre_row_num5=10000,pre_row_num6=10000,pre_row_num7=10000,
pre_row_num8=10000, pre_row_num9=10000, pre_row_num10=10000, pre_row_num11=10000,
pre_row_num12=10000,pre_row_num13=10000,pre_row_num14=10000,pre_row_num15=10000,
pre_row_num16=10000, pre_row_num17=10000, pre_row_num18=10000, pre_row_num19=10000,
pre_row_num20=10000,pre_row_num21=10000,pre_row_num22=10000,pre_row_num23=10000,
pre_row_num24=10000, pre_row_num25=10000, pre_row_num26=10000, pre_row_num27=10000,
pre_row_num28=10000,pre_row_num29=10000,pre_row_num30=10000,pre_row_num31=10000;
//port1
int p1_pre_row_num0=10000,p1_pre_row_num1=10000,p1_pre_row_num2=10000,
p1_pre_row_num3=10000,p1_pre_row_num4=10000, p1_pre_row_num5=10000,
p1_pre_row_num6=10000, p1_pre_row_num7=10000, p1_pre_row_num8=10000,
p1_pre_row_num9=10000, int p1_pre_row_num10=10000, p1_pre_row_num11=10000,
p1_pre_row_num12=10000, p1_pre_row_num13=10000, p1_pre_row_num14=10000,
p1_pre_row_num15=10000, p1_pre_row_num16=10000, p1_pre_row_num17=10000,
p1_pre_row_num18=10000,p1_pre_row_num19=10000, p1_pre_row_num20=10000,
p1_pre_row_num21=10000, p1_pre_row_num22=10000, p1_pre_row_num23=10000,
p1_pre_row_num24=10000, p1_pre_row_num25=10000, p1_pre_row_num26=10000,
p1_pre_row_num27=10000, p1_pre_row_num28=10000, p1_pre_row_num29=10000,
p1_pre_row_num30=10000, p1_pre_row_num31=10000;
int port_num;

bool rbh_ctrl (int);
bool rbh_ctrl (int addr_in)
{
 bool RBH1;

   port_num = (addr_in & 0x40)>>6;
```

```
   bank_num = (addr_in & 0xF80)>>7;
   row_num = (addr_in & 0x1FFF000)>>12;



if (port_num==0) //begin checking for port 0
{
  if (bank_num==0)
      { if(row_num==pre_row_num0) {RBH1=true;}
        else {pre_row_num0 = row_num;RBH1=false;}
      }
    else if (bank_num==1)
      { if(row_num==pre_row_num1) {RBH1=true; }
        else {pre_row_num1 = row_num;RBH1=false;}
       }
     else if (bank_num==2)
       { if(row_num==pre_row_num2) {RBH1=true; }
         else {pre_row_num2 = row_num;RBH1=false;}
       }
       ...
     /////////skept logic for bank_num 3 to 30 just for simplicity/////////

      else if (bank_num==31)
        { if(row_num==pre_row_num31) {RBH1=true; }
          else {pre_row_num31 = row_num;RBH1=false;}
        }
}//end checking for port0
else //begin checking for port 1
{
     if (bank_num==0)
        { if(row_num==p1_pre_row_num0) {RBH1=true;}
          else {p1_pre_row_num0 = row_num;RBH1=false;}
        }
     else if (bank_num==1)
       { if(row_num==p1_pre_row_num1) {RBH1=true; }
         else {p1_pre_row_num1 = row_num;RBH1=false;}
       }
     else if (bank_num==2)
       { if(row_num==p1_pre_row_num2) {RBH1=true; }
         else {p1_pre_row_num2 = row_num;RBH1=false;}
       }

       ...
     /////////skept logic for bank_num 3 to 30 just for simplicity//////////
       else if (bank_num==31)
          { if(row_num==p1_pre_row_num31) {RBH1=true; }
            else {p1_pre_row_num31 = row_num;RBH1=false;}
          }
}//end checking for port1

return RBH1;
```

```
} //end of rbh_ctrl function


//////////////////////////////////////////////////
//
// Access path: requests coming in from the CPU side
//
//////////////////////////////////////////////////
bool
Cache::access(PacketPtr pkt, CacheBlk *&blk, Cycles &lat,
              PacketList &writebacks)
{
…
blk = tags->accessBlock(pkt->getAddr(), pkt->isSecure(), lat, id);
if (!isTopLevel){if(blk) miss = false; else miss = true;}
…
}

bool
Cache::recvTimingReq(PacketPtr pkt)
{
    if (!isTopLevel)
   {
   isDebugPrint = isDebugPrint_i;
   input_addr = pkt->getAddr();
   instr_count += 1;
   isRBH = rbh_ctrl(input_addr);
   bool * ptr1;
   ptr1 = &var_test1;
   *ptr1 = isRBH;

   if(isRBH)
     {
     rbhcount += 1;
     lookupLatency = RBHLatency_CH; forwardLatency = RBHLatency_CM;
     if(miss) rbhcount_CM += 1; else rbhcount_CH += 1;
     }
   else {lookupLatency = RBMLatency_CH;forwardLatency = RBMLatency_CM; if(miss) rbmcount_CM += 1; else
rbmcount_CH += 1;}
   RBH_counter = rbhcount;
   instr_counter = instr_count;
   rbh_rate = double(RBH_counter*100)/instr_counter;

   DRAMCache_Energy = (rbhcount_CM*RBHEnergy_CM + rbhcount_CH*RBHEnergy_CH +
rbmcount_CM*RBMEnergy_CM + rbmcount_CH*RBMEnergy_CH);

   }
…
}
```

```
//////////////////////////////////////////////////
//
// Response handling: responses from the memory side
//
//////////////////////////////////////////////////


void
Cache::recvTimingResp(PacketPtr pkt)
{
  if (!isTopLevel)
   {
    isDebugPrint = isDebugPrint_i;
    input_addr = pkt->getAddr();
    instr_count += 1;
    isRBH = rbh_ctrl(input_addr);
    bool * ptr1;
    ptr1 = &var_test1;
    *ptr1 = isRBH;
    if(isRBH)
     {
      rbhcount += 1;
      forwardLatency = RBHLatency_CM;
      rbhcount_CH += 1;
     }
    else {forwardLatency = RBMLatency_CM; rbmcount_CH += 1;}
    if(isDebugPrint) cout <<"recvTimingResp::addr:"<<hex<<pkt->getAddr()<<dec<<" forwardlatency:"
<<forwardLatency<<" isRBH: "<<isRBH<<" var_test1:"<<var_test1<<"\n";
   RBH_counter = rbhcount;
    instr_counter = instr_count;
    rbh_rate = double(RBH_counter*100)/instr_counter;
   DRAMCache_Energy = (rbhcount_CM*RBHEnergy_CM + rbhcount_CH*RBHEnergy_CH +
rbmcount_CM*RBMEnergy_CM + rbmcount_CH*RBMEnergy_CH);

    }
 …
}
```

```
CacheBlk*
Cache::handleFill(PacketPtr pkt, CacheBlk *blk, PacketList &writebacks)
```

```
{
…
if(!isTopLevel) {if(isRBH) fillLatency = RBHLatency_CH; else fillLatency = RBMLatency_FILL;}
…
}

Cache::doTimingSupplyResponse(PacketPtr req_pkt, const uint8_t *blk_data,
                        bool already_copied, bool pending_inval)
{
   if(!isTopLevel) forwardLatency = RBMLatency_CM;
…
}
```

gem5/src/mem/cache/tags/tag.py

```
class BaseTags : public ClockedObject
{
 protected:
  …
   const Cycles accessLatency;
   const bool isTopLevel_tag;
   const Cycles RBMLatency_CH;
   const Cycles RBHLatency_CH;
   bool isRBH_tag;//check is RBH
```

gem5/src/mem/cache/tags/tag.py

```
class BaseTags(ClockedObject):
   type = 'BaseTags'
   abstract = True
   cxx_header = "mem/cache/tags/base.hh"
   # Get the hit latency from the parent (cache)
   hit_latency = Param.Cycles(Parent.hit_latency, "The hit latency for this cache")
   is_top_level = Param.Bool(Parent.is_top_level, "Is this cache at the top level (e.g. L1)")
   RBM_Latency_CH = Param.Cycles(Parent.RBM_Latency_CH,"row-buffer miss latency with cache hit");
   RBH_Latency_CH = Param.Cycles(Parent.RBH_Latency_CH,"row-buffer hit latency with cache hit");
   is_RBH = Param.Bool(Parent.is_RBH,"is the cache access row buffer hit?");
```

gem5/src/mem/cache/tags/base.cc

```
#include "mem/cache/glob_var.hh"
bool var_test1;
BaseTags::BaseTags(const Params *p)
   : ClockedObject(p), blkSize(p->block_size), size(p->size),
     accessLatency(p->hit_latency), cache(nullptr), warmupBound(0),
     isTopLevel_tag (p->is_top_level),
     RBMLatency_CH (p->RBM_Latency_CH),
     RBHLatency_CH (p->RBH_Latency_CH),
     isRBH_tag (p->is_RBH),
     …
```

```
{
}
```

gem5/src/mem/cache/tags/base.hh

```
class BaseTags : public ClockedObject
{
 protected:
  …
   const bool isTopLevel_tag;
   const Cycles RBMLatency_CH;
   const Cycles RBHLatency_CH;
   bool isRBH_tag;
  …
}
```

gem5/src/mem/cache/tags/base_set_assoc.hh

```
CacheBlk* accessBlock(Addr addr, bool is_secure, Cycles &lat,
                      int context_src)
  {
     …
     if(!isTopLevel_tag) {
        if(var_test1==0) lat = RBMLatency_CH;
        else lat = RBHLatency_CH;
                          }
     else lat = accessLatency;
    …

 }
```

gem5/src/sim/stat_control.cc

```
Stats::Value RBH_count;//number of all accesses with row-buffer hit
Stats::Value instr_count_total;//number of all DRAMcache row access
Stats::Value RBH_rate;//RBH_count/instr_count_total
Stats::Value DRAM_Cache_Energy;

Stats::Value RBH_count_CM;
Stats::Value RBH_count_CH;
Stats::Value RBM_count_CH;
Stats::Value RBM_count_CM;
/*global variable*/
uint64_t RBH_counter;
uint64_t instr_counter;
double rbh_rate;
double DRAMCache_Energy;
uint64_t rbhcount_CM,rbhcount_CH,rbmcount_CM,rbmcount_CH;

Global::Global()
{
  ....
```

```
  RBH_count
    .scalar(RBH_counter)
    .name("RBH_count")
    .desc("Total Number of Row buffer hit count for DRAM Cache")
    ;

  RBH_count_CM
    .scalar(rbhcount_CM)
    .name("RBH_count_CM")
    .desc("Total Number of Row buffer hit count for DRAM Cache with cache miss")
    ;
  RBH_count_CH
    .scalar(rbhcount_CH)
    .name("RBH_count_CH")
    .desc("Total Number of Row buffer hit count for DRAM Cache with cache hit(include Fill as it's always hit)")
    ;
  RBM_count_CH
    .scalar(rbmcount_CH)
    .name("RBM_count_CH")
    .desc("Total Number of Row buffer miss count for DRAM Cache with cache hit")
    ;
  RBM_count_CM
    .scalar(rbmcount_CM)
    .name("RBM_count_CM")
    .desc("Total Number of Row buffer miss count for DRAM Cache with cache miss")
    ;
  instr_count_total
    .scalar(instr_counter)
    .name("instr_counter")
    .desc("number of all DRAMcache row access")
    ;

  RBH_rate//added by Marcus
    .scalar(rbh_rate)
    .name("RBH_rate")
    .desc("Row-buffer hit rate (%)")
    ;

  DRAM_Cache_Energy
    .scalar(DRAMCache_Energy)
    .name("DRAMCache_Energy")
    .desc("Energy consumed in DRAM Cache due to accesses (nJ)")
    ;
…
}
```

gem5/src/mem/cache/global_var.hh

```
//this is a new file added
extern bool var_test1;
extern bool isDebugPrint;
extern uint64_t RBH_counter;
```

```
extern uint64_t instr_counter;
extern double rbh_rate;
extern double DRAMCache_Energy;
extern uint64_t rbhcount_CM,rbhcount_CH,rbmcount_CM,rbmcount_CH;
```

Appendix B

DRAM Cache Controller based on Tezzaron DiRAM4

B.1. Introduction

This appendix contains the RTL code written for dram cache controller mapped in Tezzaron DiRAM4. Normally the full design from figure 8.3 shouldhave been taped-out but because of I/O budget, the code presented here is a stripped down version without the necessary modules for CPU testing such as **x-bar**, **IQ**, **OQ**, pipeline **L2_arbiter** and **directory**. only the dram interface controller from figure 8.4 and FPGA control interface for testbench are presented here. All the files containing all the modules are repeated here except for the synopsys DesignWare IP modules [73] where indicated.

Here is how the directory should be organized:

benchmark_mem:

Directory containing files necessary to pre-load data from memory to L2cache

->*mem_file1.txt*: data for row 0 1 & 3

->*mem_file2.txt*: data for row 2

testbench

Directory containing testbench

->*TestFixture_preload.v*: preload data in row 0, 1, 3. Read data from 0, 1. Test refresh. Preload in row 2. Read again from row 0. Dump all content to output/mem.hex

-*TestFixture_preload_self_test*: self-test in case there is not DiRAM4 available. This will load the built-in 32B block to row0 block0

- *data_in_deserial8_to_256bit_TB.v*: 8-bit deserializer to fetch 32B block

rtl:

directory containing all source files. It contains subdirectories:

=>**Drctl**: *DRAM_controller.v, DRAM_fsm.v, L2bank_ctrl.v, Memory_Parameters.vh, col_read.v, col_write.v, column_access.v, cpx_out.v, dir.v, fillbuf_sim.v, hit_logic.v, l2cache.v, l2cache_top.v, lru.v, my_libs.v, precharge.v, preload_ctrl.v, read_request.v, read_write_enable.v, refresh.v, row_open.v, step_fsm.v, tag_read.v, tag_write.v, write_request.v*

=>**Serdes**: *DW_asymfifo_s2_sf.v, data_in_deserial8_to_256bit.v, DW_asymfifoctl_s2_sf.v data_in_serial256_to_8bit .v, DW_fifoctl_s2_sf.v, DW_ram_r_w_s_dff.v*

The following directory caontain used blocks and their RTL will not be repeated here:

=>**fifo**: *DW_fifo_s1_sf.v, DW_fifoctl_s1_sf.v, DW_ram_r_w_s_dff.v*

=>**xbar**: *W_arb_fcfs.v, cpx_with_buff_FF.v, xbar_dp_slice.v, DW_arbiter.v packet_to_all_banks.v xbar_dp_slice_cpx.v, ccx.v, packet_to_all_cpu.v, pcx_with_buff_FF.v*

=>**L2_arbiter**: *DW_arb_sp.v, DW_arbiter_l2.v, L2_arb_dp_slice.v, L2_arbiter.v*

=>**Iqueue**: *IQ.v* (not used)

=>**Oqueue**: *OQ.v* (not used)

B.2. RTL code

Testbench

TestFixture_preload.v

top module name: TestFixture

instance of: data_in_serial256_to_8bit .v, data_in_deserial8_to_256bit_TB.v, l2cache_top.v

```
`timescale 1ns/1ps
`define PCX_M1 129
`define CPX_M1 145

module TestFixture;

parameter cycle = 10;
reg clock = 0;
reg clock_ref = 0;
reg reset ;
wire l2clk;
wire l2clk_N;
wire write_clk;

reg self_test_en_i = 0;

 wire [31:0]  DI_0;
 wire[31:0]    dout_0;
 wire CQ;
 wire CQ_N;         //RdClk,
 wire QVLD; //data out valid
 wire PS_N;//CS_N,
 wire CMD1;
 wire CMD0;
 wire[6:0] AA;
 wire[5:0] BA;
 wire KD;
 wire KD_N; //data in clk

  wire push_empty_inst;
  wire pop_empty_inst;
  wire push_n_out;//added on 6/15/15 to push on-chip derser.
  wire [7 : 0] data_out_inst,data_in_L2;
  wire push_n_out_L2;
  wire[255:0] data_out_32B_L2;


 reg[255:0] pre_data_FB;
 reg[31:0] pre_addr_FB,pre_addr_FB1;
 reg[7:0] transfer_en;
 wire[7:0] FB_ready;
 integer File,File2,file,file2, line,line1;
 integer ii;
```

```
reg done_preload, done_preload2,new_row = 0;
reg[6:0] counter,counter1=0;
reg mem_L2_64B_done,done_32B;
reg[63:0] total_energy;
wire fill_en;

reg CK_div_t;
reg CK_div_n_t;
wire [3:0] step;

reg read_req;
reg [31:0] read_addr_in;
wire[23:0] dummy_di;
wire Read_clk;
initial
        begin
                 CK_div_t = 1'b0; CK_div_n_t = 1'b0;
                #(cycle*5) reset = 1'b0; read_req = 1'b0;
                #(cycle*9)  reset = 1'b1;
    self_test_en_i = 1'b0;


 File = $fopen("benchmark_mem/mem_file1.txt", "r");
 File2 = $fopen("benchmark_mem/mem_file2.txt", "r");

 end

initial begin
#(cycle*20)

while (!$feof(File))
  begin
    @ (posedge l2clk);

//this counter1 just simulate the duration of mem access in fpga to evalaute done_32B. on the real testing theis
done_32B will be given by mem ctrl
    if (~reset) begin counter1 <=0;  done_32B <=1'b0; end
          else if (~fill_en) begin counter1 <=counter1 + 1'b1; done_32B <= (counter1==40);end
    else if (fill_en) counter1 <= 0;
    if (done_32B) begin line = $fscanf(File,"%h %h",pre_addr_FB[31:0] , pre_data_FB[255:0],); end
  end
if ($feof(File)) begin done_preload <=1'b1; $fclose(File);  end


#(cycle*500)

 #(cycle*50)

                                read_req = 1'b1;
                                read_addr_in = 32'h00800000;
                                #(cycle*1) read_req = 1'b0;
```

```
                                #(cycle*110)read_req = 1'b1;
                                read_addr_in = 32'h01800000;
                                #(cycle*1) read_req = 1'b0;

                                #(cycle*70)read_req = 1'b1;
                                read_addr_in = 32'h02800000;
                                #(cycle*1) read_req = 1'b0;

                                #(cycle*70)read_req = 1'b1;
                                read_addr_in = 32'h03800000;
                                #(cycle*1) read_req = 1'b0;

        #(cycle*70)read_req = 1'b1;
                                read_addr_in = 32'h04800000;
                                #(cycle*1) read_req = 1'b0;

                                #(cycle*70)read_req = 1'b1;
                                read_addr_in = 32'h05800000;
                                #(cycle*1) read_req = 1'b0;

                                #(cycle*70)read_req = 1'b1;
                                read_addr_in = 32'h06800000;
                                #(cycle*1) read_req = 1'b0;

                                #(cycle*70)read_req = 1'b1;
                                read_addr_in = 32'h07800000;
                                #(cycle*1) read_req = 1'b0;

                                #(cycle*70)read_req = 1'b1;
                                read_addr_in = 32'h00800800;
                                #(cycle*1) read_req = 1'b0;

        #(cycle*14700)


while (!$feof(File2))
  begin
     @ (posedge l2clk);
     //this counter1 just simulate the duration of mem access in fpga to evalaute done_32B. on the real testing
theis done_32B will be given by mem ctrl
     if (~reset) begin counter1 <=0;  done_32B <=1'b0; end
           else if (~fill_en) begin counter1 <=counter1 + 1'b1; done_32B <= (counter1==40);end
     else if (fill_en) counter1 <= 0;
     if (done_32B) begin line = $fscanf(File2,"%h %h",pre_addr_FB[31:0] , pre_data_FB[255:0],); end
  end
 if ($feof(File2)) begin done_preload2 <=1'b1; $fclose(File2);  end
 #(cycle*300)

                        #(cycle*70)read_req = 1'b1;
                                read_addr_in = 32'h02800000;
```

```
                                #(cycle*1) read_req = 1'b0;

                                #(cycle*70)read_req = 1'b1;
                                read_addr_in = 32'h03800000;
                                #(cycle*1) read_req = 1'b0;

        #(cycle*70)read_req = 1'b1;
                                read_addr_in = 32'h04800000;
                                #(cycle*1) read_req = 1'b0;

                                #(cycle*200)
         file = $fopen("output/mem.hex") ;
                                        //file2 = $fopen("output/mem_odd.hex") ;
                                         for (ii=0;ii<700;ii=ii+8)
                                         begin
                $fdisplay(file, "%h %h %h %h %h %h %h %h",
MemoryPort0.Bank0[ii+0],MemoryPort0.Bank0[ii+1],MemoryPort0.Bank0[ii+2],MemoryPort0.Bank0[ii+3],Mem
oryPort0.Bank0[ii+4],MemoryPort0.Bank0[ii+5],MemoryPort0.Bank0[ii+6],MemoryPort0.Bank0[ii+7]) ;
                                         end
          $fclose(file) ;
 $finish;
end

always
          begin
                  #(0.5*cycle)        clock = ~clock;
          end

always @ (posedge l2clk) begin CK_div_t <= !CK_div_t ; end
always @ (posedge l2clk_N) begin CK_div_n_t <= !CK_div_n_t ; end

always
          begin
          #(cycle*100)         clock_ref = ~clock_ref; //1kHZ
          end

assign #(1) l2clk = clock;
assign #(1) l2clk_N = ~clock;
assign #(2.5) write_clk = clock;
assign #(1.6) Read_clk = CQ;


reg CMD1_test,CMD0_test,PS_N_test;
reg[11:0] AA_test;
reg[5:0] BA_test;
wire [17:0] A;

assign CASBar = ctrl.l2cache_ctl.mem_io.CASBar;
assign RASBar = ctrl.l2cache_ctl.mem_io.RASBar;
assign PCBar = ctrl.l2cache_ctl.mem_io.PCBar;
assign RefBar = ctrl.l2cache_ctl.mem_io.RefBar;
```

```
assign WEBar = ctrl.l2cache_ctl.mem_io.WEBar;
assign A = ctrl.l2cache_ctl.mem_io.A;

always@(*)
 begin
   if ((~PCBar)||(~CASBar && WEBar)) begin
              CMD1_test =1'b1;CMD0_test=1'b0;PS_N_test=1'b0; AA_test = A[17:6]; BA_test = A[5:0];end
            else if (~RASBar) begin
              CMD1_test=1'b1;CMD0_test=1'b1;PS_N_test=1'b0; AA_test = A[17:6]; BA_test = A[5:0];end
            else if (~CASBar && ~WEBar) begin
              CMD1_test=1'b1;CMD0_test=1'b1;PS_N_test=1'b0; AA_test = A[17:6]; BA_test = A[5:0];end
            else if (~RefBar) begin
              CMD1_test=1'b0;CMD0_test=1'b1;PS_N_test=1'b0; end
            else begin
              CMD1_test=1'b0;CMD0_test=1'b0;PS_N_test=1'b1; end
 end


 assign {DI_0[28:22],DI_0[12:6]} = 14'b0;

l2cache ctrl
(
.gclk_l2_FPGA_to_CTL_i(clock),
.write_clk_FPGA_to_CTL_i(write_clk),
.l2clk_CTL_to_DIRAM_i(l2clk),
.l2clk_N_CTL_to_DIRAM_i(l2clk_N),
.refresh_clk_FPGA_to_CTL_i(clock_ref),
.reset_FPGA_to_CTL_i(reset),
.read_req_FPGA_CTL_i(read_req),
.dout_CTL_to_DIRAM_5to0_i(dout_0[5:0]),
.dout_CTL_to_DIRAM_21to13_i(dout_0[21:13]),
.dout_CTL_to_DIRAM_31to29_i(dout_0[31:29]),
.self_test_FPGA_to_CTL_i(self_test_en_i),
.CQ_CLK_CTL_to_DIRAM_i(Read_clk),
.CQ_CLK_N_UNUSED_i(CQ_N),
.QVLD_CTL_to_DIRAM_i(QVLD),
.DI_CTL_to_DIRAM_5to0_o(DI_0[5:0]),
.DI_CTL_to_DIRAM_21to13_o(DI_0[21:13]),
.DI_CTL_to_DIRAM_31to29_o(DI_0[31:29]),
.mem_packet_CTL_to_FPGA_o(data_in_L2[7:0]),
.PS_N_CTL_to_DIRAM_o(PS_N),
.CMD1_CTL_to_DIRAM_o(CMD1),
.CMD0_CTL_to_DIRAM_o(CMD0),
.AA_CTL_to_DIRAM_o(AA),
.fill_en_CTL_to_FPGA_o(fill_en),
.mem_packet_FPGA_to_CTL_i(data_out_inst[7:0]),
.mem_ctrl_FPGA_to_CTL_i({push_n_out,addr_popping}),
.mem_ctrl_CTL_to_FPGA_o({push_n_out_L2})
);

            MEM_MODEL MemoryPort0 (
```

```
      .DO               (dout_0),
      .CQ (CQ),
      .CQ_n (CQ_N),
     .DI (DI_0),
     .CK     (l2clk),
             .CK_n     (l2clk_N), //new
               .CK_div     (CK_div_t),
               .CK_div_n     (CK_div_n_t), //new
              .AA       ({5'h0,AA_test[6:0]}),
                          .BA       (6'h0),//(BA_test),
                          .KD (l2clk),
                          .KD_N (l2clk_N),
                          .QVLD (QVLD),
               .CMD1(CMD1_test),
               .CMD0(CMD0_test),
               .PS_N(PS_N_test));

 //implement
from_offchip_ser ser_in
(
.inst_clk_push(l2clk),
.inst_clk_pop(l2clk),
.inst_rst_n(reset),
.done_32B(done_32B),
.inst_data_in(pre_data_FB[255:0]),
.inst_addr_in(pre_addr_FB[31:0]),
.read_req(read_req),
.read_addr_in(read_addr_in[31:0]),
.push_n_out(push_n_out),
.d_out_8bit(data_out_inst),
.addr_popping(addr_popping)
);

from_L2_deser deser_in
(
.inst_clk_push(l2clk),
.inst_clk_pop(l2clk),
.inst_rst_n(reset),
.inst_data_in(data_in_L2[7:0]),
.push_n_out(push_n_out_L2),
.data_out_inst(data_out_32B_L2[255:0])
);
endmodule
```

data_in_serial256_to_8bit .v
top module name: from_offchip_ser
instance of: DW_asymfifo_s2_sf.v => synopsys DesignWar: see [73] for RTL

//this is enherited from DW_asymfifo_s2_sf_inst.v

```
/* 1)when all 32B or 64B are retrieved from memory in FPGA, a local "done_32B_fpga" will play the role
of a local "push_n_fpga" to put the entire 32B in the fifo. 2)then on the next local "inst_clk_pop_fpga",
the output signal pop_empty will be asserted and will be used to local "pop_n_fpga" 8-bit in series
3)meanwhile the same pop_n_fpga from FPGA will also be sent to the on-chip deser as "push_n_out"
to start receiving 8-bit until all 32B are complete
4)when all 32B are transfered, the fpga "pop_empty_fpga" is asserted and this wiil be used to: stop pop from fpga, and sent on-chip to: stop push_n_out  and start pop_onchip_deser to collect 32B on registers */
module from_offchip_ser( inst_clk_push,inst_clk_pop,
inst_rst_n,read_req,read_addr_in,done_32B,inst_data_in, inst_addr_in, push_n_out,
d_out_8bit,addr_popping);


  parameter data_in_width = 256;
  parameter data_out_width = 8;
  parameter depth = 4;
  parameter push_ae_lvl = 1;
  parameter push_af_lvl = 1;
  parameter pop_ae_lvl = 1;
  parameter pop_af_lvl = 1;
  parameter err_mode = 0;
  parameter push_sync = 1;
  parameter pop_sync = 1;
  parameter rst_mode = 0;
  parameter byte_order = 0;
  `define bit_width_depth 1 // ceil(log2(depth))

  input inst_clk_push;
  input inst_clk_pop;
  input inst_rst_n;
  input done_32B;
  input [data_in_width-1 : 0] inst_data_in;
  input [31:0] inst_addr_in;
  input read_req;
  input [31:0] read_addr_in;
  output push_n_out;
  output [data_out_width-1 : 0] d_out_8bit;
  output addr_popping;

 reg  data_push_req_n;
 reg addr_push_req_n;
 wire data_pop_req_n, addr_pop_req_n;
 wire done_32B_delay;
 wire pop_empty_data, pop_empty_addr, addr_popping;
```

```
 wire [7:0] data_out_inst, addr_out_inst;
 wire [31:0] addr_in;
 reg read_req_latch;
//1)

 assign done_32B_delay = done_32B;

reg serializing;
always @(posedge inst_clk_push)
  begin
    if (~inst_rst_n) addr_push_req_n <= 1'b1;
    else if (done_32B_delay &&~read_req || read_req) addr_push_req_n <= 1'b0;
          else if (~addr_push_req_n ) addr_push_req_n <= 1'b1;
  end

always @(posedge inst_clk_push)
  begin
    if (~inst_rst_n) serializing <=1'b0;
    else if (~addr_pop_req_n) serializing <=1'b1;
          else  serializing <=1'b0;
  end

always @(posedge inst_clk_push)
  begin
    if (~inst_rst_n) data_push_req_n <= 1'b1;
    else if (serializing && addr_pop_req_n && ~read_req) data_push_req_n <= 1'b0;
          else if (~data_push_req_n ) data_push_req_n <= 1'b1;
  end
//2)

assign data_pop_req_n = pop_empty_data;
assign addr_pop_req_n = pop_empty_addr;
assign addr_popping = ~addr_pop_req_n;

assign push_n_out = data_pop_req_n;

assign d_out_8bit = addr_popping? addr_out_inst:data_out_inst;

assign addr_in = read_req_latch? read_addr_in:inst_addr_in;
always @(posedge inst_clk_push)
begin
  if (~inst_rst_n) begin  read_req_latch <=1'b0;end
  else if (read_req) begin  read_req_latch <=1'b1;end
  else  read_req_latch <=1'b0;
end

  // Instance of DW_asymfifo_s2_sf
  DW_asymfifo_s2_sf #(.data_in_width       (data_in_width),
```

```
  .data_out_width     (data_out_width),
  .depth              (depth),
  .push_ae_lvl        (push_ae_lvl),
  .push_af_lvl        (push_af_lvl),
  .pop_ae_lvl         (pop_ae_lvl),
  .pop_af_lvl         (pop_af_lvl),
  .err_mode           (err_mode),
  .push_sync          (push_sync),
  .pop_sync           (pop_sync),
  .rst_mode           (rst_mode),
  .byte_order         (byte_order))
  data_unit (
          .clk_push(inst_clk_push),
          .clk_pop(inst_clk_pop),
    .rst_n(inst_rst_n),
                .push_req_n(data_push_req_n),
    .flush_n(inst_flush_n),
                .pop_req_n(data_pop_req_n),
    .data_in(inst_data_in),
                .push_empty(push_empty_inst),
    .push_ae(push_ae_inst),
                .push_hf(push_hf_inst),
    .push_af(push_af_inst),
                .push_full(push_full_inst),
    .ram_full(ram_full_inst),
                .part_wd(part_wd_inst),
    .push_error(push_error_inst),
                .pop_empty(pop_empty_data),
    .pop_ae(pop_ae_inst),
                .pop_hf(pop_hf_inst),
    .pop_af(pop_af_inst),
                .pop_full(pop_full_inst),
    .pop_error(pop_error_inst),
                .data_out(data_out_inst) );

DW_asymfifo_s2_sf #(.data_in_width      (32),
  .data_out_width     (8),
  .depth              (depth),
  .push_ae_lvl        (push_ae_lvl),
  .push_af_lvl        (push_af_lvl),
  .pop_ae_lvl         (pop_ae_lvl),
  .pop_af_lvl         (pop_af_lvl),
  .err_mode           (err_mode),
  .push_sync          (push_sync),
  .pop_sync           (pop_sync),
  .rst_mode           (rst_mode),
  .byte_order         (byte_order))
  addr_unit (
```

```
            .clk_push(inst_clk_push),
            .clk_pop(inst_clk_pop),
        .rst_n(inst_rst_n),
                    .push_req_n(addr_push_req_n),
        .flush_n(1'b0),
                    .pop_req_n(addr_pop_req_n),
        .data_in(addr_in),
                    .push_empty(push_empty_inst),
        .push_ae(),
                    .push_hf(),
        .push_af(),
                    .push_full(),
        .ram_full(),
                    .part_wd(),
        .push_error(),
                    .pop_empty(pop_empty_addr),
        .pop_ae(),
                    .pop_hf(),
        .pop_af(),
                    .pop_full(),
        .pop_error(),
                    .data_out(addr_out_inst) );
Endmodule
```

data_in_deserial8_to_256bit_TB.v
top module name: from_offchip_ser

```
//I/O is enherited from DW_asymfifo_s2_sf_inst.v
/* 1)when all 32B or 64B are retrieved from memory in FPGA, a local "done_32B_fpga" will play the role
of a local "push_n_fpga" to put the entire 32B in the fifo. 2)then on the next local "inst_clk_pop_fpga",
the output signal pop_empty will be asserted and will be used to local "pop_n_fpga" 8-bit in series
3)meanwhile the same pop_n_fpga from FPGA will also be sent to the on-chip deser as "push_n_out"
to start receiving 8-bit until all 32B are complete
4)when all 32B are transfered, the fpga "pop_empty_fpga" is asserted and this wiil be used to: stop pop from fpga, and sent on-chip to: stop push_n_out  and start pop_onchip_deser to collect 32B on registers */
module from_L2_deser( inst_clk_push,inst_clk_pop,
inst_rst_n,push_n_out,inst_data_in,data_out_inst);


  parameter data_in_width = 8;
  parameter data_out_width = 256;
  parameter depth = 4;
```

```
  parameter push_ae_lvl = 1;
  parameter push_af_lvl = 1;
  parameter pop_ae_lvl = 1;
  parameter pop_af_lvl = 1;
  parameter err_mode = 0;
  parameter push_sync = 1;
  parameter pop_sync = 1;
  parameter rst_mode = 0;
  parameter byte_order = 0;
  `define bit_width_depth 1 // ceil(log2(depth))

  input inst_clk_push;
  input inst_clk_pop;
  input inst_rst_n;
  input push_n_out;
  input [data_in_width-1 : 0] inst_data_in;
  output [data_out_width-1 : 0] data_out_inst;

 wire inst_push_req_n;
 wire inst_pop_req_n;
 reg[5:0] count_data;
 reg[2:0] count_addr;

//1)
wire addr_popping;
assign addr_popping = 1'b0;
 assign inst_push_req_n = push_n_out;

always @(posedge inst_clk_push)
  begin
    if (~inst_rst_n) count_data<=0;// <= 1'b1;
    else if (~push_n_out && ~addr_popping) count_data <= count_data +  1'b1;
          else if (push_n_out) count_data<=0;
  end


reg[255:0] data_PB,data_PB_out;
reg[31:0] addr_PB,addr_PB_out;


always @(posedge inst_clk_push)
begin
  if (~inst_rst_n) data_PB <= 256'h0;
  else if (~inst_push_req_n)
   begin
     case( count_data )
       0 :  begin data_PB[255:248] <= inst_data_in[7:0]; end
            1 :  begin data_PB[247:240] <= inst_data_in[7:0]; end
```

```
      2 :  begin data_PB[239:232] <= inst_data_in[7:0]; end
           3 :  begin data_PB[231:224] <= inst_data_in[7:0]; end
           4 :  begin data_PB[223:216] <= inst_data_in[7:0]; end
           5 :  begin data_PB[215:208] <= inst_data_in[7:0]; end
      6 :  begin data_PB[207:200] <= inst_data_in[7:0]; end
           7 :  begin data_PB[199:192] <= inst_data_in[7:0]; end
           8 :  begin data_PB[191:184] <= inst_data_in[7:0]; end
           9 :  begin data_PB[183:176] <= inst_data_in[7:0]; end
           10 :  begin data_PB[175:168] <= inst_data_in[7:0]; end
           11 :  begin data_PB[167:160] <= inst_data_in[7:0]; end
           12 :  begin data_PB[159:152] <= inst_data_in[7:0]; end
           13 :  begin data_PB[151:144] <= inst_data_in[7:0]; end
           14 :  begin data_PB[143:136] <= inst_data_in[7:0]; end
           15 :  begin data_PB[135:128] <= inst_data_in[7:0]; end
           16 :  begin data_PB[127:120] <= inst_data_in[7:0]; end
           17 :  begin data_PB[119:112] <= inst_data_in[7:0]; end
           18 :  begin data_PB[111:104] <= inst_data_in[7:0]; end
           19 :  begin data_PB[103:96] <= inst_data_in[7:0]; end
           20 :  begin data_PB[95:88] <= inst_data_in[7:0]; end
           21 :  begin data_PB[87:80] <= inst_data_in[7:0]; end
           22 :  begin data_PB[79:72] <= inst_data_in[7:0]; end
           23 :  begin data_PB[71:64] <= inst_data_in[7:0]; end
           24 :  begin data_PB[63:56] <= inst_data_in[7:0]; end
           25 :  begin data_PB[55:48] <= inst_data_in[7:0]; end
           26 :  begin data_PB[47:40] <= inst_data_in[7:0]; end
           27 :  begin data_PB[39:32] <= inst_data_in[7:0]; end
           28 :  begin data_PB[31:24] <= inst_data_in[7:0]; end
           29 :  begin data_PB[23:16] <= inst_data_in[7:0]; end
           30 :  begin data_PB[15:8] <= inst_data_in[7:0]; end
           31 :  begin data_PB[7:0] <= inst_data_in[7:0]; end

    endcase
   end
   else if (count_data == 32) begin data_PB_out <= data_PB; /*addr_out_inst <=
addr_PB_out;done_32B <= 1'b1;*/end
end

assign data_out_inst = data_PB_out;
endmodule
```

mem_model.v

top module name: MEM_MODEL

```
`timescale 1ns / 1ps

module MEM_MODEL (
                        DO, //Q[31:0]
                        CQ,
```

```
                        CQ_n,
                        DI, //D[31:0]
                        CK, // CK, CK#
                        CK_n,
                        CK_div,
                        CK_div_n,
                        AA, //{A[17:0],B[5:0]}
                        BA,
                        KD,
                        KD_N, //data in clk
                        QVLD, //data out valid
                        CMD0,
                        CMD1,
                        PS_N
                        );

              `include "parameter.vh"
output    [data_bits - 1 : 0] DO; //
output                        CQ;
      output                        CQ_n;
      output  QVLD;
input     [data_bits - 1 : 0] DI;
input                         CK;
      input                         CK_n;
      input                         KD;
      input                         KD_N;
      input                         CK_div;
      input                         CK_div_n;
input     [row_bits - 1 : 0] AA;
      input     [bank_bits - 1 : 0] BA;
      input CMD1;
      input CMD0;
      input PS_N;

      wire     [addr_bits - 1 : 0] A = {BA,AA};
               reg       [data_bits - 1 : 0] Bank0 [mem_size - 1 : 0];

reg       [col_bits - 1 : 0] Col_addr [9 : 0];
reg                 [1 : 0] Command [9 : 0];
reg      [addr_bits - 1 : 0] B0_row_addr;

reg      [data_bits - 1 : 0] DI_dm, DO_temp;
reg       [col_bits - 1 : 0] Col_temp, Burst_counter;

reg                          ACT_b0;
reg                          PC_b0;
reg                          Dqs_out;
```

```
reg                     DI_EN;
reg                     DO_EN;

reg             [1 : 0] Rw_command;
reg    [addr_bits - 1 : 0] Row, Row_dqs;
reg     [col_bits - 1 : 0] Col, Col_dqs;
reg     [col_bits - 1 : 0] Col_brst, Col_brst_dqs;
wire            [3 : 0] DM = 4'b0;
    wire CK_div_buf;
    assign # quarter_cycle CK_div_buf = CK_div;
    wire    ACT_EN  = ~PS_N & CMD1 & CMD0;
wire    REF_EN  = ~PS_N & ~CMD1 & CMD0;
wire    PC_EN   = ~PS_N & CMD1 & ~CMD0;
wire    RD_EN   = ~PS_N & CMD1 & ~CMD0;
wire    WR_EN   = ~PS_N & CMD1 & CMD0;

            // Burst type
wire    sequential = 1'b1;
  `define DEBUG
            `ifdef BL4
              // Turn on Debug messages
              wire [3:0]    Burst_length = 4;
            `else
              wire [3:0]    Burst_length = 8;
            `endif

            `ifdef DEBUG
              wire    Debug = 1;
            `else
              wire    Debug = 0;
            `endif
    wire    RdClk;
assign  # (tSS +  tDV -0.002)DO  = DO_temp;
    assign   # (tSS +  tDV -0.002)QVLD  = DO_EN|RdClk;
    assign  CQ_n = ~CQ;

    assign RdClk = (Burst_counter >= 1 && Burst_counter <= 8)? CK:1'b0;

    assign # (tSS-0.001) CQ = CK;
// Timing Check
integer  RCC_chk = 0, WCC_chk = 0;
integer  WRR_chk = 0;
time    RF_chk = 0;
time    RAC_chk0 = 0;
time    PC_chk = 0;

initial begin
  ACT_b0 = 1'b0;
```

```
    PC_b0 = 1'b0;
    Dqs_out = 1'b0;
    DO_temp = {data_bits{1'bz}};
    {DI_EN, DO_EN} = 2'b0;
    {RCC_chk, WCC_chk, RF_chk} = 3'b0;
    RAC_chk0 = 1'b0;
    PC_chk = 1'b0;
    WRR_chk = 1'b0;

    $timeformat (-9, 1, " ns", 12);
end



always @ (posedge CK) begin
    // Internal Commamd Pipelined
    Command[0] = Command[1];
    Command[1] = Command[2];
    Command[2] = Command[3];
    Command[3] = Command[4];
    Command[4] = Command[5];
    Command[5] = Command[6];
    Command[6] = Command[7];
    Command[7] = Command[8];
    Command[8] = Command[9];
    Command[9] = `NOP;

    Col_addr[0] = Col_addr[1];
    Col_addr[1] = Col_addr[2];
    Col_addr[2] = Col_addr[3];
    Col_addr[3] = Col_addr[4];
    Col_addr[4] = Col_addr[5];
    Col_addr[5] = Col_addr[6];
    Col_addr[6] = Col_addr[7];
    Col_addr[7] = Col_addr[8];
    Col_addr[8] = Col_addr[9];
    Col_addr[9] = 0;


    // tWRR counter
    WRR_chk = WRR_chk +  1;
    WCC_chk = WCC_chk +  1;
    RCC_chk = RCC_chk +  1;



     // Auto Refresh
    if ((REF_EN === 1'b1) && (CK_div_buf == 1'b0)) begin
```

```
            if (Debug) $display ("at time %t ARF  : Auto Refresh", $time);
            // Auto Refresh to Auto Refresh
            if ($time - RF_chk < tCYC) begin
                $display ("at time %t ERROR: tCYC violation during Auto Refresh", $time);
            end
            // Precharge to Auto Refresh
            if ($time - PC_chk < tPCRF) begin
                $display ("at time %t ERROR: tPCRF violation during Auto Refresh", $time);
            end
            // Precharge to Auto Refresh
            if (PC_b0 === 1'b0 ) begin
                $display ("at time %t ERROR: All banks must be Precharge before Auto Refresh",
$time);
            end
            // Record Current Refresh time
            RF_chk = $time;
        end


        // Active Block (Latch Row Address)
        if ((ACT_EN === 1'b1)&& (CK_div_buf == 1'b0)) begin
            if (PC_b0 === 1'b1) begin
                {ACT_b0, PC_b0} = 2'b10;
                B0_row_addr = A [addr_bits - 1 : 0];
                RAC_chk0 = $time;
                if (Debug) $display ("at time %t ACT  : Bank = 0 Row = %d",$time, A);
                // Precharge to Activate Bank 0
                if ($time - PC_chk < tPCR) begin
                    $display ("at time %t ERROR: tPCR violation during Activate", $time);
                end
            end else if ( PC_b0 === 1'b0) begin
                $display ("at time %t ERROR: Bank 0 is not Precharged.", $time);
            end

            // AutoRefresh to Activate
            if ($time - RF_chk < tRFR) begin
                $display ("at time %t ERROR: tRFR violation during Activate", $time );
            end
        end

        // Precharge Block
        if ((PC_EN === 1'b1)&& (CK_div_buf == 1'b0)) begin
            {PC_b0, ACT_b0} = 2'b10;
            if (Debug) $display ("at time %t PC   : Bank = 0",$time);

            // tWRR violation check for Write
            if (WRR_chk < tWRR) begin
                $display ("at time %t ERROR: tWRR violation during Precharge", $time);
```

```
            end

            // tRPC violation check for Write
            if ($time - RAC_chk0 < tRPC) begin
               $display ("at time %t ERROR: tRPC violation during Precharge", $time);
            end

            // tRFPC violation check for Write
            if ($time - RF_chk  < tRFPC) begin
               $display ("at time %t ERROR: tRFPC violation during Precharge", $time);
            end

            // tWCC check (Read to Write)
            if (RCC_chk < tCRPC) begin
               $display ("at time %t ERROR: tCRPC violation during Precharge", $time);
            end
            // tWCC check (Write to Write)
            if (WCC_chk < tCWPC) begin
               $display ("at time %t ERROR: tCWPC violation during Precharge", $time);
            end

            // Terminate a WRITE immediately
            if (DI_EN === 1'b1) begin
               DI_EN = 1'b0;
            end

            // Record Current PC time
            PC_chk = $time;
         end

         // Read, Write, Column Latch
         if (RD_EN === 1'b1 || WR_EN === 1'b1) begin
            // Check to see if bank is open (ACT)
            if ( PC_b0 === 1'b1) begin
               $display("at time %t ERROR: Cannot Read or Write - Bank 0 is not Activated",
$time);
            end

            // Activate to Read or Write (RAS_N to CAS_N)
            if (A[1:0]===2'b0)begin
                     if ( $time - RAC_chk0 < tRAC_A)
                        $display("at time %t ERROR: tRAC_A (Aligned Access) violation during
Read or Write", $time);
                  end
                  else begin
                    if ( $time - RAC_chk0 < tRAC_UA)
                        $display("at time %t ERROR: tRAC_UA (Unaligned Access) violation during
Read or Write", $time);
```

```
            end
        // Read Command
        if ((RD_EN === 1'b1)&& (CK_div_buf == 1'b1)) begin
           // CAS Latency pipeline
           if (tRL===4) begin
                    Command[4] = `READ;
                  Col_addr[4] = A;
                end

           // tRCC check (Read to Read)
           if (RCC_chk < tRCC) begin
              $display ("at time %t ERROR: tRCC violation during Read", $time);
           end
           // tWCC check (Write to Read)
           if (WCC_chk < tWCC) begin
              $display ("at time %t ERROR: tWCC violation during Read", $time);
           end
           RCC_chk = 0;

        // Write Command
        end else if ((WR_EN === 1'b1)&& (CK_div_buf == 1'b1)) begin
                     Command[0] = `WRITE;
                   Col_addr[0] = A;
                                                  //end
           if (RCC_chk < tRCC) begin
              $display ("at time %t ERROR: tRCC violation during Write", $time);
           end
           // tWCC check (Write to Write)
           if (WCC_chk < tWCC) begin
              $display ("at time %t ERROR: tWCC violation during Write", $time);
           end
           WCC_chk = 0;

        end
     end

    Rw_command = Command[0];
  end

// Latch address for Read or Write
  task Latch_address;
     begin
        Col_dqs  = Col_addr[0];
        Col_brst_dqs = Col_addr[0];
        Row_dqs = B0_row_addr;
     end
  endtask
```

```
    always @ (Rw_command) begin
    //turn on DI_EN/DO_EN
        if (Rw_command === `READ ) begin
            Latch_address;
            @ (posedge CK );
            Row = Row_dqs;
            Col = Col_dqs;
            Col_brst = Col_brst_dqs;
            Burst_counter = 0;
            DI_EN = 1'b0;
            DO_EN = 1'b1;
        end else if (Rw_command === `WRITE) begin
            Latch_address;
            @ (posedge CK);
            Row = Row_dqs;
            Col = Col_dqs;
            Col_brst = Col_brst_dqs;
            Burst_counter = 0;
            DI_EN = 1'b1;
            DO_EN = 1'b0;
        end
    end

    always @ (posedge CK) begin //always @ (posedge CK or negedge CK ) begin//always @
(posedge CK_div or negedge CK_div ) begin//
        #0.002;                                          // Delay to avoid race condition with
Rw_command
        if (DI_EN === 1'b1) begin                        // Writing Data to Memory
            // Array buffer
             DI_dm [31 : 0] = Bank0 [{Row, Col}];//DI_dm [127 : 0] = Bank0 [{Row, Col}];
            // Dqm operation
            if (DM[0]  == 1'b0) DI_dm [ 7 : 0]    = DI [ 7 : 0];
            if (DM[1]  == 1'b0) DI_dm [15 : 8]    = DI [15 : 8];
            if (DM[2]  == 1'b0) DI_dm [23 : 16]   = DI [23 : 16];
            if (DM[3]  == 1'b0) DI_dm [31 : 24]   = DI [31 : 24];

            // Write to memory
            Bank0 [{Row, Col}] = DI_dm [31 : 0];//Bank0 [{Row, Col}] = DI_dm [127 : 0];
            // Output result
            if (DM == 4'hf) begin //if (DM == 16'hffff) begin
                if (Debug) $display("at time %t WRITE: Bank = 0 Row = %d, Col = %d, Data = Hi-
Z due to DM", $time,  Row, Col);
            end else begin
                if (Debug) $display("at time %t WRITE: Bank = 0 Row = %d, Col = %d, Data = %h,
DM = %b", $time,  Row, Col, DI_dm, DM);
                // Record tWRR time
                WRR_chk = 0;
            end
```

```
        // Advance burst counter subroutine
        Burst;
      end else if (DO_EN === 1'b1) begin              // Reading Data from Memory
        DO_temp = Bank0[{Row, Col}];
        if (Debug) $display("at time %t READ : Bank = 0 Row = %d, Col = %d, Data = %h",
$time, Row, Col, DO_temp);
        Burst;

      end else begin
        DO_temp = {data_bits{1'bz}};
                    Burst_counter = 0;
      end
    end

    task Burst;
    //generate col;  turn off DI_EN & DO_EN
      begin
        // Advance Burst Counter
        Burst_counter = Burst_counter +  1;

        // Burst Type
        if (sequential === 1'b1) begin                          // Sequential Burst
          Col_temp = Col +  1;
        end
        // Burst Length
        if (Burst_length === 4) begin                                                  // Burst
Length = 4
          Col [1 : 0] = Col_temp [1 : 0];
        end else if (Burst_length === 8) begin   //end else if (Burst_length === 8) begin
// Burst Length = 8
          Col [2 : 0] = Col_temp [2 : 0];
        end
        // Data Counter
                                    if (Burst_length === 4) begin
          if (Burst_counter >= 4) begin
            DI_EN = 1'b0;
            DO_EN = 1'b0;
          end
        end else if (Burst_length === 8) begin //end else if (Burst_length === 8) begin
          if (Burst_counter >= 8) begin
            DI_EN = 1'b0;
            DO_EN = 1'b0;
          end
        end
      end
    endtask

    specify
```

```
    specparam
            tCH =  0.49,//1.15,                    // Clock High-Level Width
            tCL =  0.49,//1.15,                    // Clock Low-Level Width
            tCK =  1,//2.5,                      // Clock Cycle Time
            tDS =  0.175,//0.3,                    // Data-in Setup Time
            tDH =  0.175,//0.15,                   // Data-in Hold Time
            tAS =  0.175,//0.3,                    // Address Setup Time
            tAH =  0.175,//0.3,                    // Address Hold Time
            tIS =  0.175,//0.3,                   // Input Setup Time
            tIH =  0.175;//0.3;                    // Input Hold Time
    $width    (posedge CK,        tCH     );
    $width    (negedge CK,        tCL     );
    $period   (negedge CK,        tCK     );
    $period   (posedge CK,        tCK     );
    $setuphold(posedge CK,          DI,    tDS, tDH);
    $setuphold(negedge CK,          DI,    tDS, tDH);
              $setuphold(posedge CK,   PS_N,  tIS, tIH);
    $setuphold(posedge CK,   CMD0, tIS, tIH);
    $setuphold(posedge CK,   CMD1, tIS, tIH);
    $setuphold(posedge CK,   A,      tAS, tAH);
  endspecify

endmodule
```

l2cache_top.v:
top module name: l2cache
instance of: l2cache.v

```
module l2cache
(
input gclk_l2_FPGA_to_CTL_i,
input l2clk_CTL_to_DIRAM_i,
input l2clk_N_CTL_to_DIRAM_i,
input write_clk_FPGA_to_CTL_i,
input refresh_clk_FPGA_to_CTL_i,
input CQ_CLK_CTL_to_DIRAM_i, //output clock from DiRAM will be used to latch data (dout)
into internal registers
input reset_FPGA_to_CTL_i,
input read_req_FPGA_CTL_i,
input[5:0] dout_CTL_to_DIRAM_5to0_i,
input[21:13] dout_CTL_to_DIRAM_21to13_i,
input[31:29] dout_CTL_to_DIRAM_31to29_i,
input self_test_FPGA_to_CTL_i,

input CQ_CLK_N_UNUSED_i, //won't be needing this
input QVLD_CTL_to_DIRAM_i,
output[5:0] DI_CTL_to_DIRAM_5to0_o,
```

```
output[21:13] DI_CTL_to_DIRAM_21to13_o,
output[31:29] DI_CTL_to_DIRAM_31to29_o,
output[7:0] mem_packet_CTL_to_FPGA_o,
output PS_N_CTL_to_DIRAM_o,
output CMD1_CTL_to_DIRAM_o,
output CMD0_CTL_to_DIRAM_o,
output[6:0] AA_CTL_to_DIRAM_o,
output fill_en_CTL_to_FPGA_o,
input[7:0] mem_packet_FPGA_to_CTL_i,
input[1:0] mem_ctrl_FPGA_to_CTL_i,
output mem_ctrl_CTL_to_FPGA_o
);

l2cache_wrap l2cache_ctl
(
.gclk_l2_i          (gclk_l2_FPGA_to_CTL_i),
.l2clk_i            (l2clk_CTL_to_DIRAM_i),
.l2clk_N_i          (l2clk_N_CTL_to_DIRAM_i),
.write_clk_i        (write_clk_FPGA_to_CTL_i),
.refresh_clk_i      (refresh_clk_FPGA_to_CTL_i),
.reset_i            (reset_FPGA_to_CTL_i),
.read_req_i         (read_req_FPGA_CTL_i),
.dout_5to0_i        (dout_CTL_to_DIRAM_5to0_i[5:0]),
.dout_21to13_i      (dout_CTL_to_DIRAM_21to13_i[21:13]),
.dout_31to29_i      (dout_CTL_to_DIRAM_31to29_i[31:29]),
.self_test_en_i     (self_test_FPGA_to_CTL_i),
.CQ_i               (CQ_CLK_CTL_to_DIRAM_i),
.CQ_N_i             (CQ_CLK_N_UNUSED_i),
.QVLD_i             (QVLD_CTL_to_DIRAM_i),
.DI_5to0_o          (DI_CTL_to_DIRAM_5to0_o[5:0]),
.DI_21to13_o        (DI_CTL_to_DIRAM_21to13_o[21:13]),
.DI_31to29_o        (DI_CTL_to_DIRAM_31to29_o[31:29]),
.L2_mem_packet_o    (mem_packet_CTL_to_FPGA_o[7:0]),
.PS_N_o             (PS_N_CTL_to_DIRAM_o),
.CMD1_o             (CMD1_CTL_to_DIRAM_o),
.CMD0_o             (CMD0_CTL_to_DIRAM_o),
.AA_o               (AA_CTL_to_DIRAM_o[6:0]),
.fill_en_o          (fill_en_CTL_to_FPGA_o),
.mem_L2_packet_i    (mem_packet_FPGA_to_CTL_i[7:0]),
.mem_L2_ctrl_i      (mem_ctrl_FPGA_to_CTL_i[1:0]),
.L2_mem_ctrl_o      (mem_ctrl_CTL_to_FPGA_o)
);

endmodule
```

l2cache.v:
top module name: l2cache_wrap
instance of: preload_ctrl.v, L2bank_ctrl.v

```
// by Marcus Tshibangu
// this is the main top module for L2 DiRAM cache controller

`timescale 1ns/1ps

`define PCX_M1 129
`define CPX_M1 145
module l2cache_wrap
(
//global signals PADs
input reset_i, //active-low
input gclk_l2_i,
input write_clk_i,
input l2clk_i,
input l2clk_N_i,
input refresh_clk_i,
input read_req_i,
input self_test_en_i,
//DRAM Bank0 I/O PADs
 input[31:29] dout_31to29_i,
 input[21:13] dout_21to13_i,
 input[5:0] dout_5to0_i,
 input CQ_i,
 input CQ_N_i,
 input QVLD_i,
 output[31:29] DI_31to29_o,
 output[21:13] DI_21to13_o,
 output[5:0] DI_5to0_o,
 output PS_N_o,
 output CMD1_o,
 output CMD0_o,
 output[6:0] AA_o,
 output fill_en_o,

 //FPGA: external I/O for the chip
 input[7:0] mem_L2_packet_i, //to transfer addr and data
 input[1:0] mem_L2_ctrl_i,  //to control transfer
 output L2_mem_ctrl_o,
 output[7:0]L2_mem_packet_o

);
//to/from cpu0
wire [7:0] pcx_spc0_grant_px;
wire [7:0] spc0_pcx_req_pq;
wire [`PCX_M1:0] spc0_pcx_data_pa;
wire [7:0] pcx_spc1_grant_px;
wire [7:0] spc1_pcx_req_pq;
```

```
wire [`PCX_M1:0] spc1_pcx_data_pa;
wire pcx_sctag0_data_rdy_px1;
wire[7:0] transfer_en;
wire[31:0] pre_addr_FB;
wire[255:0] pre_data_FB;
wire bank_en;
wire[31:0] read_addr;
wire read_en;
wire[17:0] A_0;
wire[4:0] dummy_AA;
wire[31:0] DI_tmp;

reg CK_div;
reg CK_div_n;

wire l2clk_temp;
assign DI_31to29_o = DI_tmp[31:29];
assign DI_21to13_o = DI_tmp[21:13];
assign DI_5to0_o = DI_tmp[5:0];

always @ (posedge l2clk_i) begin if (~reset_i) CK_div <= 1'b0; else CK_div <= !CK_div ; end
always @ (posedge l2clk_N_i) begin if (~reset_i) CK_div_n <= 1'b0; else CK_div_n <=
!CK_div_n ; end

preload_ctrl preload (
.l2clk(l2clk_i),
.reset(reset_i),
.data_in(mem_L2_packet_i[7:0]),
.addr_popping(mem_L2_ctrl_i[0]),
.push_n_out(mem_L2_ctrl_i[1]),
.read_req (read_req_i),
//output
.read_en(read_en),
.read_addr(read_addr),
.pre_addr_FB(pre_addr_FB[31:5]),
.pre_data_FB(pre_data_FB[255:0]),
.bank_en(bank_en)
);

 L2bank_ctrl BANK0_ctrl
(
.gclk_l2                  (gclk_l2_i),
.reset                   (reset_i),
.l2clk                   (l2clk_i),
.l2clk_N                    (l2clk_N_i),
.write_clk                (write_clk_i),
.CK_div                    (CK_div),
.CK_div_n                      (CK_div_n),
```

```
.refresh_clk              (refresh_clk_i),
.self_test_en   (self_test_en_i),
//DRAM Bank ctrl
.RefBar                   (RefBar_0),
.CASBar                   (CASBar_0),
.RASBar                   (RASBar_0),
.PCBar                    (PCBar_0),
.WEBar                    (WEBar_0),
.DI                      (DI_tmp),
.A                       (A_0),
.CQ                       (CQ_i),
.CQ_N                     (CQ_N_i),
.QVLD                     (QVLD_i),
.dout
({dout_31to29_i[31:29],7'b0,dout_21to13_i[21:13],7'b0,dout_5to0_i[5:0]}),
.read_req (read_req_i),
.read_en(read_en),
.read_addr(read_addr),

.pre_addr_FB(pre_addr_FB[31:5]),
.pre_data_FB(pre_data_FB[255:0]),
.bank_en(bank_en),
.fill_en(fill_en_o),
.L2_mem_packet(L2_mem_packet_o),
.L2_mem_ctrl(L2_mem_ctrl_o)
);

DiRAM_io mem_io (
.l2clk_N (l2clk_N_o),
.RefBar                   (RefBar_0),
.CASBar                   (CASBar_0),
.RASBar                   (RASBar_0),
.PCBar                    (PCBar_0),
.WEBar                    (WEBar_0),
.A                       (A_0),
.PS_N    (PS_N_o),
.CMD1    (CMD1_o),
.CMD0    (CMD0_o),
.AA      ({dummy_AA,AA_o})
          );

endmodule

module DiRAM_io
(
//DRAM Bank0 I/O
 input l2clk_N,
 input  RefBar,
```

```
 input  CASBar,
 input  RASBar,
 input  PCBar,
 input  WEBar,
 input [17:0]  A,
 output reg PS_N,
 output reg CMD1,
 output reg CMD0,
 output reg[11:0] AA

 );



 reg write_tmp, write;
always @(posedge l2clk_N)
  begin  write_tmp <= ~WEBar; write <= write_tmp;end

 always@(*)
 begin
    if ((~PCBar)||(~CASBar && WEBar)) begin
            CMD1=1'b1;CMD0=1'b0;PS_N=1'b0; AA = A[17:6];end
          else if (~RASBar) begin
            CMD1=1'b1;CMD0=1'b1;PS_N=1'b0; AA = A[17:6]; end
          else if (write) begin
            CMD1=1'b1;CMD0=1'b1;PS_N=1'b0; AA = A[17:6];end

          else if (~RefBar) begin
            CMD1=1'b0;CMD0=1'b1;PS_N=1'b0; end
          else begin
            CMD1=1'b0;CMD0=1'b0;PS_N=1'b1; end
 end

 endmodule
```

preload_ctrl.v
top module name: preload_ctrl
instance of: data_in_deserial8_to_256bit.v

```
module preload_ctrl (
input l2clk,
input reset,
input [7:0] data_in,
input addr_popping,
input push_n_out,
input read_req,
```

```
//output
output read_en,
output [31:0] read_addr,
output reg[31:5] pre_addr_FB,
output reg[255:0] /*[511:0]*/ pre_data_FB, //
output reg/*[7:0]*/ bank_en
                    );

wire [31:0] addr_in;
wire [127:0] data;
reg done_32B_ff;
wire done_32B;


wire [255:0] data_out_32B;
wire [31:0] addr_out_inst;

always @(posedge l2clk)
begin
  if (done_32B)
    begin
          bank_en <= 1'b1;
          pre_data_FB <= data_out_32B;
          pre_addr_FB <= addr_out_inst[31:5];
         end

  else bank_en <= 1'b0;
end

from_offchip_deser deser_in
(
.inst_clk_push(l2clk),
.inst_clk_pop(l2clk),
.inst_rst_n(reset),
.inst_data_in(data_in[7:0]),
.push_n_out(push_n_out),
.addr_popping(addr_popping),
.done_32B(done_32B),
.read_req(read_req),
.read_en(read_en),
.read_addr(read_addr),
.addr_out_inst(addr_out_inst),
.data_out_inst(data_out_32B[255:0])
);

endmodule
```

data_in_deserial8_to_256bit.v.v

top module name: from_offchip_deser l

```
//I/O enherited from DW_asymfifo_s2_sf_inst.v
/* 1)when all 32B or 64B are retrieved from memory in FPGA, a local "done_32B_fpga" will play the role
of a local "push_n_fpga" to put the entire 32B in the fifo. 2)then on the next local "inst_clk_pop_fpga",
the output signal pop_empty will be asserted and will be used to local "pop_n_fpga" 8-bit in series
3)meanwhile the same pop_n_fpga from FPGA will also be sent to the on-chip deser as "push_n_out"
to start receiving 8-bit until all 32B are complete
4)when all 32B are transfered, the fpga "pop_empty_fpga" is asserted and this wiil be used to: stop pop from fpga, and sent on-chip to: stop push_n_out  and start pop_onchip_deser to collect 32B on registers */
module from_offchip_deser( inst_clk_push,inst_clk_pop, inst_rst_n,push_n_out,inst_data_in, addr_popping,addr_out_inst,read_addr,read_req,read_en,data_out_inst,done_32B);
  parameter data_in_width = 8;
  parameter data_out_width = 256;
  parameter depth = 4;
  parameter push_ae_lvl = 1;
  parameter push_af_lvl = 1;
  parameter pop_ae_lvl = 1;
  parameter pop_af_lvl = 1;
  parameter err_mode = 0;
  parameter push_sync = 1;
  parameter pop_sync = 1;
  parameter rst_mode = 0;
  parameter byte_order = 0;
  `define bit_width_depth 1

  input inst_clk_push;
  input inst_clk_pop;
  input inst_rst_n;
  input push_n_out;
  input [data_in_width-1 : 0] inst_data_in;
  input addr_popping;
  input read_req;
  output reg read_en;
  output reg done_32B;
  output reg[31:0] addr_out_inst;
  output reg [31:0] read_addr;
  output [data_out_width-1 : 0] data_out_inst;

 wire inst_push_req_n;
 wire inst_pop_req_n;
 reg[5:0] count_data;
 reg[2:0] count_addr;
```

```
 reg read_req_latch;
 reg data_count_disable;

//1)

 assign inst_push_req_n = push_n_out;
 assign inst_pop_req_n = (count_data==32)? 1'b0:1'b1;

//2)

always @(posedge inst_clk_push)
  begin
    if (~inst_rst_n) count_data<=0;// <= 1'b1;
    else if (~push_n_out && ~addr_popping) count_data <= count_data +  1'b1;
          else if (push_n_out) count_data<=0;
  end

always @(posedge inst_clk_push)
  begin
    if (~inst_rst_n) count_addr<=0;// <= 1'b1;
    else if (addr_popping) count_addr <= count_addr +  1'b1;
          else count_addr<=0;
  end

reg[255:0] data_PB,data_PB_out;
reg[31:0] addr_PB,addr_PB_out;

always @(posedge inst_clk_push)
begin
  if (~inst_rst_n) begin addr_PB <= 32'h0; addr_PB_out <=0;end
  else if (addr_popping)
   begin
     case( count_addr )
       0 :  begin addr_PB[31:24] <= inst_data_in[7:0]; end
            1 :  begin addr_PB[23:16] <= inst_data_in[7:0]; end
       2 :  begin addr_PB[15:8] <= inst_data_in[7:0]; end
            3 :  begin addr_PB[7:0] <= inst_data_in[7:0]; end
     endcase
   end
  else if (count_addr == 4 ) addr_PB_out <= addr_PB;
   //if (count_addr == 4 /*&& read_req*/) read_addr <= addr_PB;
end

always @(posedge inst_clk_push)
begin
  if (~inst_rst_n) begin read_addr <= 32'b0; read_en <=1'b0;end
```

```
  else if (count_addr == 4 &&read_req_latch) begin read_addr <= addr_PB; read_en
<=1'b1;end
  else  begin /*read_addr <= addr_PB; */read_en <=1'b0;end
end

always @(posedge inst_clk_push)
begin
  if (~inst_rst_n) begin data_count_disable <=1'b0;end
  else if (read_req_latch) data_count_disable <=1'b1;
  else if (count_data == 32) begin data_count_disable <=1'b0;end
end

always @(posedge inst_clk_push)
begin
  if (~inst_rst_n) begin  read_req_latch <=1'b0;end
  else if (read_req) begin  read_req_latch <=1'b1;end
  else  if (count_addr == 4) begin read_req_latch <=1'b0;end
end

always @(posedge inst_clk_push)
begin
  if (~inst_rst_n) begin  data_PB <= 256'h0; addr_out_inst <=0;data_PB_out <=0;end
  else if (~inst_push_req_n)
   begin
     case( count_data )
       0 :  begin data_PB[255:248] <= inst_data_in[7:0]; end
            1 :  begin data_PB[247:240] <= inst_data_in[7:0]; end
       2 :  begin data_PB[239:232] <= inst_data_in[7:0]; end
            3 :  begin data_PB[231:224] <= inst_data_in[7:0]; end
            4 :  begin data_PB[223:216] <= inst_data_in[7:0]; end
            5 :  begin data_PB[215:208] <= inst_data_in[7:0]; end
       6 :  begin data_PB[207:200] <= inst_data_in[7:0]; end
            7 :  begin data_PB[199:192] <= inst_data_in[7:0]; end
            8 :  begin data_PB[191:184] <= inst_data_in[7:0]; end
            9 :  begin data_PB[183:176] <= inst_data_in[7:0]; end
            10 :  begin data_PB[175:168] <= inst_data_in[7:0]; end
            11 :  begin data_PB[167:160] <= inst_data_in[7:0]; end
            12 :  begin data_PB[159:152] <= inst_data_in[7:0]; end
            13 :  begin data_PB[151:144] <= inst_data_in[7:0]; end
            14 :  begin data_PB[143:136] <= inst_data_in[7:0]; end
            15 :  begin data_PB[135:128] <= inst_data_in[7:0]; end
            16 :  begin data_PB[127:120] <= inst_data_in[7:0]; end
            17 :  begin data_PB[119:112] <= inst_data_in[7:0]; end
            18 :  begin data_PB[111:104] <= inst_data_in[7:0]; end
            19 :  begin data_PB[103:96] <= inst_data_in[7:0]; end
            20 :  begin data_PB[95:88] <= inst_data_in[7:0]; end
            21 :  begin data_PB[87:80] <= inst_data_in[7:0]; end
            22 :  begin data_PB[79:72] <= inst_data_in[7:0]; end
```

```
            23 :  begin data_PB[71:64] <= inst_data_in[7:0]; end
            24 :  begin data_PB[63:56] <= inst_data_in[7:0]; end
            25 :  begin data_PB[55:48] <= inst_data_in[7:0]; end
            26 :  begin data_PB[47:40] <= inst_data_in[7:0]; end
            27 :  begin data_PB[39:32] <= inst_data_in[7:0]; end
            28 :  begin data_PB[31:24] <= inst_data_in[7:0]; end
            29 :  begin data_PB[23:16] <= inst_data_in[7:0]; end
            30 :  begin data_PB[15:8] <= inst_data_in[7:0]; end
            31 :  begin data_PB[7:0] <= inst_data_in[7:0]; end

     endcase
   end
   else if (count_data == 32) begin data_PB_out <= data_PB; addr_out_inst <=
addr_PB_out;end
end

always @(posedge inst_clk_push)
begin
  if (~inst_rst_n) begin done_32B <= 1'b0; end
  else if ((count_data == 32) &&~data_count_disable) begin  done_32B <= 1'b1;end
  else done_32B <= 1'b0;
  end

assign data_out_inst = data_PB_out;
endmodule
```

l2bank_ctrl.v
top module name: l2bank_ctrl
instance of: DRAM_controller.v

```
timescale 100ps/1ps
//`timescale 1ns/100ps
`define PCX_M1 129
`define CPX_M1 145

module L2bank_ctrl (
 input reset, //active-low
 input l2clk,
 input l2clk_N,
 input gclk_l2,
 input write_clk,
 input CK_div,
 input CK_div_n,
 input refresh_clk,
 input CQ,
 input CQ_N,    //RdClk,
 input QVLD,
 input self_test_en,
```

```
 input read_req,
 input[31:0] read_addr,
 input read_en,
 output  RefBar,
 output  CASBar,
 output  RASBar,
 output  PCBar,
 output  WEBar,
 output [17:0]  A,
 output [31:0]  DI,
 input[31:0]    dout,
 input[255:0] pre_data_FB,
 input[31:5] pre_addr_FB,
 input bank_en,
 output fill_en,
 output L2_mem_ctrl,
 output[7:0]L2_mem_packet
 );

`include "rtl/drctl/Memory_Parameters.vh"

//IQ internal
wire  [`PCX_M1+1:0]  IQ_arb_data_c1;
wire          IQ_arb_request_c1; //IQ send request to L2 arbiter
wire          arb_IQ_stall;
wire          arb_IQ_grant;
wire          IQ_empty;
//OQ internal
wire          OQ_arb_stall;
wire  [`CPX_M1+3:0]  ctrl_OQ_data;
wire  OQ_full;
//arbiter all internal
wire  [`PCX_M1+1:0]  arb_ctrl_data_c2;
wire  [`PCX_M1+1:0]  FB_arb_data;
wire  [`PCX_M1+1:0]  MB_arb_data;
wire          MB_arb_request; //MB send request to L2 arbiter
wire          FB_arb_request;//FB send request to L2 arbiter
wire          ctrl_arb_do_arbitration;
wire          MB_arb_stall;
wire arb_IQ_granted; //grant to IQ from L2 arbiter

//DRAM Bank ctrl; include MB/FB/WBB
DRAM_controller Tezz_ctrl(
//global
.l2clk(l2clk),
.l2clk_N(l2clk_N),
.CK_div                  (CK_div),
.CK_div_n                   (CK_div_n),
```

```
.gclk_l2(gclk_l2),
.write_clk(write_clk),
.refresh_clk(refresh_clk),
.reset(reset),
.self_test_en(self_test_en),
.read_req (read_req),
.read_en(read_en),
.read_addr(read_addr),
.RefBar1(RefBar),
.CASBar1(CASBar),
.RASBar1(RASBar),
.PCBar1(PCBar),
.WEBar1(WEBar),
.DI_to_dram_port(DI),
.A1(A),
.CQ                    (CQ),
.CQ_N                  (CQ_N),
.QVLD                  (QVLD),
.dram_port_dout(dout),

//added for testing
.pre_addr_FB(pre_addr_FB[31:5]),
.pre_data_FB0(pre_data_FB[255:128]),
.pre_data_FB1(pre_data_FB[127:0]),

.bank_en(bank_en),
.fill_en(fill_en),
.L2_mem_packet(L2_mem_packet),
.L2_mem_ctrl(L2_mem_ctrl)

);
endmodule
```

DRAM_controller.v

top module name: DRAM_controller

instance of: precharge.v, row_open.v, column_access.v, dram_fsm.v, read_request.v, tag_read.v, tag_write.v, step.v, hit_logic.v, cpx_out.v, read_write_enable.v, lru.v, write_request.v, fillbuf_sim.v, refresh.v,

```
//`timescale 1ns/1ps
`define PCX_M1 129
`define CPX_M1 145

module DRAM_controller(
//DRAM I/O #1&2
 input l2clk,
 input l2clk_N,
```

```
 input CK_div,
 input CK_div_n,
 input gclk_l2, //global l2 clock
 input write_clk,
 input refresh_clk,
 input reset,
 input self_test_en,
 output  RefBar1,
 output  CASBar1,
 output  RASBar1,
 output  PCBar1,
 output  WEBar1,
 output [17:0]  A1,
 input CQ,
 input CQ_N,    //RdClk,
 input QVLD,
 output [31:0]  DI_to_dram_port,
 input[31:0]    dram_port_dout,
 input read_req,
 input[31:0] read_addr,
 input read_en,

 //added for testing only
 input[127:0] pre_data_FB0,
 input[127:0] pre_data_FB1,
 input[31:5] pre_addr_FB,
 input bank_en,
 output fill_en,
 output L2_mem_ctrl,
 output[7:0]L2_mem_packet

);

`include "rtl/drctl/Memory_Parameters.vh"

//Tag ctrl I/O #4
 wire  [17:0]   index0;
 wire           rd_en0;
 wire           wr_en0;
 wire  [8:0]      lkup_tag0;
 wire [39:0] addr;
 wire data_exist;
wire [3:0] step;
 assign addr[39:0] = {8'b0,read_addr};
 assign lkup_tag0 = addr[31:23];
 assign index0 = addr[22:5];
 wire [17:0] previous_row;
 wire [6:0] current_state,next_state;
```

```
 wire CSBar;
 wire RefBar;
 wire CASBar;
 wire RASBar;
 wire PCBar;
 wire CSBar_pc;
 wire CSBar_ras;
 wire CSBar_cas;
 wire CSBar_ref;
 wire WEBar;
 reg [17:0] A;
 wire[17:0] A_ras;
 wire[17:0] A_cas;
 wire[3:0] replace_way;
 wire ctrl_arb_done;

wire hit;
wire burst_done_tag;

//tag groups register
wire[127:0]    reg_tag_group0;   //col0
wire[127:0]    reg_tag_group1;   //col1
wire[127:0]    reg_tag_out0;   //col0
wire[127:0]    reg_tag_out1;   //col1
//data register
wire[127:0]    reg_data0;   //data from DRAM
wire[127:0]    reg_data1;
wire [127:0] DI_to_dram_port_t, DI_to_dram_port_w;
wire[6:0] data_address;
wire burst_done;
//added for testing only
wire[127:0] data_wr0_FB;
wire[127:0] data_wr1_FB;
wire[31:5] addr_out;
wire[7:0] data_out_inst;
wire push_n_out;
wire burst_done_0,burst_done_ff;
assign L2_mem_ctrl = push_n_out;
assign L2_mem_packet = data_out_inst;

wire wr_to_rd;
wire refresh_needed;
wire [17:0] A1_temp;
wire CSBar1;

assign A1_temp = A;
assign   A1[17:0] = A1_temp[17:0];
```

```
always @(*)
begin
  if (~RASBar) A[17:0] = A_ras;//from 1 to 2
  else if (step==2) A[12:6] = A_cas[12:6];
  else if (step==3 && (rd_en0||wr_en0) && hit) A[12:6] = data_address;
  else if ((step==3)&& fill_en) A[12:6] = 8*replace_way +  8;
  else if (step==7) begin {A[17:13],A[5:0]} = {A_ras[17:13],A_ras[5:0]};A[12:6] =
A_cas[12:6];end
  else A = A;
end

//step0: initially precharge
precharge Pre(
.l2clk_N(l2clk_N),
.CK_div                 (CK_div),
.reset(reset),
.refresh_needed(refresh_needed),
.step(step),
.current_state(current_state),
.next_state(next_state),
.PCBar1(PCBar1),
.PCBar(PCBar)
);

row_open RAS(
.l2clk_N(l2clk_N),
.CK_div                 (CK_div),
.reset(reset),
.current_state(current_state),
.step(step),
.previous_row_ras(previous_row[17:0]),
.rd_en0(rd_en0),
.wr_en0(wr_en0),
.fill_en(fill_en),
.index0(index0),
.addr_in_FB(addr_out[22:5]),
.RASBar(RASBar),
.RASBar1(RASBar1),
.A(A_ras[17:0])
);

column_access cas(
.l2clk_N(l2clk_N),
.CK_div                 (CK_div),
.reset(reset),
.step(step),
.current_state(current_state),
.next_state(next_state),
```

```
.rd_en0(rd_en0),
.wr_en0(wr_en0),
.fill_en(fill_en),
.hit(hit),
.burst_done_tag(burst_done_tag),
.burst_done_ff(burst_done),
.same_tag(same_tag),
.same_row(same_row),
.refresh_needed(refresh_needed),
.previous_row_cas(previous_row[17:0]),
.new_row(index0[17:0]),
.WEBar(WEBar),
.WEBar1(WEBar1),
.CASBar(CASBar),
.CASBar1(CASBar1),
.A(A_cas[17:0])
);


dram_fsm fsm0(
.l2clk(l2clk),
.reset(reset),
.CASBar(CASBar),
.RASBar(RASBar),
.PCBar(PCBar),
.RefBar(RefBar),
.WEBar(WEBar),
.current_state(current_state),
.next_state(next_state)
);

read_request read_data(
.l2clk(l2clk),
.reset(reset),
.CQ                     (CQ),
.QVLD                     (QVLD),
.step(step),
.burst_done(burst_done),
.current_state(current_state),
.dram_port_dout(dram_port_dout[31:0]),
.reg_data0(reg_data0[127:0]),
.reg_data1(reg_data1[127:0])
                    );

tag_read tag_rd(
.l2clk(l2clk),
.reset(reset),
.CQ                     (CQ),
```

```
.QVLD                    (QVLD),
.step(step),
.burst_done0(burst_done_0),
.burst_done_ff(burst_done_tag),
.current_state(current_state),
.dram_port_dout(dram_port_dout[31:0]),
//Tag output
.tag0(reg_tag_group0[127:0]),
.tag1(reg_tag_group1[127:0])
                    );

tag_write tag_update(
.l2clk(l2clk),
.gclk_l2(gclk_l2),
.write_clk(write_clk),
.reset(reset),
.step(step),
.current_state(current_state),
.burst_done_tag(burst_done_0),
.same_row(same_row),
.fill_en(fill_en),
.replace_way(replace_way[3:0]),
.replace_addr(pre_addr_FB[31:23]),
.reg_tag_group0(reg_tag_group0[127:0]),
.reg_tag_group1(reg_tag_group1[127:0]),
//modified tag output
.reg_tag_out0(reg_tag_out0[127:0]),
.reg_tag_out1(reg_tag_out1[127:0]),
.DI_dram_port(DI_to_dram_port_t[31:0])
                    );

status_fsm status_reg (
.l2clk (l2clk),
.reset(reset),
.CK_div    (CK_div),
.CK_div_n   (CK_div_n),
.CASBar(CASBar),
.RASBar(RASBar),
.PCBar(PCBar),
.RefBar(RefBar),
.WEBar(WEBar),
.current_state(current_state),
.hit(hit),
.burst_done_tag(burst_done_tag),
.burst_done(burst_done),
.burst_done_ff(burst_done_ff),
.done_bit(ctrl_arb_done),
.rd_en0(rd_en0),
```

```
.wr_en0(wr_en0),
.fill_en(fill_en),
.refresh_needed(refresh_needed), //added 3/27/14
.same_tag(same_tag),
.same_row(same_row),
.data_exist(data_exist),
.step(step));

//new addition
hit_logic tag_lookup(
 //input
.l2clk(l2clk),
.reset(reset),
.step(step),
.rd_en(rd_en0),
.wr_en(wr_en0),
.fill_en(fill_en),
.wr_to_rd(wr_to_rd),
.burst_done0(burst_done_0),
.burst_done_tag(burst_done_tag),
.lkup_tag0(lkup_tag0[8:0]),
.previous_row(previous_row[17:0]),
.index0(index0),
.addr_in_FB(addr_out[22:5]),
.reg_tag_in0(reg_tag_out0[127:0]),
.reg_tag_in1(reg_tag_out1[127:0]),
//output
.same_tag(same_tag),
.same_row(same_row),
.data_exist(data_exist),
.hit(hit),
.data_address(data_address[6:0])
                    );

cpx_out data_return(
 //input
.inst_clk_push(l2clk),
.inst_clk_pop(l2clk),
.inst_rst_n(reset),
.step(step),
.burst_done_ff(burst_done),
.data_exist(data_exist),
.reg_data0(reg_data0[127:0]),
.reg_data1(reg_data1[127:0]),
//output
.push_n_out(push_n_out),//added on 6/15/15 to push on-chip derser.
.d_out_8bit(data_out_inst)
```

```
                    );

read_write_enable rd_wr(
 //input
.l2clk(l2clk),
.reset(reset),
.step(step),
.ctrl_arb_done(ctrl_arb_done),
.read_en(read_en),
 //output
.wr_to_rd(wr_to_rd),
.rd_en0(rd_en0),
.wr_en0(wr_en0)
                    );

pLRU plru0(
 //input
.l2clk (l2clk),
.reset(reset),
.step(step),
.same_row(same_row),
.fill_en(fill_en),
.half_tag15(reg_tag_out1[116:112]),
.burst_done_tag(burst_done_0),
//output
.replace_way(replace_way[3:0])
                    );

write_request write_data(
.gclk_l2 (gclk_l2),
.write_clk(write_clk),
.reset(reset),
.step(step),
.current_state(current_state),
.fill_en(fill_en),
.reg_data0(reg_data0[127:0]),
.reg_data1(reg_data1[127:0]),
.data_wr0_FB(data_wr0_FB[127:0]),
.data_wr1_FB(data_wr1_FB[127:0]),
//output
.DI_dram_port(DI_to_dram_port_w[31:0])
                    );
fill_buffer FB ( //not yet designed
.l2clk (l2clk),
.reset(reset),
.step(step),
.addr_in(pre_addr_FB[31:5]), //tag+ row only because we already know bank
.data_in0(pre_data_FB0[127:0]),
```

```
.data_in1(pre_data_FB1[127:0]),
.rd_en0(rd_en0),
.bank_en(bank_en),
.self_test_en(self_test_en),
//output
.fill_en(fill_en), //set this when all 64B arrived and start store
.addr_out(addr_out[31:5]),
.data_wr0(data_wr0_FB[127:0]),
.data_wr1(data_wr1_FB[127:0])
);

refresh_logic refr (
.l2clk (l2clk),
.l2clk_N(l2clk_N),
.CK_div                 (CK_div),
.refresh_clk(refresh_clk),
.reset(reset),
.current_state(current_state),
.RefBar(RefBar),
.RefBar1(RefBar1),
.refresh_needed(refresh_needed)
);

assign DI_to_dram_port = (step==6)? DI_to_dram_port_t[31:0]:DI_to_dram_port_w[31:0];

endmodule
```

precharge.v

top module name: precharge

```
module precharge (
input l2clk_N,
input CK_div,
input reset,
input refresh_needed,
input [3:0] step,
input [6:0] current_state,
input [6:0] next_state,
output  PCBar1,
output  PCBar
);
`include "rtl/drctl/Memory_Parameters.vh"
//step0: initially precharge at startup to allow ras to be ready

  reg pc_ready;
  reg PCBar_new;
```

```
  assign  PCBar = PCBar_new;
  assign  PCBar1 = PCBar_new;

  always @(posedge l2clk_N)
  begin
     if (reset==1'b0)
        begin
         PCBar_new <= 1'b1;
        end
           else if ((step == 0) && ~CK_div)
         begin
          PCBar_new  <= 1'b0;//~pc_ready;
         end
           else if ((step==6) && (current_state==eColumnWriteWait5 )&& ~CK_div)
         begin
          PCBar_new  <= 1'b0;
         end
     else
         begin
          PCBar_new  <= 1'b1;
         end
  end
endmodule
```

row_open.v

top module name: row_open

```
module row_open (
input l2clk_N,
input CK_div,
input reset,
input [3:0] step,
input rd_en0,
input wr_en0,
input fill_en,
input [6:0] current_state,
input [17:0] index0,
input [17:0] addr_in_FB,
output  RASBar1,
output  RASBar,
output reg [17:0] previous_row_ras,
output  [17:0] A
);

`include "rtl/drctl/Memory_Parameters.vh"
reg ras_ready;
reg RASBar_new;
```

```
reg[17:0] A_new;

  assign  RASBar = RASBar_new;
  assign  RASBar1 = RASBar_new;

always @(posedge l2clk_N)
  begin
     if (~reset)
        begin
          RASBar_new <= 1'b1;
          A_new[17:0] <= 18'b0;
        end

             else if ((rd_en0||wr_en0||fill_en)&&(((step==1)&& (current_state ==
ePreChargeWait9))||((step==10)&&(current_state == eRefreshWait18))) && (~CK_div))
         begin
          RASBar_new  <= 1'b0;//~ras_ready;
          if (rd_en0||wr_en0) A_new[17:0] <= index0[17:0];
                      else if (fill_en) A_new[17:0] <= addr_in_FB[17:0];
         end
      else
         begin
           RASBar_new  <= 1'b1; //if (PCBar == 1'b0)
           A_new[17:0] <= A_new[17:0];
         end
  end

always @(posedge l2clk_N)
  begin
     if (~reset) previous_row_ras <= 0;
     else if ((fill_en||rd_en0||wr_en0)&&(step==2))previous_row_ras <= A_new[17:0];
   end
  assign A = A_new;
endmodule
```

column_access.v
top module name: column_access

```
module column_access (
input l2clk_N,
input CK_div,
input reset,
input [3:0] step,
input [6:0] current_state,
input [6:0] next_state,
input rd_en0,
input wr_en0,
input fill_en,
```

```
input hit,
input burst_done_tag,
input burst_done_ff,
input same_tag,
input same_row,
input refresh_needed,
output reg WEBar,
output  WEBar1,
output reg CASBar,
output  CASBar1,
input [17:0] previous_row_cas,
input [17:0] new_row,
output reg[17:0] A

);

`include "rtl/drctl/Memory_Parameters.vh"

assign CASBar1 = CASBar;
assign WEBar1 = WEBar;
wire rd_en, wr_en;
wire cas_ready;
assign  rd_en = rd_en0;
assign  wr_en = wr_en0;
assign cas_ready = ((current_state == eRowAddressWait2)||(current_state ==
eRowAddressWait3)||(current_state == eColumnWriteWait5)||(current_state ==
eBurstReadWait8))? 1'b1: 1'b0;
always @(posedge l2clk_N)
  begin
    if (~reset)
          begin
            CASBar <= 1'b1;
            WEBar <= 1'b1;
          end

     else if (rd_en && ~fill_en && ~wr_en)
         begin
           //to read tag after row open
           if ((step == 2) && (CK_div))
             begin //go to step3 to lookup tag
              CASBar  <= ~cas_ready;
              WEBar <= 1'b1;
              A[12:6] <= 0;
             end

           //read data if tag lookup hit: we use "eColumnWriteWait1" if we come from a WRITE
           else if ((step == 3) && hit && (current_state == eBurstReadWait8 )&& (CK_div))
//to read data
```

```
            begin //go to step4 to read data
             CASBar  <= ~cas_ready;
             WEBar <= 1'b1;
            end

          else if ((step == 3) && hit && (current_state == eColumnWriteWait5)/*&& (CASBar
== 1'b1)*/&& (CK_div)) //
            begin //go to step4 to read data
             CASBar  <= ~cas_ready; //access column if coming from a WRITE
             WEBar <= 1'b1;
                        end
             else if ((step == 7) && (previous_row_cas != new_row) && (cas_ready) /*&&
(CASBar == 1'b1)*/&& (CK_div))
            begin //update tag (step6) before precharge to avoid loosing data
             CASBar  <= ~cas_ready;
             WEBar <= ~cas_ready;
             A[12:6] <= 0;
            end

           else
            begin
              CASBar  <= 1'b1; //if (PCBar == 1'b0)
              WEBar <= 1'b1;
              A[17:0] <= A[17:0];
            end
         end

     else if (~rd_en && ~fill_en && wr_en)
        begin
          //to read tag after row open
          if ((step == 2)/*&& (CASBar == 1'b1)*/&& (CK_div)) //to read tag
            begin
              CASBar  <= ~cas_ready;
              WEBar <= 1'b1;//still read tag
              A[12:6] <= 0;
            end

          //write data if tag lookup hit
          else if ((step == 3) && hit && (current_state == eBurstReadWait8 || current_state
== eColumnWriteWait5)&& (CK_div)) //to write data
            begin //go to step4 to read "before" write data (RMW:read-modify-write)
               CASBar  <= ~cas_ready;
               WEBar <= 1'b1;
            end

          else if ((step == 4) && burst_done_ff && (current_state == eBurstReadWait8 )&&
(CK_div)) //to write data
            begin //go to step5 to write data
```

```
              CASBar  <= ~cas_ready;
              WEBar <= ~cas_ready;
            end


             else if ((step == 7) && (previous_row_cas != new_row) && (CK_div)) //to write
tag
            begin
              CASBar  <= ~cas_ready;
              WEBar <= ~cas_ready;
              A[12:6] <= 0;
            end

          else
            begin
              CASBar  <= 1'b1; //if (PCBar == 1'b0)
              WEBar <= 1'b1;
              A[17:0] <= A[17:0];
            end
        end

           else if (fill_en && ~rd_en && ~wr_en)
        begin
          //to read tag after row open
          if ((step == 2)&& (CK_div)) //to read tag
            begin
              CASBar  <= ~cas_ready;
              WEBar <= 1'b1;//still read tag
              A[12:6] <= 0;
            end
          //write data if tag lookup hit
          else if ((step == 3) && (burst_done_tag||same_row) && (current_state ==
eBurstReadWait8 || current_state == eColumnWriteWait5)&& (CK_div)) //to write data
             begin //go to step5 to write data
              CASBar  <= ~cas_ready;
              WEBar <= ~cas_ready;
            end

             else if ((step == 7) && (~same_row) && (CK_div)) //to read tag
            begin
              CASBar  <= ~cas_ready;
              WEBar <= ~cas_ready;
              A[12:6] <= 0;
            end

          else
            begin
              CASBar  <= 1'b1; //if (PCBar == 1'b0)
```

```
            WEBar <= 1'b1;
            A[17:0] <= A[17:0];
          end
      end
    else if (~fill_en && ~rd_en && ~wr_en)
      begin
        if (refresh_needed && (step==7))
          begin
            CASBar  <= ~cas_ready;
            WEBar <= ~cas_ready;
            A[12:6] <= 0;
          end
                    else
          begin
            CASBar  <= 1'b1; //if (PCBar == 1'b0)
            WEBar <= 1'b1;
            A[17:0] <= A[17:0];
          end
      end
  end//always

endmodule
```

dram_fsm.v

top module name: dram_fsm

```
`timescale 100ps/1ps

module dram_fsm(
 input l2clk,
 input reset,
 input  RefBar,
 input  CASBar,
 input  RASBar,
 input  PCBar,
 input  WEBar,
 output reg [6:0] current_state,
 output reg [6:0] next_state
);

`include "rtl/drctl/Memory_Parameters.vh"
//FSM
always@(posedge l2clk)
        begin
                if(~reset)
                        current_state <= eIdle;
                else
                        current_state <= next_state;
        end
```

```
always @(*)
	begin
		case(current_state)
			eIdle:
				begin
					if( PCBar == 1'b0)
						next_state = ePreCharge;
					else
						next_state = eIdle;

				end
			ePreCharge:
				begin
					if( RefBar == 1'b0)
						begin
						$display("ERROR:PreCharge to Refresh Delay
Violated, tPCRF");
						next_state = ePreChargeWait1;
						end
					else if( RASBar == 1'b0)
						begin
						$display("ERROR:PreCharge to RAS Delay
Violated, tPCR");
						next_state = ePreChargeWait1;
						end
					else
						next_state = ePreChargeWait1;

				end
			eRefresh:
				begin
					if( RASBar == 1'b0)
						begin
						$display("ERROR:Refresh to RAS delay violated,
tRFR");
						next_state = eRefreshWait1;
						end
					else if( PCBar == 1'b0)
						next_state = ePreCharge;
					else
						next_state = eRefreshWait1;
				end
			eRowAddress:
				begin
					if( CASBar == 1'b0)
						begin
```

```
                                        $display("ERROR : RAS to CAS Delay Violated
,tRAC");
                                        next_state = eRowAddressWait1;
                                        end
                                else if( PCBar == 1'b0)
                                        begin
                                        $display("ERROR : RAS to Precharge Delay
Violated ,tRPC");
                                        next_state = eRowAddressWait1;
                                        end
                                else
                                        next_state = eRowAddressWait1;
                        end

                eColumnWrite:
                        begin
                                if( PCBar == 1'b0)
                                        begin
                                        $display("ERROR: Column Write to Precharge
delay violated, tCWPC");
                                        next_state = eBurstWriteWait1;
                                        end
                                else if( CASBar == 1'b0)
                                        begin
                                        $display("ERROR: Succesive Column Write delay
violated, tWCC");
                                        next_state = eBurstWriteWait1;
                                        end
                                else
                                        next_state = eBurstWriteWait1;
                        end

                eColumnRead:
                        begin
                                if( PCBar == 1'b0)
                                        begin
                                        $display("ERROR: Column Read to Precharge
delay violated, tCRPC");
                                        next_state = eBurstReadWait1;
                                        end
                                else if( (CASBar == 1'b0 && WEBar == 1'b1))
                                        begin
                                        $display("ERROR: Succesive Column Read delay
violated, tRCC");
                                        next_state = eBurstReadWait1;
                                        end
                                else if( (CASBar == 1'b0 && WEBar == 1'b0))
                                        begin
```

```
                                        $display("ERROR: Delay between Read & Write
Violated , tRCC");
                                        next_state = eBurstReadWait1;
                                        end
                                else
                                        next_state = eBurstReadWait1;
                        end

                ePreChargeWait1:
                        begin
                                if( RefBar == 1'b0)
                                        begin
                                        $display("ERROR:PreCharge to Refresh Delay
Violated, tPCRF");
                                        next_state = ePreChargeWait2;
                                        end
                                else if( RASBar == 1'b0)
                                        begin
                                        $display("ERROR:PreCharge to RAS Delay
Violated, tPCR");
                                        next_state = ePreChargeWait2;
                                        end
                                else
                                        next_state = ePreChargeWait2;
                        end

                ePreChargeWait2:
                        begin
                                if( RefBar == 1'b0)
                                        begin
                                        $display("ERROR:PreCharge to Refresh Delay
Violated, tPCRF");
                                        next_state = ePreChargeWait3;
                                        end
                                else if( RASBar == 1'b0)
                                        begin
                                        $display("ERROR:PreCharge to RAS Delay
Violated, tPCR");
                                        next_state = ePreChargeWait3;
                                        end
                                else
                                        next_state = ePreChargeWait3;
                        end

                ePreChargeWait3:
                        begin
                                if( RefBar == 1'b0)
                                        begin
```

```
                                        $display("ERROR:PreCharge to Refresh Delay
Violated, tPCRF");
                                        next_state = ePreChargeWait4;
                                        end
                                else if( RASBar == 1'b0)
                                        begin
                                        $display("ERROR:PreCharge to RAS Delay
Violated, tPCR");
                                        next_state = ePreChargeWait4;
                                        end
                                else
                                        next_state = ePreChargeWait4;
                        end

                ePreChargeWait4:
                        begin
                                if( RefBar == 1'b0)
                                        begin
                                        $display("ERROR:PreCharge to Refresh Delay
Violated, tPCRF");
                                        next_state = ePreChargeWait4;
                                        end
                                else if( RASBar == 1'b0)
                                        begin
                                        $display("ERROR:PreCharge to RAS Delay
Violated, tPCR");
                                        next_state = ePreChargeWait5;
                                        end
                                else
                                        next_state = ePreChargeWait5;
                        end

                ePreChargeWait5:
                        begin
                                if( RefBar == 1'b0)
                                        begin
                                        $display("ERROR:PreCharge to Refresh Delay
Violated, tPCRF");
                                        next_state = ePreChargeWait6;
                                        end
                                else if( RASBar == 1'b0)
                                        begin
                                        $display("ERROR:PreCharge to RAS Delay
Violated, tPCR");
                                        next_state = ePreChargeWait6;
                                        end
                                else
                                        next_state = ePreChargeWait6;
```

```
                end

            ePreChargeWait6:
                begin
                    if( RefBar == 1'b0)
                        begin
                        $display("ERROR:PreCharge to Refresh Delay
Violated, tPCRF");
                        next_state = ePreChargeWait7;
                        end
                    else if( RASBar == 1'b0)
                        begin
                        $display("ERROR:PreCharge to RAS Delay
Violated, tPCR");
                        next_state = ePreChargeWait7;
                        end
                    else
                        next_state = ePreChargeWait7;
                end

            ePreChargeWait7:
                begin
                    if( RefBar == 1'b0)
                        begin
                        $display("ERROR:PreCharge to Refresh Delay
Violated, tPCRF");
                        next_state = ePreChargeWait8;
                        end
                    else if( RASBar == 1'b0)
                        begin
                        $display("ERROR:PreCharge to RAS Delay
Violated, tPCR");
                        next_state = ePreChargeWait8;
                        end
                    else
                        next_state = ePreChargeWait8;
                end

            ePreChargeWait8:
                begin
                    if( RefBar == 1'b0)
                        begin
                        $display("ERROR:PreCharge to Refresh Delay
Violated, tPCRF");
                        next_state = ePreChargeWait9;
                        end
                    else if( RASBar == 1'b0)
                        begin
```

```
                                        $display("ERROR:PreCharge to RAS Delay
Violated, tPCR");
                                        next_state = ePreChargeWait9;
                                        end
                                else
                                        next_state = ePreChargeWait9;
                        end

                ePreChargeWait9:  //refresh and RAS ready
                        begin
                                if( RefBar == 1'b0)
                                        next_state = eRefresh;
                                else if( RASBar == 1'b0)
                                        next_state = eRowAddress;

                                else
                                        next_state = ePreChargeWait9;
                        end

                eRefreshWait1:
                        begin
                                if( RASBar == 1'b0)
                                        begin
                                        $display("ERROR:Refresh to RAS delay violated,
tRFR");
                                        next_state = eRefreshWait2;
                                        end
                                else if( RefBar == 1'b0)
                                    begin
                                         $display("ERROR:Refresh to RAS delay
violated, tRFR");
                                         next_state = eRefreshWait2;
                                         end
                                else
                                        next_state = eRefreshWait2;
                        end

                eRefreshWait2:
                        begin
                                if( RASBar == 1'b0)
                                        begin
                                        $display("ERROR:Refresh to RAS delay violated,
tRFR");
                                        next_state = eRefreshWait3;
                                        end
                                else if( RefBar == 1'b0)
                                    begin
```

```
                                        $display("ERROR:Refresh to RAS delay
violated, tRFR");
                                        next_state = eRefreshWait3;
                                        end
                                else
                                        next_state = eRefreshWait3;
                        end

                eRefreshWait3:
                        begin
                                if( RASBar == 1'b0)
                                        begin
                                        $display("ERROR:Refresh to RAS delay violated,
tRFR");
                                        next_state = eRefreshWait4;
                                        end
                                else if( RefBar == 1'b0)
                                    begin
                                        $display("ERROR:Refresh to RAS delay
violated, tRFR");
                                        next_state = eRefreshWait4;
                                        end
                                else
                                        next_state = eRefreshWait4;
                        end
                eRefreshWait4:
                        begin
                                if( RASBar == 1'b0)
                                        begin
                                        $display("ERROR:Refresh to RAS delay violated,
tRFR");
                                        next_state = eRefreshWait5;
                                        end
                                else if( RefBar == 1'b0)
                                    begin
                                        $display("ERROR:Refresh to RAS delay
violated, tRFR");
                                        next_state = eRefreshWait5;
                                        end
                                else
                                        next_state = eRefreshWait5;
                        end

                eRefreshWait5:
                        begin
                                if( RASBar == 1'b0)
                                        next_state = eRefreshWait6;
                                else if( RefBar == 1'b0)
```

```
                                    begin
                                        $display("ERROR:Refresh to RAS delay
violated, tRFR");
                                        next_state = eRefreshWait6;
                                        end
                                else
                                        next_state = eRefreshWait6;
                        end

                eRefreshWait6:
                        begin
                                if( RASBar == 1'b0)
                                        next_state = eRefreshWait7;
                                else if( RefBar == 1'b0)
                                    begin
                                        $display("ERROR:Refresh to RAS delay
violated, tRFR");
                                        next_state = eRefreshWait7;
                                        end
                                else
                                        next_state = eRefreshWait7;
                        end
                eRefreshWait7:
                        begin
                                if( RASBar == 1'b0)
                                        next_state = eRefreshWait8;
                                else if( RefBar == 1'b0)
                                    begin
                                        $display("ERROR:Refresh to RAS delay
violated, tRFR");
                                        next_state = eRefreshWait8;
                                        end
                                else
                                        next_state = eRefreshWait8;
                        end

                eRefreshWait8:
                        begin
                                if( RASBar == 1'b0)
                                        next_state = eRefreshWait9;
                                else if( RefBar == 1'b0)
                                    begin
                                        $display("ERROR:Refresh to RAS delay
violated, tRFR");
                                        next_state = eRefreshWait9;
                                        end
                                else
                                        next_state = eRefreshWait9;
```

```
                end

            eRefreshWait9:
                begin
                    if( RASBar == 1'b0)
                        next_state = eRefreshWait10;
                    else if( RefBar == 1'b0)
                        begin
                            $display("ERROR:Refresh to RAS delay
violated, tRFR");
                            next_state = eRefreshWait10;
                            end
                    else
                        next_state = eRefreshWait10;
                end

            eRefreshWait10:
                begin
                    if( RASBar == 1'b0)
                        next_state = eRefreshWait11;
                    else if( RefBar == 1'b0)
                        begin
                            $display("ERROR:Refresh to RAS delay
violated, tRFR");
                            next_state = eRefreshWait11;
                            end
                    else
                        next_state = eRefreshWait11;
                end

            eRefreshWait11:
                begin
                    if( RASBar == 1'b0)
                        next_state = eRefreshWait12;
                    else if( RefBar == 1'b0)
                        begin
                            $display("ERROR:Refresh to RAS delay
violated, tRFR");
                            next_state = eRefreshWait12;
                            end
                    else
                        next_state = eRefreshWait12;
                end

            eRefreshWait12:
                begin
                    if( RASBar == 1'b0)
                        next_state = eRefreshWait13;
```

```
                    else if( RefBar == 1'b0)
                        begin
                            $display("ERROR:Refresh to RAS delay
violated, tRFR");
                            next_state = eRefreshWait13;
                            end
                    else
                            next_state = eRefreshWait13;
                end

            eRefreshWait13:
                begin
                    if( RASBar == 1'b0)
                            next_state = eRefreshWait14;
                    else if( RefBar == 1'b0)
                        begin
                            $display("ERROR:Refresh to RAS delay
violated, tRFR");
                            next_state = eRefreshWait14;
                            end
                    else
                            next_state = eRefreshWait14;
                end

            eRefreshWait14:
                begin
                    if( RASBar == 1'b0)
                            next_state = eRefreshWait15;
                    else if( RefBar == 1'b0)
                        begin
                            $display("ERROR:Refresh to RAS delay
violated, tRFR");
                            next_state = eRefreshWait15;
                            end
                    else
                            next_state = eRefreshWait15;
                end

            eRefreshWait15:
                begin
                    if( RASBar == 1'b0)
                            next_state = eRefreshWait16;
                    else if( RefBar == 1'b0)
                        begin
                            $display("ERROR:Refresh to RAS delay
violated, tRFR");
                            next_state = eRefreshWait16;
                            end
```

```
                        else
                                next_state = eRefreshWait16;
                end

        eRefreshWait16:
                begin
                        if( RASBar == 1'b0)
                                next_state = eRefreshWait17;
                        else if( RefBar == 1'b0)
                            begin
                                $display("ERROR:Refresh to RAS delay
violated, tRFR");
                                next_state = eRefreshWait17;
                                end
                        else
                                next_state = eRefreshWait17;
                end

        eRefreshWait17:
                begin
                        if( RASBar == 1'b0)
                                next_state = eRefreshWait18;
                        else if( RefBar == 1'b0)
                            begin
                                $display("ERROR:Refresh to RAS delay
violated, tRFR");
                                next_state = eRefreshWait18;
                                end
                        else
                                next_state = eRefreshWait18;
                end

        eRefreshWait18:
                begin
                        if( RASBar == 1'b0)
                                next_state = eRowAddress;//eRefreshWait19;
                        else if( RefBar == 1'b0)
                            begin
                                next_state = eRefresh;//eRefreshWait19;
                                end
                        else
                                next_state = eRefreshWait18;
                end

        eRowAddressWait1:
                begin
                        if( CASBar == 1'b0)
                                begin
```

```
                                        $display("ERROR : RAS to CAS Delay Violated
,tRAC");
                                        end
                                else if( PCBar == 1'b0)
                                        begin
                                        $display("ERROR : RAS to Precharge Delay
Violated ,tRPC");
                                        next_state = eRowAddressWait2;
                                        end
                                else
                                        next_state = eRowAddressWait2;
                        end

                eRowAddressWait2:
                        begin
                                if( (CASBar == 1'b0 && WEBar == 1'b1))
                                        begin
                                        next_state = eColumnRead;
                                        end
                                else if( (CASBar == 1'b0 && WEBar == 1'b0))
                                        begin
                                        next_state = eColumnWrite;
                                        end
                                else if( PCBar == 1'b0)
                                        begin
                                        $display("ERROR : RAS to Precharge Delay
Violated ,tRPC");
                                        next_state = eRowAddressWait3;
                                        end
                                else
                                        next_state = eRowAddressWait3;
                        end

                eRowAddressWait3:
                        begin
                                if( (CASBar == 1'b0 && WEBar == 1'b1))
                                        next_state = eColumnRead;
                                else if( (CASBar == 1'b0 && WEBar == 1'b0))
                                        next_state = eColumnWrite;
                                else if( PCBar == 1'b0)
                                        begin
                                        $display("ERROR : RAS to Precharge Delay
Violated ,tRPC");
                                        next_state = eRowAddressWait4;
                                        end
                                else
                                        next_state = eRowAddressWait4;
                        end
```

```
                eRowAddressWait4:
                        begin
                                if( (CASBar == 1'b0 && WEBar == 1'b1))
                                        next_state = eColumnRead;
                                else if( (CASBar == 1'b0 && WEBar == 1'b0))
                                        next_state = eColumnWrite;
                                else if( PCBar == 1'b0)
                                        begin
                                        $display("ERROR : RAS to Precharge Delay
Violated ,tRPC");
                                        next_state = eRowAddressWait5;
                                        end
                                else
                                        next_state = eRowAddressWait5;
                        end

        eRowAddressWait5:
                        begin
                                if( (CASBar == 1'b0 && WEBar == 1'b1))
                                        next_state = eColumnRead;
                                else if( (CASBar == 1'b0 && WEBar == 1'b0))
                                        next_state = eColumnWrite;
                                else if( PCBar == 1'b0)
                                        begin
                                        $display("ERROR : RAS to Precharge Delay
Violated ,tRPC");
                                        next_state = eRowAddressWait6;
                                        end
                                else
                                        next_state = eRowAddressWait6;
                        end
                eRowAddressWait6:
                        begin
                                if( (CASBar == 1'b0 && WEBar == 1'b1))
                                        next_state = eColumnRead;
                                else if( (CASBar == 1'b0 && WEBar == 1'b0))
                                        next_state = eColumnWrite;
                                else if( PCBar == 1'b0)
                                        begin
                                        $display("ERROR : RAS to Precharge Delay
Violated ,tRPC");
                                        next_state = eRowAddressWait7;
                                        end
                                else
                                        next_state = eRowAddressWait7;
                        end
```

```
                eRowAddressWait7:
                        begin
                                if( (CASBar == 1'b0 && WEBar == 1'b1))
                                        next_state = eColumnRead;
                                else if( (CASBar == 1'b0 && WEBar == 1'b0))
                                        next_state = eColumnWrite;
                                else if( PCBar == 1'b0)
                                        begin
                                        $display("ERROR : RAS to Precharge Delay
Violated ,tRPC");
                                        next_state = eRowAddressWait8;
                                        end
                                else
                                        next_state = eRowAddressWait8;
                        end

                eRowAddressWait8:
                        begin
                                if( (CASBar == 1'b0 && WEBar == 1'b1))
                                        next_state = eColumnRead;
                                else if( (CASBar == 1'b0 && WEBar == 1'b0))
                                        next_state = eColumnWrite;
                                else if( PCBar == 1'b0)
                                        begin
                                        next_state = ePreCharge;//eRowAddressWait5;
                                        end
                                else
                                        next_state = eRowAddressWait8;
                        end
                eBurstReadWait1:
                        begin
                                if( PCBar == 1'b0)
                                        begin

                                        $display("ERROR: Column Read to Precharge
delay violated, tCRPC");
                                        next_state = eBurstReadWait2;
                                        end
                                else if( CASBar == 1'b0)
                                        begin
                                        $display("ERROR: Succesive Column Read delay
violated, tRCC");
                                        next_state = eBurstReadWait2;
                                        end
                                else next_state = eBurstReadWait2;
                        end

                eBurstReadWait2:
```

```
                        begin
                                if( PCBar == 1'b0)
                                        begin

                                        $display("ERROR: Column Read to Precharge
delay violated, tCRPC");
                                        next_state = eBurstReadWait3;
                                        end
                                else if( CASBar == 1'b0)
                                        begin
                                        $display("ERROR: Succesive Column Read delay
violated, tRCC");
                                        next_state = eBurstReadWait3;
                                        end
                                else next_state = eBurstReadWait3;
                        end

                eBurstReadWait3:
                        begin
                                if( PCBar == 1'b0)
                                        begin

                                        $display("ERROR: Column Read to Precharge
delay violated, tCRPC");
                                        next_state = eBurstReadWait4;
                                        end
                                else if( CASBar == 1'b0)
                                        begin
                                        $display("ERROR: Succesive Column Read delay
violated, tRCC");
                                        next_state = eBurstReadWait4;
                                        end
                                else next_state = eBurstReadWait4;
                        end

                eBurstReadWait4:
                        begin
                                if( PCBar == 1'b0)
                                        begin

                                        $display("ERROR: Column Read to Precharge
delay violated, tCRPC");
                                        next_state = eBurstReadWait5;
                                        end
                                else if( CASBar == 1'b0)
                                        begin
                                        $display("ERROR: Succesive Column Read delay
violated, tRCC");
```

```
                                        next_state = eBurstReadWait5;
                                        end
                                else next_state = eBurstReadWait5;
                        end

                eBurstReadWait5:
                        begin
                                if( PCBar == 1'b0)
                                        begin

                                        $display("ERROR: Column Read to Precharge
delay violated, tCRPC");
                                        next_state = eBurstReadWait6;
                                        end
                                else if( CASBar == 1'b0)
                                        begin
                                        $display("ERROR: Succesive Column Read delay
violated, tRCC");
                                        next_state = eBurstReadWait6;
                                        end
                                else next_state = eBurstReadWait6;
                        end

                eBurstReadWait6:
                        begin
                                if( PCBar == 1'b0)
                                        begin

                                        $display("ERROR: Column Read to Precharge
delay violated, tCRPC");
                                        next_state = eBurstReadWait7;
                                        end
                                else if( CASBar == 1'b0)
                                        begin
                                        $display("ERROR: Succesive Column Read delay
violated, tRCC");
                                        next_state = eBurstReadWait7;
                                        end
                                else next_state = eBurstReadWait7;
                        end

                eBurstReadWait7:
                        begin
                                if( PCBar == 1'b0)
                                        begin
                                        $display("ERROR: Column Read to Precharge
delay violated, tCRPC");
                                        next_state = eBurstReadWait8;
```

```
                                        end

                                else if( (CASBar == 1'b0 && WEBar == 1'b1))
                                        begin
                                        next_state = eColumnRead;
                                        end
                                else if( (CASBar == 1'b0 && WEBar == 1'b0))
                                        begin
                                        next_state = eColumnWrite;
                                        end
                                else next_state = eBurstReadWait8;

                        end

                eBurstReadWait8:
                        begin
                                if( PCBar == 1'b0)
                                        begin
                                        next_state = ePreCharge;
                                        end

                                else if( (CASBar == 1'b0 && WEBar == 1'b1))
                                        begin
                                        next_state = eColumnRead;
                                        end
                                else if( (CASBar == 1'b0 && WEBar == 1'b0))
                                        begin
                                        next_state = eColumnWrite;
                                        end
                                else next_state = eBurstReadWait8;

                        end

                eBurstWriteWait1:
                        begin
                                if( PCBar == 1'b0)
                                        begin
                                        $display("ERROR: Column Write to Precharge
delay violated, tCWPC");
                                        next_state = eBurstWriteWait2;
                                        end
                                else if( CASBar == 1'b0)
                                        begin
                                        $display("ERROR: Succesive Column Write delay
violated, tWCC");
                                        next_state = eBurstWriteWait2;
                                        end
```

```
                else
                        next_state = eBurstWriteWait2;
        end
    eBurstWriteWait2:
        begin

                        if( PCBar == 1'b0)
                                begin
                                $display("ERROR: Column Write
to Precharge delay violated, tCWPC");
                                next_state = eBurstWriteWait3;
                                end

                        else if( CASBar == 1'b0)
                                begin
                                $display("ERROR: Succesive
Column Write delay violated, tWCC");
                                next_state = eBurstWriteWait3;
                                end

                        else
                                next_state = eBurstWriteWait3;

        end
    eBurstWriteWait3:
        begin
                if( PCBar == 1'b0)
                        begin
                        $display("ERROR: Column Write to
Precharge delay violated, tCWPC");
                        next_state = eBurstWriteWait4;
                        end
                else if( CASBar == 1'b0)
                        begin
                        $display("ERROR: Succesive Column
Write delay violated, tWCC");
                        //next_state = eColumnWrite;
                        end

                else
                        next_state = eBurstWriteWait4;
        end

    eBurstWriteWait4:
        begin
                if( PCBar == 1'b0)
                        begin
```

```
                                        $display("ERROR: Column Write to
Precharge delay violated, tCWPC");
                                        next_state = eBurstWriteWait5;
                                        end
                                else if( CASBar == 1'b0 )
                                        begin
                                        $display("ERROR: Succesive Column
Write delay violated, tWCC");
                                        next_state = eBurstWriteWait5;
                                        end
                                else next_state = eBurstWriteWait5;
                        end

        eBurstWriteWait5:
                        begin
                                if( PCBar == 1'b0)
                                        begin
                                        $display("ERROR: Column Write to
Precharge delay violated, tCWPC");
                                        next_state = eBurstWriteWait6;
                                        end
                                else if( CASBar == 1'b0 )
                                        begin
                                        $display("ERROR: Succesive Column
Write delay violated, tWCC");
                                        next_state = eBurstWriteWait6;
                                        end
                                else next_state = eBurstWriteWait6;
                        end

                eBurstWriteWait6:
                        begin
                                if( PCBar == 1'b0)
                                        begin
                                        $display("ERROR: Column Write to
Precharge delay violated, tCWPC");
                                        next_state = eColumnWriteWait1;
                                        end
                                else if( CASBar == 1'b0 )
                                        begin
                                        $display("ERROR: Succesive Column
Write delay violated, tWCC");
                                        next_state = eColumnWriteWait1;
                                        end
                                else next_state = eColumnWriteWait1;
                        end

        eColumnWriteWait1:
```

```
                        begin
                                if( PCBar == 1'b0)
                                        begin
                                        $display("ERROR: Column Write to Precharge
delay violated, tCWPC");
                                        next_state = eColumnWriteWait2;
                                        end
                                else if( (CASBar == 1'b0 && WEBar == 1'b0))
                                        next_state = eColumnWrite;
                                else if( (CASBar == 1'b0 && WEBar == 1'b1))
                                        next_state = eColumnRead;
                                else
                                        next_state = eColumnWriteWait2;
                        end

                eColumnWriteWait2:
                        begin
                                if( PCBar == 1'b0)
                                        begin
                                        $display("ERROR: Column Write to Precharge
delay violated, tCWPC");
                                        next_state = eColumnWriteWait3;
                                        end
                                else if( (CASBar == 1'b0 && WEBar == 1'b0))
                                        next_state = eColumnWrite;
                                else if( (CASBar == 1'b0 && WEBar == 1'b1))
                                        next_state = eColumnRead;
                                else
                                        next_state = eColumnWriteWait3;
                        end

                eColumnWriteWait3:
                        begin
                                if( PCBar == 1'b0)
                                        begin
                                        $display("ERROR: Column Write to Precharge
delay violated, tCWPC");
                                        next_state = eColumnWriteWait4;
                                        end
                                else if( (CASBar == 1'b0 && WEBar == 1'b0))
                                        next_state = eColumnWrite;
                                else if( (CASBar == 1'b0 && WEBar == 1'b1))
                                        next_state = eColumnRead;
                                else
                                        next_state = eColumnWriteWait4;
                        end
                    eColumnWriteWait4:
                        begin
```

```
                                        if( PCBar == 1'b0)
                                                begin
                                                next_state = ePreCharge;
                                                end
                                        else if( (CASBar == 1'b0 && WEBar == 1'b0))
                                                next_state = eColumnWrite;
                                        else if( (CASBar == 1'b0 && WEBar == 1'b1))
                                                next_state = eColumnRead;
                                        else
                                                next_state = eColumnWriteWait5;
                                end
                        eColumnWriteWait5:
                                begin
                                        if( PCBar == 1'b0)
                                                begin
                                                next_state = ePreCharge;
                                                end
                                        else if( (CASBar == 1'b0 && WEBar == 1'b0))
                                                next_state = eColumnWrite;
                                        else if( (CASBar == 1'b0 && WEBar == 1'b1))
                                                next_state = eColumnRead;
                                        else
                                                next_state = eColumnWriteWait5;
                                end
                endcase
        end
endmodule
```

read_request.v

top module name: read_request

instance of: col_read.v

```
module read_request (
input l2clk,
input reset,
input CQ,
input QVLD,
input[3:0] step,
input[6:0] current_state,
input[31:0] dram_port_dout,
output burst_done,
output[127:0] reg_data0,
output[127:0] reg_data1
                );

col_read data_read1(
.CQ(CQ),
.QVLD(QVLD),
.reset(reset),
```

```
.step(step),
.step_cst(4'h4),
.burst_done(burst_done),
.current_state(current_state),
.dram_port_dout(dram_port_dout[31:0]),
.data0(reg_data0[127:0]),
.data1(reg_data1[127:0])
);
endmodule
```

col_read.v
top module name: col_read

```
`timescale 1ns/1ps

module col_read (
input CQ,
input QVLD,
input reset,
input[3:0] step,
input[3:0] step_cst,
input[31:0] dram_port_dout,
input[6:0] current_state,
output [127:0] data0,
output [127:0] data1,
output  burst_done
                    );

`include "rtl/drctl/Memory_Parameters.vh"

reg[2:0] burst_counter0; //= 3'b000;
wire[2:0] burst_counter_even;
wire ReadClk_even;
wire ReadClk_even_temp;
reg burst_done_tmp;
wire burst_done_tmp2;

reg [31:0] data_in0,data_in1,data_in2,data_in3,data_in4,data_in5,data_in6,data_in7;
wire [31:0] data_in00,data_in11,data_in22,data_in33,data_in44,data_in55,data_in66,data_in77;

assign #0.275 ReadClk_even = CQ;

//burst counter
always @(posedge ReadClk_even or negedge reset)
        begin
           if (~reset) burst_counter0 <= 3'b0;
                  else if((step == step_cst) && QVLD/*(current_state >=
eBurstReadWait4)*/&&(burst_counter0 != 3'd7))
```

```
                    burst_counter0 <= burst_counter0 +  3'b001;
                else     burst_counter0 <= 3'b0;
        end

assign burst_counter_even = burst_counter0;

always @(posedge ReadClk_even)
begin
  if (step == step_cst)  //all commands sent for CAS1
   begin
     case( burst_counter_even )
       0 :  begin if (QVLD) data_in0[31:0] <= dram_port_dout[31:0]; end
            1 :  begin if (QVLD) data_in1[31:0] <= dram_port_dout[31:0]; end
       2 :  begin if (QVLD) data_in2[31:0] <= dram_port_dout[31:0]; end
            3 :  begin if (QVLD) data_in3[31:0] <= dram_port_dout[31:0]; end
            4 :  begin if (QVLD) data_in4[31:0] <= dram_port_dout[31:0]; end
            5 :  begin if (QVLD) data_in5[31:0] <= dram_port_dout[31:0]; end
       6 :  begin if (QVLD) data_in6[31:0] <= dram_port_dout[31:0]; end
            7 :  begin if (QVLD) data_in7[31:0] <= dram_port_dout[31:0]; end
     endcase
   end
end

always @(posedge ReadClk_even)
 begin
   if (~reset) burst_done_tmp <= 1'b0;
   else if (burst_counter_even == 3'h7) burst_done_tmp <= 1'b1;
   else burst_done_tmp <= 1'b0;
 end

assign burst_done = burst_done_tmp;
assign data1[127:0] = {data_in7,data_in6,data_in5,data_in4};
assign data0[127:0] = {data_in3,data_in2,data_in1,data_in0};

endmodule
```

tag_read.v
top module name: tag_read
instance of: col_read.v

```
module tag_read (
input l2clk,
input reset,
input CQ,
input QVLD,
input[3:0] step,
input[31:0] dram_port_dout,
input[6:0] current_state,
```

```
output[127:0] tag0,
output[127:0] tag1,
output  burst_done0,
output  burst_done_ff
                    );

wire burst_done;
reg burst_done_ff_tmp, burst_done_ff_tmp0,burst_done_ff_tmp1,burst_done_ff_tmp2;

always@(posedge l2clk)
      begin
            if (~reset) begin burst_done_ff_tmp0 <=1'b0; end
       else if (step==3 && burst_done) begin burst_done_ff_tmp0 <=1'b1; end
            else burst_done_ff_tmp0 <= 1'b0;
      end
 always@(posedge l2clk)
      begin
            if (~reset) begin burst_done_ff_tmp <=1'b0;end
       else if (burst_done_ff_tmp0) begin burst_done_ff_tmp <=1'b1;  end
            else burst_done_ff_tmp <= 1'b0;
      end

assign burst_done0 = burst_done;
assign burst_done_ff = burst_done_ff_tmp|burst_done_ff_tmp0;

col_read data_read1(
.CQ(CQ),
.QVLD(QVLD),
.reset(reset),
.step(step),
.step_cst(4'h3),
.burst_done(burst_done),
.current_state(current_state),
.dram_port_dout(dram_port_dout[31:0]),
.data0(tag0[127:0]),
.data1(tag1[127:0])
);

endmodule
```

tag_write.v
top module name: tag_write
instance of: col_write.v

```
module tag_write (
input l2clk,
input gclk_l2,
input write_clk,
```

```
input reset,
input[3:0] step,
input [6:0] current_state,
input[3:0] replace_way,
input burst_done_tag,
input same_row,
input fill_en,
input[31:23] replace_addr,
input[127:0] reg_tag_group0,
input[127:0] reg_tag_group1,
output[127:0] reg_tag_out0,
output[127:0] reg_tag_out1,
output[31:0] DI_dram_port

                    );


wire [127:0] data_wr0_tag;//4 column of the 64B block
wire [127:0] data_wr1_tag;
reg [127:0] reg_tag_local0;
reg [127:0] reg_tag_local1;
wire [127:0] reg_tag_updated0;
wire [127:0] reg_tag_updated1;
wire burst_done_write;

always @(posedge l2clk)
  begin
        if (step==3 && ~same_row && burst_done_tag) //fetch new tag for new row
          begin
                reg_tag_local0 <= reg_tag_group0;
                reg_tag_local1 <= reg_tag_group1;
          end
        else if (step==7 ) //after every write update valid/tag
          begin
                reg_tag_local0 <= reg_tag_updated0;
                reg_tag_local1 <=
{12'b0,replace_way[3:0],reg_tag_updated1[111:0]};//reg_tag_local1 <= reg_tag_updated1;
          end
   end

update_tag_reg tag_0_7(
.reg_tag(reg_tag_local0[127:0]) ,
.step(step),
.l2clk(l2clk),
.replace_addr(replace_addr[31:23]),
.start_way(4'd0),
.fill_en(fill_en),
.current_state(current_state),
```

```
.replace_way(replace_way[3:0]),
.reg_tag_updated(reg_tag_updated0[127:0])
                );

update_tag_reg tag_8_15(
.reg_tag(reg_tag_local1[127:0]) ,
.step(step),
.l2clk(l2clk),
.replace_addr(replace_addr[31:23]),
.start_way(4'd8),
.fill_en(fill_en),
.current_state(current_state),
.replace_way(replace_way[3:0]),
.reg_tag_updated(reg_tag_updated1[127:0])
                );

assign reg_tag_out0 = reg_tag_local0[127:0];
assign reg_tag_out1 = reg_tag_local1[127:0];
assign data_wr0_tag = reg_tag_local0[127:0];
assign data_wr1_tag = reg_tag_local1[127:0];

col_write write_t(
.gclk_l2(gclk_l2),
.write_clk(write_clk),
.reset(reset),
.step(step[3:0]),
.step_cst(4'h6),
.current_state(current_state),
.data_wr0(data_wr0_tag[31:0]),
.data_wr1(data_wr0_tag[63:32]),
.data_wr2(data_wr0_tag[95:64]),
.data_wr3(data_wr0_tag[127:96]),
.data_wr4(data_wr1_tag[31:0]),
.data_wr5(data_wr1_tag[63:32]),
.data_wr6(data_wr1_tag[95:64]),
.data_wr7(data_wr1_tag[127:96]),
//output
.DI_dram_port(DI_dram_port[31:0])
                    );
endmodule

module update_tag_reg (
input [127:0] reg_tag,
input [3:0] step,
input l2clk,
input [31:23] replace_addr,
input [3:0] start_way,
input fill_en,
```

```
input [6:0] current_state,
input [3:0] replace_way,
output reg[127:0] reg_tag_updated
              );
`include "rtl/drctl/Memory_Parameters.vh"
/*************************************************************************************
*******/
//Valid/dirty/replace bits are updated depending on the type of operation:
//for fill_en, valid is update as V=1, dirty and Replace are reset as D=0, R=0, tag is written
//for Store, Dirty and Replace are updated as D=1, R=1, valid remain same
//for Load, only Replace is updated (if not previously set) as R=1, valid and dirty remain
same
/*************************************************************************************
*******/

always@(posedge l2clk)
  begin
    if (step == 4 || step == 5) //at fill_en we update tag block with info from mem address
           begin
             case (replace_way)
                  start_way: begin
                             if(step == 5 && fill_en && (current_state==eColumnWriteWait1))
reg_tag_updated[127:0] <= {reg_tag[127:16],1'b1,6'b0,replace_addr[31:23]}; //for fill_en
                      else if(step == 5 && ~fill_en)reg_tag_updated[127:0] <=
{reg_tag[127:15],2'b11,4'b0,reg_tag[8:0]}; //for store
                             else if((step==4)&&
(current_state==eBurstReadWait8))reg_tag_updated[127:0] <=
{reg_tag[127:14],1'b1,4'b0,reg_tag[8:0]}; //for load
                    end

                  start_way+ 1: begin
                             if(step == 5 && fill_en && (current_state==eColumnWriteWait1))
reg_tag_updated[127:0] <= {reg_tag[127:32],1'b1,6'b0,replace_addr[31:23],reg_tag[15:0]};
                             else if(step == 5 && ~fill_en) reg_tag_updated[127:0] <=
{reg_tag[127:31],2'b11,4'b0,reg_tag[24:0]};
                             else if((step==4)&& (current_state==eBurstReadWait8))
reg_tag_updated[127:0] <= {reg_tag[127:30],1'b1,4'b0,reg_tag[24:0]};
                    end

                  start_way+ 2: begin
                             if(step == 5 && fill_en && (current_state==eColumnWriteWait1))
reg_tag_updated[127:0] <= {reg_tag[127:48],1'b1,6'b0,replace_addr[31:23],reg_tag[31:0]};
                             else if(step == 5 && ~fill_en) reg_tag_updated[127:0] <=
{reg_tag[127:47],2'b11,4'b0, reg_tag[40:0]};
                             else if((step==4)&& (current_state==eBurstReadWait8))
reg_tag_updated[127:0] <= {reg_tag[127:46],1'b1,4'b0, reg_tag[40:0]};
                    end
```

```
                start_way+3: begin
                        if(step == 5 && fill_en && (current_state==eColumnWriteWait1))
reg_tag_updated[127:0] <= {reg_tag[127:64],1'b1,6'b0,replace_addr[31:23],reg_tag[47:0]};
                        else if(step == 5 && ~fill_en) reg_tag_updated[127:0] <=
{reg_tag[127:63],2'b11,4'b0,reg_tag[56:0]};
                        else if((step==4)&& (current_state==eBurstReadWait8))
reg_tag_updated[127:0] <= {reg_tag[127:62],1'b1,4'b0,reg_tag[56:0]};
                  end

                start_way+4: begin
                        if(step == 5 && fill_en && (current_state==eColumnWriteWait1))
reg_tag_updated[127:0] <= {reg_tag[127:80],1'b1,6'b0,replace_addr[31:23],reg_tag[63:0]};
                        else if(step == 5 && ~fill_en) reg_tag_updated[127:0] <=
{reg_tag[127:79],2'b11,4'b0,reg_tag[72:0]};
                        else if((step==4)&& (current_state==eBurstReadWait8))
reg_tag_updated[127:0] <= {reg_tag[127:78],1'b1,4'b0,reg_tag[72:0]};
                  end

                start_way+5: begin
                        if(step == 5 && fill_en && (current_state==eColumnWriteWait1))
reg_tag_updated[127:0] <= {reg_tag[127:96],1'b1,6'b0,replace_addr[31:23],reg_tag[79:0]};
                        else if(step == 5 && ~fill_en) reg_tag_updated[127:0] <=
{reg_tag[127:95],2'b11,4'b0,reg_tag[88:0]};
                        else if((step==4)&& (current_state==eBurstReadWait8))
reg_tag_updated[127:0] <= {reg_tag[127:94],1'b1,4'b0,reg_tag[88:0]};
                  end

                start_way+6: begin
                        if(step == 5 && fill_en && (current_state==eColumnWriteWait1))
reg_tag_updated[127:0] <=
{reg_tag[127:112],1'b1,6'b0,replace_addr[31:23],reg_tag[95:0]};
                        else if(step == 5 && ~fill_en) reg_tag_updated[127:0] <=
{reg_tag[127:111],2'b11,4'b0,reg_tag[104:0]};
                        else if((step==4)&& (current_state==eBurstReadWait8))
reg_tag_updated[127:0] <= {reg_tag[127:110],1'b1,4'b0,reg_tag[104:0]};
                  end

                start_way+7: begin
                        if(step == 5 && fill_en && (current_state==eColumnWriteWait1))
reg_tag_updated[127:0] <= {1'b1,6'b0,replace_addr[31:23],reg_tag[111:0]};
                        else if(step == 5 && ~fill_en) reg_tag_updated[127:0] <=
{reg_tag[127],2'b11,4'b0,reg_tag[120:0]};
                        else if((step==4)&& (current_state==eBurstReadWait8))
reg_tag_updated[127:0] <= {reg_tag[127:126],1'b1,4'b0,reg_tag[120:0]};
                  end
                default: begin reg_tag_updated[127:0] <= reg_tag[127:0];end
//mask_tag[15:0] = mask[15:0];end
                endcase
```

```
        end
 end

endmodule
```

col_write.v
top module name: col_write

```
`timescale 1ns/1ps

module col_write (
input gclk_l2,
input write_clk,
input reset,
input[3:0] step,
input[6:0] current_state,
input[3:0] step_cst,
input [31:0] data_wr0,
input [31:0] data_wr1,
input [31:0] data_wr2,
input [31:0] data_wr3,
input [31:0] data_wr4,
input [31:0] data_wr5,
input [31:0] data_wr6,
input [31:0] data_wr7,
output reg[31:0] DI_dram_port
                    );

`include "rtl/drctl/Memory_Parameters.vh"

reg [31:0] DI_dram_port_even;

reg[2:0] burst_counter0;
reg[2:0] burst_counter1;

wire DI_clk ;
wire burst_count_en;
wire burst_count_cond,DI_data_en,no_max_burst_count,continue_count;

assign DI_clk = gclk_l2;
assign write_burst_count = write_clk;
assign burst_count_cond = (current_state >= eBurstWriteWait1) & (current_state <=
eColumnWriteWait2) && (step == step_cst);
```

```
assign DI_data_en = (current_state >= eColumnWrite) & (current_state <=
eColumnWriteWait1) && (step == step_cst);

assign burst_count_en = burst_count_cond? 1'b1:1'b0;
assign no_max_burst_count = (burst_counter0 < 3'd7);
assign continue_count = no_max_burst_count && burst_count_en;

always @(posedge write_burst_count or negedge reset)
          begin
       if (~reset) burst_counter0 <= 3'b0;
                  else if(continue_count)          burst_counter0 <= burst_counter0 +  1'b1;
                  else     burst_counter0 <= 3'b0;
          end

//step7: data group
always @(posedge DI_clk)
begin
  if (DI_data_en)
   begin
     case( burst_counter0 )
       0 :  DI_dram_port[31:0] <= data_wr0[31:0];
       1 :  DI_dram_port[31:0] <= data_wr1[31:0];
       2 :  DI_dram_port[31:0] <= data_wr2[31:0];
            3 :  DI_dram_port[31:0] <= data_wr3[31:0];
            4 :  DI_dram_port[31:0] <= data_wr4[31:0];
            5 :  DI_dram_port[31:0] <= data_wr5[31:0];
       6 :  DI_dram_port[31:0] <= data_wr6[31:0];
            7 :  DI_dram_port[31:0] <= data_wr7[31:0];
            default :  DI_dram_port[31:0] <= 32'h0;
     endcase
   end
  else  DI_dram_port_even[31:0] <= 32'h0;


end

endmodule
```

step_fsm.v
top module name: status_fsm

```
`timescale 1ns/1ps

module status_fsm(
 input l2clk,
 input reset,
 input CK_div,
input CK_div_n,
```

```
 input  RefBar,
 input  CASBar,
 input  RASBar,
 input  PCBar,
 input  WEBar,
 input [6:0] current_state, //from DRAM_fsm.v
 input hit,
 input rd_en0,
 input wr_en0,
 input fill_en,
 input same_tag,
 input same_row,
 input data_exist,
 input refresh_needed,
 output reg[3:0] step,
 output reg done_bit,
 input burst_done_tag,
 input burst_done,
 output  burst_done_ff
);

`include "rtl/drctl/Memory_Parameters.vh"

reg data_read_bit,data_write_bit,tag_read_bit,tag_read_bit_ff,tag_write_bit;
reg[3:0] step_temp;//reg[4:0] step_temp;
reg done_tmp;

buffer #(1) burst_delay (
.in(burst_done),
.out(burst_done_ff));

always@(posedge l2clk)
        begin
                if(~reset)      done_bit <= 1'b0;
                else begin
                   if (step_temp==7 && step !=7) done_bit <= 1'b1;
                   else if(step==7) done_bit <=1'b0;
                end
        end

always@(posedge l2clk)
  begin
    if(~reset) step <= 4'b0;
    else step <= step_temp;
  end

always @(*)
        begin
```

```
                case(step)
                        0: //start-up @reset
                        begin
                          if( PCBar == 1'b0) step_temp = 1;
            else  step_temp = 0;
                    end
                        1: //PRECHARGE
                        begin
                          if ( current_state == ePreChargeWait9)
                            begin
                    if( RefBar == 1'b0) step_temp = 10;
                    else if( RASBar == 1'b0) step_temp = 2;
                    else step_temp = 1;
                                    end
                    end

                        2: //RAS
                        begin
                          if ((current_state == eRowAddressWait2) && (CASBar == 1'b0))
step_temp = 3;
                          else if ((current_state == eRowAddressWait3) && (CASBar == 1'b0))
step_temp = 3;
            else if ((current_state == eRowAddressWait4) && (PCBar == 1'b0)) step_temp = 3;
            else step_temp = 2;
                    end

                        3: //CAS TAG READ
                        begin
                          if ((current_state == eBurstReadWait8))//here either
                            begin
                                    if (~same_row)
                                     begin
                                      if((CASBar == 1'b0) && (WEBar == 1'b1) &&
burst_done_tag && hit && (rd_en0 || wr_en0)&& CK_div) step_temp = 4; //read request
after hit or write request: go to read first then on step4 go to write
                                      else if((CASBar == 1'b0) && (WEBar == 1'b0) &&
burst_done_tag && fill_en && CK_div) step_temp = 5; //fill request no need for hit (preload)
                                      else if((CASBar == 1'b1) && (WEBar == 1'b1) &&
burst_done_tag &&  ~hit && ~fill_en) step_temp = 7; //miss
                 else step_temp = 3;
                end
                                    else if(same_row)
                                     begin
                                      if((CASBar == 1'b0) && (WEBar == 1'b1) && hit &&
(rd_en0 || wr_en0)&& CK_div) step_temp = 4; //read request after hit or write request: go
to read first then on step4 go to write
                                      else if((CASBar == 1'b0) && (WEBar == 1'b0) && fill_en
&& CK_div) step_temp = 5; //fill request no need for hit (preload)
```

```
                                    else if((CASBar == 1'b1) && (WEBar == 1'b1) &&  ~hit &&
~fill_en) step_temp = 7; //miss
                 else step_temp = 3;
                end
                                  end
            if ((current_state == eColumnWriteWait5))
                              begin
                                    if((CASBar == 1'b0) && (WEBar == 1'b1) && hit && rd_en0
&& CK_div && same_row) step_temp = 4; //read request after hit in same row
                                    else if((CASBar == 1'b0) && (WEBar == 1'b0) && hit &&
wr_en0 && CK_div && same_row) step_temp = 5; //write request after hit same row
                                    else if((CASBar == 1'b0) && (WEBar == 1'b0) && fill_en &&
CK_div && same_row) step_temp = 5; //fill request same row (no burst_done_tag was
reached because no tag access)
                                    else if((CASBar == 1'b1) && (WEBar == 1'b1) && ~hit &&
~fill_en) step_temp = 7; //miss
                else step_temp = 3;
                                  end
                    end

                         4: //CAS DATA READ
                         begin
                           if ((current_state == eBurstReadWait8))//here either
                              begin
                                if((CASBar == 1'b0) && (WEBar == 1'b0) && wr_en0 && CK_div)
step_temp = 5; //write request after hit: go to read first then on step4 go to write
                 else if((CASBar == 1'b1) && (WEBar == 1'b1) &&  rd_en0 && (burst_done_ff))
step_temp = 7;
                 else step_temp = 4;
               end
                    end

                         5: //CAS DATA WRITE
                         begin
                           if (current_state == eColumnWriteWait4) step_temp = 7;
                           else step_temp = 5;

                    end

                         6: //CAS TAG UPDATE
                         begin
                           if ((current_state == eColumnWriteWait5) && ( PCBar == 1'b0))
step_temp = 1;
            else step_temp = 6;
                    end

                         7: //DONE
                         begin
```

```
                        if ((current_state == eBurstReadWait8))//after finishing read
                          begin
                                if((CASBar == 1'b0) && (WEBar == 1'b0) && CK_div)
step_temp = 6; //tag write request
                                else if((CASBar == 1'b1) && (WEBar == 1'b1) &&
(rd_en0||wr_en0) && same_row) step_temp = 3; //same row no need to access tag
              else step_temp = 7;
                               end
           else if ((current_state == eColumnWriteWait5)) //after finishing write
                          begin
                                if((CASBar == 1'b0) && (WEBar == 1'b0) && CK_div)
step_temp = 6; //tag write request
                                else if((CASBar == 1'b1) && (WEBar == 1'b1) &&
(rd_en0||wr_en0||fill_en)&&same_row) step_temp = 3; //same row no need to access tag
              else step_temp = 7;
                               end
                    end

                        10: //REFRESH
                        begin
                          if ( current_state == eRefreshWait18)
                            begin
              if( RASBar == 1'b0) step_temp = 2;
                                else if( RefBar == 1'b0) step_temp = 10;
              else step_temp = 10;
             end
                    end

                        default: //default
                        begin

                    end

                endcase
        end

endmodule

module buffer (out, in);
  parameter                 SIZE = 1;

  output      [SIZE-1:0]      out;
  input       [SIZE-1:0]      in;
  assign out = in;
endmodule
```

hit_logic.v
top module name: hit_logic

```
module hit_logic (
  input l2clk,
  input reset,
  input [3:0] step,
  input [127:0] reg_tag_in0,
  input [127:0] reg_tag_in1,
  input [8:0] lkup_tag0,
  input [17:0] previous_row,
  input [17:0] index0,
  input [17:0] addr_in_FB,
  input rd_en,
  input wr_en,
  input fill_en,
  input wr_to_rd,
  input burst_done0,
  input burst_done_tag,
  output reg[6:0] data_address,
  output same_tag,
  output same_row,
  output  data_exist,
  output  hit

                );
 wire[14:0]   way_hit;
 wire[15:0]    tag_way0;
 wire[15:0]    tag_way1;
 wire[15:0]    tag_way2;
 wire[15:0]    tag_way3;
 wire[15:0]    tag_way4;
 wire[15:0]    tag_way5;
 wire[15:0]    tag_way6;
 wire[15:0]    tag_way7;

 wire[15:0]    tag_way8;
 wire[15:0]    tag_way9;
 wire[15:0]    tag_way10;
 wire[15:0]    tag_way11;
 wire[15:0]    tag_way12;
 wire[15:0]    tag_way13;
 wire[15:0]    tag_way14;

 wire  hit0_3;
 wire  hit4_7;
 wire  hit8_11;
 wire  hit12_15;
```

```
wire  hit0_7;
wire  hit8_15;
reg  hit_tmp;
reg   [8:0]      lkup_tag_pre;
reg same_row_tmp, same_tag_tmp,burst_done_ff_tmp0;
assign   tag_way14  =  reg_tag_in1[111:96];
assign   tag_way13  =  reg_tag_in1[95:80];
assign   tag_way12  =  reg_tag_in1[79:64];
assign   tag_way11  =  reg_tag_in1[63:48];
assign   tag_way10  =  reg_tag_in1[47:32];
assign   tag_way9  =  reg_tag_in1[31:16];
assign   tag_way8  =  reg_tag_in1[15:0];

assign   tag_way7  =  reg_tag_in0[127:112];
assign   tag_way6  =  reg_tag_in0[111:96];
assign   tag_way5  =  reg_tag_in0[95:80];
assign   tag_way4  =  reg_tag_in0[79:64];
assign   tag_way3  =  reg_tag_in0[63:48];
assign   tag_way2  =  reg_tag_in0[47:32];
assign   tag_way1  =  reg_tag_in0[31:16];
assign   tag_way0  =  reg_tag_in0[15:0];

assign way_hit[0] = (tag_way0[8:0] == lkup_tag0[8:0]) & tag_way0[15];
assign way_hit[1] = (tag_way1[8:0] == lkup_tag0[8:0]) & tag_way1[15];
assign way_hit[2] = (tag_way2[8:0] == lkup_tag0[8:0]) & tag_way2[15];
assign way_hit[3] = (tag_way3[8:0] == lkup_tag0[8:0]) & tag_way3[15];
assign way_hit[4] = (tag_way4[8:0] == lkup_tag0[8:0]) & tag_way4[15];
assign way_hit[5] = (tag_way5[8:0] == lkup_tag0[8:0]) & tag_way5[15];
assign way_hit[6] = (tag_way6[8:0] == lkup_tag0[8:0]) & tag_way6[15];
assign way_hit[7] = (tag_way7[8:0] == lkup_tag0[8:0]) & tag_way7[15];
assign way_hit[8] = (tag_way8[8:0] == lkup_tag0[8:0]) & tag_way8[15];
assign way_hit[9] = (tag_way9[8:0] == lkup_tag0[8:0]) & tag_way9[15];
assign way_hit[10] = (tag_way10[8:0] == lkup_tag0[8:0]) & tag_way10[15];
assign way_hit[11] = (tag_way11[8:0] == lkup_tag0[8:0]) & tag_way11[15];
assign way_hit[12] = (tag_way12[8:0] == lkup_tag0[8:0]) & tag_way12[15];
assign way_hit[13] = (tag_way13[8:0] == lkup_tag0[8:0]) & tag_way13[15];
assign way_hit[14] = (tag_way14[8:0] == lkup_tag0[8:0]) & tag_way14[15];

assign hit0_3 = way_hit[0] | way_hit[1] | way_hit[2] | way_hit[3];
assign hit4_7 = way_hit[4] | way_hit[5] | way_hit[6] | way_hit[7];
assign hit8_11 = way_hit[8] | way_hit[9] | way_hit[10] | way_hit[11];
assign hit12_15 = way_hit[12] | way_hit[13] | way_hit[14] ;

assign hit0_7 = hit0_3 | hit4_7;
assign hit8_15 = hit8_11 | hit12_15;

always@(posedge l2clk)
    begin
```

```
            if (~reset) begin burst_done_ff_tmp0 <=1'b0; end
       else if (step==3 && burst_done0) begin burst_done_ff_tmp0 <=1'b1; end
            else burst_done_ff_tmp0 <= 1'b0;
      end

always @(*)
   begin
     if ((step == 3) && ~fill_en && (burst_done_tag||same_row))  hit_tmp = (~hit0_7 &
hit8_15)|(hit0_7 & ~hit8_15);
           else hit_tmp = 1'b0;
   end
assign hit = hit_tmp;

always @(*)
 begin
   if (~reset) begin data_address = 7'h0;end
   else if (step==3)
    begin
        if      (way_hit[0]) begin data_address = 8; end //8i+ 8
        else if (way_hit[1]) begin data_address = 16; end
        else if (way_hit[2]) begin data_address = 24; end
        else if (way_hit[3]) begin data_address = 32; end
        else if (way_hit[4]) begin data_address = 40; end
        else if (way_hit[5]) begin data_address = 48; end
        else if (way_hit[6]) begin data_address = 56; end
        else if (way_hit[7]) begin data_address = 64; end
        else if (way_hit[8]) begin data_address = 72; end
        else if (way_hit[9]) begin data_address = 80; end
        else if (way_hit[10]) begin data_address = 88; end
        else if (way_hit[11]) begin data_address = 96; end
        else if (way_hit[12]) begin data_address = 104; end
        else if (way_hit[13]) begin data_address = 112; end
        else if (way_hit[14]) begin data_address = 120; end

    end
end

 always @(posedge l2clk)
  begin
    if (~reset) lkup_tag_pre <= 9'h1FF;
    else if ((step==4 && rd_en) || (step==5 && wr_en)) lkup_tag_pre <= lkup_tag0;
  end

reg same_en_ff, same_en;
always@(posedge l2clk)
  begin
    if(~reset) same_en_ff <= 1'b0;
    else if(step==7 && ~same_en_ff && ~same_en) same_en_ff <=1'b1;
```

```
    else same_en_ff <=1'b0;
 end

always @(*) begin
  if (~reset) same_en = 1'b0;
  else if (step==7 && same_en_ff) same_en = 1'b1;
  else if (step==3) same_en = 1'b0; end

always@(*)
  begin
    if(~reset) same_tag_tmp = 1'b0;
    else if(lkup_tag_pre==lkup_tag0 && step==7 &&(rd_en||wr_en)&& same_en)
same_tag_tmp = 1'b1;
    else if(step==4 || step==5) same_tag_tmp = 1'b0;
   end

 always@(*)
  begin
    if(~reset) same_row_tmp = 1'b0;
    else if(previous_row==index0 && (step==7 ) &&(rd_en||wr_en) && same_en)
same_row_tmp = 1'b1;
    else if(previous_row==addr_in_FB && (step==7 ) &&(fill_en) && same_en) same_row_tmp
= 1'b1;
    else if(step==4 || step==5) same_row_tmp = 1'b0;
   end

assign  same_tag = same_tag_tmp;
assign  same_row = same_row_tmp;
reg data_exist_tmp;

 always@(*)
  begin
    if(~reset) data_exist_tmp = 1'b0;
    else if(same_row && same_tag && ~wr_to_rd && step==3 && ~data_exist_tmp)
data_exist_tmp = 1'b1;
    else if(step==4 ) data_exist_tmp = 1'b0;
   end

reg data_exist_ff;
always @(posedge l2clk)
  begin
    if(~reset) data_exist_ff <= 1'b0;
    else if (~data_exist_ff) data_exist_ff <= data_exist_tmp;
    else data_exist_ff <= 1'b0;
 end
assign data_exist = data_exist_tmp | data_exist_ff;

endmodule
```

cpx_out.v
top module name: cpx_out
instance of: DW_asymfifo_s2_sf => this is open source synopsys design-ware for asym fifo

```
module cpx_out( inst_clk_push,inst_clk_pop, inst_rst_n, push_n_out, d_out_8bit,
burst_done_ff,data_exist,reg_data0,reg_data1, step);

  parameter data_in_width = 256;
  parameter data_out_width = 8;
  parameter depth = 4;
  parameter push_ae_lvl = 1;
  parameter push_af_lvl = 1;
  parameter pop_ae_lvl = 1;
  parameter pop_af_lvl = 1;
  parameter err_mode = 0;
  parameter push_sync = 1;
  parameter pop_sync = 1;
  parameter rst_mode = 0;
  parameter byte_order = 0;
  `define bit_width_depth 1 // ceil(log2(depth))

  input inst_clk_push;
  input inst_clk_pop;
  input inst_rst_n;
  output push_n_out;
  output [data_out_width-1 : 0] d_out_8bit;
 input [3:0] step;
 input burst_done_ff;
 input data_exist;
 input[127:0]    reg_data0;
 input[127:0]    reg_data1;

 reg  data_push_req_n;
 reg addr_push_req_n;
 wire data_pop_req_n, addr_pop_req_n;
 wire done_32B_delay;
 wire pop_empty_data, pop_empty_addr, addr_popping;
 wire [7:0] data_out_inst, addr_out_inst;
 wire [data_in_width-1 : 0] inst_data_in;

 assign inst_data_in = {reg_data1,reg_data0};
 assign done_32B_delay = (step == 4) & (burst_done_ff | data_exist);//done_32B;

always @(posedge inst_clk_push)
  begin
    if (~inst_rst_n) data_push_req_n <= 1'b1;
    else if (done_32B_delay) data_push_req_n <= 1'b0;
          else if (~data_push_req_n ) data_push_req_n <= 1'b1;
```

```
  end

assign data_pop_req_n = pop_empty_data;
assign push_n_out = data_pop_req_n;
assign d_out_8bit = data_out_inst;

 DW_asymfifo_s2_sf #(.data_in_width       (data_in_width),
    .data_out_width     (data_out_width),
    .depth              (depth),
    .push_ae_lvl        (push_ae_lvl),
    .push_af_lvl        (push_af_lvl),
    .pop_ae_lvl         (pop_ae_lvl),
    .pop_af_lvl         (pop_af_lvl),
    .err_mode           (err_mode),
    .push_sync          (push_sync),
    .pop_sync           (pop_sync),
    .rst_mode           (rst_mode),
    .byte_order         (byte_order))
    data_unit (
              .clk_push(inst_clk_push),
              .clk_pop(inst_clk_pop),
        .rst_n(inst_rst_n),
                    .push_req_n(data_push_req_n),
        .flush_n(inst_flush_n),
                    .pop_req_n(data_pop_req_n),
        .data_in(inst_data_in),
                    .push_empty(push_empty_inst),
        .push_ae(push_ae_inst),
                    .push_hf(push_hf_inst),
        .push_af(push_af_inst),
                    .push_full(push_full_inst),
        .ram_full(ram_full_inst),
                    .part_wd(part_wd_inst),
        .push_error(push_error_inst),
                    .pop_empty(pop_empty_data),
        .pop_ae(pop_ae_inst),
                    .pop_hf(pop_hf_inst),
        .pop_af(pop_af_inst),
                    .pop_full(pop_full_inst),
        .pop_error(pop_error_inst),
                    .data_out(data_out_inst) );

endmodule
```

read_write_enable.v
top module name: read_write_enable

```
module read_write_enable (
```

```
 input l2clk,
 input reset,
 input [3:0] step,
 input ctrl_arb_done,
 input read_en,
 output rd_en0,
 output wr_en0,
 output reg wr_to_rd );

reg rd_en;
reg wr_en;
reg startup_req;

 assign rd_en0 = rd_en; //maybe use FF
 assign wr_en0 = wr_en;

always @ (posedge l2clk)
begin
   if(~reset) begin startup_req <= 1'b1; end
   else if(rd_en || wr_en) startup_req <= 1'b0;
end

always@(*)begin

 if(~reset) begin rd_en =1'b0; wr_en =1'b0; wr_to_rd = 1'b0; end
 else if(((step==1) && startup_req)|| (step==10)) begin
           if(read_en)
      begin
       rd_en = 1'b1;
       wr_en = 1'b0;
    end

    else if (ctrl_arb_done)
      begin
        rd_en = 1'b0;
        wr_en = 1'b0;
      end
  end
 else if(step==7) begin
           if(read_en)
      begin
       rd_en = 1'b1;
       wr_en = 1'b0;
    end

    else if (ctrl_arb_done)
      begin
        rd_en = 1'b0;
```

```
      wr_en = 1'b0;
    end
  end

 else if (step==4 && wr_to_rd) wr_to_rd = 1'b0;

 end

 endmodule
```

lru.v
top module name: pLRU

```
`timescale 100ps/1ps

module pLRU
 (
   input l2clk,
   input reset,
   input[3:0] step,
   input fill_en,
   input same_row,
   input[4:0] half_tag15, //last 4-bit contains record of last row filled (MRU) and first 6-bit a
constant showing is row ever accessed before

   input burst_done_tag,
   output [3:0] replace_way
 );
 wire [3:0] MRU;
 reg[3:0] replace_way_tmp;
 reg start_up;
 //reg row_used;

 assign MRU = half_tag15[3:0];
 assign replace_way = replace_way_tmp;

 always @ (posedge l2clk)
  begin
   if (~reset) start_up <=1'b1;
   else if (fill_en && (step == 1 || step == 10)) start_up <=1'b1; //added to stop confusion
when filling a new row. or we can make replace_way a FF
   else if (step==7) start_up <=1'b0;
  end

 always @ (posedge l2clk)
  begin
   if (~reset) begin replace_way_tmp <= 4'b0; end
```

```
   else if (fill_en && (step == 3) /*&& (~same_row || ~same_tag)*/ &&
(burst_done_tag||same_row))
    begin

                    if(~start_up && (replace_way_tmp < 15)) replace_way_tmp <= MRU[3:0] +
1'b1;
        else replace_way_tmp <= 4'b0;
    end
  end

endmodule
```

write_request.v
top module name: write_request
instance of: col_write.v

```
module write_request (
input gclk_l2,
input write_clk,
input reset,
input[3:0] step,
input[6:0] current_state,
input fill_en,
input [127:0] reg_data0,
input [127:0] reg_data1,
input [127:0] data_wr0_FB, //coming from FB after all 64B are in
input [127:0] data_wr1_FB,
output[31:0] DI_dram_port

                    );
reg [31:0] data_wr0_iq;
reg [31:0] data_wr1_iq;
reg [31:0] data_wr2_iq;
reg [31:0] data_wr3_iq;
reg [31:0] data_wr4_iq;
reg [31:0] data_wr5_iq;
reg [31:0] data_wr6_iq;
reg [31:0] data_wr7_iq;

wire [31:0] data_wr0;
wire [31:0] data_wr1;
wire [31:0] data_wr2;
wire [31:0] data_wr3;
wire [31:0] data_wr4;
wire [31:0] data_wr5;
wire [31:0] data_wr6;
wire [31:0] data_wr7;
```

```
always @(*)
 begin
  if (step == 5'h4 && ~fill_en)
    begin
{data_wr7_iq,data_wr6_iq,data_wr5_iq,data_wr4_iq,data_wr3_iq,data_wr2_iq,data_wr1_iq,data
_wr0_iq} = {reg_data1,reg_data0}; end
  else begin
      data_wr0_iq =data_wr0_iq;
      data_wr1_iq=data_wr1_iq;
      data_wr2_iq=data_wr2_iq;
      data_wr3_iq=data_wr3_iq;
      data_wr4_iq=data_wr4_iq;
      data_wr5_iq=data_wr5_iq;
      data_wr6_iq=data_wr6_iq;
      data_wr7_iq=data_wr7_iq;
      end
 end

assign data_wr0[31:0] = (fill_en & step==5) ? data_wr0_FB[127:96]:data_wr0_iq;
assign data_wr1[31:0] = (fill_en & step==5) ? data_wr0_FB[95:64]:data_wr1_iq;
assign data_wr2[31:0] = (fill_en & step==5) ? data_wr0_FB[63:32]:data_wr2_iq;
assign data_wr3[31:0] = (fill_en & step==5) ? data_wr0_FB[31:0]:data_wr3_iq;
assign data_wr4[31:0] = (fill_en & step==5) ? data_wr1_FB[127:96]:data_wr4_iq;
assign data_wr5[31:0] = (fill_en & step==5) ? data_wr1_FB[95:64]:data_wr5_iq;
assign data_wr6[31:0] = (fill_en & step==5) ? data_wr1_FB[63:32]:data_wr6_iq;
assign data_wr7[31:0] = (fill_en & step==5) ? data_wr1_FB[31:0]:data_wr7_iq;

col_write data_write(
.gclk_l2(gclk_l2),
.write_clk(write_clk),
.reset(reset),
.step(step),
.step_cst(4'h5),
.current_state(current_state),
.data_wr0(data_wr0[31:0]),
.data_wr1(data_wr1[31:0]),
.data_wr2(data_wr2[31:0]),
.data_wr3(data_wr3[31:0]),
.data_wr4(data_wr4[31:0]),
.data_wr5(data_wr5[31:0]),
.data_wr6(data_wr6[31:0]),
.data_wr7(data_wr7[31:0]),
.DI_dram_port(DI_dram_port[31:0])
                    );

endmodule
```

fillbuf_sim.v

top module name: fill_buffer

```
module fill_buffer (
input l2clk,
input reset,
input[3:0] step,
input [31:5] addr_in,
input[127:0] data_in0, //from preload module
input[127:0] data_in1,
input bank_en,
input rd_en0,
input self_test_en,
//output
output reg fill_en, //to tell ctrl to start data write
output reg[31:5] addr_out,
output reg[127:0] data_wr0,
output reg[127:0] data_wr1
                    );

reg bank_en_ff,self_test_en_ff;

always @(posedge l2clk)
begin
  if (~reset)
   begin
      addr_out[31:5] <= 27'h0;
            bank_en_ff <=1'b0;
            self_test_en_ff <=1'b0;
    end
  if ((step == 7 || step == 1 || step == 10)  && bank_en)
   begin
      data_wr0[127:0] <= data_in0[127:0];
            data_wr1[127:0] <= data_in1[127:0];
      addr_out[31:5] <= addr_in[31:5];
            bank_en_ff <=1'b1;
            self_test_en_ff <=1'b0;
    end
  else if ((step == 7 || step == 1 || step == 10)  && self_test_en)
   begin
      data_wr0[127:0] <= 128'h614d6268_6f43656b_22232425_f874676e;
            data_wr1[127:0] <= 128'h29312873_00010203_33343536_6f436501;
      addr_out[31:5] <= 27'h0;
            self_test_en_ff <=1'b1;
            bank_en_ff <=1'b0;
    end

  else begin bank_en_ff <=1'b0; self_test_en_ff <=1'b0;end
end
```

```
reg filling;
always@(*)
  begin
    if(~reset) begin fill_en = 1'b0; filling = 1'b0;end
    else if((bank_en_ff||self_test_en_ff) ) begin fill_en = 1'b1;filling = 1'b1;end
          else if(step==5) filling = 1'b0;
    else if((step==7)&&~filling) fill_en = 1'b0;
   end

endmodule
```

refresh.v

top module name: refresh_logic

```
//refresh counter is 32ms which is the worst case of temperature 120C
//for 1GHZ clk counter_done = 32,000,000 ns = 25'h1E84800
//for 1MHZ clk counter_done = 32,000 us = 16'h7D00
//for 100kHZ clk counter_done = 3,200 = 12'hC80
//for 10kHZ clk counter_done = 320 = 10'h140
//for 1kHZ clk counter_done = 32 = 6'h20

module refresh_logic (
  input l2clk,
  input l2clk_N,
  input CK_div,
  input refresh_clk,
  input reset,
  input [6:0] current_state,
  output reg RefBar,
  output  RefBar1,
  output reg refresh_needed
                );

 `include "rtl/drctl/Memory_Parameters.vh"

 reg[5:0] refresh_counter;

 wire refresh_ready;
 reg counter_reset_n;
 reg CSBar_ref;
 wire reset_count;
assign reset_count = ~reset | ~counter_reset_n;

always @(posedge refresh_clk or posedge reset_count)
   begin
     if (reset_count ) refresh_counter <= 0;
     else if (refresh_counter < 32) refresh_counter <= refresh_counter +  1'b1;
```

```
    else refresh_counter <= 0;
  end
 reg delay1,delay2;
always @(posedge l2clk_N)
  begin
    if (~reset) begin refresh_needed <= 1'b0; counter_reset_n <= 1'b1; delay1 <=1'b0;delay2
<=1'b0;end
          else if (refresh_counter >= 32) begin refresh_needed <= 1'b1; delay1
<=1'b1;counter_reset_n <= ~counter_reset_n; end
          else if (current_state == eRefreshWait18 && delay2) begin refresh_needed <= 1'b0;
delay1 <=1'b0; delay2 <=1'b0;counter_reset_n <= 1'b1; end
          if (delay1) delay2<=1'b1;
  end

always @(posedge l2clk_N)
  begin
    if (~reset) begin RefBar <= 1'b1; CSBar_ref <= 1'b1; end
    else if (refresh_needed && ~CK_div && RefBar)
      begin
        RefBar <= ~refresh_ready;
                  CSBar_ref <= ~refresh_ready;
      end
    else  begin RefBar <= 1'b1; CSBar_ref <= 1'b1; end
  end

assign refresh_ready = (current_state == ePreChargeWait9 || current_state ==
eRefreshWait18)? 1'b1:1'b0;
assign RefBar1 = RefBar;

endmodule
```

B.3. Post-Synthesis Output File: Bank0-Row0-to-3

8002a001 80048003 80068005 80088007 800a8009 800c800b 800e800d 0000800f => Tag0
600d6028 6003602b 20232025 e034602e 20312033 00010003 20342036 60036001 =>Row0 start
60346035 60136032 0033603a 20202020 60252020 202e6020 602f6022 20202002
20336039 60206029 000a6033 602e6033 4006400d 60356021 20302033 a0340003
202c4024 20336028 60206025 20336039 20342028 20256031 00000029 20202004
20256021 00000000 602e6033 4006400d 60356021 20312033 00000000 602e6005
00000000 602e6033 4006400d 60356021 20342033 00000000 20202020 60222006
0000202c 60246023 60256020 60286027 20242020 00000020 602e6033 40064007
60332020 202c6025 60236029 60256025 60252034 0000000a 60216033 602f2008
00000009 00000000 00106018 0000202f 002b2000 00180000 00000000 00020009
00020010 00000000 00000000 00000000 00108010 0000e032 002b2000 00180010
40020011 00000000 00200000 00000000 00108011 0000e032 002b2000 0018a011
00020012 00000000 00300000 001e0000 00108012 0000e032 002b2000 00180012
00030013 00000000 00000000 00034000 00108013 0000e032 002b2000 00180013
00040014 00064000 00000000 00090000 00108014 0000e032 002b2000 00180014
00050015 00000000 00390000 00050000 00108015 0000e032 002b2000 00180015 => Row0 end
8002a001 80048003 80068005 80088007 800a8009 800c800b 800e800d 0000800f => Tag1
00060016 00056000 00000000 00022000 00108016 0000e032 002b2000 00180016 =>Row1 start
60060017 00056000 00000000 00022000 00108017 0000e032 002b2000 00180017
e03f6034 e03fe02f a012000f 00008008 4000c00a 8012c00e 00008008 80120018
20004028 40020008 c03fc00b a0104000 20180000 00004020 c03fc02b 0001c019
40020006 00008008 0000600a 00008037 0006800a 4022600c 00008008 00000020
8006e00a 00004003 00004003 00004003 000d0017 00004003 0010602b 000e0021
00000003 00000003 000da017 00000003 000ea017 000fa017 8020a007 00000022
e01b6012 20204003 0000e00b 8006800b c03ec00b 003fc00b 003fc00b 00004023
0020a00c 801f800b c03ec00b 200f4028 40020008 40020008 40020008 60014024
c03f0038 c0140020 e03f200b a009801b c03fc016 c029e03f e03f400b 201aa025
c03fc018 a0210020 e03fc033 c03be00b c03f602a 80140000 e03f803f 00142026
c03f4000 60090020 e03fc01f e029c03c c03f8014 4008e03f e03fc000 a0320027
c03f001f 001c0020 e03f0039 203fe02d c03fc018 20100000 e03f0004 80384028
c03f0031 402f0020 e03f800f e00d601f c03f2028 601e0000 e03fc00a 600be029
c03f003f 80300000 e03fe03a e0216011 c03f6005 80060000 e03f0009 c035e030 =>Row1 end
80028001 80048003 80068005 80088007 800a8009 800c800b 800e800d 000e800f => Tag2
00060016 00056000 00000000 00022000 00108016 0000e032 002b2000 00180032 =>Row2 start
60060017 00056000 00000000 00022000 00108017 0000e032 002b2000 00180033
e03f6034 e03fe02f a012000f 00008008 4000c00a 8012c00e 00008008 80120034
20004028 40020008 c03fc00b a0104000 20180000 00004020 c03fc02b 0001c035
40020006 00008008 0000600a 00008037 0006800a 4022600c 00008008 00000036
8006e00a 00004003 00004003 00004003 000d0017 00004003 0010602b 000e0037
00000003 00000003 000da017 00000003 000ea017 000fa017 8020a007 00000038
e01b6012 20204003 0000e00b 8006800b c03ec00b 003fc00b 003fc00b 00004039
0020a00c 801f800b c03ec00b 200f4028 40020008 40020008 40020008 60014000
c03f0038 c0140020 e03f200b a009801b c03fc016 c029e03f e03f400b 201aa001

c03fc018 a0210020 e03fc033 c03be00b c03f602a 80140000 e03f803f 00142002
c03f4000 60090020 e03fc01f e029c03c c03f8014 4008e03f e03fc000 a0320003
c03f001f 001c0020 e03f0039 203fe02d c03fc018 20100000 e03f0004 80384004
c03f0031 402f0020 e03f800f e00d601f c03f2028 601e0000 e03fc00a 600be005
c03f003f 80300000 e03fe03a e0216011 c03f6005 80060000 e03f0009 c035e006 =>Row2 end
X0Xx8001 X0XxX0Xx X0XxX0Xx X0XxX0Xx X0XxX0Xx X0XxX0Xx X0XxX0Xx 0000X0Xx => Tag3
c03f801d 60330000 e03f600a 6010c003 c03fa013 20140000 e03f800c 203b6031 =>Row3 start