STUDY AND ANALYSIS OF POWER OPTIMIZATION TECHNIQUES FOR HIGH-SPEED CACHE MEMORY ARCHITECTURE

M.Tech Thesis

By

ROHIT KUMAR LILHARE

DISCIPLINE OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

JUNE 2021

STUDY AND ANALYSIS OF POWER OPTIMIZATION TECHNIQUES FOR HIGH-SPEED CACHE MEMORY ARCHITECTURE

A THESIS

Submitted in partial fulfillment of the requirements for the award of the degree

of

Master of Technology

by

ROHIT KUMAR LILHARE

DISCIPLINE OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

JUNE 2021

INDIAN INSTITUTE OF TECHNOLOGY INDORE

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the thesis entitled **STUDY AND ANALYSIS OF POWER OPTIMIZATION TECHNIQUES FOR HIGH SPEED CACHE MEMORY ARCHITECTURE** in the partial fulfillment of the requirements for the award of the degree of **MASTER OF TECHNOLOGY** and submitted in the **DISCIPLINE OF ELECTRICAL ENGINEERING, Indian Institute of Technology Indore**, is an authentic record of my own work carried out during the time period from JULY 2019 to JUNE 2021 under the supervision of Dr. Swaminathan Ramabadran, Assistant Professor, IIT INDORE and Mr. Abhishek Sakharwade, Staff Engineer, QUALCOMM BANGLORE

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other institute.

Rohit 04/06/2021

Signature of the student with date
(ROHIT KUMAR LILHARE)

This is to certify that the above statement made by the candidate is correct to the best of my/our knowledge.

[illegible] 04/06/2021

Signature of the Supervisor of
M.Tech. thesis (with date)
(DR. SWAMINATHAN R.)

[illegible] 4/6/2021

Signature of the Supervisor of
M.Tech. thesis (with date)
(MR. ABHISHEK SAKHARWADE)

ROHIT KUMAR LILHARE has successfully given his/her M.Tech. Oral Examination held on **4 June 2021**.

[illegible] 4/6/2021 [illegible] 04/06/2021

Signature(s) of Supervisor(s) of M.Tech. thesis
Date: 04/06/2021

[illegible] (Swaminathan R)

Convener, DPGC
Date: 04/06/2021

Dr. Puneet Gupta [illegible]

Signature of PSPC Member 1
Date: 7 June 2021

[illegible]

Signature of PSPC Member 2
Date: 4/6/2021

Acknowledgements

I would like to express my sincere gratitude to my thesis supervisor Dr. Swaminathan Ramabadran for his constant support, patience, encouragement, and immense knowledge. His guidance helped me in conducting the research and writing of this M.Tech dissertation.

I would also like to thank my external supervisor Mr. Abhishek Sakharwade for giving me the opportunity to work under him and his direction. I wish to express gratitude to my supervisor for providing good platform at QUALCOMM to perform my work.

I am thankful to the members of my thesis committee: Dr. Srivathsan Vasudevan and Dr. Puneet Gupta for their encouragement and insightful comments. Their questions and suggestions helped me to widen my knowledge from various perspectives.

I am thankful to all the faculty members of the Department of Electrical Engineering for their support. I must also thank my fellow batch mates and all other friends for making my stay at IIT Indore delightful. I also thank my family for their unceasing encouragement and support throughout this journey.

Rohit Kumar Lilhare
M.Tech (communication and signal processing)
1902102005
Discipline of Electrical Engineering, IIT INDORE

Abstract

Power optimization in cache memory is very attractive area of research to achieve data transfer at high speed between the high-speed processor and low speed main memory (DRAM). The processor accesses the main memory through cache memory to enhance system performance. This thesis demonstrates that cache usage is an important measure of performance and power with an emphasis on the order of access to the cache line. In this thesis, we mainly focused on cache memory optimization through L1 data cache memory. For that we did experiment on 32KB size of cache memory architecture with 64B cache size and 4-way set associative cache and divide this 32KB size of cache memory into smaller size of caches using 2KB, 4KB, 8KB and 16KB size of logical banks to organize 32KB memory. After that we compared all 2KB, 4KB, 8KB and 16KB logical banks based on the number of comparators used, number of shared or individual address decoder used, number of multiplexers required in each logical bank. After that we evaluate the miss rate in data transfer, cost in each logical bank organization, bandwidth required in each organization and memory traffic in each organization. Sleep mode finite machine concept is also used as per the Qualcomm memory specs to save the power and in that clock is used only when memory is busy in reading and writing of data, else clock is off as per memory timing specification. With the help of all these observations we decided 4KB and 8KB logical bank organization is better suited for one big 32KB size logical bank organization.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

SRAM	Static Random-Access Memory
DRAM	Dynamic Random-Access Memory
ECHP	Early Cache Hit Predictor
NTW	No Tag Matching Write
LA	Locality-based Allocation
FSM	Finite State Machine
LB	Logical Bank
CPU	Central Processing Unit
AMAT	Average Memory Access Time
ALU	Arithmetic Logic Unit

Chapter 1

Introduction

The Memory system is a vital component of any high-performance computer system. Memories in computing, refers to the physical devices used to store data (e.g. program state information) or programs (sequences of instructions) on a temporary or permanent basis in a computer or other digital electronic device. The term primary memory, that operate at high-speed (i.e. RAM), as a compared from secondary memory, is used to denote the information in physical systems. Secondary memory can be accessed via large storage devices such as hard discs, floppy disk storage media, optical storage devices which offers greater capacity but slow to accessing data. In contrast to primary memory, secondary memory is not directly accessible via the processor. Primary memory access through the processor is done by utilizing address and data busses, while the input and output channels serve to access the secondary memory. Data and instructions that are stored inside the memory are accessed using the memory addresses which denote the location of each element inside the memory [1].

1.1 Memory Hierarchy

Memories can be organized based on access time, cost of the memory, size of the memory. For perceived execution speed of the processor, memory speed is often the major component, since the processor can only execute data and instruction as fast as memory provides. It is obvious that the design will aim for a large-sized, high-speed memory for a given cost. The solution to this problem is offered by hierarchical memory design.

A typical memory hierarchy as shown in figure 1.1, as one moves down the hierarchy, the cost per bit decreases, the access time increases, the capacity increases and frequency the memory access by the processor decreases. Hierarchy design of memory is based on two principles: first is the principle of locality (both temporal and spatial), and second is the cost or performance of the memory. Temporal locality principal says that it is likely for recently accessed memory locations is being accessible sometime in the not-too-distant future. The principal of Spatial locality says that memory locations are likely to be accessible within a short span of time that are nearby [2]. Memory hierarchy work well, because programs prefer to access the storage at any high level more often than they access the storage at the next lower level. As a result, storage at the next level might be slower, but bigger and less expensive per bit. Thus, smaller, faster and more expensive memories are therefore supported with bigger, less costly, slower memories. The total effect is a vast pool of memory which thus costs the same as cheap storage at the lower level of the hierarchy but feeds data to processes at the same rate as fast storage at the top [1][2].

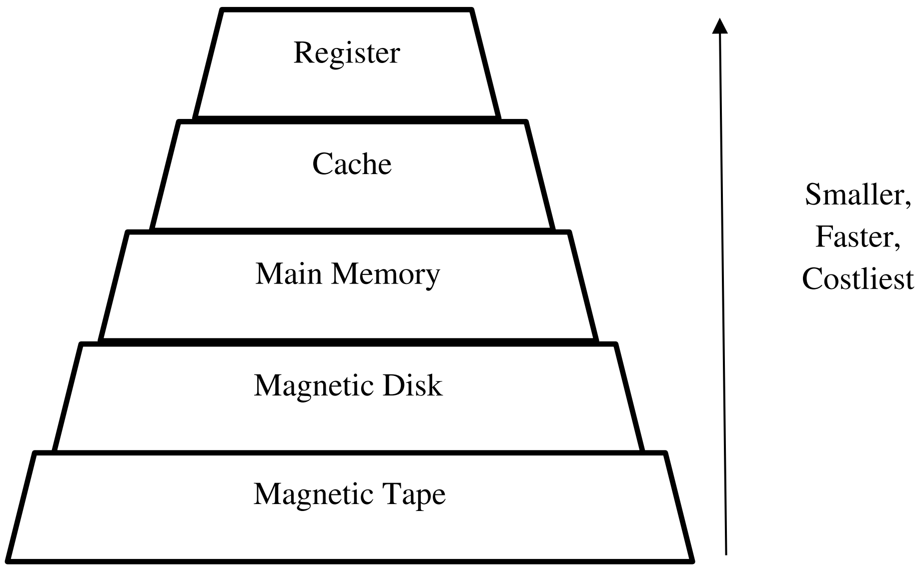

Figure 1.1 Memory Hierarchy [2]

1.2 Motivation

As the gap between performance of processor and main memory increases continuously, it causes a significant roadblock to exascale computing. Memory performance has lagged behind the processor performance, and today due to the advent of data-intensive applications particularly, becoming a bottleneck. Figure 1.2 [3] shows this performance gap over past three decades.

With the advent of data intensive applications, memory is fast becoming the critical bottleneck for system performance and is not scaling at the same rate as the computing performance.

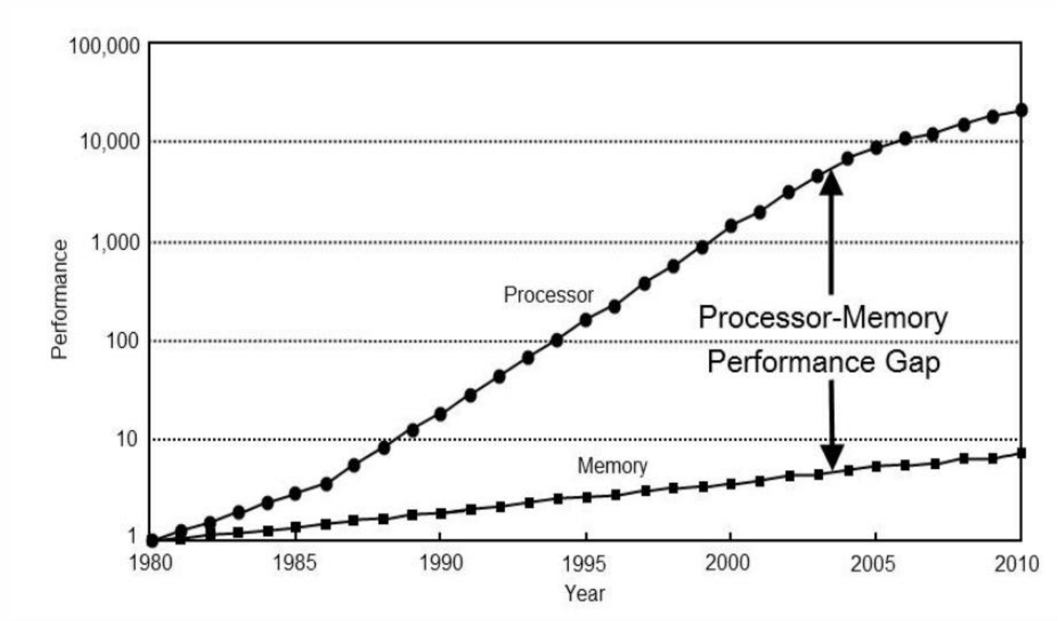

Figure 1.2 Performance Curve [3]

Basic architecture of memory and processor is shown in figure 1.3. In this figure processor connected with a main memory module (DRAM) whose performance is far behind in comparison to the processor. The performance of processor cores increases at a faster pace. Compared to large capacity main memory, the processor is generally able executes operations on operands faster than main memory access time. Although semiconductor memory exists that can function at rates equivalent to the processor's, it is not cost-effective to offer very high-speed semiconductor memory for the main memory, taking in view the capacity of the main memory. The performance of the processor and to the system as whole, can indeed be bottleneck due to the presence of slower memory.

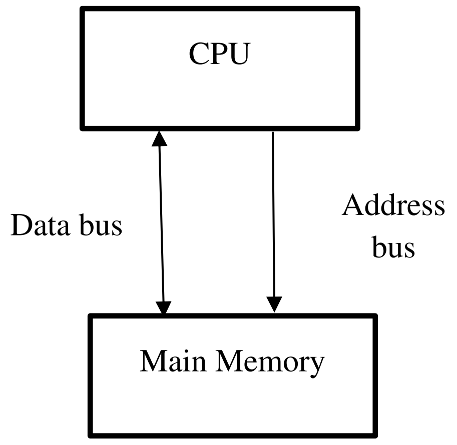

Figure1.3 Basic memory and processor architecture [6]

The problem of performance gap between main memory and processor can be taken the edge off by inserting a small block of high-speed memory known as a cache (SRAM) between the processor and the main memory and figure 1.4 shows that representation. The program issues effective addresses and read and write requests, and these requests are satisfied by memory. The request is unknown to the program, whether it is the cache (SRAM) or main memory (DRAM) that processes. Usually, instructions and data in cache memories can be referenced in 10 to 25 percentage of the time taken to access main memory [5]. Thus, the execution rate of the system to be substantially increased by the cache memories. If majority of memory location accesses are cached memory locations, then the average latency of memory accesses will be nearer to the cache memory latency than to the main memory [6].

So, cache memory employs as bridging the performance gap between the high-speed processor and slow speed main memory. The processor cache is a hardware component that stores frequently accessed data. High bandwidth and low latency of memory with respect to the processor can be achieved by the memory caches. In this thesis we discuss techniques for power optimization that involve in enhancing the performance of the high-speed cache memory organization.

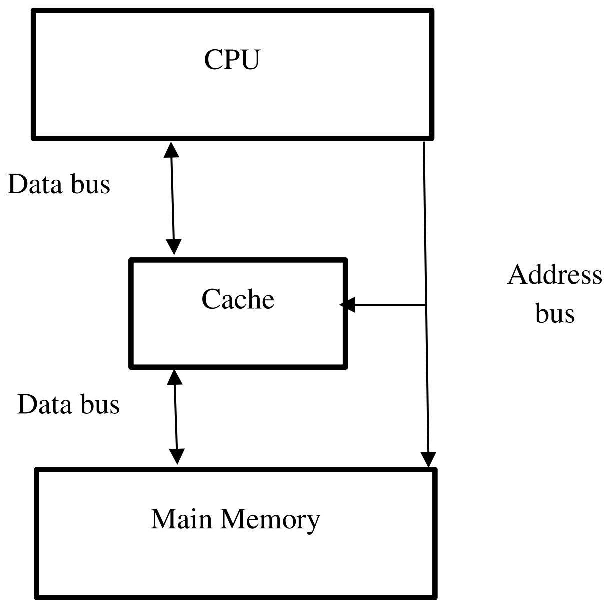

Figure1.4 Basic memory and processor with cache [6]

1.3 Organization of Thesis

In the further part of the thesis is organized in the following fashion:

- Chapter 2: In this chapter we will discussed about the related work that is already done with respect to the high-speed cache memory performance enhancement.

- Chapter 3: In this chapter we will focus on the terms that is very important terms related to the cache memory. After that we will add the mapping technique for cache memory. Then we will cover the writing policy for the cache.

- Chapter 4: In this chapter we will discuss about the proposed technique for the implementation of 32Kb cache. And other power saving option that can be done in the cache memory organization

- Chapter 5: In this chapter, we will capture the various test bench results for the different organization and the comparison table for different logical banks organizations.

- Chapter 6: In this chapter, we will conclude what we have done and what possibly future work can be done in this field.

1.4 Summary

In this chapter, we have discussed about the evolution of the memory hierarchy and how cache memory comes into picture for the evolution of memory performance. Then the motivation behind the cache memory optimization technique. And in the last how we organized the thesis for further parts.

Chapter 2

Literature Survey

In this chapter, we discuss the existing works related to the theme of the proposal. In [7], the authors proposed a technique in which the effectiveness of the target banks improved by continuously adjusting its associativity. Also, the expense of sending requests is improved by using network distance as an extra parameter for target selection. The methodology used here has varied working set requirements, as well as uneven distribution of data across cache banks, which is dynamically handled to determine which banks should be switched off. Target bank will get the content of shutdown bank that will also, take care of any future requests for this closed bank. The target bank manages the shutdown bank's additional load, so will get a greater number of requests. Proposed technique achieved 23 percentage EDP minimization for a 4MB lower level of caches and 43 percentage reduction in static energy with a performance constraint of 3 percentage.

In [8], A.D. Jebaseeli, M. Kiruba proposed a new cache architecture to improve the accuracy of cache miss prediction by a partial tag enhanced-bloom filter. Significant amounts of energy will be saved without sacrificing performance abasement by allowing the L2 cache to function in a direct mapping mode while write hits and minimizing tag comparison of cache miss prediction; there will be no necessitated to use the L2 cache if a cache miss is detected. L2 cache does not functional for all tags, and the partial bloom filter predicts cache misses. On average, the suggested methodology saves 63 percent of the energy in L2 cache while incurring just 0.02 percent to the area overhead.

In [9], J. Lee and S. Kim, proposed a filter data cache design in order to efficiently integrate the filter cache into the data cache hierarchy. They identified that since the filter cache is being used for data cache, misses occur considerably. Those misses increase cache latency and loading superfluous data that effects cost performance and energy consumption. Three approaches are used in the described filter data cache design to minimize miss costs: ECHP, LA, and NTW. When compared to the filter cache, the suggested filter data cache decreases energy consumption by 21 percent, and the ALU's energy usage on average by 27.2 percentage. The filter data cache depicts an immense benefit on energy as the size of the L0/L1 caches grows, and it has minimal operating cost in terms of area and power leakage.

In paper [10], A. G. Kumar, D. A. Janeera, M. Ramesh proposed novel cache architecture so called Tag-Bloom cache designed to minimize power usage. Every way with in L2 cache has a tag associated to it, is contained in way tag array in FLC. During successive write hits, the L2 cache is being accessed as a direct mapped cache, reducing power consumption. This method enhances the efficiency of both cache miss and cache hit prediction. To increase the efficiency of cache hit as well as cache miss predictions, the Bloom filter examines these way tags. Thus, with a minimum area overhead they achieved energy effectual cache design without sacrificing efficiency deterioration.

In paper [11], C. H. Ting, Y. H. Kao and J. D. Huang proposed the concept of sequential way access, on each set-associative cache access ensuring minimal energy usage and high performance to limit the number of ways that can be activated. The proposed techniques eliminate future accesses once a hit is identified after accessing each way in sequence. It also reduced the heavy fan out load of the hit signal, as a result of a more sophisticated cache control mechanism, which prevents the cache cycle time against increasing. The experiments demonstrate that when compared to a traditional 2-way set-associative cache of size 32KB, the same size 2-way sequential access set-associative cache consumes 24 percentage less energy with no performance compromise.
In paper [12], Z. Chuanjun, V. Frank, N. Walid shown that changing a cache's line size

to a specific program is seemed to be an incredibly efficient approach for minimizing memory access energy for embedded devices. By adding this configuration to a cache architecture is straight forward and embedded system designers can select the best configuration simply. They have worked on a configurable cache architecture with not only have a variable line size as well as provides flexible number of ways to access it with maintaining same total size. Experiment result showed that owing to memory access, adjusting the line size to a specific program can minimize energy dissipation by 50 percentage.

In paper [13], T. Ishihara, F. Fallah proposed architecture of a non-uniform cache for minimizing power requirements for memory systems. Non uniform cache provides differing associativity levels (i.e., the number of cache-ways) across distinct cache sets. In this paper, authors also proposed a programming approach for decreasing cache-tag and cache-way requests that are redundant. The proposed approach evaluates the optimal number of cache-ways for each cache-set and produces non-uniform cache memory object code. Result showed that the methodology has the potential to minimize power usage by up to 76 percent in comparison to the conventional method of memory systems.

In paper [14], M. Mutyam, C. J. Janraj, T. Warrier, T. V. Kalyan proposed way-sharing cache design, in which two cache sets exchange certain cache ways such that high-demand sets obtain the most cache ways than low-demand sets. Regardless of the set-by-set requirements, in standard k-way set-associative caches, every set has n cache ways at its allotments, although demands for these cache ways might vary between cache sets. Using this feature, way-sharing cache architecture has been proposed in which by permitting cache ways to be shared across two cache sets, as high as 41 percentage dynamic power savings with negligible performance penalty is achieved.

In paper [15], Z. Chuanjun, V. Frank, N. Walid proposed technique in which three software-configurable settings in the cache can be customized to the specific purposes. The cache's associativity may be adjusted to be direct-mapping, 2-way, or 4-way set-associative mapping using a unique mechanism called way concatenation, is first one. The total size of a cache may be varied by having to shut down ways is the second configurable parameter. Third and last one is, the line size of the cache can be customized to 16 bytes, 32 bytes, or 64 bytes. Configurable cache adjusted to each program ended up saving energy for every program with an overall energy savings of over 40 percentage associated to memory access when compared to the standard 4-way set associative cache and a traditional direct-mapping cache.

Chapter 3

Theoretical Foundation of Cache Memory

Cache memory have been playing an important role in bridging the performance gap between the high-speed processor and slow speed main memory. Cache memory is special type SRAM placed between the slow main memory and the fast processor. Cache is the first or highest level of memory structure that the processor encounters upon leaving an address. Cache memories are smaller in size, cache memories are employed in modern computer systems to temporarily store sections of the main memory's (DRAM) contents that are currently be in use.

When request is generated to a memory, the cache memory will get the request first from processor. Then the check will be made to determine whether the data is present in the cache and if present, then the data will be sent to the processor. If data is not there, a block with some fixed number of words (based on cache size) from main memory is read into the cache memory and then the word will be sent to the processor. The concept behind the fast cache memory is similar to virtual memory that some active part of data (frequently used data) of a slow main memory is get stored in a cache memory. It is the implementation of cache and virtual memory that made the difference otherwise both rely on the correlation features seen in address reference sequences, so conceptually the same but because of the speed requirements of cache their implementations are totally different. Because of principle of locality of reference, the cache memory to be effective by storing only the subset (i.e., subset of recently used instructions and data) of the slow main memory. With reference to the principle of locality of reference, to satisfy a single memory reference When a block of data is fetched into the cache, there is a chance that subsequent accesses to that memory location or other data in the block will occur. So that the upcoming requests for the respective data may be served more quickly because cache stores the recently accessed data [4][6].

In a particular processor design, the term word is referring for the unit of data used, a word is represented by fixed size group of bits. In the form of word, the transfer of traffic to and from the processor and the cache takes place. And in the form of block between the cache and main memory, the traffic is transferred. With respect to the cache these cache blocks are referred to as cache lines. Usually, a word size varies between 8, 16, 32 and 64 bits, in the modern processors.

3.1 Cache Memory Architecture

Cache memory has three levels of cache. Levels describe its closeness and accessibility to the processor. L1 is called as the first level of cache (FLC) and L2 and L3 is called as the last level of cache (LLC). Figure 3.1 showed that the three levels between the fast processor and the slow speed memory.

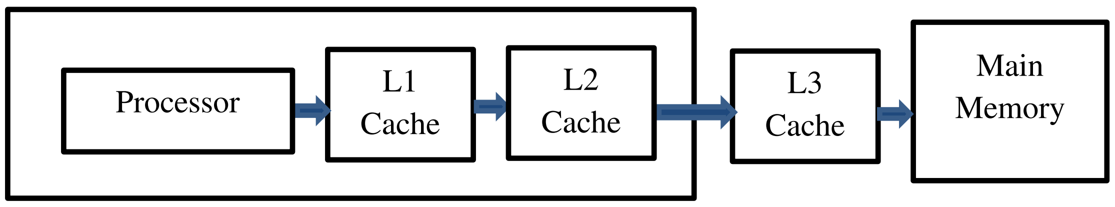

Figure 3.1 3 Levels of cache between processor and main memory

For L1 cache, instruction and data caches are separated, whereas for last levels of cache, instructions and data cache are unified. L1 and L2 are present inside processor and L3 are located outside the processor. With the help of FLCs, we can minimize access latency and while with the help of LLCs reduce the numbers of off-chip accesses and cache miss-rate. By design L1 caches are smaller in size and employ parallel lookup and tag arrays and have smaller associativity. LLCs, on the other hand, are significantly bigger, use serial or phased data and tag array lookups, and have stronger associativity.

FLCs consume a bigger portion of their power in the form of dynamic power due to their lower size and quantity of accesses, while L2 and L3 caches spend the majority of their power in terms of leakage power [16].

The basic cache memory architecture shown in the figure 3.2[16].

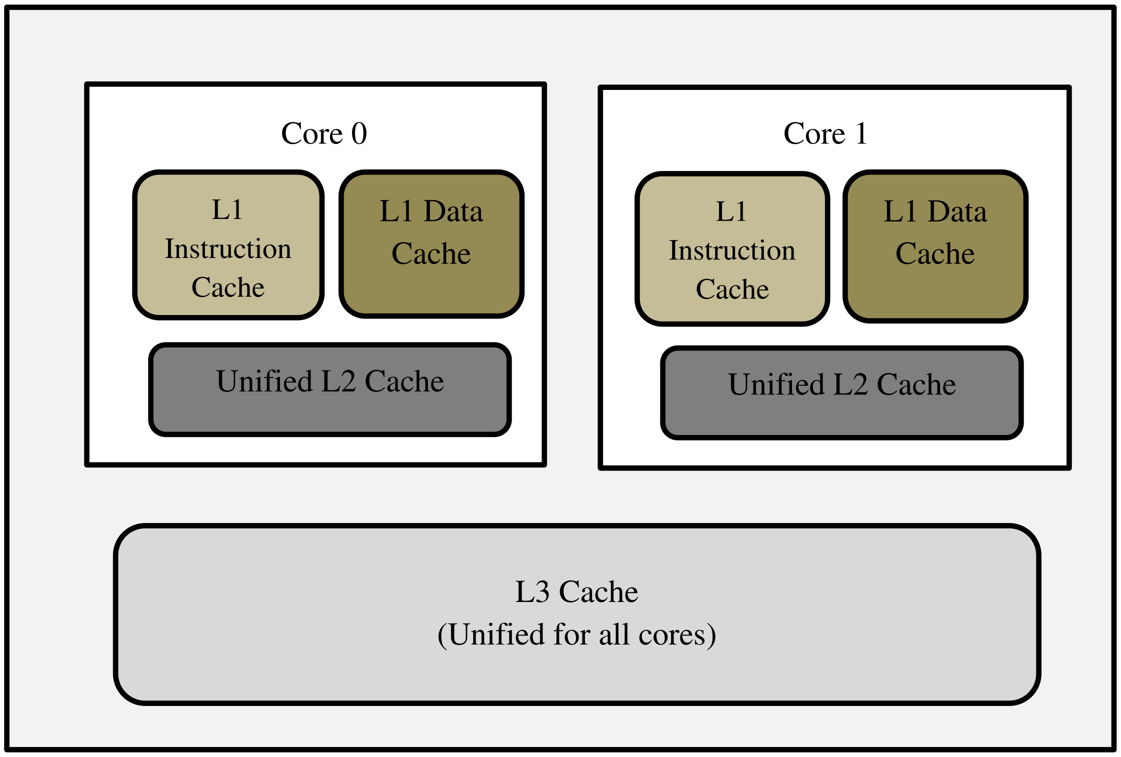

Figure 3.2 Basic cache memory architecture for 2 cores

In this figure we have taken the example of processor which has two core. L1 data cache and L1 instruction cache are separated for each core and for each core has there is unified L2 cache memory. And L3 cache memory is unified for all the cores. A unified cache is more flexible because it offers greater hit ratio compared to the splits caches and it can flexibly accommodate either data or instruction and the program may have a larger fraction of instructions than data or vice versa. The fundamental benefit of a split cache is that it avoids conflict in the cache between the instruction fetch/decode unit and the execution unit. In this thesis, we will mainly focus on L1 data cache memory.
In figure 3.3 we have listed some basic functions of L1 cache and in the figure 3.4, we have listed the functions of L2 and L3 cache memory [4].

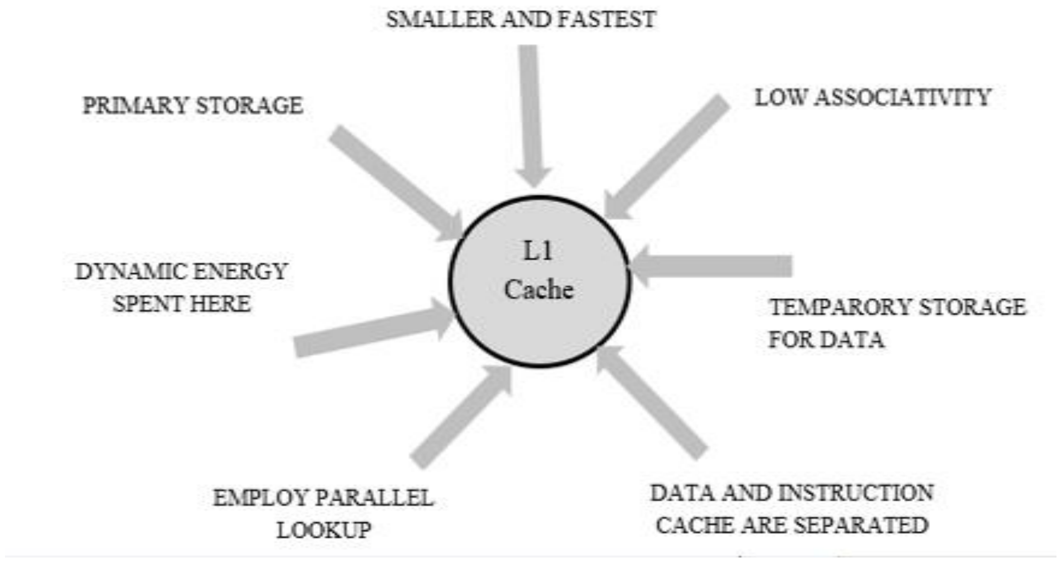
SMALLER AND FASTEST
PRIMARY STORAGE
LOW ASSOCIATIVITY
L1 Cache
DYNAMIC ENERGY SPENT HERE
TEMPARORY STORAGE FOR DATA
EMPLOY PARALLEL LOOKUP
DATA AND INSTRUCTION CACHE ARE SEPARATED

Figure 3.3 Functions of L1 cache

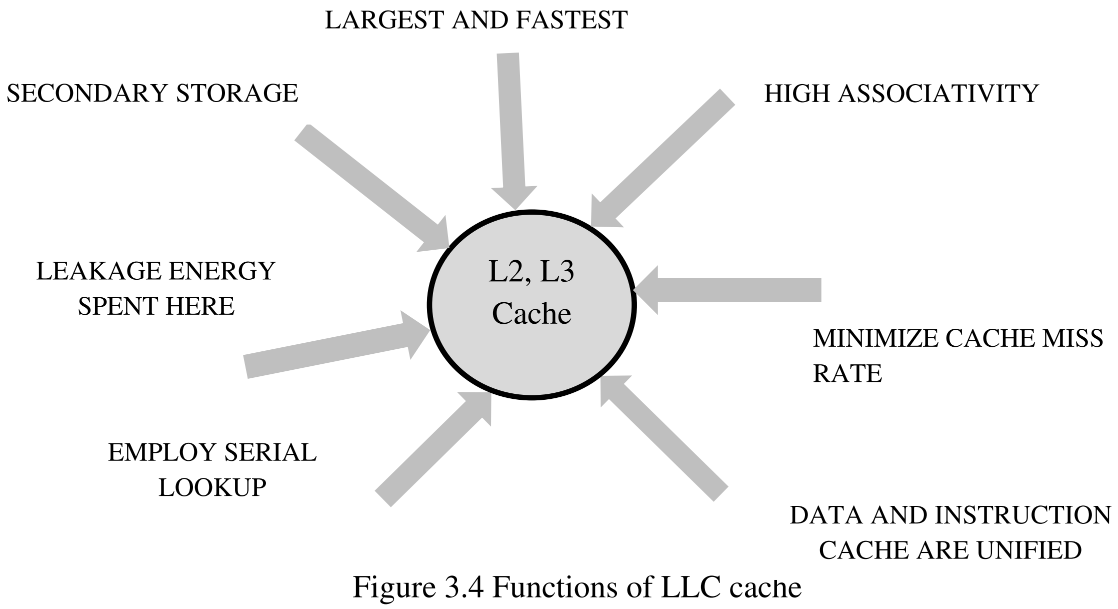

Figure 3.4 Functions of LLC cache

3.2 Terms Related to Cache Memory

Cache Hit: If the data or instruction requested by the processor is present in the cache, then it is termed as cache hit. The data is taken from that level and propagated up the hierarchy.

$$Hit\ Rate = \frac{\textbf{Number of hits}}{\textbf{Total number of memory access}}$$

Cache Miss: If the required data or instruction requested by the processor is not present in the cache, then it is termed as cache miss and request goes down a level until found. A cache miss at any level may overwrite old data when the requested new data is propagated up the hierarchy which result longer execution time.

Access Time: It indicates the time for representing the address and retrieve the appropriate data from memory.

Memory Cycle Time: It is the time required by the memory to recover before next access.

Cache Lines: The unit of data transfer between the cache memory and main memory is represented by cache line. Most commonly used cache line is 64 bytes. Caches lines contain copies of blocks of data from main memory.

Average Memory Access Time:

AMAT = Hit Time + (Miss Rate * Miss Penalty access)

Hit Time = Hit Rate * Access Time

The various types of misses are there in cache memory which is categorized as:

- Compulsory misses: These are known as first references misses. When first access to the block happens then compulsory misses occur. From the upper-level memory, data must be fetched at least once to be present in the cache.
- Conflict misses: Conflict misses occurs during the mapping of two different data to the same cache line.
- Coherency misses: Coherency misses occur when the data updated by the external processor or input output device.
- Capacity misses: Capacity misses occur when the data and instructions required for

the program exceeds the cache size.

3.3 Cache Memory Organizations

A cache memory is a special type static random-access memory (SRAM), where copies of information currently used by instructions and data, loaded from the slow speed main memory stores by the computer hardware. The cache has more expensive implementation technology but has significantly shorter access time than the main memory. If the cache memory is used again for information fetched to it, then the access time to it will be much shorter than in the case of the main memory. If the same information is stored in it, then the program will execute faster because of the cache memory [18].

Cache memory operation governs the principle of locality of reference. There are two methods for retrieving programs from main memory and storing them in cache.

Temporal Locality: Temporal locality means currently being fetched instruction or data may be needed soon. To avoid searching of that data in main memory, instruction or data should be stored in the cache memory [17].

There is a high probability that if some data is referenced, then that data will be brought up again in the near future.

Spatial Locality: Spatial locality refers to the present memory location where an instruction or data is being retrieved, which may be required in the near future. Spatial locality is slightly differing from temporal. The focus of spatial locality was on nearly situated memory locations, but the focus of temporal locality was on the actual memory locations being fetched [17].

The cache mapping techniques used to bring the main memory's contents into cache and are then referenced by the processor. It is basically an algorithm by which cache lines are mapped from main memory blocks. The number of cache lines is often less than the number of main memories blocks [18].

To determine which main memory block occupies a cache line in cache memory, mapping methods are required. According to the mapping policy cache memory is organized. Three different types of cache organizations commonly used based on mapping functions.

3.3.1 Direct Mapped Cache:

Implementation of direct mapped cache is very simple because every single main memory block is mapped to single cache line. In this mapping technique, every memory block of main memory is allotted for specific cache line. If the main memory block that in recently present in cache line, and new block of data is required by processor, then the previous block is deleted. The address bits are divided into three segments like as offset bits, index bits and tag bits. index bits are used to access the cache directory, offset bits used get the specified byte within a cache line and tag bits field must be compared with the tag bits of respective data block associated with it, to ensure a hit [19][20].

If CPU generates request to a memory, the index bit is used for accessing the specific line of the cache memory. The tag bit of that address is then compared with the tag address of the cache memory. If this tag is match with the tag address field, the word is present in the cache and cache hit occurs. Else, a cache miss will be occurred. If cache miss occurs, then required word will be taken from the main memory. And by changing the existing tag with the new one, it will be saved in the cache memory. In figure 3.5 block number J is selected for line L based on the tag and index bits. Direct mapped cache has high conflict misses because of lack of associativity but have better access time.

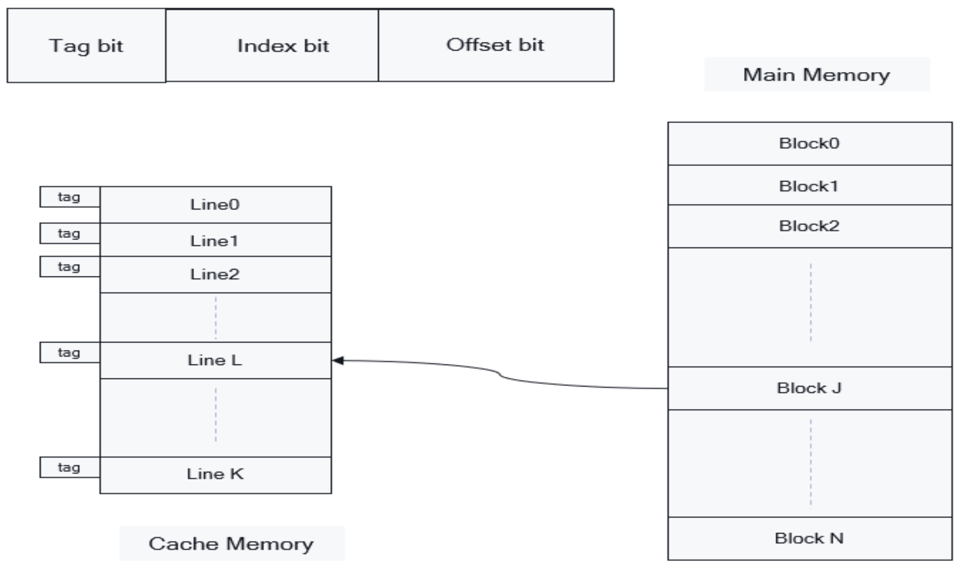

Figure 3.5 Direct mapped cache

3.3.2 Fully Associative Mapped Cache:

In fully associative mapped cache technique, the block of main memory can be placed into any of the cache line in cache memory, which is freely available. Fully mapped cache technique is more flexible compared to the direct mapped cache, but implementation of fully associative mapping is more complex. Address bit is divided into two parts one is tag bit field which is used for identifying the given block uniquely and other is offset bits that is used get the specified byte within a cache line. when CPU generate the request to cache, the sent address request is compared to each entry in the tag array to determine whether the data referenced for the operation is there in cache or not. If requested address is found in cache memory, data belong to that address location in the cache is fetched and returned back to the processor, otherwise a miss occurs, then required data will be taken from the main memory [21][22].

In the figure 3.6 the block J can be placed any of the cache line which is free in the cache memory.

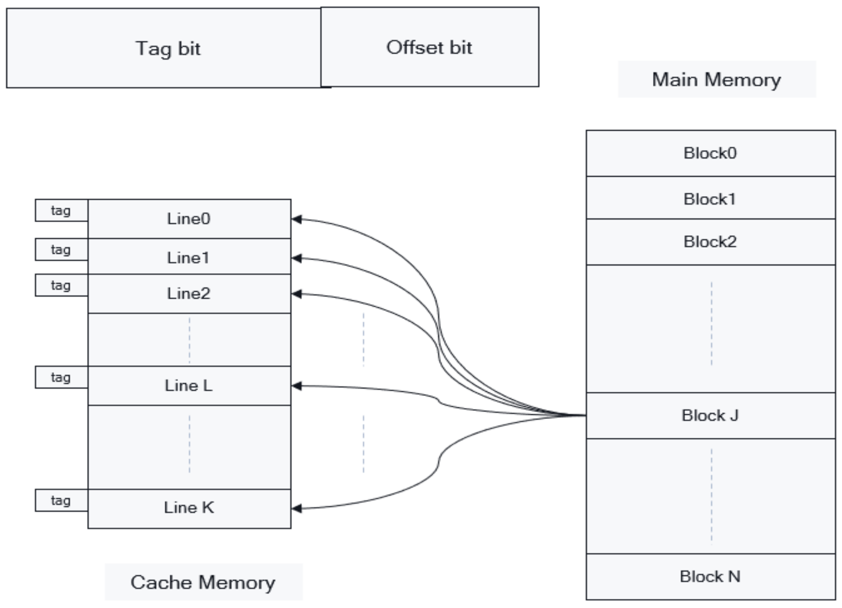

Figure 3.6 Fully associative mapped cache

3.3.3 K-Way Set Associative Mapped Cache:

In the K-way set associative mapped cache technique, according to the number of sets cache lines are grouped and K number of lines is present in each set. A specific block of the main memory can be mapped to one specific set in the cache memory. And block may be mapped into any one of the ways within the set that depends on the size of K. Typical size of K varies between 2-ways, 4-ways, 8 ways and 16 ways. Set way associative mapped cache is tradeoff between fully associative mapped cache and the direct mapped cache. Address bit is divided into three parts, tag bits field, set bits field and offset bit field. If request is generated from the processor, then set bits field is used to select one particular set in the cache memory and then with the set based on the tag bits field will be decided from which way, we have to get the data. If the tag bits are not matched with any of the ways, then cache miss will occur, and the request will be sent to the main memory to get that data from the particular location [23][24].

Figure 3.7 is the example of 4-way set associative cache memory that shows reading of data from the given address, based on the set bits will select the set from the cache memory. Then we will compare the tag bit from the each of the ways in the selected set and result will generate by the comparator and with the help of the multiplexer data will be sent to the processor and with the OR gate we will decide cache hit or miss occurs.

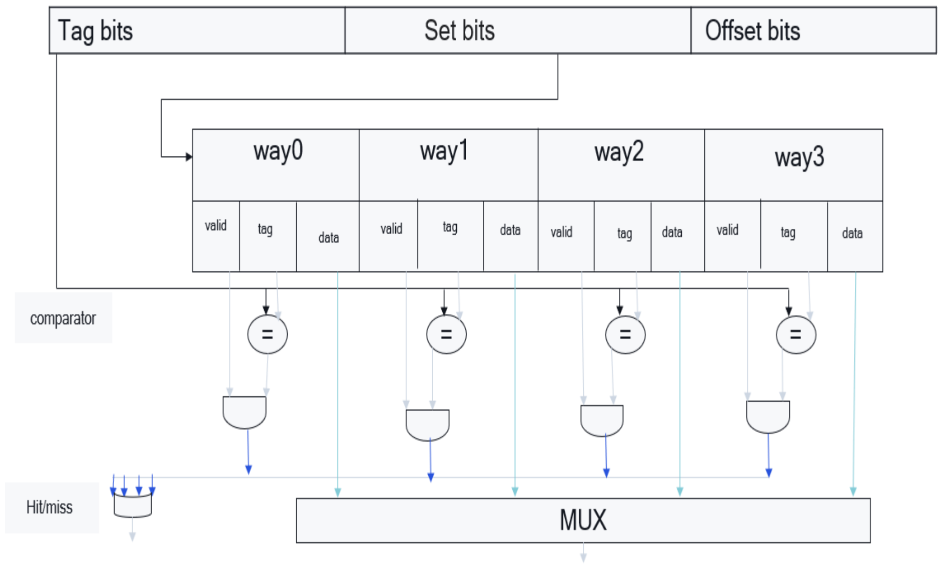

Figure 3.7 4-way set associative cache mapping

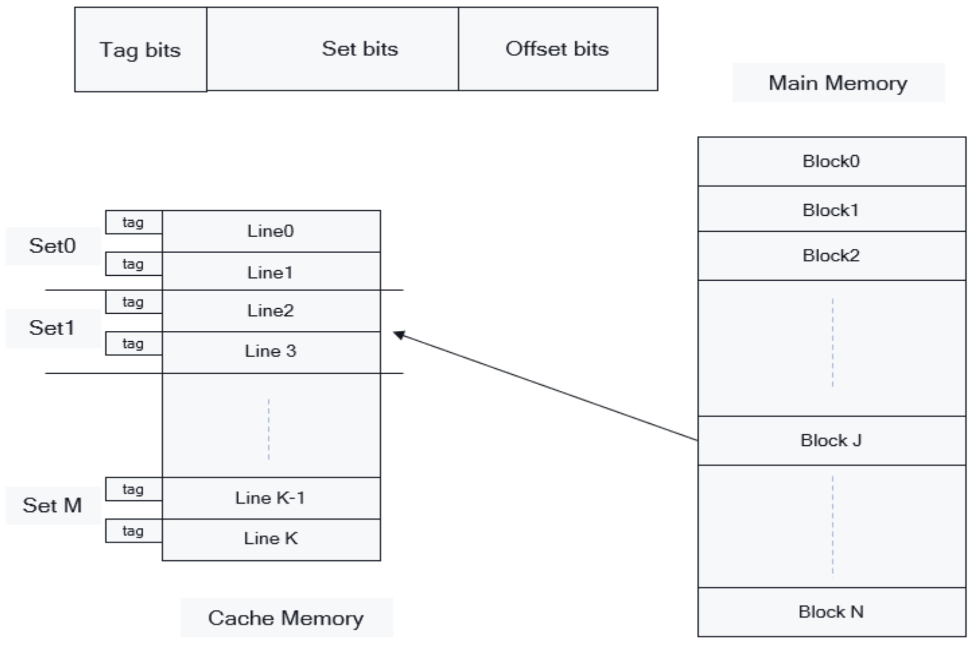

Figure 3.8 2-way set associative mapping

Figure 3.8 show the organization of 2-way set associative mapping, in which block number J is mapped to the set number 1 of the cache memory.

Cache memory with the higher associative organization leads to more complex hardware requirements but will have a lower miss rate as each set has more blocks, so chances of a conflict between two addresses is less. Fully associative cache makes complete use of their capacity but relying on expensive search process, so they have higher hardware cost. To search in K-way set-associative cache is easier than fully-associative cache.

3.3.4 Writing Policy in Cache Memory

While performing the read operation on cache memory it will not change data present in the memory. But performing any write operation on the cache will changes the content data of the cache memory. So, it is very important to perform any write on cache memory carefully. Assume a value is already cached and we want to overwrite it, but only changing the cached value would result in a discrepancy between the cached copy of data and the original data value preserved in main memory [27]. In cache memory, to ensure cache memory and main memory coherence, two major write policies are used:

Write Through Policy: Write-through policy is the most commonly methods to be used for writing into the cache. In write-through method, when the cache memory is updated, it also updates the main memory at the same time. So, at any time, the main memory will contain the same data which is available in the cache memory. Write through policy is a slow process because of excessive write and every time it needs to access main memory. This method will perform good, if there are few writes to a given block and if there are multiple writes to the given block, then it will be disadvantageous [26].

Write Back Policy: Write-back policy also used for cache memory writing. In this policy only the cache memory location is updated during a write operation while following write-back method. Updated location is marked by a flag, when update in cache occurs. The flag is named as dirty bit or modified bit. When a dirty cache line is thrown out, then only the stored data will be written to main memory [25].

Inconsistency may occur in write back techniques because of different copies of data in cache and main memory, which is the only limitation of this technique. Flag bit is set if cache memory is replaced by the word, then word will be written into main memory. The idea behind this method is based on the fact that during write operation in cache memory, the word present in the cache memory may be accessed several times before also, so reduce the number of references to main memory [27].

Chapter 4

Power Optimization in Cache Memory

4.1 Design Specification of Memory Wrapper

Designed configurable memory wrapper to enable analysis of different configuration with respect to power, area and performance for target IP. The features of the memory wrapper include the configurable cache size with the default setting of cache size used is 64 bytes. For the design of memory wrapper design we have used configurable size logical bank (LB). Our design mainly focused on design of 32KB size of memory wrapper. So if we use logical bank of size 8KB, then we will have requirement of 4 logical banks. For the design of memory wrapper, we have used configurable address decoder architecture. It is the job of decoder to locate the selected memory block. The address decoder architecture will be shared between all the logical banks. In our design we have used 32KB memory wrapper with 64 data bits, then we have 12-bit address and LB of 8KB having 10-bit address. So last two bit of address used for the decoding logic and allow logical bank to be placed in any 8KB section of 32Kb address space.

Port to the memory wrapper:

- **Input Ports**
    - Clock and Reset
    - Write/Read Enable
    - Write Address
    - Active Input
    - Write Data
- **Output Port**
    - Read Data

Sleep Control logic also added inside the memory wrapper design. Sleep control logic provides the power saving feature in the memory design whenever there is no reading or writing is going on (means memory is in idle mode). Sleep control logic count the idle number of clock pulse and put all logical banks into the sleep.

4.2 Top Level Block Diagram of the Memory Wrapper

Figure 4.1 show the top-level block diagram of the memory wrapper. In the top-level block diagram, we have shown input ports to the memory wrapper and read out as the output port to the wrapper.

In the block diagram we have shown memory controller, which drives input interface to the logical banks via finite state machine (FSM). Also, memory controller gets the data from the input port and drive it to the logical banks as per the memory protocol.

Sleep controller is responsible for the efficiently managing power mode (power up and power) of the memory wrapper. It uses the activity counter for transition through various sleep state to save power. Sleep controller maintains the activity counter per logical banks. It is following the Qualcomm memory timing specification for enter and exits from the sleep mode and with the help of this we have designed sleep mode FSM.

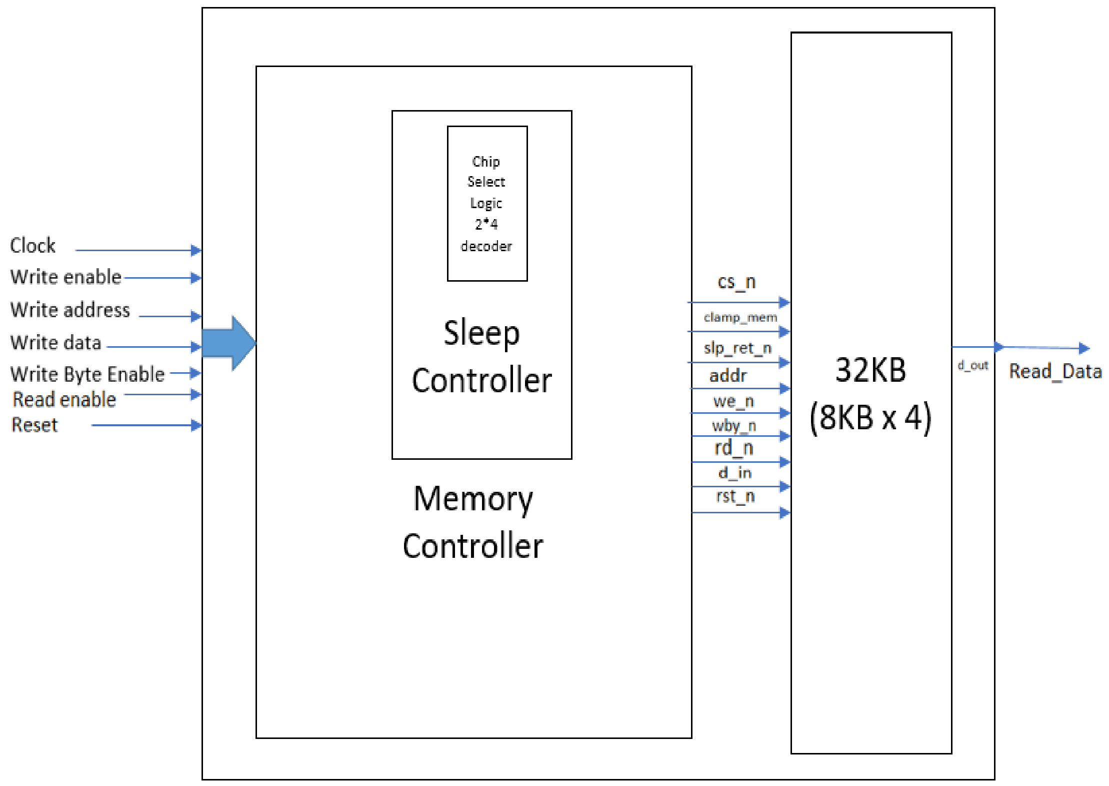

Figure 4.1 Top level block diagram of memory wrapper

4.3 Sleep Mode Finite State Machine

In figure 4.2 we have shown FSM for Sleep mode diagram. Usually memory is in normal mode, when it is doing reading and writing operation. But when memory is idle, this means it is not doing any operation and for that to save the power in the idle mode, we have designed the sleep mode FSM. In this FSM we have used counters, where one is idle counter and other is the state change counter. When memory is in idle state (i.e., no read and write), the idle counter will check for the idle number of clock pulse and if the configurable number of idle clock pulse is reached (default idle number of clock pulse 5 is used), then the memory goes into the sleep mode stages. Sleep mode starts from the state zero (STG0), where it will spend configurable amount of time and then change the state from STG0 to state one (STG1). It will spend the defined amount of time in state one then will move to the STG2 with the help of state change counter. In STG2 sleep retention will be low to retain the data. After this stage wake up state will start and in state three (STG3) it will spend some time and after that in the last state which is dummy state (STG4) it will spend defined amount of time, because after the wake up, first clock cycle will be invalid for read and write operation. After this state idle counter will again check for the idle number of clock pulse and if defined idle clock is reached, then it will remain in the sleep state, otherwise will go into the normal mode and memory again starts the read and write operation.

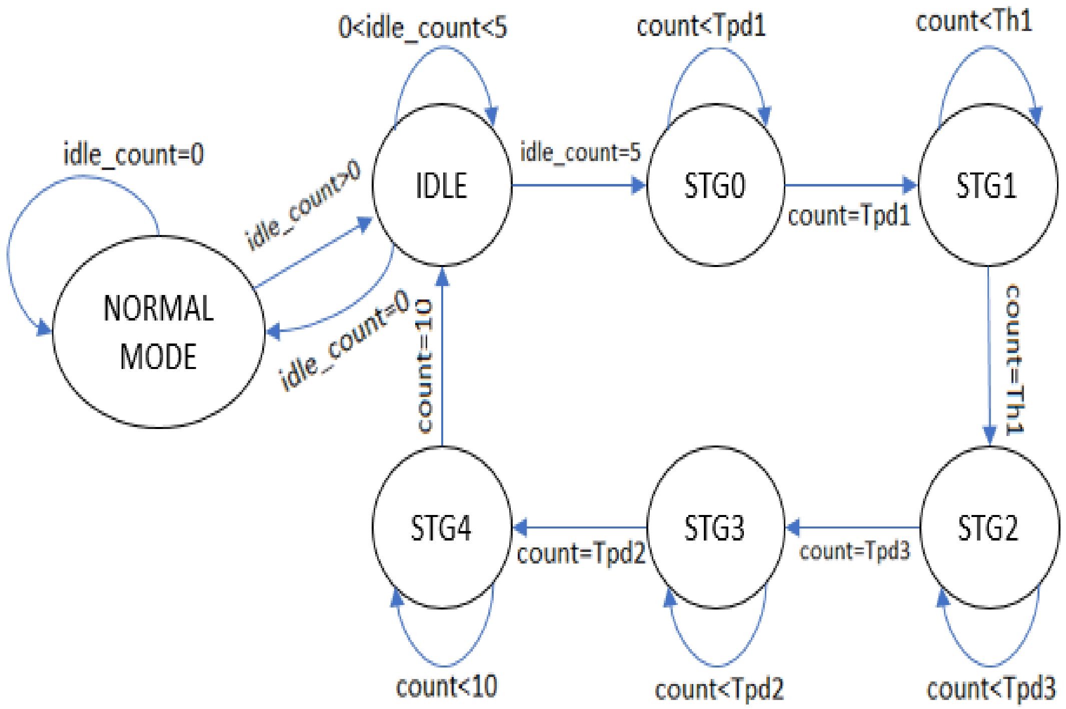

Figure 4.2 Sleep mode FSM

4.4 Different Configuration of LB for 32KB L1 Data Cache Design

4.4.1 Specifications for 32KB L1 Data Cache

For the design of 32KB L1 data cache, in this dissertation we have used 4-way set associative organization and cache line size of 64 bytes and address bit width of 64 bit so total number of cache line require will be 512 (cache line = size of cache memory divided by cache line size) and total number of sets required is 128 (number of sets = cache line divided by K-way set associative). Total address bit is divided into three part, first is offset bit field that will be of size 6 (offset bits = $\log_2$(cache line size)), second is set bit field that will be size of 7 (set bits = $\log_2$(number of sets)) and third one is tag bit field that will be of size 51 (tag bit = total address bits – (offset bits + set bits)).

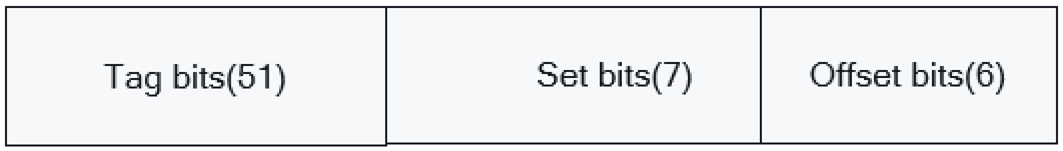

Figure 4.3 address bits

4.4.2 Design 32KB with 2KB LB

To design 32KB memory with 2KB, there will be 16 LB required. We need to get 128 set with 64-byte cache line so each LB will have 8 set with 64-byte cache size. LB selection to be done by the 4 MSB of set bit and least 3 bits of set field will be used to select one of the sets, this job will be done by the decoder. 4*16 decoder is required to select one of the LB and 3*8 decoder is used for selecting one set. After that from the selected set based on the tag bit, selection of one way from four ways will be done. Tag bit is simultaneously comparing are the tags from each way, to do this job in one clock cycle then all the send through the multiplexer and multiplexer chose data from the one of the ways and send it to the processor. From the given address, tag bit is compared with all the tag bit of each way and comparison is done by comparator and then cache hit or miss occurrence is decided. If cache miss occurs, then request will go to the main memory and data will be taken and also replaced in cache memory by write through policy.

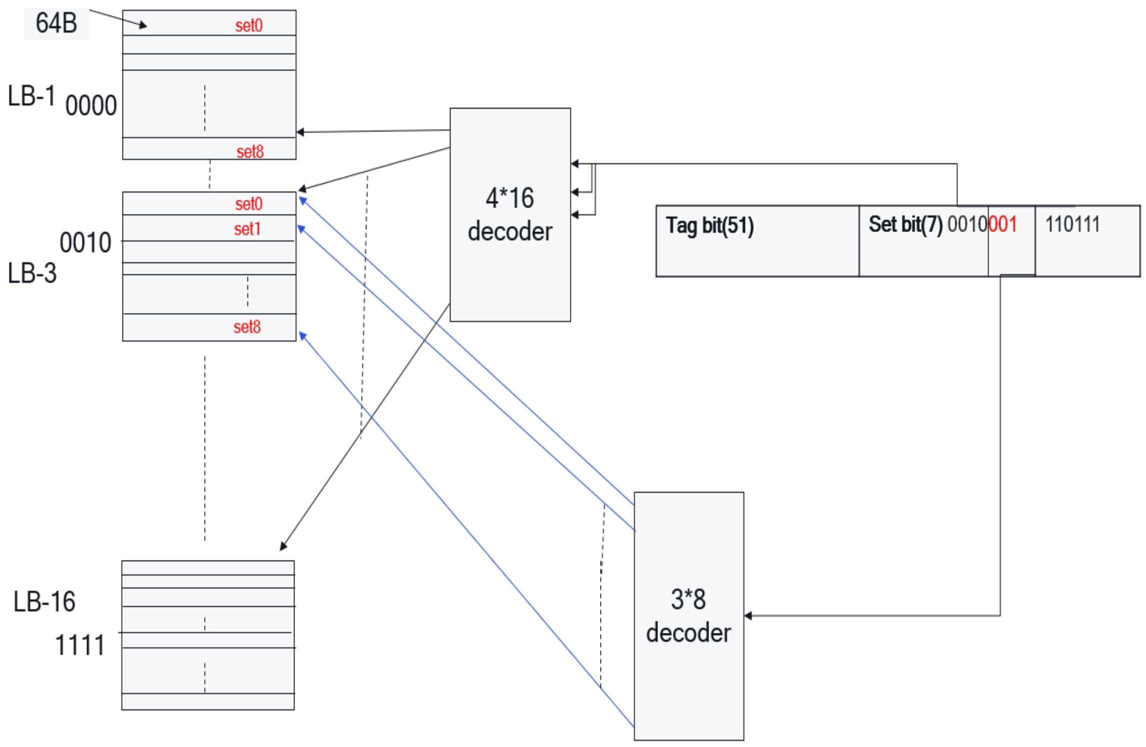

Figure 4.3 2KB logical bank organization

In figure 4.3 we have shown how to logical bank is organized. If the request is generated by the processor, then the address set has set bits 0010001. Based on the 4 MSB bits, LB 3 is selected and by the least 3 bits of sets, set number 1 is selected. Then by figure 4.4 we have shown selection of particular for sending the data. The tag bits from the given address is compared with all ways tag bits and then put in the comparator with the valid bit, where the valid bit represents whether data is present or not. Based on the decision, hit or miss is done and with the help of multiplexer, data is send to the processor.

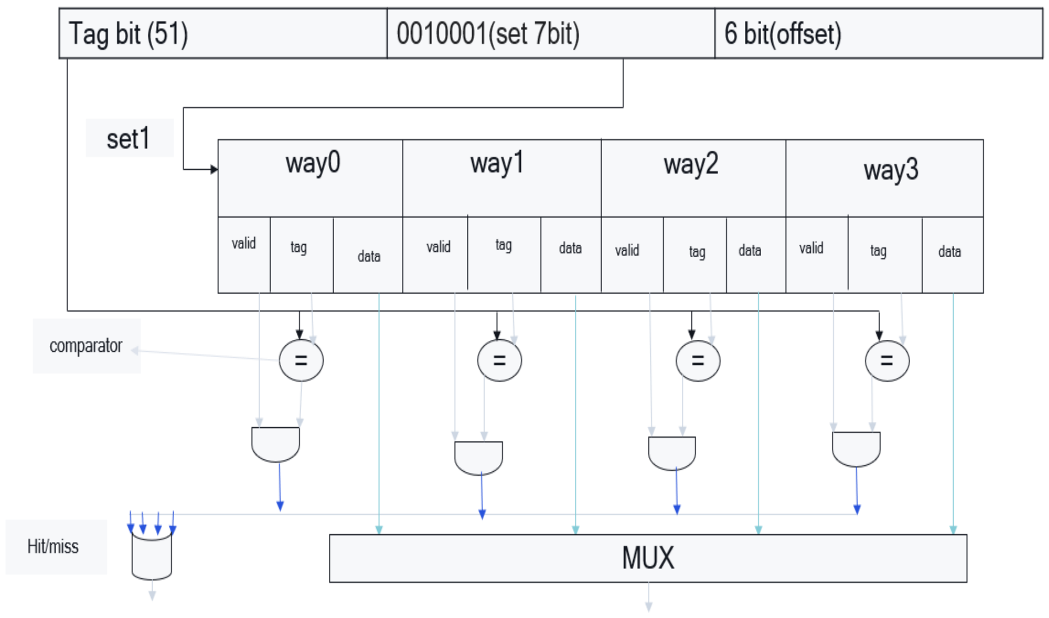

Figure 4.4 Ways representation in the sets for 2KB LB

4.4.3 Design 32KB with 4KB LB

To design 32KB memory with 4KB, 8 LB is required. We need to get 128 set with 64-byte cache line so that each LB will have 16 set with 64-byte cache size. LB selection is done by the 3 MSB bits of set bits and least 4 bits of set field will be used to select one of the sets. Note that this job will be done by the decoder. 3*8 decoder is required to select one of the LB and 4*16 decoder is used for selecting one set. Rest of the procedure is same as we have discussed for the 2KB LB.

In figure 4.5 we have shown a way to organize logical bank of size 4KB for the 32KB cache memory. If the request is generated by the processor to the cache memory and in the address sent by the processor, set field has set bits 1101101. After that based on the 3 MSB bits, LB 7 is selected and by the least 4 bits of set field, set number 7 is selected. Figure 4.6 shows ways inside the particular set and in that we have shown the selection of particular way for sending the data. Also, the tag bits from the given address is compared with all ways tag bits and then put in the comparator with the

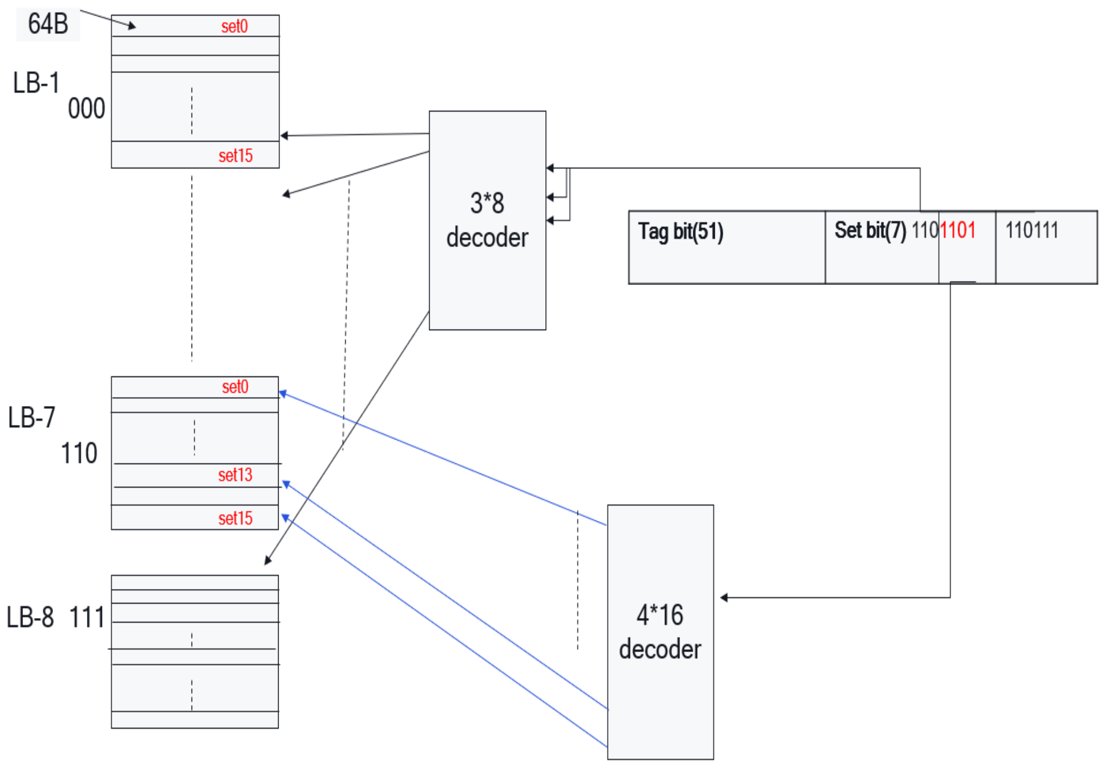

Figure 4.5 4KB logical bank organization

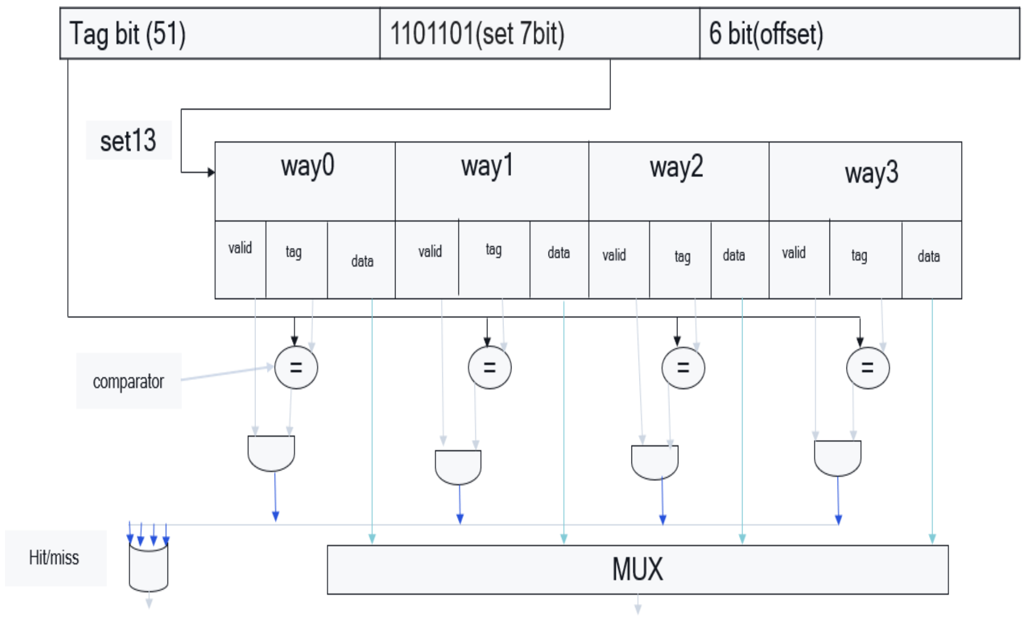

Figure 4.6 Ways representation in the sets for 4KB LB

valid bit, where valid bit represents whether data is present or not. Based on that decision, hit or miss is done and with the help of multiplexer, data is sent to the processor.

4.4.4 Design 32KB with 8KB LB

To design 32KB memory with 8KB, 4 LB is required. We need to get 128 set with 64-byte cache line so that each LB will have 32 set with 64-byte cache size. LB selection is done by the 2 MSB bits of set bits and least 5 bits of set field will be used to select one of the sets. Note that this job will be done by the decoder. 2*4 decoder is required to select one of the LB and 5*32 decoder is used for selecting one set. Rest of the procedure is same as we have discussed for the 2KB LB.

In figure 4.7 we have shown a way to organize logical bank of size 8KB for the 32KB cache memory. If the request is generated by the processor to the cache memory and in the address sent by the processor, set field has set bits 1011000. After that based on the 2 MSB bits, LB 3 is selected and by the least 5 bits of set field, set number 24 is selected. Figure 4.8 shows ways inside the particular set and in that we have shown the selection of particular way for sending the data. Also, the tag bits from the given address is compared with all ways tag bits and then put in the comparator with the valid bit, where valid bit represents whether data is present or not. Based on that decision, hit or miss is done and with the help of multiplexer, data is sent to the processor.

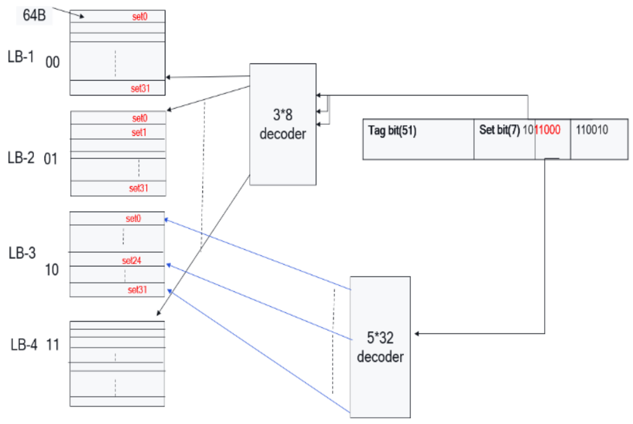

Figure 4.7 8KB logical bank organization

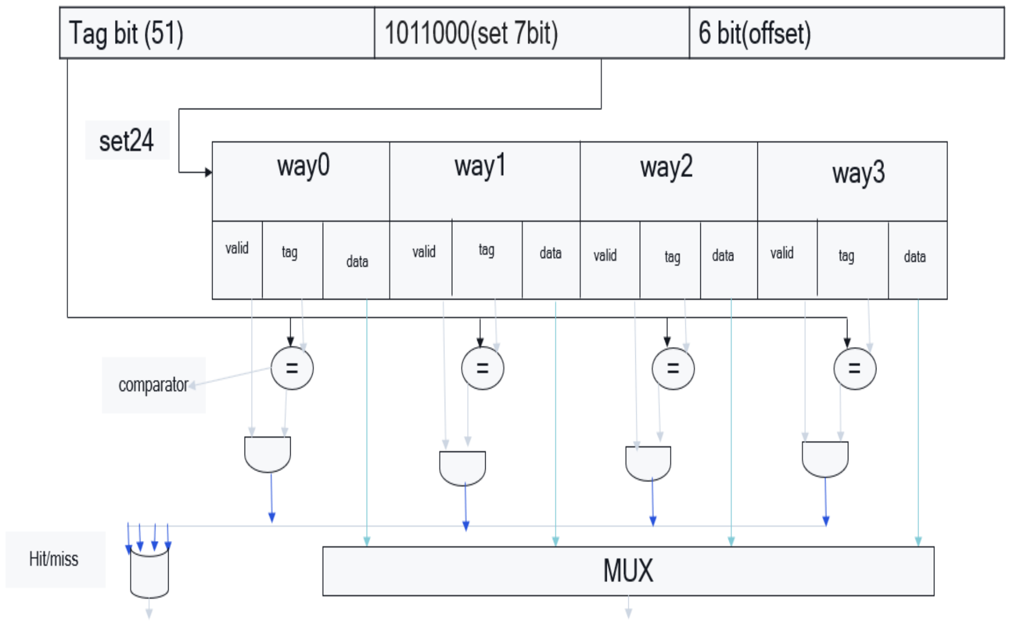

Figure 4.8 Ways representation in the sets for 8KB LB

4.4.5 Design 32KB with 16KB LB

To design 32KB memory with 16KB, 2 LB is required. We need to get 128 set with 64-byte cache line so that each LB will have 64 set with 64-byte cache size. LB selection is done by the 1 MSB bits of set bits and least 6 bits of set field will be used to select one of the sets. Note that this job will be done by the decoder. 1*2 decoder is required to select one of the LB and 6*64 decoder is used for selecting one set. Rest of the procedure is same as we have discussed for the 2KB LB.

In figure 4.9 we have shown a way to organize logical bank of size 16KB for the 32KB cache memory. If the request is generated by the processor to the cache memory and in the address sent by the processor, set field has set bits 1011110. After that based on the 1 MSB bits, LB 2 is selected and by the least 6 bits of set field, set number 30 is selected. Figure 4.10 shows ways inside the particular set and in that we have shown the selection of particular way for sending the data. Also, the tag bits from the given address is compared with all ways tag bits and then put in the comparator with the valid bit, where valid bit represents whether data is present or not. Based on that decision, hit or miss is done and with the help of multiplexer, data is sent to the processor.

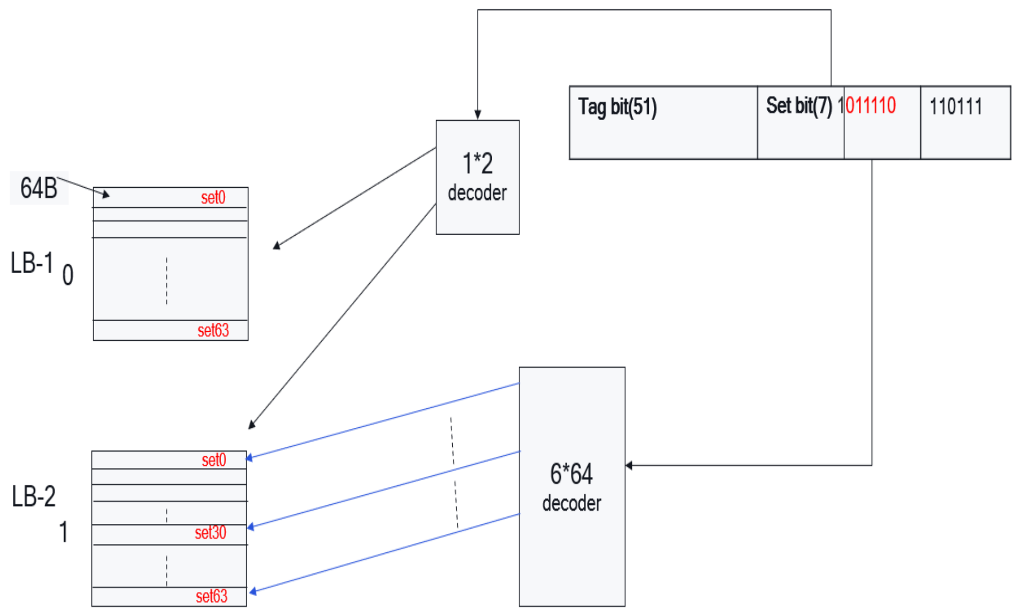

Figure 4.9 16KB logical bank organization

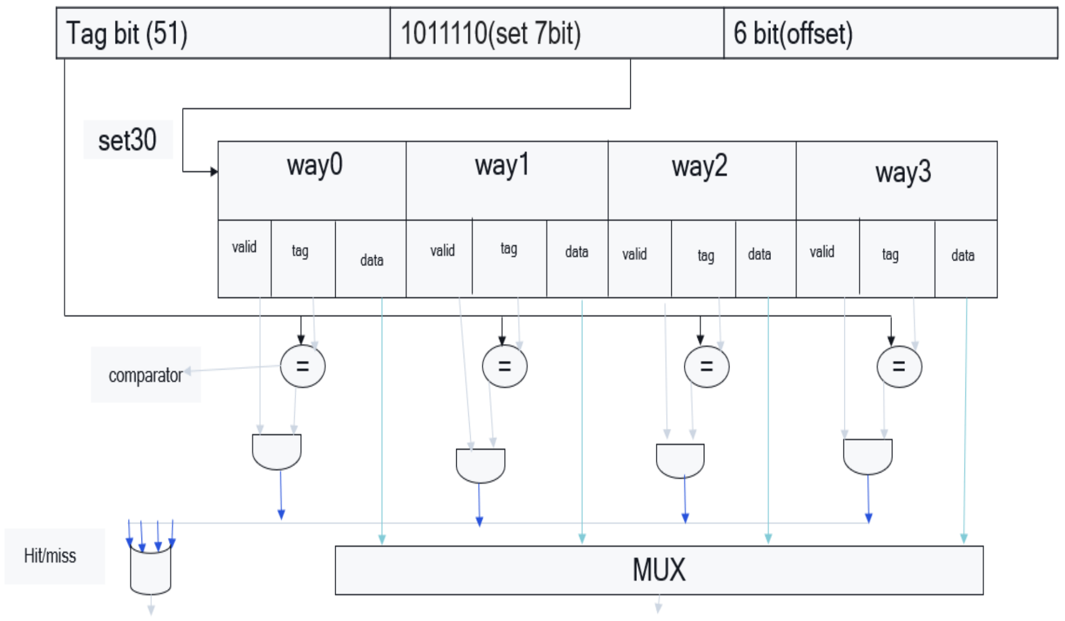

Figure 4.10 Ways representation in the sets for 16KB LB

Chapter 5

Results and Discussions

For the purposes of implementation, any cache memory configuration can be chosen. In this thesis the following standards are considered:

Cache memory 32KB

Cache size 64B

4-way set associativity

Total number of sets 128

Address width 64 bits

Tag field 51 bits

Set field 7 bits

Offset field 6 bits

Clock Frequency 1.8GHz to 2.4GHz

5.1 Simulation Wave from Top level

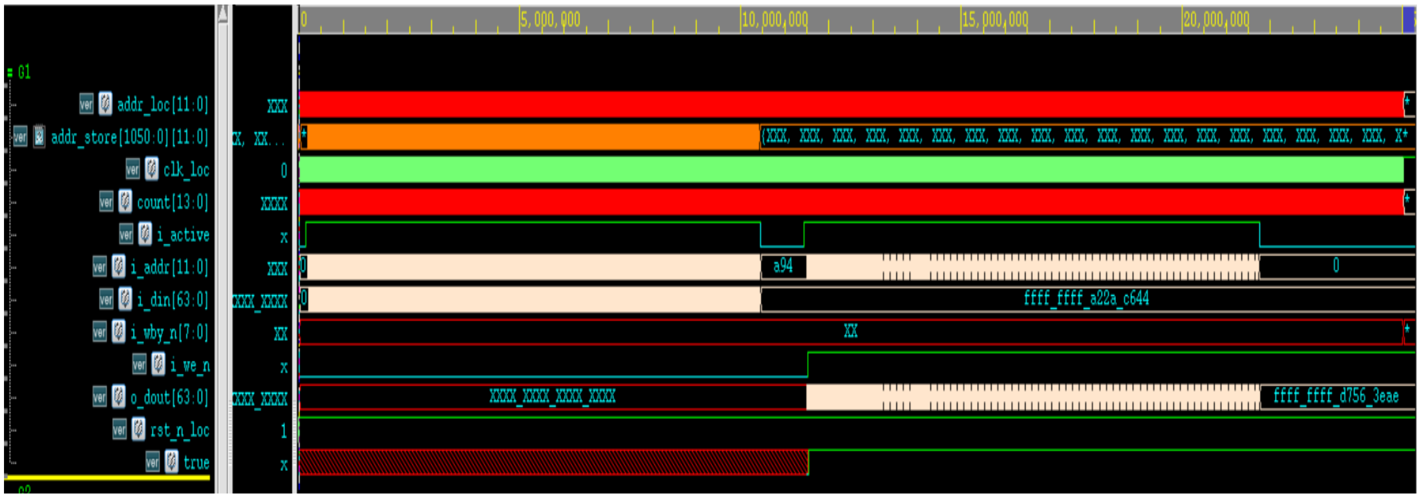

Figure 5.1 Simulation waveform of top-level design of 32KB memory

5.2 Simulation Wave Form of Logical Bank Size of 2KB

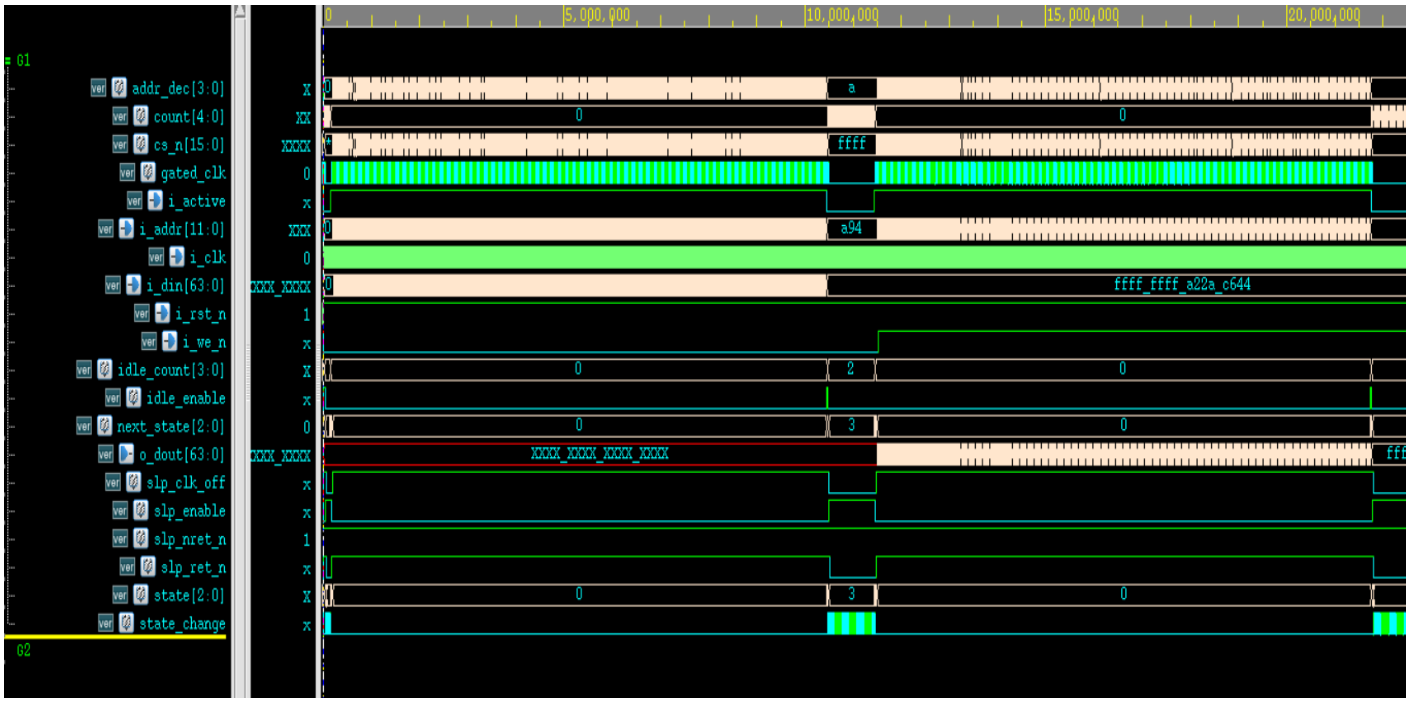

Figure 5.2 Simulation waveform of LB of size 2KB

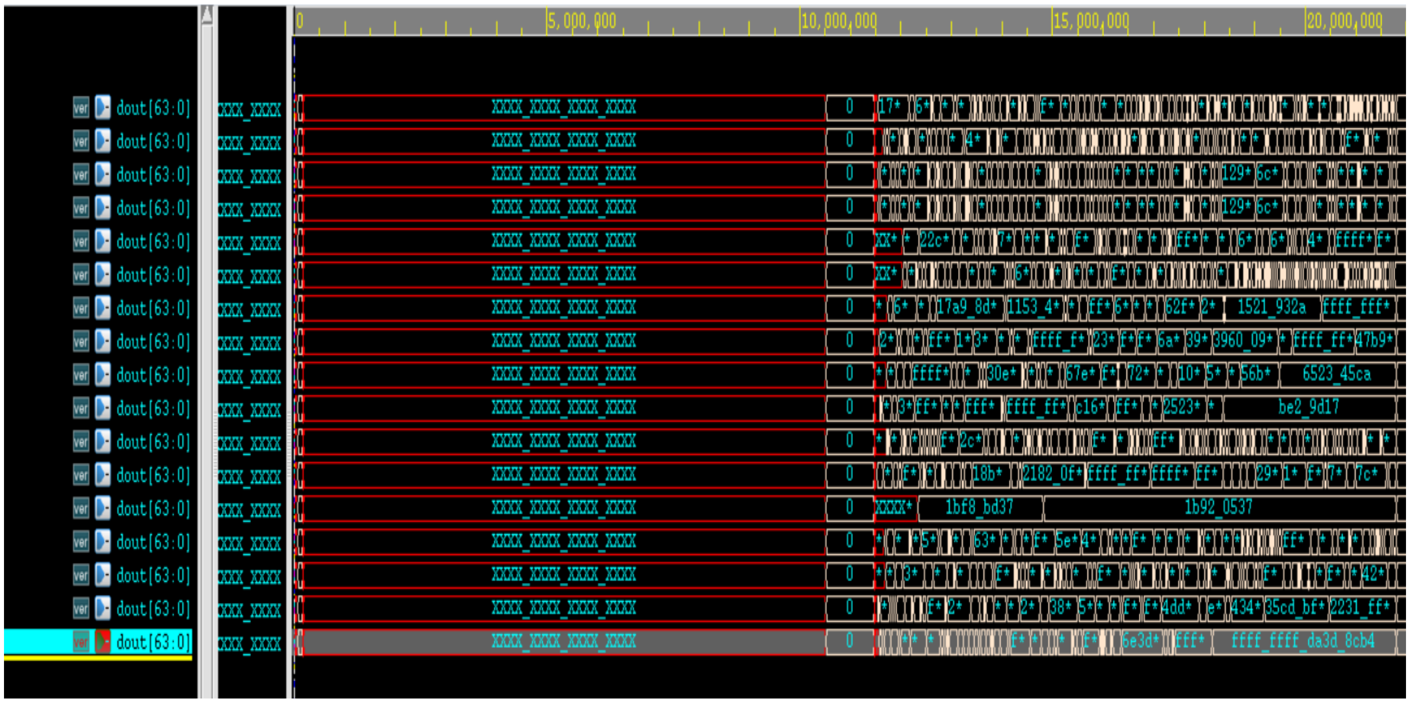

Figure 5.3 Simulation waveform of dout of 2KB each

5.3 Simulation Wave form of Logical Bank Size of 4KB

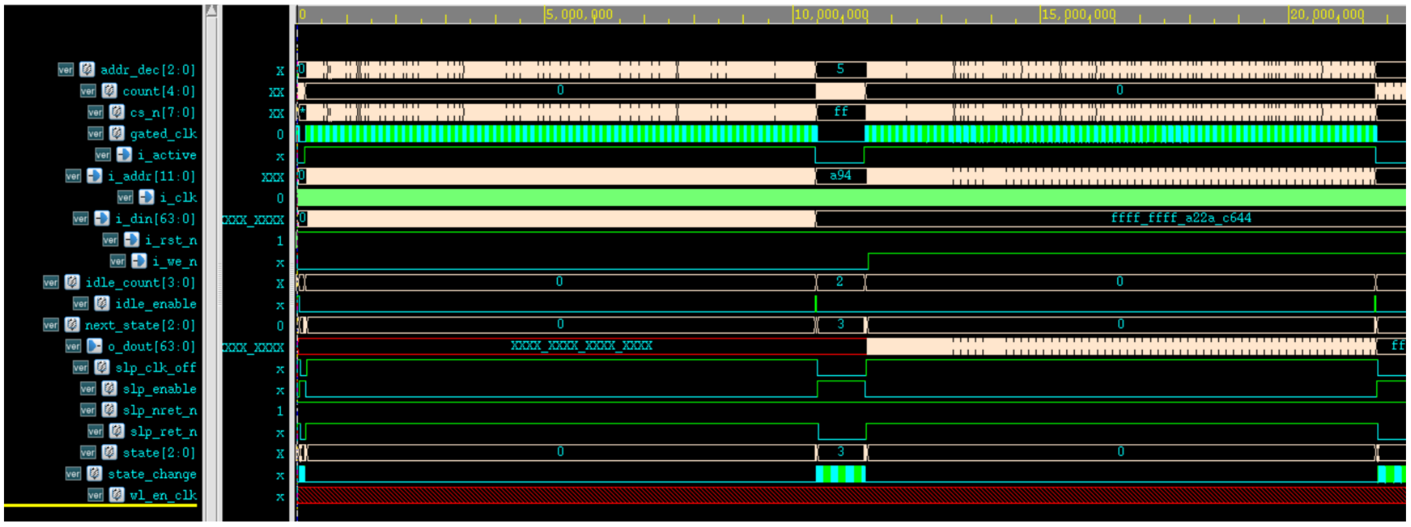

Figure 5.4 Simulation waveform of LB of size 4KB

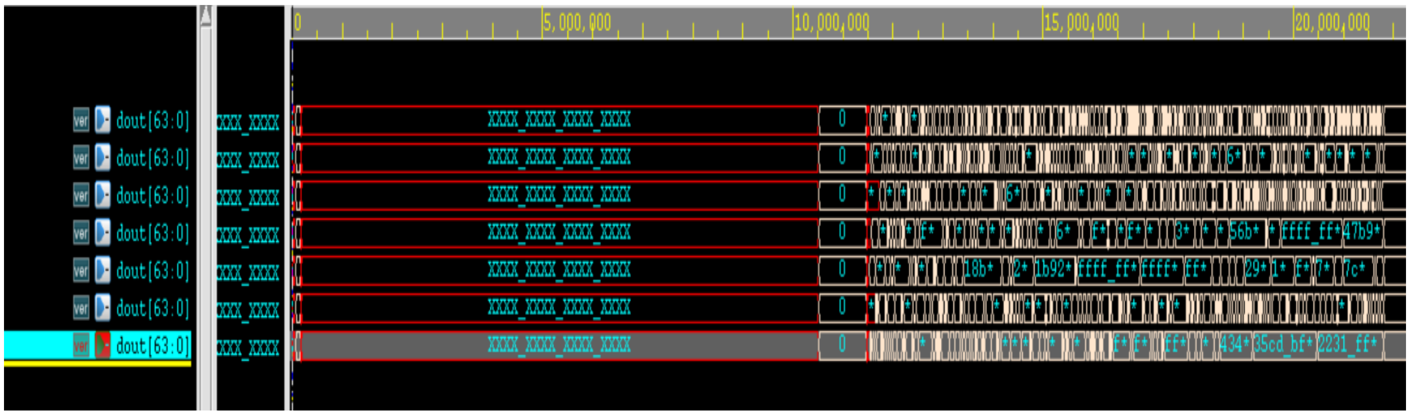

Figure 5.5 Simulation waveform of dout of 4KB each

5.4 Simulation Wave Form of Logical Bank Size of 8KB

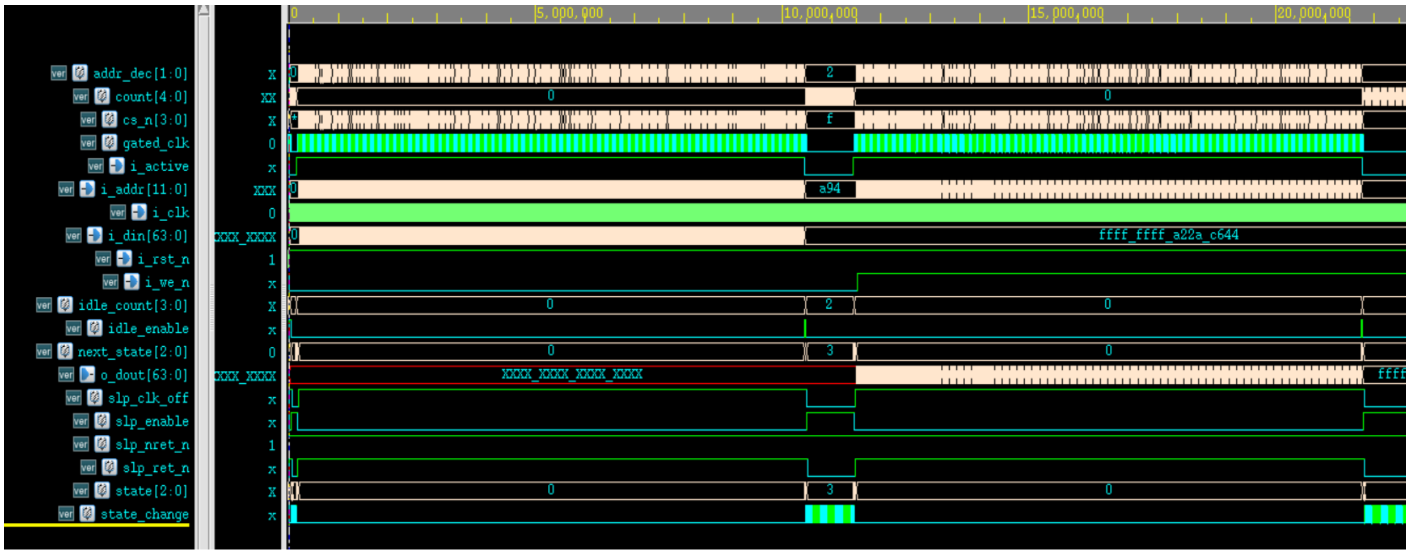

Figure 5.6 Simulation waveform of LB of size 8KB

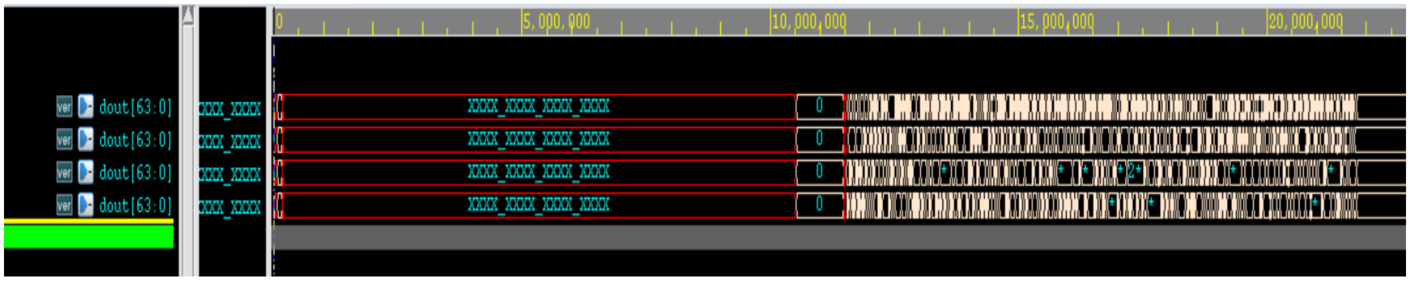

Figure 5.7 Simulation waveform of dout of 8KB each

5.5 Simulation Wave form of Logical Bank Size of 16KB

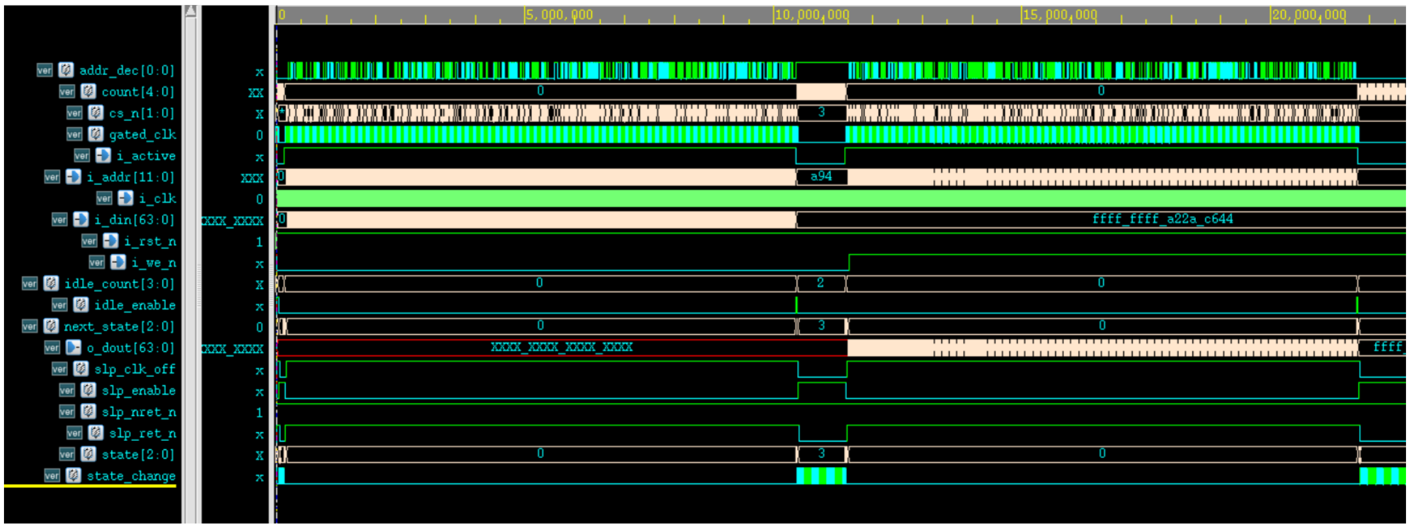

Figure 5.8 Simulation waveform of LB of size 16KB

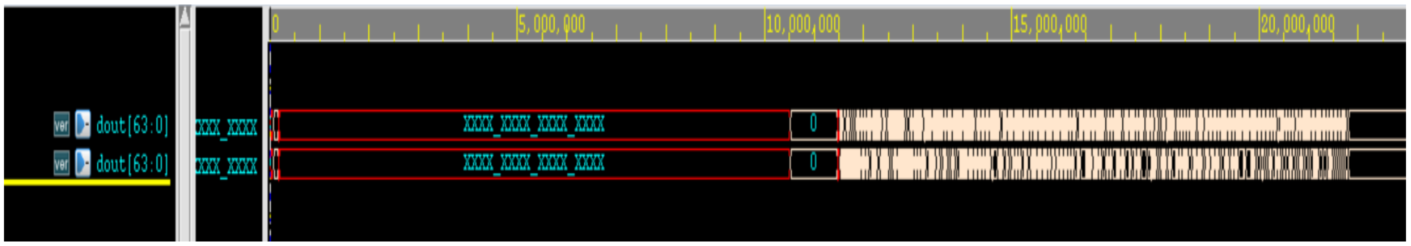

Figure 5.9 Simulation waveform of dout of 16KB each

In the waveform, we have shown input and output from top level, memory level and logical bank. From the top level, we have shown read and write operations to one data input and one data output and another signal active is used to activate logical bank. In each logical banks, the memory wrapper wave form signals are there, which is used in the sleep mode FSM. In the third waveform, read data is taken out from each logical bank.

5.6 Comparison of different configurations based on hardware requirements

Size of Memory (Total size 32KB)	Hardware Requirements		
	No. of Comparator	No. of Decoder + Shared Decoder	No. of Multiplexer
2KB LB	64	4*16+3*8	16(4*1) + 16(8*1)
4KB LB	32	3*8+4*16	8(4*1) + 8(16*1)
8KB LB	16	2*4+5*32	4(4*1) + 4(32*1)
16KB LB	8	1*2+6*64	2(4*1) + 2(64*1)

Table 5.1 Comparison on hardware requirements

In the comparison table of different logical bank based on the hardware requirement, comparison has been provided for each logical bank with respect to number of comparators, decoder and multiplexer used. Based on the comparison, 4KB and 8KB logical bank is found better in comparison to 2KB and 16 KB logical bank to form 32KB memory.

Chapter 6

Conclusions and Future Work

In this thesis detailed analysis on the cache memory is done and we have discussed different configuration of logical bank for 32KB cache memory and a block diagram is also provided for organization of 32KB cache memory. Comparison with all tag of each way is done in one clock cycle. We have designed memory wrapper based on the sleep mode FSM to save power in the idle mode. Further, design of different configuration of cache is also done and then compared them based on the hardware requirement. Based on the comparison, we come into conclusion that 4KB and 8KB logical bank perform well compared to other logical bank.

In future we will analysis different way to set associative mapping with this design to improve the performance and will focus on the miss rate minimization specially because miss rate impacts not only on the execution time but also on optimize power. We will also observe this design by varying the cache size for better performance. After adding all this features, we again compare all the logical bank to form 32KB based on the power and we can more accurately comment on the design of logical bank, which performs the best among all logical banks.

References

[1] C. C. Liu, I. Ganusov, M. Burtscher, S. Tiwari. Bridging the Processor Memory Performance Gap with 3D IC Technology. Design and Test of Computers, IEEE, 22, 556-564, 2005.

[2] C. Carvalho. The Gap between Processor and Memory speeds. In IEEE International Conference on Control and Automation, 2002.

[3] C. Srilatha, C. V. Guru Rao. A Novel Approach for Estimation and Optimization of Memory in Low Power Embedded Systems. International journal of computer theory and engineering, 5, 581-587, 2009.

[4] A. J. Mith. Cache memories. ACM Computing Surveys (CsUR), 1982.

[5] D. A. Patterson, J. L. Hennessy. (1996), Computer Architecture: A Quantitative Approach.

[6] V. P. Heuring and H. F. Jordan. (2004) Computer systems Design and Architecture, 3rd edition, India, Pearson Education.

[7] S. Chakraborty, S. Das, H. K. Kapoor. Performance Constrained Static Energy Reduction Using Way-Sharing Target-Banks. IEEE International Parallel and Distributed Processing Symposium Workshops, 2015.

[8] A. D. Jebaseeli, M. Kiruba. Design of Low Power L2 Cache Architecture Using Partial Way Tag Information. International Conference on Green Computing Communication and Electrical Engineering (ICGCCEE), 2014.

[9] J. Lee and S. Kim. Filter Data Cache: An Energy-Efficient Small L0 Data Cache Architecture Driven by Miss Cost Reduction. IEEE Transactions on Computers, 64, 2015.

[10] A. G. Kumar, D. A. Janeera, M. Ramesh. Power and Performance Efficient Secondary Cache Using Tag Bloom Architecture. International Conference on Electronics and Communication System (lCECS), 2014.

[11] C. H. Ting, J. D. Huang, Y. U. Kao. Cycle-time-aware sequential way-access set- associative cache for low energy consumption. IEEE Asia Pacific Conference on Circuits and Systems,854-857, 2008.

[12] Z. Chuanjun, V. Frank, N. Walid. Energy benefits of a configurable line size cache for embedded systems. IEEE Computer Soc. Annu. Symp. 87-91, 2003.

[13] T. Ishihara, F. Fallah. A Non-Uniform Cache Architecture for Low Power System Design. Int. Symp. on Low Power Electronics and Design, 363 - 368, 2005.

[14] M. Mutyam, C. J. Janraj, T. Warrier, T. V. Kalyan. Way haring et Associative Cache Architecture. 25th International Conference on VLSI Design (VLSID), 251-256, 2012.

[15] C. Zhang, F. Vahid and W. Najjar. A highly configurable cache for low energy embedded systems. In Journal ACM Trans. on Embedded Computing systems. 4, 363-387, 2005.

[16] O. A. Emmanuel, B. Washington and O. A. Michael. Architectural Techniques for Improving the Power Consumption of NoC-Based CMPs: A Case Study of Cache and Network Layer. Journal of Low Power Electronics and Applications. 2017.

[17] S. Gupta, P. Xiang, Y. Yang, H. Zhou. Locality Principle Revisited: A Probability-Based Quantitative Approach. IEEE 26th International Parallel and Distributed Processing Symposium, 2012.

[18] A. Alshegaifi, C. H. Huang. A Locality-Aware, Energy-Efficient Cache Design for Large-Scale Multi-Core Systems. IEEE International Conference on Internet of Things and IEEE Green Computing and Communications and IEEE Cyber, Physical and Social Computing and IEEE Smart Data. 2018.

[19] Chuanjun Zhang. An efficient direct mapped instruction cache for application-specific embedded systems, Third IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'05). 2005.

[20] N. P. Jouppi. Improving Direct-Mapped Cache Performance by The Addition of a Small Fully Associative Cache and Prefetch Buffers. IEEE The 17th Annual International Symposium on Computer Architecture. 1990**.**

[21] Y. Lee, J. Kim, H. Jang, H. Yang, J. Kim, J. Jeong. A fully associative, Tagless DRAM cache. ACM/IEEE 42nd Annual International Symposium on Computer

Architecture (ISCA). 2015.

[22] S Mishra, T. V. Mahendra, A. Dandapat. A Quasi-Static Ternary Fully Associative Cache Tag with Selective Match line Evaluation for Wire Speed Applications. IEEE Transactions on Circuits and Systems I, 63, 2016.

[23] R. E. Kessler, R. Jooss, A. Lebeck. Inexpensive Implementations of Set-Associativity. The 16th Annual International Symposium on Computer Architecture . 2002

[24] C. J. Janraj, T. V. Kalyan, T. Warrier; M, Mutyam. Way Sharing Set Associative Cache Architecture. 25th International Conference on VLSI Design. 2012.

[25] D. Zhang, J. Lei, M. Zhao, X. Gao, Z. Jia. Write-Back Aware Shared Last-Level Cache Management for Hybrid Main Memory. 53nd ACM/EDAC/IEEE Design Automation Conference (DAC). 2016.

[26] P. Guironnet, D. Massas, F. Petrot. Comparison of Memory Write Policies for Noc Based Multicore Cache Coherent Systems. 2008 Design, Automation and Test in Europe.

[27] Y. P. Liang, T. Y. Chen, Y. H. Chang, S. H. Chen, P. Y. Chen, W. K. Shih. Rethinking Last-level-cache Write-back Strategy for MLC STT-RAM Main Memory with Asymmetric Write Energy. IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED). 2019.