Different Access Mechanisms for Set-Associative Cache Architecture for Reduced Power Consumption

Dissertation submitted in the partial fulfilment of requirement for the award of degree of

Master of Technology

in

VLSI Design

Submitted By:

Ankita Pandey

Roll No.: 601161023

Under the Guidance of:

Dr. Sanjay Sharma

Professor

ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT

THAPAR UNIVERSITY

(Established under the section 3 of UGC Act, 1956)

PATIALA – 147004 (PUNJAB)

July-2013

DECLARATION

I hereby declare that the work which is being presented in the dissertation entitled, **"Different Access Mechanisms for Set-Associative Cache Architecture for Reduced Power Consumption"** in partial fulfillment of the requirement for the award of degree of Master of Technology in VLSI Design submitted in Electronics and Communication Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Dr. Sanjay Sharma, Professor, ECED and refers other researcher's work which are duly listed in the reference section.

The matter presented in this dissertation has not been submitted in any other University/Institute for the award of degree.

Date: 15/07/2013

(Ankita Pandey)

Roll No: 601161023

It is certified that the above statement made by the student is correct to the best of my knowledge and belief.

15.7.2013.

(Dr. Sanjay Sharma)

Professor

ECED, Thapar University

Countersigned by:

Head

ECED, Thapar University

Patiala-147004

Dean of Academic Affairs

Thapar University

Patiala- 147004

ACKNOWLEDGEMENT

First of all, I would like to express my gratitude to **Dr. Sanjay Sharma, Professor**, Electronics and Communication Engineering Department, Thapar University, Patiala for his patient guidance and support throughout this report. I am truly very fortunate to have the opportunity to work with him. I found this guidance to be extremely valuable.

I am also thankful to our **Head of the Department, Professor (Dr.) Rajesh Khanna** as well as **PG Coordinator, Dr. Kulbir Singh, Associate Professor,** Electronics and Communication Engineering Department. I would like to thank entire faculty and staff of Electronics and Communication Engineering Department and then friends who devoted their valuable time and helped me in all possible ways towards successful completion of this work. I thank all those who have contributed directly or indirectly to this work.

Ankita Pandey
VLSI Design
601161023
TU, Patiala

TABLE OF CONTENTS

ABSTRACT

In recent years, power consumption has become one of the most critical design concern. Power dissipation by microprocessors is getting larger, which leads to serious problems in terms of allowable temperature and performance improvement. Cache memory is effective in bridging the growing speed gap between a processor and relatively slow external main memory. Today, almost all the commercial processors, not only high performance microprocessors but embedded ones, have on-chip cache memories. The processor accesses the main memory (DRAM) via the cache memory in order to improve the performance of the system. This dissertation demonstrates that cache utilization is an important performance and power-related metric by focusing on the ordering of cache line accesses.

Set-associative cache organization achieves lower miss rates resulting in improved system's performance but at the same time result in significant power dissipation. Hence, In order to achieve low power consumption, novel set-associative cache architectures have been proposed in which the process of accessing the tag sub-arrays and the data sub-arrays associated with the respective ways of four-way set associative cache architecture takes place in two phases unlike the conventional set-associative cache architectures. This paper also proposes a valid bit pre-checking architectural scheme for conventional parallel set-associative cache architecture in order to achieve low power consumption. The behavioral modeling of the architectures was done using Verilog HDL. The simulation and synthesis results illustrate that the proposed phased parallel-access and phased sequential-access set-associative cache architectures showed significant power reduction of 28% and 35.36% on an average as compared to conventional parallel-access and conventional sequential-access set-associative cache architectures. The conventional parallel set-associative architecture with valid bit pre-check technique also showed an average reduction of 19.88% in power consumption over conventional parallel-access set-associative architecture.

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

SRAM	Static Random Access Memory
DRAM	Dynamic Random Access Memory
FIFO	First In First Out
LFU	Least Frequently Used
LRU	Least Recently Used
HDL	Hardware Description Language

CHAPTER 1
INTRODUCTION

1.1 MEMORY

In computing, memory refers to the physical devices used to store programs (sequences of instructions) or data (e.g. program state information) on a temporary or permanent basis for use in a computer or other digital electronic device. The term primary memory is used to denote the information in physical systems which function at high-speed (i.e. RAM), as a distinction from secondary memory. The secondary memory is accessible in the form of mass storage devices such as hard disk, floppy disk storage media, optical storage devices which are slow to access but offer higher capacity. Unlike primary memory, secondary memory is not directly accessed through CPU. The accessing of the primary memory through CPU is done by making use of address and data buses, whereas input/ output channels are used 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.

There are two types of memory accesses:

- *reads* which give the value of an element stored at a defined address,
- *writes w*hich store a value at a given address.

The access could concern only one word or some words, depending on the executed instruction. Thus, the memory can be treated as monolithic block with storage capacity limited by the number of address bits, and with the ability to load and store instructions and data but in the real world, cost, speed, size, power consumption complicate the picture [1].

1.2 MEMORY HIERARCHY

The problem of designing memory system involves three primary parameters, (1) the cost of the memory, (2) the size of the memory, and (3) the access time. Memory devices built using different technologies such as semiconductor (SRAM and DRAM), magnetic and optical storage devices, and tape drives, exhibit different values of the above parameters. Across this spectrum of technologies, the following relationships hold:

- Faster access time, greater cost per bit
- Greater capacity, smaller cost per bit
- Greater capacity, slower access time

It is obvious that the design will aim for a large-sized, high-speed (small access time) memory for a given cost. Therefore, a monolithic memory system built using any one of the above technology will not meet the desired system's requirements. The solution to this problem is offered by *hierarchical memory design*.

The memory system can be modelled as a hierarchy of storage devices with different capacities, costs, and access times. A typical memory hierarchy consists of five levels as shown in figure 1.1. As one goes down the hierarchy, the following occur:

a. Decreasing cost per bit
b. Increasing capacity
c. Increasing access time
d. Decreasing frequency of access of the memory by the processor

Memory hierarchies work because programs tend to access the storage at any particular level more frequently than they access the storage at the next lower level. So the storage at the next level can be slower, and thus larger and cheaper per bit. Thus, smaller, more expensive faster memories are supplemented by larger, cheaper and slower memories. The overall effect is a large pool of memory that costs as much as the cheap storage near the bottom of the hierarchy, but that serves data to programs at the rate of the fast storage near the top of the hierarchy[1][2].

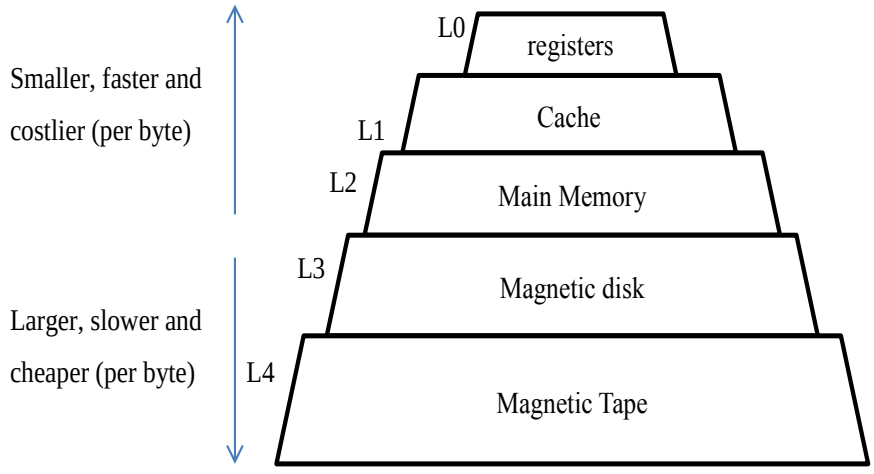


Figure 1.1 The Memory Hierarchy.

Suppose that the processor has access to two levels of memory. Level 1 denoted by L1 contains 1000 words and has an access time of 0.01μs, and level 2 denoted by L2 contains 100,000 words and has an access time of 0.1μs. Assume that the word to be accessed is in level 1, then the processor accesses it directly. If it is in level 2, then the word is first transferred in level 1 and then accessed by the processor. Ignoring, the time required for the processor whether the word is in level 1 or level 2, figure 1.2 shows the general shape of the curve that covers this situation. The figure 1.2 shows the average access time to a two level memory as a function of the Hit rate H, where

H = fraction of all memory accesses that are found in the faster memory (e.g. cache)

T1 = access time to level 1

T2 = access time to level 2

As can be seen, from figure 1.2, for high percentages of level 1 access, the average total access time is much closer to that of level 1 than that of level 2. In this example, suppose 95% of the memory accesses are found in the cache. Then, the average access time to a word can be expressed as:

$$(0.95)(0.01\ \mu s) + (0.05)(0.01\ \mu s + 0.1\ \mu s) = 0.0095 + 0.0055 = 0.015\ \mu s$$

Therefore, the resultant average access time is much closer to 0.01 μs than to 0.1 μs as desired. The use of two levels of memory to reduce access time works in principle, but only if the above conditions (a) through (d) apply.

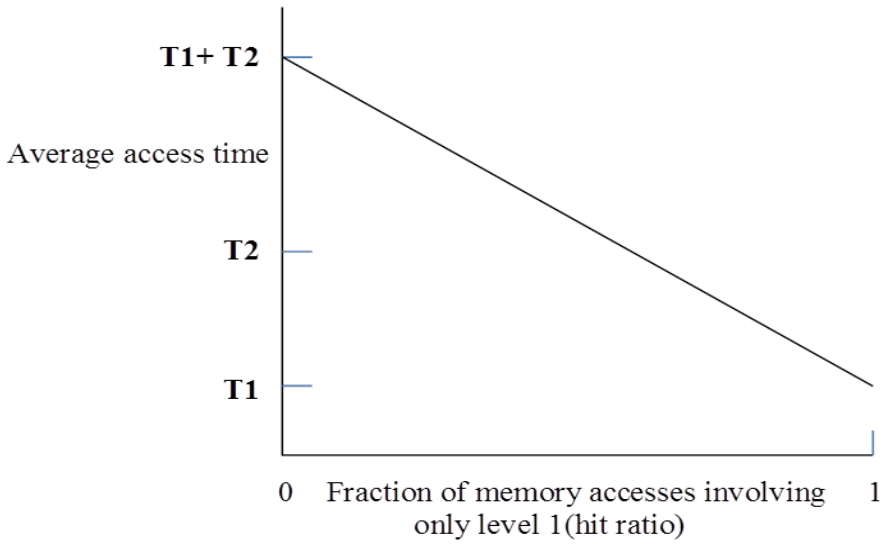


Figure 1.2 Performance of a simple Two-level memory.

1.2.1 LOCALITY OF REFERENCE

The basis of the validity of condition (d) described above is a principle known as locality of reference. During the course of execution of the program, the memory accesses by the processor, for both the instruction and data, tend to cluster. Program typically contains a number of iterative loops and subroutines. Once a loop or subroutine is entered, there are repeated references to a small set of instructions. Over a long period of time, the clusters in use change, but over a short period of time, the processor is primarily working with fixed clusters of memory references.

Accordingly, it is possible to organize data across the hierarchy such that the percentage of accesses to each successively lower level is substantially less than that of the level above. Considering the two-level memory example described above, suppose that the level 2 memory contains all instructions and data. The current clusters can be temporarily placed in level 1. From time to time, one of the cluster from level 1 have to be swapped back to level 2 to make a room for the new cluster coming in to level 1. However, on an average most references will be to instruction and data contained in level 1. The principle can be applied across more than two levels of hierarchy. The principle of locality has two components [2] [14]:

- *Spatial locality of reference* abstracts the tendency of a process to make future references in the neighbourhood of the last reference.
- *Temporal locality of reference* states that there is a tendency of a process to repeatedly refer certain addresses over a small period of time.

1.3 THE NEED OF CACHES

1.3.1 THE PROCESSOR-MEMORY PERFORMANCE GAP

Continuous advances in semiconductor technology have been enabling more powerful and sophisticated processor. The growth in performance for high-performance processor has been improving at roughly 60% per year. However, the speed of the main memory has followed a much lower growth rate of about 10% per year. This has led to what is referred to as the processor-memory performance gap which has been growing with about 40-50% each year since 1980. The performance gap grows exponentially as shown in figure 1.3 and it acts as a primary obstacle to improved system's performance [3][4].

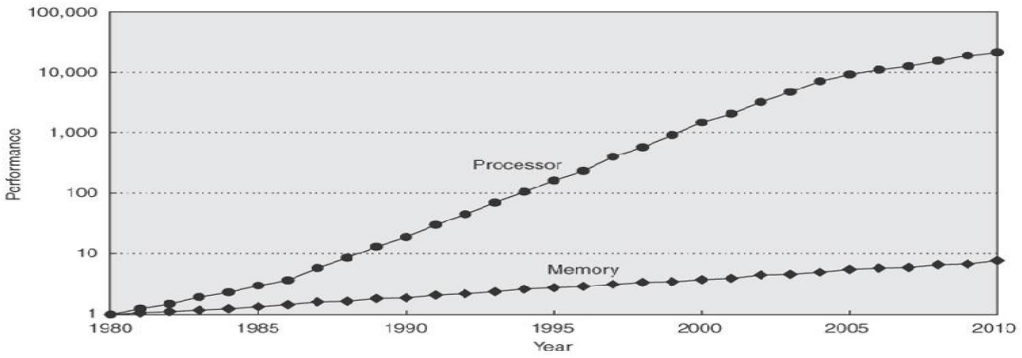


Figure 1.3 The processor-memory performance gap.

Figure 1.4 shows a basic architecture. It consists of a main memory module (DRAM) whose performance is far behind in comparison to the connected processor. The performance of processor cores increases at a faster pace. Processors are generally able to perform operations on operands faster than the access time of large capacity main memory. Though, semiconductor memory which can operate at speeds comparable with the speed of operation of the processor exists, but taking in view the capacity of the main memory, it is not economical to provide the main memory with very high speed semiconductor memory. Thus, the presence of slower memory can indeed be a bottleneck to the performance of the processor and to the system as a whole.

The problem can be alleviated by introducing a small block of high speed memory called a cache (SRAM) between the main memory and the processor as shown in figure 1.5. The program issues effective addresses and read and write requests, and these requests are satisfied by memory. Whether it is the cache (SRAM) or main memory (DRAM) that processes the request is unknown to the program. Instructions and data in cache memories can usually be referenced in 10 to 25 % of the time required to access main memory. Thus, cache memories permit the execution rate of the machine to be substantially increased. As long as most memory accesses are cached memory locations, the average latency of memory accesses will be closer to the cache latency than to the latency of main memory [5][6].

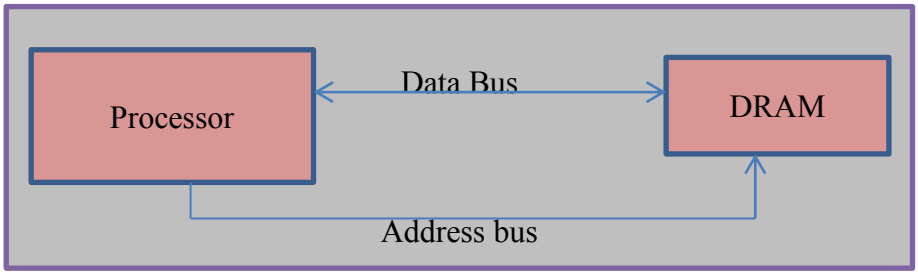


Figure 1.4 Processor and Main Memory (DRAM).

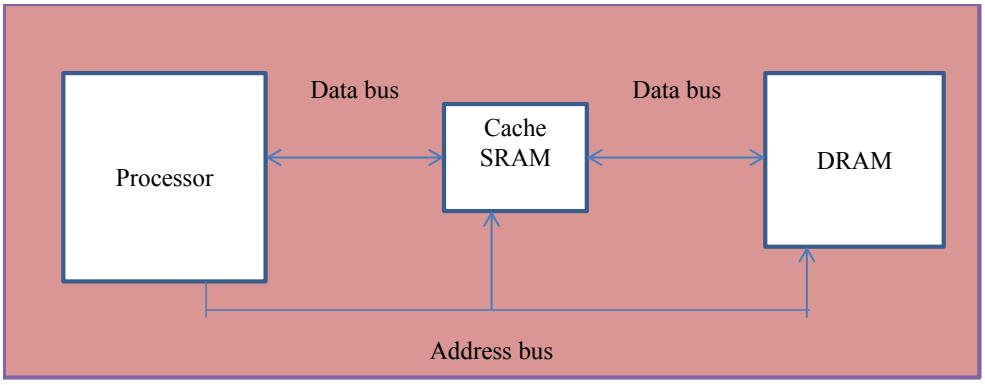


Figure 1.5 Processor, Main memory (DRAM) and Cache (SRAM).

1.4 PROBLEM FORMULATION

Cache memories are employed in embedded architectures to bridge the performance gap which arises due to latency between the processor cycle time and memory access delay. As the speed gap between the memory and CPU continues to increase, modern processors tend to enlarge their caches in order to reduce the number of accesses to long latency memories. For this reason, the caches have been a major consumer of total chip power. In digital and especially in embedded systems because of the portable characteristic of these devices reducing power consumption is one of the most important challenges. Several studies have shown that cache memories account for about 50% of the total power consumed in these systems [7]. On the other hand the temperature of a chip is also severely tied with the magnitude of power dissipation. More power consumption cause to create more temperature effects which consequently lead to increase in cooling cost and degradation of reliability of the system as a whole.

The performance of a given cache architecture is largely determined by the behaviour of the application using that cache. A set-associative cache is commonly used in modern systems for its ability to reduce cache conflict misses which leads to improved system's performance. However, a conventional set-associative cache implementation is not power-efficient as it involves the redundancy of accessing large tag and data sub-arrays which dominate the power consumption of the cache architecture to a large extent. Higher associativity of a set-associative cache will help in reducing the probability of block contention, however, the larger power dissipation will be incurred. Therefore, designing of the cache architecture which can have an optimum performance per power for the processors, will be applicable.

1.5 OBJECTIVE

As applications have become more complex, design of systems employing cache memories have created a research area for low power. Thus, the dissertation embodies the following objectives:

(i) To study the various parameters related to cache memory.

(ii) To study the various factors which contribute significantly to the unnecessary power consumption in cache memory.

(iii) To study various synthesis and simulation tools with the aim of implementing low power cache architectures.

(iv) To implement cache architectures which can give reduced power with optimum performance.

1.6 ORGANIZATION OF THE DISSERTATION

As explained in the previous paragraph, the aim of the work is to implement a low-power, simple, efficient and easy to design set-associative cache architectures. In order to achieve the same, the dissertation has been divided into different chapters:

Chapter 2 gives an overview regarding the work done in the field of cache memories with the aim of achieving power reduction.

Chapter 3 outlines the theoretical background of the cache memory which includes its contribution and importance in enhancing system's performance. The chapter further gives a brief description about the basic principles followed by cache memory for its functionality and how different parameters associated with the cache memory contribute towards the power consumption.

Chapter 4 outlines the proposed architectures of cache memory in order to achieve low power consumption and their comparison with the conventional cache architecture.

Chapter 5 contains simulation and synthesis results obtained using XILINX 8.2i of the base cache architectures and the proposed cache architectures. The results illustrate the functionality, device utilization summaries and the power consumption analysis of the base and the proposed cache architectures. The chapter gives the description about the reasons behind the variation of power among the different cache architectures.

Chapter 6 sums up the final conclusion of the proposed work and suggests some ideas for the future works.

CHAPTER 2

LITERATURE SURVEY

Albonesi, D.H. (1999) [8] The paper proposed selective cache ways which provide the ability to disable a subset of the ways in a set associative cache during periods of modest cache activity, while the full cache may remain operational for more cache-intensive periods. Selective cache ways exploited the fact that cache requirements may vary considerably between applications, as well as during the execution of an individual application. Selective cache ways exploits the sub-array partitioning of set associative caches in order to provide the capability to disable ways of the cache during periods where full cache functionality is not required to achieve good performance. Because this approach leverages the sub-array partitioning that is already present for performance reasons, only minor changes to a conventional cache are required, and therefore, full-speed cache operation can be maintained. The hardware and software aspects of this approach are described and demonstrated that a 40% reduction in overall cache power dissipation can be achieved for 4-way set associative caches with less than 2% overall performance degradation.

Powell, M.D., Agarwal, A., Vijaykumar, T. N., Falsafi, B., Roy, K. (2001) [9] In this paper, two previously-proposed techniques, way-prediction and selective direct-mapping are applied to reducing L1 cache dynamic energy while maintaining high performance. The techniques predict the matching way and probe only the predicted way and not all the ways, achieving power savings. While these techniques were originally proposed to improve set-associative cache access times, this is the first paper to apply them in reducing cache power consumption. The effectiveness of these techniques in reducing L1 d-cache, L1 i-cache, and overall processor power consumption is evaluated. Using these techniques, caches achieve the energy-delay of sequential access while maintaining the performance of parallel access. Relative to parallel access of L1 i-and d-caches, the techniques achieve overall processor energy-delay reduction of 8%, while perfect way-prediction with no performance degradation achieves 10% reduction. The performance degradation of the techniques is less than 3%, compared to an aggressive, 1-cycle, 4-way, parallel access cache.

Yang, J., Gupta, R. (2002) [10] They proposed the design of the Frequent Value Cache(FVC) in which storing a frequent value requires few bits as they are stored in encoded form while

all other values are stored in un-encoded form using 32 bits. With this encoding, the data array part of the cache is split into a smaller array and a larger array. The smaller array holds the short encoded form for frequent values and partial word of the same width for the non-frequent values. The larger array holds the rest part of the word for both frequent and non-frequent values. If a frequent value is accessed only the first data array is accessed, otherwise an additional cycle is needed to access the second data array. Experiments show that on an average FVC provides 28.8% energy reduction for L1 cache and 3.38% increase in execution time delay over a conventional cache.

Zhang, C., Vahid, F., Najjar, W. (2003) [11] They have shown that tuning a cache's line size to a particular program is an extremely effective method of reducing memory access energy for embedded systems. Adding such configurability to a cache architecture is straightforward, and selecting the best configuration can be done fairly simply by embedded system designers. For this reason, the authors have worked on a configurable cache architecture that not only has a configurable line size, but also provide configurable number of ways (while maintaining the same total size). They showed that tuning the line size to a particular program can reduce energy dissipation due to memory access by 50%.

Min, R., Jone, W., Hu, Y. (2004) [12] They proposed a low power cache design, namely phased tag cache, for reducing the power consumption of set-associative caches. In the phased tag cache, the tag is compared in two phases. A small part of the tag is compared in the first phase to determine the data way in which a memory reference falls into. The remaining bits of the tag are compared in the second phase to verify if the result from the first phase is valid. This method eliminates most of the unnecessary activities on the entire tag. Simulation results suggested that the phased tag cache design has small impact on the cache performance. The power model shows the phased tag design reduces the power consumption by 30-50% compared to conventional caches. The phased tag cache design reduces the power consumption of set-associative caches.

Chen, H.C., Chiang, J.S. (2005) [13] The proposed Way predicting cache using valid bit pre-decision architectural scheme performs its way prediction based on the original MRU(Most recently used) bits plus valid bits. The technique uses the valid bits from data memory to pre-decide the disabling of unnecessary tag sub-arrays and data sub-arrays. In the architecture of the way-predicting cache, they have used a way-predictor and a MRU table, where the MRU table records the information about the most recently used block for each set in a cache, and

the way predictor can maintain the status of the MRU table and send the enable signals to all tag sub-arrays and data sub-arrays. Therefore, for a set-associative cache with sub-block placement, when the cache is accessed, in additional to tag checking of all ways, the corresponding valid bits of all ways must be checked together. In this paper, focusing on the way-predicting cache with sub-block placement, fortunately, the valid bits from the data sub-arrays can be used to pre-eliminate the unnecessary probes at each cache access. Moreover, it can make the original rest enabled ways at the second cycle become the first-cycle access if the valid bit of the MRU way does not exist. The proposed way-predicting cache can be applied to the parallel architecture. The way-predictor decides which ways with tag sub-arrays and data sub-arrays should be disabled according as their corresponding valid bits are "0". WPD-V cache still has an about 18% improved rate in average power dissipation.

Kim, C.H., Chung, S.W., Jhon, C.S. (2006) [14] They proposed a new instruction cache architecture, named Partitioned Power-aware instruction cache (PP-cache), for reducing dynamic power consumption in the instruction cache by partitioning it to small sub-caches. When a request comes into the PP-cache, only one sub-cache is accessed by utilizing the locality of applications. In the meantime, the other sub-caches are not activated. The proposed PP-cache reduces dynamic power consumption by reducing the activated cache size and eliminating the power consumed in tag matching. The simulation results show that the PP-cache reduces dynamic power consumption by 34–56%.

Ting, C.H., Huang, J.D., Kao, Y.U. (2008) [15] This paper exploited the concept of sequential way access to reduce the number of ways being activated on each access of set-associative cache for low energy consumption while maintaining performance. The proposed architecture accesses each way in sequence, and then eliminates subsequent accesses if a hit is detected. The proposed scheme reduced the heavy fan out load of the hit signal, which suppresses the possible increase of cache cycle time due to more complicated cache control mechanism. The experimental results show that a 32KB 2-way sequential way access set-associative cache reduces the energy consumption by 24% compared against a conventional 2-way set-associative cache with the same size at virtually no performance loss.

Tseng, C.Y., Chen, H.C. (2009) [16] In this paper, the power reduction is achieved by using MRU (Most Recently Used*)* table which is used to record most recently used block for each index to improve cache hit rate and which would help in reducing the number of tag comparisons, and increase set-associative hit rate. The paper also described and used

(Modified Pseudo) MPLRU algorithm which reduces the number of tag comparison and also reduced the hardware complexity and cache miss rate. The experimental results obtained demonstrated that the prediction hit rate reach 90.15%, thus save 64.12% energy.

Xu, C., Zhang, G., Hao, S. (2009) [17] They introduced a new way-prediction scheme for achieving low power consumption and high performance for set-associative instruction cache. The proposed scheme is based on the different cases of how program execution proceeds. By predicting for the sequential instruction flow and non-sequential one respectively, high way prediction hit rate was achieved. Since this way-prediction doesn't increase the cache access time, it is referred as Fast way-prediction cache (FWPC). The experimental results came from FPGA showed that the proposed FWPC has a very high way prediction hit rate, which is 97.932% on average. At the same time, the energy dissipation of instruction cache is reduced by 64.83% compared to conventional cache at the cost of 0.2% performance penalty, in which the way prediction miss only cost 0.38% energy.

Zhang, C., Vahid, F., Najjar, W. (2005) [18] The cache has three software-configurable parameters that can be tuned to particular applications. First, the cache's associativity can be configured to be direct-mapped, two-way, or four-way set-associative, using a novel technique called way concatenation. Second, the cache's total size can be configured by shutting down ways. Finally, the cache's line size can be configured to have 16, 32, or 64 bytes. In this paper, configurable cache tuned to each program saved energy for every program compared to a conventional four-way set-associative cache as well as compared to a conventional direct-mapped cache, with an average energy savings related to memory access of over 40%.

Ghosh, M., Ozer, E., Ford, S., Biles, S., Lee, H.S. (2009) [19] This paper presented an efficient use of the counting segmented Bloom Filter called Way Guard to reduce significant dynamic cache power by filtering out unnecessary way lookup in a set-associative cache. Even though the scheme incurs extra hardware, the Way Guard only comprises of a bit-vector and counters. Querying a Way Guard only involves checking the bit vector and this consumes much less energy than looking for the address in the Tag RAM it is guarding. Since, the filter must be checked prior to the Tag RAMs being selected, there is a potential performance penalty. However, since the filter is fast, the filter access and the Tag RAM selection process can be potentially contained within a cycle. In the worst case, the filter would add one extra cycle to the cache access time. The proposed technique can be efficiently applied to all levels

of the cache hierarchy, obtaining substantial energy savings of up to 70% in both instruction and data L1 caches, and up to 65% for a unified L2 cache.

Zhang, C., Vahid, R., Yang, J., Najjar, W. (2005) [20] They proposed a way-halting cache which is a four-way set-associative cache that stores the four lowest-order bits of the tags of all ways into a fully associative memory, called the halt tag array. The lookup in the halt tag array is done in parallel with, and is no slower than, the set-index decoding. The halt tag array predetermines which tags cannot match due to their low-order 4 bits mismatching. The technique provided 55% savings of memory-access related energy consumption on an average over a conventional four-way set-associative cache are obtained, while imposing no performance overhead and only 2% cache area overhead.

Janraj, C.J., Kalyan, T.V., Warrier, T., Mutyam, M. (2012) [21] They proposed way-sharing cache architecture, wherein two cache sets share some cache ways so that highly demanded sets can get more cache ways than that of under-utilized sets. In conventional n-way set-associative caches, irrespective of the set-wise demand, each set has n-cache ways at its disposal, but cache sets may exhibit non uniform demand for these cache ways. Exploiting this property way sharing cache architecture has been proposed where in by allowing sharing of cache ways among a pair of cache sets, dynamic power savings as high as 41% with negligible performance penalty is achieved.

Ishihara, T., Fallah, F. (2005) [22] They proposed a non-uniform cache architecture for reducing the power consumption of memory systems. The non-uniform cache allows different associativity values (i.e. the number of cache-ways) for different cache-sets. An algorithm determines the optimum number of cache-ways for each cache-set and generates object code suitable for the non-uniform cache memory. The paper also proposed a compiler technique for reducing redundant cache-way accesses and cache-tag accesses. Experiments demonstrate that the technique can reduce the power consumption of memory systems by up to 76% compared the conventional method.

Gu, J., Guo, H., Li, P. (2011) [23] Aiming at reduced power consumption of the on-chip cache, a Reduced One-Bit Tag Instruction Cache (ROBTIC) is proposed in this paper, where the cache size is judiciously reduced and the cache tag field only contains the least significant bit of the full-tag. A cache operational control scheme for ROBTIC is developed so that with the one-bit cache tag, the program locality can still be efficiently exploited. For applications

where most of the memory accesses are localized, the proposed cache can achieve similar performance as a traditional full-tag cache, however, the power consumption of the cache can be significantly reduced due to the much smaller cache size, narrower tag array (just one bit), and tinier tag comparison circuit being used. The proposed design can reduce up to 27.3% dynamic power consumption and 30.9% area of the traditional cache. With the cache size customization, a further 47.8% power saving can be achieved.

CHAPTER 3

GENERALITIES ABOUT CACHE

3.1 BASIC PRINCIPLES OF CACHE MEMORY

Cache is the name given to the highest or first level of the memory hierarchy encountered once the address leaves the processor. Cache memories are small, high speed buffer memories used in modern computer systems to hold temporarily those portions of the contents of the main memory which are believed to be currently in use.

Traffic to and from the processor and the cache is in the form of *words*. In computing, word is a term for the unit of data used by a particular processor design. A word is basically a fixed-sized group of bits. Modern processors, including embedded systems, usually have a word size of 8, 16, 32 and 64 bits. Traffic between the cache and main memory is in the form of blocks. These cache blocks are referred to as *cache lines* discussed later. The mode of transfer of information among the processor, main memory and cache is shown in the figure 3.1.

The idea of cache memory is similar to virtual memory with the respect, that some active portion of a low-speed memory is stored in a higher-speed cache memory. The difference between cache and virtual memory is a matter of implementation. The two notions are conceptually the same because they both rely on the correlation properties observed in sequences of address references but their implementations are totally different because of the speed requirements of cache. When a memory request is generated, the request is first presented to the cache memory. A check is made to determine whether the word is present in the cache, if so the word is delivered to the processor. If not, a block of main memory consisting of some fixed number of words is read into the cache and then the word is delivered to the processor. The principle of locality of reference allows the cache memory to be effective by storing only the subset of the main memory (the subset being recently used data and instructions). Because of this phenomenon of locality of reference, when a block of data is fetched in to the cache to satisfy a single memory reference, it is likely that there will be future references to that same memory location or to other words in the block. Thus, cache stores the recently accessed data so that the future requests for the respective data can be served faster A typical cache organization is shown in figure 3.2 [2][6].

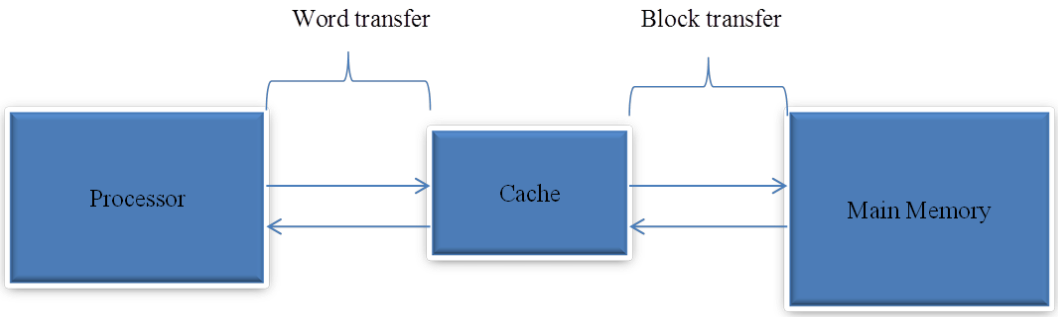


Figure 3.1 Cache memory Principles.

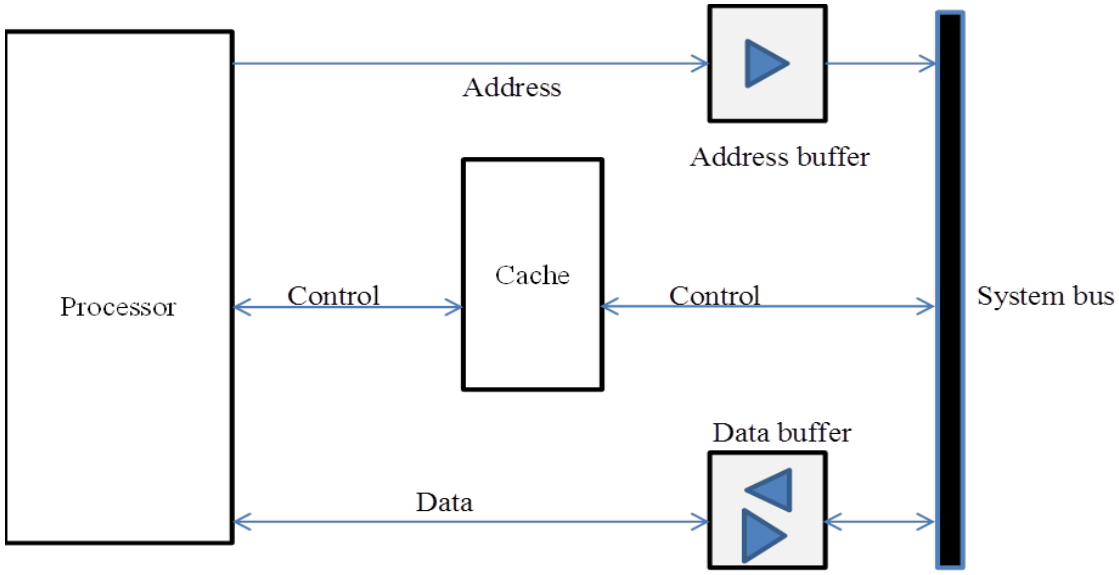


Figure 3.2 Typical Cache Organization.

3.1.1 UNIFIED VERSUS SPLIT CACHES

If there is a single cache at a given level that holds both data and instructions, then it is called as the *unified cache*. If the cache at a given level is split into two caches, one for storing the instructions and another for storing the data, then this approach is called as the *split cache*. For a given overall size, a unified cache is more flexible due to the following reasons:

- It offers higher hit ratio than the split caches. The reason being the working set of a program may have a larger fraction of data than instructions or vice versa. A unified cache can flexibly accommodate either data or instruction.
- Only one cache needs to be design and implemented.

The key advantage of the split cache is that it eliminates contention for the cache between the instruction fetch/decode unit and the execution unit. Figures 3.3 (a) and 3.3(b) give an overview of the split and unified cache architectures respectively [1][24].

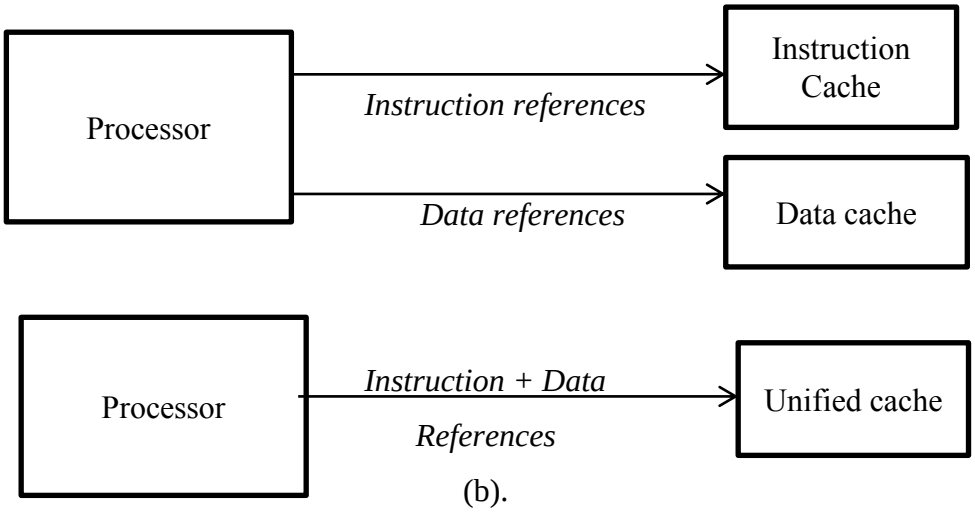


Figure 3.3 Two scenarios for memory hierarchy.

(a). Split cache (b). Unified Cache

3.1.2 REPLACEMENT POLICIES

Cache replacement policy is a major design parameter of any memory hierarchy. The efficiency of the replacement policy affects both the hit rate and the access latency of a cache system. In order to make room for the new entry on a cache miss, the cache may have to evict one of the existing entries. The heuristic that it uses to choose the entry to evict is called the replacement policy. The fundamental problem with any replacement policy is that it must predict which existing cache entry is least likely to be used in the future but predicting the future is difficult, so there is no perfect way to choose among the variety of replacement policies available.

One popular replacement policy, least-recently used (LRU), replaces the least recently accessed entry. Marking some memory ranges as non-cacheable can improve performance, by avoiding caching of memory regions that are rarely re-accessed. First in First Out (FIFO) is also one of the replacement policy used when the number of blocks to keep track increases. In this case LRU becomes complicated to implement. The FIFO scheme eliminates the oldest block from cache. The least frequently used (LFU) replacement policy keeps track of the frequency of access of the different blocks that are resident in the cache and chooses for replacement the block with the least frequency of access. Though the access frequency of a block is a better indicator of the usefulness of a block, the disadvantage of LFU algorithm is that it may replace a newly brought block because its frequency of access is small while the

block may be needed in the future. Obviously, a good replacement should choose a block that is unlikely to be referred in future [2][25].

3.1.3 DIFFERENT TYPES OF MISSES

If the data or instruction requested by the processor is contained in the cache, it is called a *cache hit*, otherwise the situation is termed as *cache miss*. In case of a miss, the data has to be fetched from its original storage location, which results in longer execution time. The various types of misses of the cache memory can be categorized as:

- *Compulsory misses*: These are the misses on a cold start. Data must be fetched at least once from the upper level memory to be present in the cache.
- *Capacity misses*: These misses occur when the working set (data/instructions required for the program) exceeds the cache size.
- *Conflict misses*: Such misses result from the mapping of two different items to the same cache line.
- *Coherency misses*: In a multiprocessing environment, the coherency protocol may invalidate a line. The next lookup for this line or block will be a miss.

In the interests of completeness, the fourth C has since been introduced when dealing with cache-coherent multiprocessor systems [26].

3.1.4 CACHE READ AND WRITE POLICIES

When the processor makes a memory reference, it may be for a memory read or for a memory write. In each of these two cases there can be a hit or a miss in the cache. When a data is located in the cache, the system has two copies: one in the main memory, one in the cache.

- *Cache hit on a memory read:* This is the simplest of all the cases. The referred data is accessed from the cache and delivered to the processor. There is no need to update main memory on the cache read operation.
- *Cache miss on a memory read:* For read operations that cause a cache miss, the item is retrieved from main memory and copied into the cache. During the short period available before the main-memory operation is complete, some other item in cache is removed to make room for the new item.

The process of read hit and read miss on accessing the cache memory are diagramatically depicted by figure 3.4 and figure 3.5 [2][27].

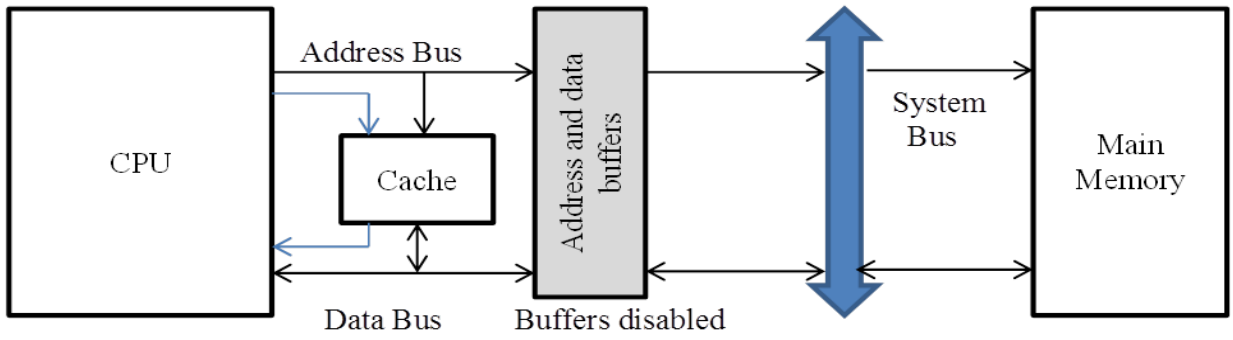


Figure 3.4 Read hit.

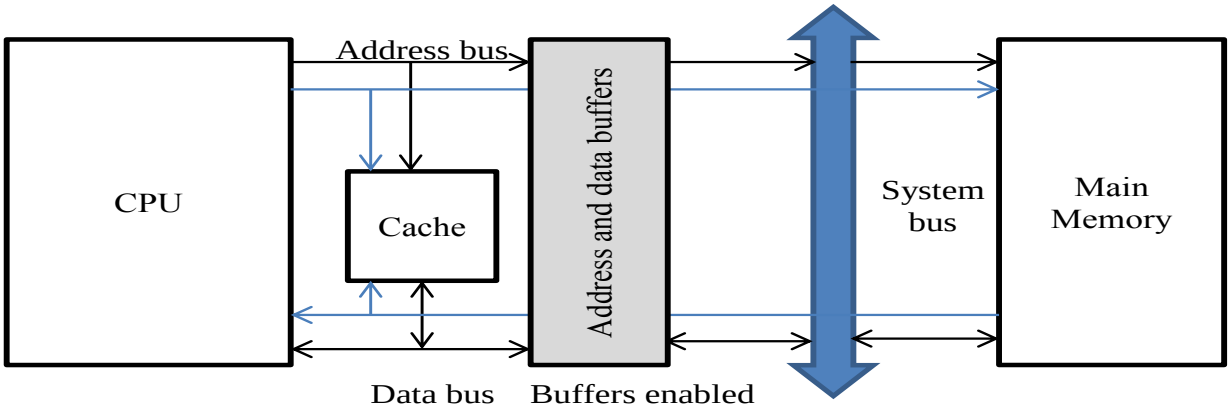


Figure 3.5 Read miss.

However, in general, writing can occur to cache words and it is possible that the cache word and its corresponding copy held in the main memory may be different. It is therefore, necessary to keep the cache and the main memory copy identical. Two different policies govern the write:

- *write through:* the data/instruction is written both in the cache and in the main memory. The copy of data in the main memory is never different from that of the copy in the cache. This will be a good strategy, if there are only few writes to a given block, but it will be disadvantageous if there are multiple writes.
- *write back:* the information is written in the main memory only when the line is removed from the cache. To avoid useless back writings in the main memory, the cache lines are provided with a dirty bit. If it is set to 1, it means that the data/instruction has been modified in the cache. Otherwise, the data is clean and does not need to be written back in the main memory on eviction. The concept of dirty bit is shown in figure 3.6.

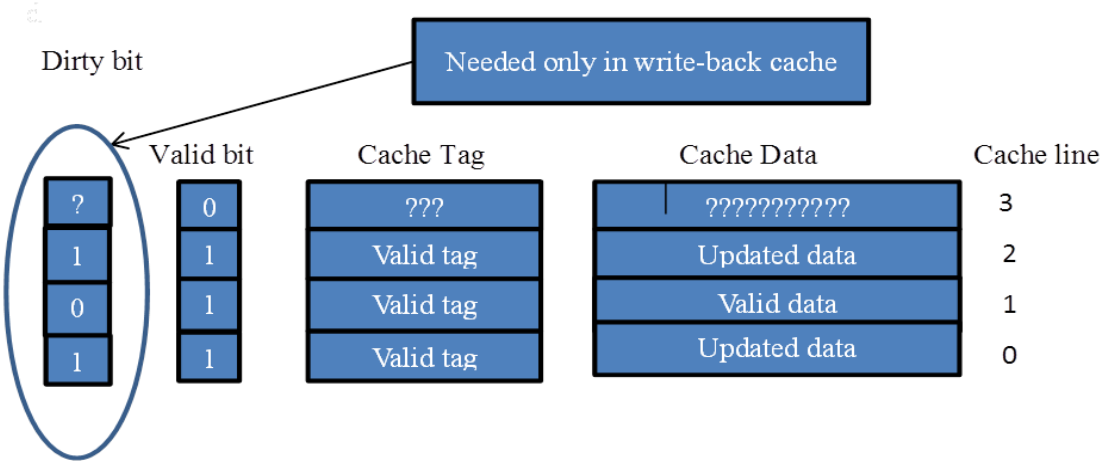


Figure 3.6 Concept of dirty-bit in write-back cache.

The main problem with the write through policy that it generates a lot of unnecessary traffic. It leads to a situation where the processor may have to wait for the write buffer to perform the writes but with this policy, the main memory has always an up-to-date copy of the data/instruction, which simplifies coherency [2]. The process of write miss is depicted by figure 3.7. The concepts of write-back and write-through on a write hit are shown by figures 3.8 and 3.9 respectively [27].

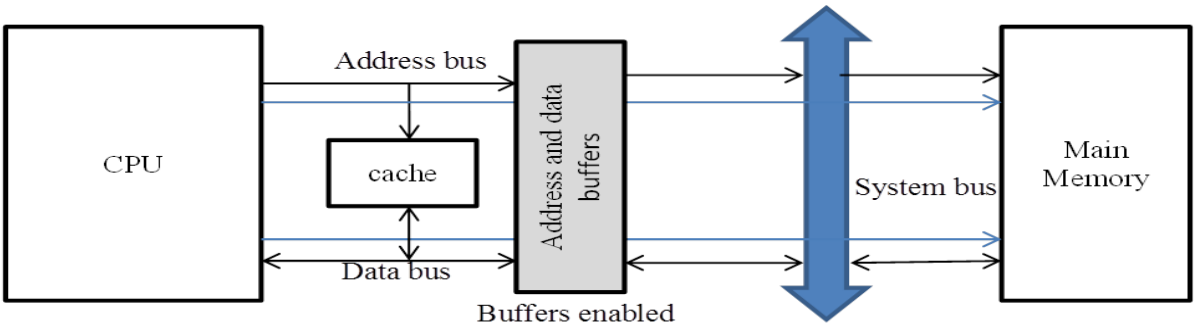


Figure 3.7 Write miss.

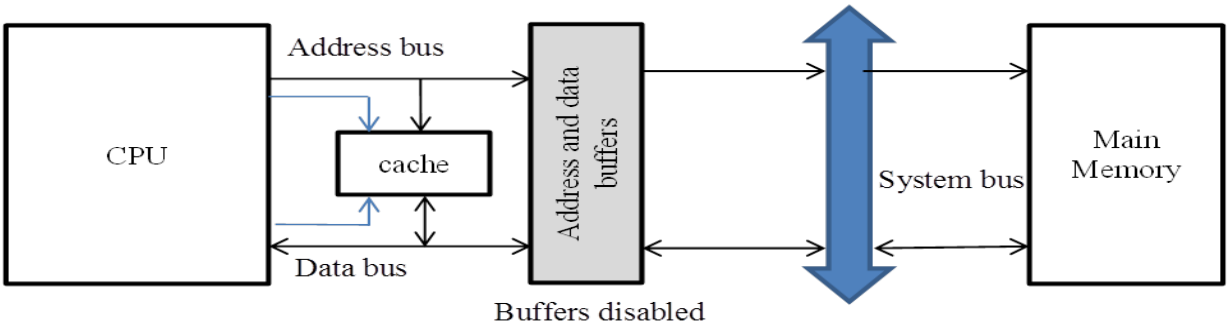


Figure 3.8 Cache hit in a write-back cache.

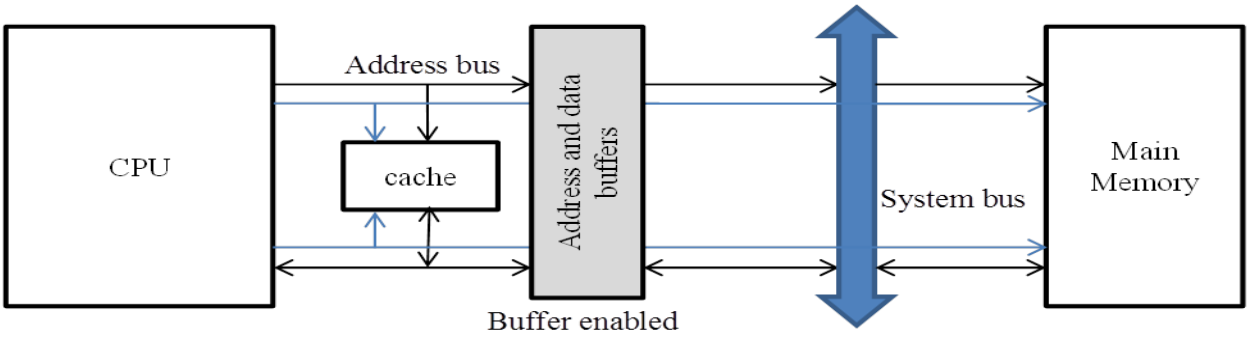


Figure 3.9 Cache hit in a write-through cache.

3.1.5 CACHE SIZE

According to the rules of memory hierarchy, the size of the cache should be small enough so that the overall average cost per bit is close to that of main memory alone and large enough so that the overall average access time is close to that of the cache alone The improved performance of the cache memory depends greatly on the miss rate. Miss rate reduction represents the classical approach in improving cache behaviour. The global miss rate for the system decreases as the cache size increases because capacity and conflict misses are reduced but for large caches sizes, the miss rate is stabilized as shown in figure 3.10.

This stabilization is due to the compulsory and coherence misses, which are independent of cache size. Though, increasing cache's size reduces miss rate, there are several motivations for minimizing the cache size. The larger the cache, the larger number of gates involved in addressing the cache. The result is that the larger caches tend to be slightly slower than that of the smaller ones, even when built with the same integrated circuit technology. Also, the larger number of gates associated with the large cache size contribute significantly towards increased power consumption. The available chip and board area also limits the cache size.

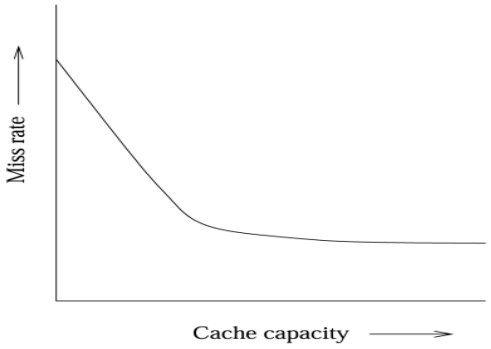


Figure 3.10 The variation of miss rate with Cache size.

3.1.6 LINE SIZE

Another design element is the line size. Data is transferred between memory and cache in blocks of fixed size, called cache lines. The contents of a cache memory is stored as set of cache lines. In other words, cache line size can be defined as the cache location having fixed data width. It specifies the minimum number of bits that is to be written into the cache. These cache lines store data of the lower memory level. When a cache line is copied from memory into the cache, a cache entry is created.

When a block of data is retrieved and placed in the cache, not only the desired word but some of the adjacent words are also retrieved. As the block size or cache's line size increases from very small to larger sizes, the hit ratio will at first increases, because of the principle of locality, according to which the data in the vicinity of the referenced word are likely to be referenced in the near future. As the block size increases, more useful data are brought in the cache. The hit ratio begins to decrease however, as the block becomes even bigger. Two specific effects come into play:

- Larger blocks reduce the number of blocks that can be paced inside the cache. When the block size becomes large enough for the same cache size, there are fewer blocks available inside the cache, and consequently there are more potential conflicts misses. Because each block fetch overwrites the older cache contents, a small number of blocks results in data being overwritten shortly after they are fetched.
- As a block becomes larger, each additional word is farther from the requested word and is less likely to be needed in the near future.

Figure 3.11 shows the variation of miss rate for various block sizes. From this graph, it can be seen that at first, the magnitude of miss rate tends to decrease when the block size is increased. It is because the larger blocks take advantage of spatial locality. When a program exhibits higher spatial locality, then a larger line size cache can reduce cache misses. However, without spatial locality, a large line size fetches many unnecessary bytes, which not only lengthens cache fill time, but may also evict needed bytes from the cache, thus increasing off-chip memory accesses and stalls. Increasing block size along with the cache size decreases the degree of miss rate. However, increase in cache's size in turn leads to increase in the power consumption [18].

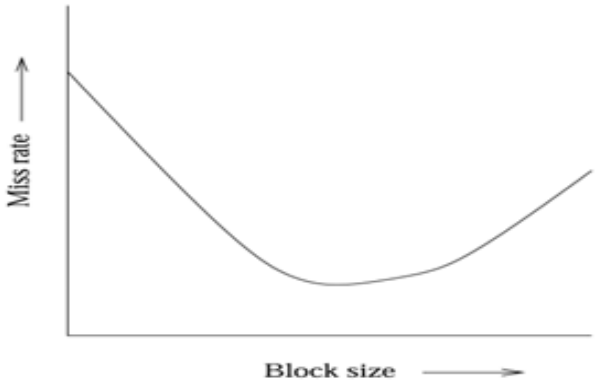


Figure 3.11 Variation of miss rate with cache's line size.

3.2 CACHE MEMORY ORGANIZATIONS

Cache Mapping is a method by which the contents of the main memory are brought in to the cache and are then referenced by the CPU. It is basically an algorithm which is needed to map main memory blocks into cache lines. Main memory consists of 2^n addressable words, with each word having a unique n-bit address. For mapping purposes, the memory is considered to be consisting of fixed length blocks of K words each. That is, there are $M = 2^n / K$ blocks. Cache consists of C lines of K words each, and the number of lines is considerably less than the number of main memory blocks (C<<M). At any time, some subset of the blocks of memory resides in lines in the cache. If a word in a block of memory is read, that block is transferred to one of the lines in the cache. Because, there are more number of blocks than cache lines, an individual line cannot be uniquely and permanently dedicated to a particular block. Thus, each line includes a tag that identifies which particular block is currently being stored. Figure 3.12 shows the process of referring to a typical cache memory. Each reference to a cell in memory is presented to the cache. The cache searches its directory of address tags shown in the figure to see if the item is in the cache. If the item is not in the cache, a miss occurs [1][2] .

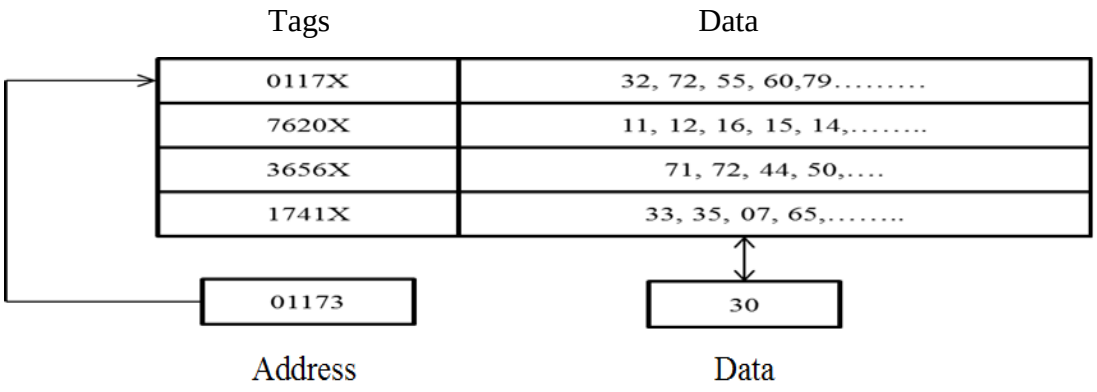


Figure 3.12 A cache-memory reference. The tag 0117X matches address 01173, so the cache returns the item in the position X=3 of the matched block.

Methods are needed to determine which main memory block occupies a cache line. The cache memory is organized according to the mapping policy followed. Following are the three different types of commonly used cache organizations based on mapping functions.

3.2.1 DIRECT-MAPPED CACHE ORGANIZATION

In direct-mapped cache organization, a given main memory block can be placed in one and only one place in the cache. Here, each memory block is mapped to a particular location inside the cache. The lower order address bits (index field) denoted by k bits are used to access the cache directory. Since multiple memory addresses map in to the same location in the cache directory, the upper line address bits (tag field) must be compared with the tag bits associated with the respective data block to ensure a hit. The lower order bits (word field) denoted by m bits are then compared to get the specified byte within a cache line. If a comparison is not valid, the result is a cache miss, or simply a miss. Figure 3.13 represents the direct mapped cache organization [26][28].

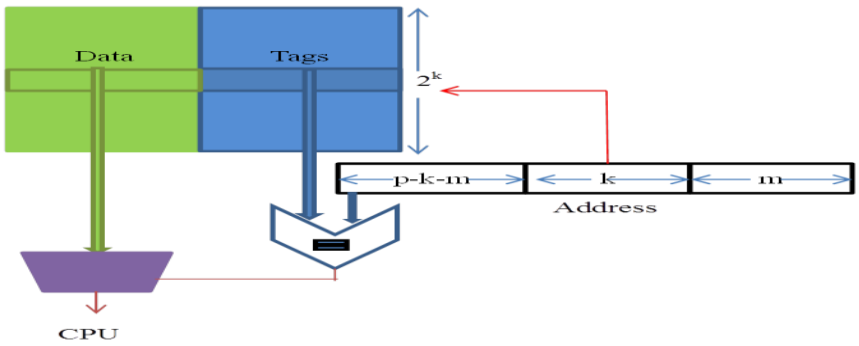


Figure 3.13 Direct-Mapped Cache organization.

3.2.2 FULLY-ASSOCIATIVE CACHE ORGANIZATION

Fully-associative cache organization form the other extreme, where a given main memory block can be placed anywhere in the cache. A fully associative cache requires the cache to be composed of associative memory holding both the memory address and the data for each cached line. After being placed in the cache, a given block is identified uniquely by its main memory block number, refer to as tag field. As shown in figure 3.14, when a memory operation is sent to the cache, the address of the request must be compared to each entry in the tag array using the internal logic of the associative memory to determine whether the data referenced by the operation is contained in the cache. If the requested address id found (a directory hit), the data block belonging to the corresponding location in the cache is fetched and returned to the processor, otherwise a miss occurs. If a match is found, the corresponding data is read out [27][28].

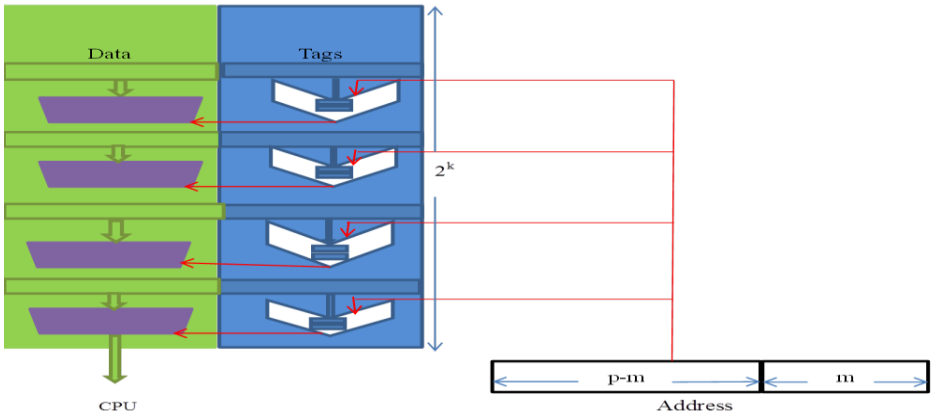


Figure 3.14 Fully-Associative Cache organization.

3.2.3 SET-ASSOCIATIVE CACHE ORGANIZATION

Block set-associative share the properties of both the previous mapping functions. The set-associative cache is similar to the direct-mapped cache, however now there are multiple choices: two four or more blocks from a given group in the main memory can occupy the same group in the cache depending on the fact that the whether the cache is 2 way, 4 way or 8 way set-associative. The mapping allows a limited number of blocks, with the same index and different tags, in the cache and can therefore be considered as a compromise between a fully-associative cache and a direct-mapped cache. In the set-associative cache organization,

the cache is divided into sets of blocks. A four-way set associative cache would have four blocks in each set. The number of blocks in a set is known as the associativity or set size. Each block in each set has a stored tag which together with the index, completes the identification of the block. Each of these data block with its respective tag bits corresponds to a particular location in sub-cache. The total cache array is formed by the collection of these sub-caches. In the set-associative cache, as in the direct mapped all the sub-arrays can be accessed simultaneously, together with the cache directory. If any of the entries in the cache directory match the reference address, and there is a hit, the particular data block is selected and the desired word is gated back to the processor [23][29]. A two-way set-associative organization is shown in Figure 3.15.

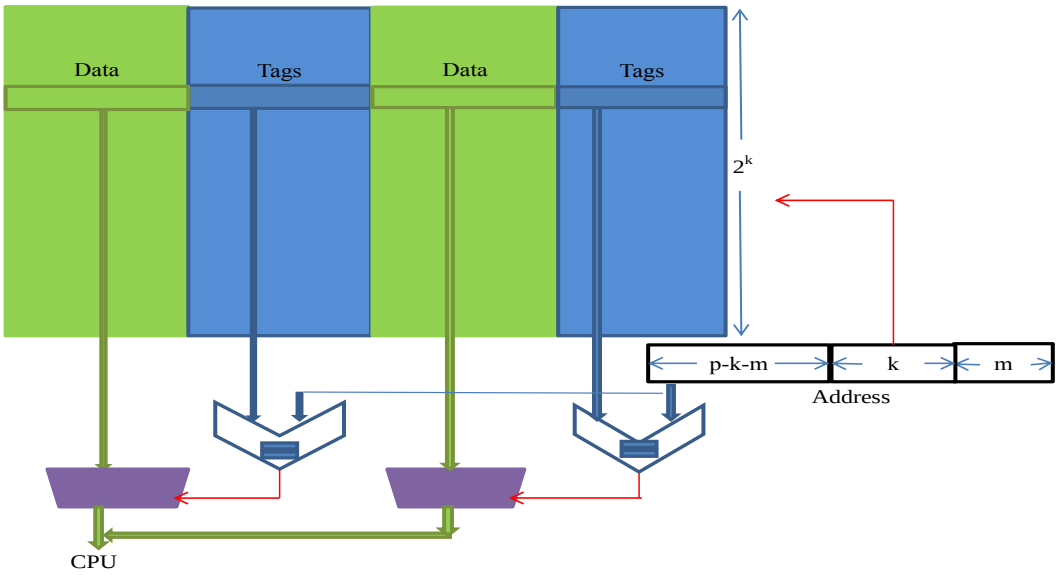


Figure 3.15 Set-Associative Cache organization.

3.2.4 IMPACT OF ASSOCIATIVITY ON CACHE PERFORMANCE

Associativity divides a cache into several ways. The higher associative organization of cache memory leads to more complex hardware requirements, but a highly-associative cache will have a lower miss rate as each set has more blocks, so there is less chance of a conflict between two addresses. Though the fully-associative cache makes complete use of their capacity but the major drawback lies in the expensive search process requiring a higher hardware cost. N-way set-associative cache organization is easier to search than fully-associative cache. Direct-mapped caches have more conflict misses due to their lack of associativity, but they have better access times than their associative counter parts. The

direct-mapped cache has the advantage of simplicity but the obvious disadvantage that only a single block from a given group can be placed in the cache at any given time. Performance-oriented applications want the highest associativity possible but these applications want the associativity such that the power savings from added ways, due to improved hit rate, outweigh the increased power consumption per access. For processors employing highly associative caches, the power consumption gets even worse as N-tag comparisons are needed for each parallel lookup of an N-way cache. Increasing the number of ways to two or even four improves a cache's hit rate, but beyond four ways, the improvement is typically not great. For example conventional direct-mapped cache accesses only one tag array and one data array per cache access, whereas a conventional four-way set-associative cache accesses four tags arrays and four data arrays per cache access. Thus, a conventional direct-mapped cache consumes much less dynamic power per access than a set-associative cache. Selecting optimal associativity is important, because changing associativity has a significant impact on cache performance, cost and on the performance of the system as a whole [20][30][28].

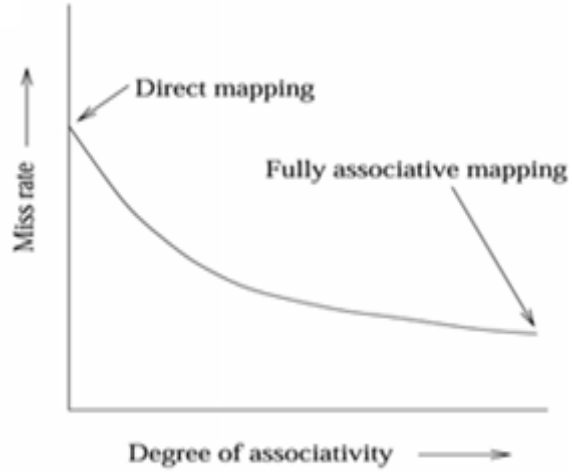


Figure 3.16 Variation of miss rate with associativity

CHAPTER 4

CACHE MEMORY DESIGN FOR REDUCED POWER

4.1 INTRODUCTION

The designing of cache architecture involves the following steps:

(a) **Cache Memory specifications:** This may include stating the type of cache organization, the size of the cache, cache line size, word size for the processor and the bit-length of main memory address.

(b) **Tag, Index and Offset bits calculation:** The main memory address is split into Tag, Index and Offset fields. The sizes of these fields are dependent on the bit-length of the main memory address, cache size, cache line size and associativity of the cache memory.

(c) **Implementation:** This involves the description of the mode of operations of various cache architectures with the aim of producing the software code for the architectures based on the given specifications.

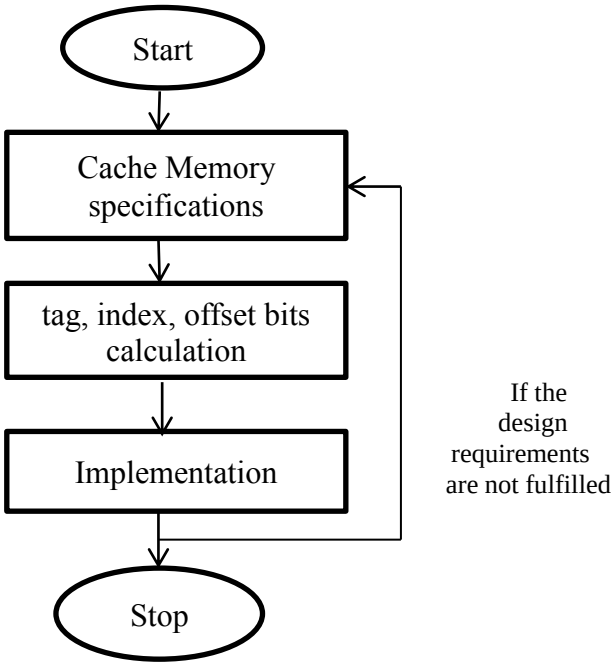


Figure 4.1 Design Steps for Cache architecture.

4.1.1 CACHE MEMORY SPECIFICATIONS

(i) Associativity
(ii) Line size
(iii) Cache size
(iv) Word size of the processor
(v) Bit-length of Main Memory address

In this dissertation, four way set–associative organization with cache size and cache-line size of 16Kbyte, and 32 byte respectively has been considered for designing of the cache architectures. The cache architectures differ with respect to their operational modes. The bit-length of main memory address is considered to be 32 bit and the word size for the processor is considered to be of 8 bits.

4.1.2 CALCULATION OF TAG, INDEX AND OFFSET BITS

All the information needed to locate the data in the cache is given in the main memory address. The 32 bit memory address is split up into various fields which are used for locating data in the cache. The fields are named as tag, index and offset fields as shown in figure 4.2. The sizes of these fields are based on the specifications of the cache memory.

If the block size is B then $\mathbf{b = \log_2 B}$ bits will be needed in the address to specify the block offset. If S is the number of sets in our cache, then the set index field has $\mathbf{s = \log_2 S}$ bits. If l is the length of the address (in bits), then the number of tag bits are **t = l -- b -- s.** The bit-length of main memory address is assumed to be 32 bit. Considering, the cache size and cache line size of 16Kbyte and 32 byte respectively, for a set-associative cache having associativity of four, the number of sets can be calculated as:

Number of sets = Cache size / Line size * associativity

$$\textbf{Number of sets} = 2^4 * 2^{10} / 2^5 * 2^2 = 128$$

Therefore, the bits size of tag field, index field and offset field can be calculated as

Bit size of index field = $\log_2 128$ = 7 bits

Bit size of offset field= $\log_2 32$ = 5 bits

Bit size of tag field = 32 – 7 – 5 = 20 bits

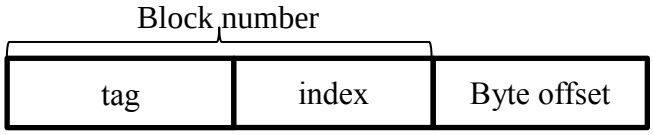


Figure 4.2 Address partition.

Thus, the considered four way set-associative cache organization consists of 128 sets. The main memory address is partitioned into 7 bit index field, 20 bit tag field and an offset field of 5 bits. Summarizing the above details, the 5 bit offset field determines the location of the desired word (byte) in the cache line or block. The 7 bit set index field is used to determine which cache set to look at. The remaining bits 20 bits are used for the tag which is the main memory block number used to uniquely identify a given block after being placed inside the cache.

4.1.3 IMPLEMENTATION

4.1.3.1 BACKGROUND & APPROACH

The amount of power consumed in accessing a cache $P_{(Cache)}$ can be approximated by the following equation:

$$\mathbf{P_{(Cache)} = P_{(Decode)} + P_{(SRAM\ array)}}$$

$$\mathbf{P_{(Cache)} \approx P_{(SRAM\ array)} = N_{(Tag)} \times P_{(Tag)} + N_{(Data)} \times P_{(Data)}}$$

- $\mathbf{P_{(Decode)}}$ is the average power consumed in decoding the main memory address requested by the processor.
- $\mathbf{P_{(SRAM\ array)}}$ is the average power consumed for accessing of the SRAM array (tag memory and data memory), per cache access.
- $\mathbf{N_{(Tag)}}$, $\mathbf{N_{(Data)}}$ are the number of tag sub-arrays and data sub-arrays to be activated for a cache access.
- $\mathbf{P_{(Tag)}}$, $\mathbf{P_{(Data)}}$ indicate the power consumed in accessing of tag sub-array and data sub-array respectively.

The power consumption of the address decoder is about three order of magnitude smaller than that of other components. Therefore, it is assumed that the power consumed in accessing the cache (P_{Cache}) depends significantly on the power consumed in accessing of the SRAM array ($P_{SRAM\ array}$).

As the cache memory has been organized as four-way set-associative, there are four possible places in which a given main memory block can reside, and all the places must be searched associatively. Each entry in a way consists of a valid bit (VALID), tag bits (TAG), and cache line to store memory data (DATA). Figure 4.3 gives an overview of typical details of cache memory structure.

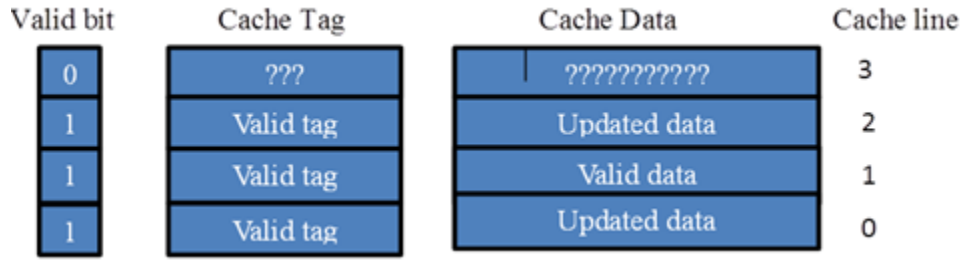


Figure 4.3 Cache Memory details.

Therefore, each way constitutes a data sub-array, a tag sub-array and a valid bit bank. The total cache array is formed by the collection of these sub-arrays. Based on the specifications considered in this dissertation, it can be inferred that each way associated with the four-way set-associative cache organization consists of a 128 X 20 bit tag memory consisting of corresponding tag bits associated with each data block stored inside the cache, a 256 X 1-bit memory for storing the valid bits associated with each cache line stored. The valid bit keeps the track whether the information inside the data block is valid or not. A 128 X 256-bit data memory is also associated with each way. As the cache's line size is considered to be of 32 byte, there are $2^5 = 32$ locations to search for, in order find a 8 bit word from the data block stored inside the cache. The 5-bit offset field becomes essentially a cache address, specifying where to find a word if indeed it is in the cache. A 7x128 decoder is also implemented to decode the index field consisting of 7 bits. The decoding of the index bits is done to address a particular set location to be accessed inside the cache directory.

As described above, both tag and data memory arrays usually dominate the power consumption of the cache component [31]. More ways associated with the cache organization means more lookups per access, and hence more power consumed per access. In fact, most of the power consumed in concurrent lookups of all the ways is redundant as the requested data can only be present in one particular way. This redundancy provides a good opportunity for power saving. Also, from the above specifications it can be inferred that $P_{(tag)} = 0.078P_{(data)}$, because the ratio of the tag size to the cache-line size is 20 : 256 (in terms of bits), or 0.078 : 1. Thus, efficient schemes for accessing the tag and data memory of the respective ways of

the four-way set associative cache organization are required to get reduced power consumption with optimum performance. The following sections describe the modelling of both the base four way set-associative and the proposed four-way set- associative cache architectures, the variation of power consumption among different accessing schemes employed by the respective cache architectures and how the proposed architectures help in reducing the power consumption. The architectures differ due their varying access mechanism. The section describes

4.2 BASE CACHE ARCHITECTURES

4.2.1 CONVENTIONAL SET-ASSOCIATIVE CACHE ARCHITECTURES

In the traditional four-way set-associative cache organization each cache set is divided into four partitions, each with its own tag sub-array and data sub-array. The power consumed in cache-access depends on the power dissipated for the SRAM access, as explained above. In the set-associative cache algorithm, the processor searches for data or instruction sequentially or in parallel after jumping to vicinity called set.

Once the address is decoded, to get the location of the particular cache's set to be accessed, tag bits and data bits from the tag sub-array and data sub-array respectively, associated with the four ways, are read out concurrently or sequentially. It is to be noted that no two data blocks that map in to the same cache's set can have the same tag number. The cache also maintains a valid bit for each data block stored, to keep track of whether the information inside the corresponding block is valid. Tag bits are compared in parallel or sequential manner to select the correct way and the corresponding valid bit is check so as to generate the hit signal. In conventional set-associative caches, the data sub-arrays of the ways are activated and are accessed regardless of hits or misses resulting in unnecessary power consumption. Two types of conventional set-associative organizations are explained in the following section.

4.2.1.1 CONVENTIONAL PARALLEL-ACCESS SET-ASSOCIATIVE CACHE ARCHITECTURE

In conventional parallel-access cache the tag bits are compared in parallel manner and hence it requires only one clock cycle to complete the comparison. If the cache is four-way set-

associative, four comparators for each way are required for the tag bits comparison. This will reduce the access time but consume more power by making all the comparators to work concurrently and by accessing unwanted data bits and tag bits. Figure 4.4 shows how the cache is accessed in a conventional parallel architectural scheme. The location of the particular cache's set is obtained after decoding the index bits.

After the set isolation, when the clock is enabled the tag bits from all the ways are taken and are compared with the tag field of the main memory address requested by the processor. At the same time the data blocks are also gated out from the respective data sub-arrays and are stored inside the corresponding buffers of the same size as shown in the figure. If there is a tag match and if the corresponding valid bit for the data block is set, indicating that the information inside the block is valid, then the buffer corresponding to the hit way gets enabled. The data block which was already driven from the data sub-array to the buffer are connected to the input lines of the multiplexer. The offset bits acting as selection lines of the multiplexer select the desired byte corresponding to the block offset field of the main memory address. The requested data byte is then available in the data bus in the same clock cycle and is forwarded to the processor. The output of the OR gate indicates whether cache access resulted in hit or miss based on the fact that whether one or none of the tag bits of all the ways that are accessed match with the tag field of the main memory address. The conventional parallel-access scheme used for the set-associative cache is good for the performance, but it is not optimized from the view point of power consumption. The parallel data arrays access before knowing the result of tag comparison would result in a lot of unnecessary way activities and, thus, large power consumption.

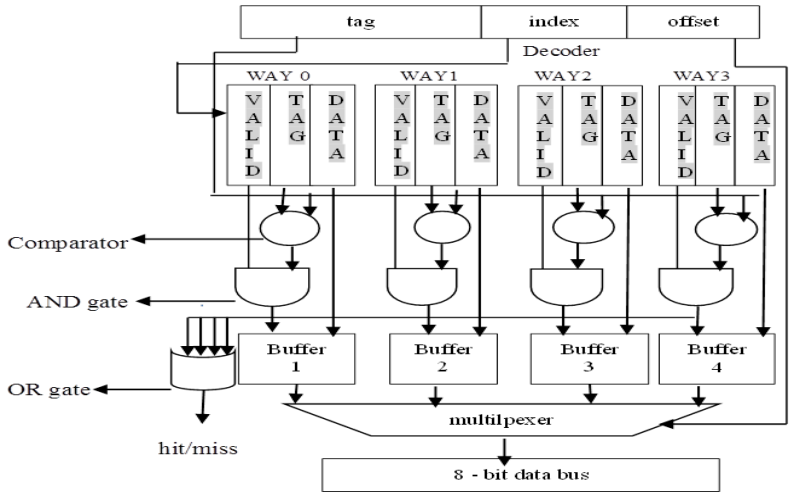


Figure 4.4 Conventional parallel-access set-associative cache architecture (Clock cycle 1).

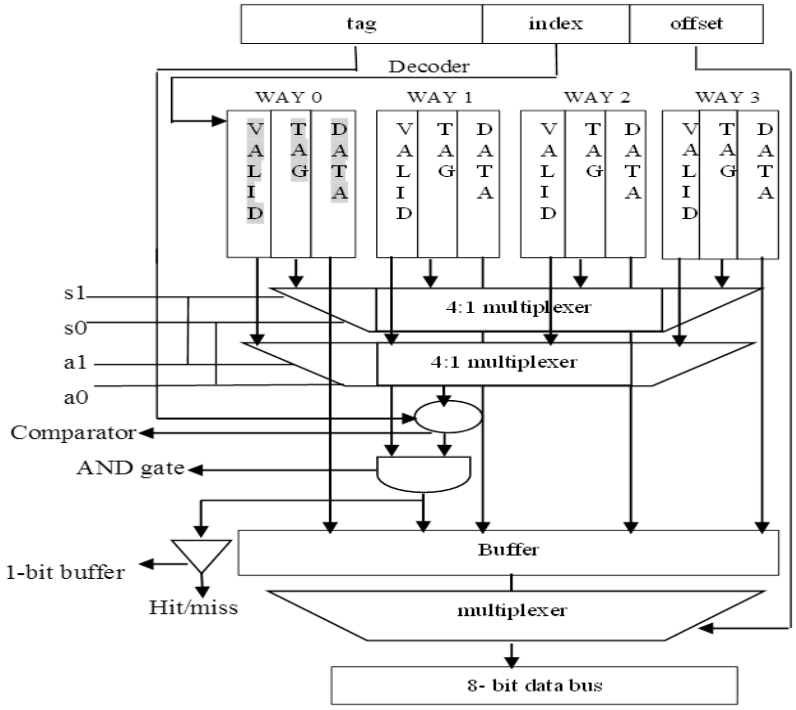
4.2.1.2 CONVENTIONAL SEQUENTIAL-ACCESS SET-ASSOCIATIVE CACHE ARCHITECTURE

To reduce redundant memory accesses while maintaining the cycle time, the concept of sequential way access of set-associative cache is exploited. The proposed architecture sequentially searches each way in cycle-by-cycle fashion. The access sequence terminates when a match is detected and subsequent accesses on remaining ways are then saved. It concurrently accesses both the tag and data arrays of the currently searched way in order to maintain the cycle time.

Figure 4.5 depicts the conventional sequential set-associative architecture. Here, each way of the four way set-associative cache memory comprises of a tag sub-array and a data sub-array. The valid bits are also present, 1 bit for each data block stored inside the cache. Therefore, a valid bit bank is also associated with each way. When a memory request is made by the processor, the index field is decoded and directs the search to the correct set. After jumping to the correct set, the tag bits and the valid bits of all the four ways of the corresponding set are gated out and are connected to the input lines of separate 4:1 multiplexers. The selection lines of these multiplexers are connected to each other and act as a counter. The counter keeps the track of the number of clock cycles required which in turn is dependent on the number of ways that are accessed to get the required data block. The first way is always probed first for all cache accesses, and the remaining ways are accessed in sequence in the following cycles if its previous way misses.

In the first clock cycle, the counter is set to 00, the tag bits of the WAY0 are selected and compared with the tag bits of the incoming main memory address using the comparator, at the same time the corresponding valid bit is also selected. During the same clock cycle, data access is made and the data block is gated out from the data sub-array. The data block is stored inside the buffer irrespective of a hit or miss. The enable of the buffer is the output of the AND gate. If there is a tag hit and if the valid bit is set, the buffer is enabled and the data bits inside the block are connected to the input lines of the multiplexer. The offset field acting as selection lines select the required data byte from the data bits and the selected 8-bit data is sent to the data bus. The whole process needs only one clock cycle for its completion. If the tag comparison of the first way resulted in a miss then the tag bits of the next way are compared, the counter will be incremented i.e. in each clock cycle different tag bits are compared and the corresponding data bits are fed to the input of the buffer. This is repeated until there is a hit or when all the ways in the set are compared. The worst case of this type of

cache architecture is when the desired data block is present in the last way. The output of the buffer which is composed of 1–bit indicates whether there is a tag match (hit) resulted while accessing any of the four ways of the four-way set-associative cache architecture. In the sequential set-associative type of architecture only one tag sub-array and data sub-array need to be accessed at a time resulting in considerably less hardware.



Accessed sub-array Un-accessed sub-array

Figure 4.5 Conventional sequential-access set-associative cache architecture (Clock cycle 1).

4.3 PROPOSED CACHE ARCHITECTURES

4.3.1 PHASED SET-ASSOCIATIVE CACHE ARCHITECTURES

The power consumption of conventional set-associative cache architectures is much higher because the large data sub-arrays of the unwanted ways are also accessed. As explained above, each way comprises of a tag sub-array and a data sub-array. In phased cache architecture the tag bits and the data bits are accessed in two phases i.e. in two clock cycles. In the first clock cycle the tag bits are compared with the tag field of the requested main memory address coming from the CPU and in the second clock cycle, the data sub-array of only the desired way is accessed. A more power conscious phased cache first checks tags,

and only reads the data out from the correct way. The phased cache only probes data sub-array of one way instead of all, reducing accesses to large data sub-arrays. While significant power is saved by using phased set-associative cache architectures, more number of clock cycles are required for their operation. Thus, they have slower speed of execution in comparison to the conventional set-associative architectures.

4.3.1.1 PHASED SEQUENTIAL-ACCESS SET-ASSOCIATIVE CACHE ARCHITECTURE

Figure 4.6 shows the phased sequential set-associative cache architecture. In this cache architecture, the tag sub-array of the four ways corresponding to the selected cache's set are accessed sequentially and the data sub-array corresponding of only that particular way is accessed whose tag bits comparison results in a hit.

After the decoding of the index bits, the tag bits of all the ways in the selected set are fed to the input lines of the multiplexer. The valid bits from the valid bit bank of all the four ways are also connected to the input lines of another multiplexer. The selection lines of these multiplexers act as a counter. This counter keeps the count of the number of clock cycles required which is dependent on the number of ways that are accessed to get the desired data block. Initially the counter is set to 00 and so the tag bits of WAY 0 are compared with the tag bits from the CPU and the generated output is fed to the input of AND gate along with the corresponding valid bit. This is done in the first cycle as presented in figure 4.6(a). In the second cycle, if the tag bits of WAY0 get matched with the tag bits corresponding to the tag field of the main memory address and if the valid bit corresponding to the data block is set, then only the data sub-array of WAY0 is accessed and the desired data block is fed to the input of the buffer as shown in figure 4.6(b). The buffer will be enabled in the same clock cycle and the required data bits corresponding to the data block is fed to the input lines of the multiplexer where the offset field is then used to select the required byte from the corresponding block. The multiplexer routes the desired byte to the data bus. Thus, the requested byte will be available in the data bus in two clock cycles. This is the best case of phased sequential cache. The worst case is when the required data is stored in WAY3, in this case it will take five clock cycles to get the desired data byte. The first way is always accessed first and the subsequent sequential accesses to the next ways are made depending on the tag bits comparison results from the previous ways. The output of the 1-bit buffer indicates whether there is any hit resulted in the subsequent accessing of any of the four

ways.

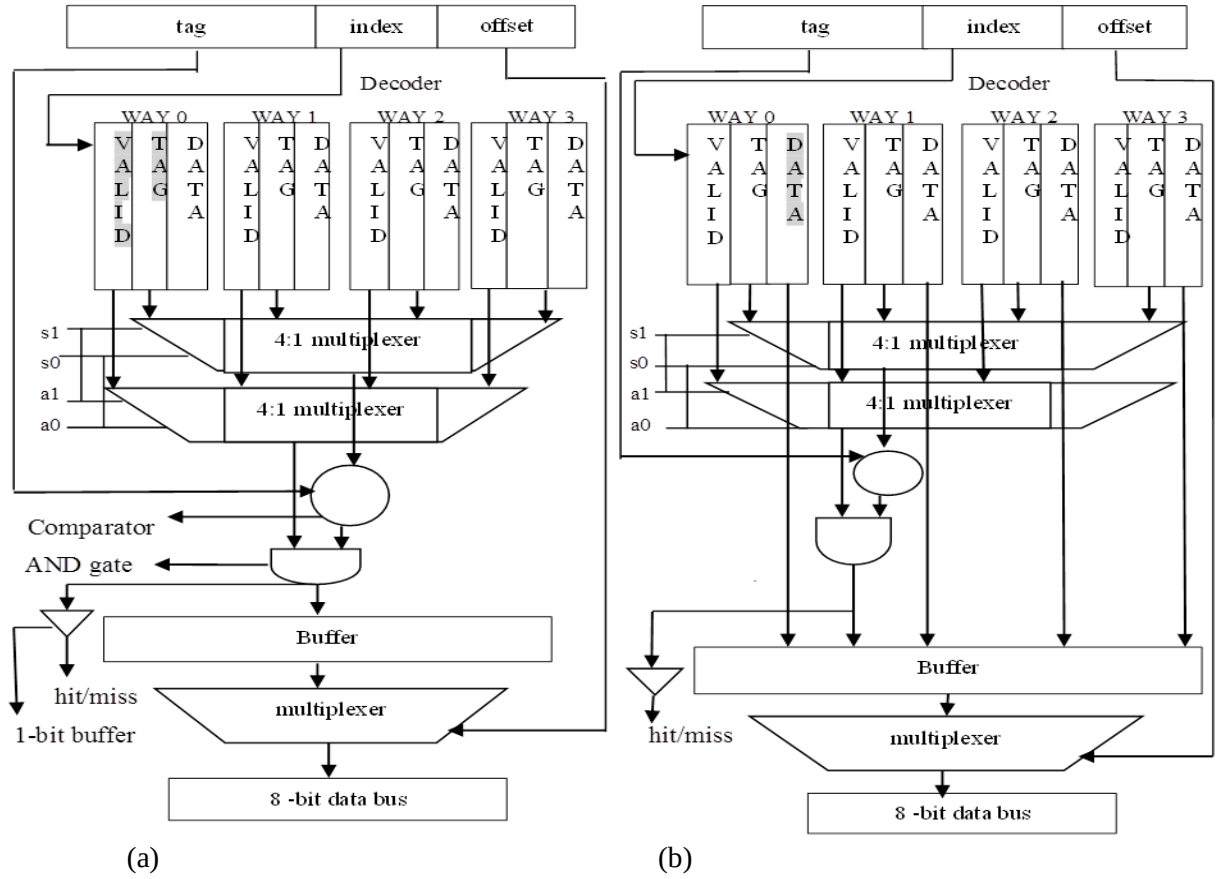


Figure 4.6 Phased sequential-access set-associative cache architecture: (a) Clock cycle 1 (b) Clock cycle 2.

4.3.1.2 PHASED PARALLEL-ACCESS SET-ASSOCIATIVE CACHE ARCHITECTURE

Figure 4.7 presents the phased parallel set-associative cache architecture. As shown in figure 4.7(a), in the first clock cycle, the tag sub-arrays of all the four ways are accessed corresponding to the desired cache's set location and the respective tag bits from all the four ways are compared with the tag field of the main memory address coming from the processor using the tag comparators. If the tag bits comparison resulted in a hit, and the associated valid bit is also set, then in the second clock cycle, the corresponding data sub-array is accessed and the data block for the hit way is gated out and get connected to the input lines of the multiplexer. The bits corresponding to the offset field of the address is then used to select the

desired word or byte within the data block. The data byte will be available in the data bus in the same clock cycle as shown in figure 4.7(b).

The phased parallel cache architecture avoids unwanted access of the data sub-arrays thus reducing the power consumption. The phased cache only probes one data sub-array instead of four data sub-arrays. As the architecture is of parallel type the number of comparators and gates used are equal to the number of ways in the set.

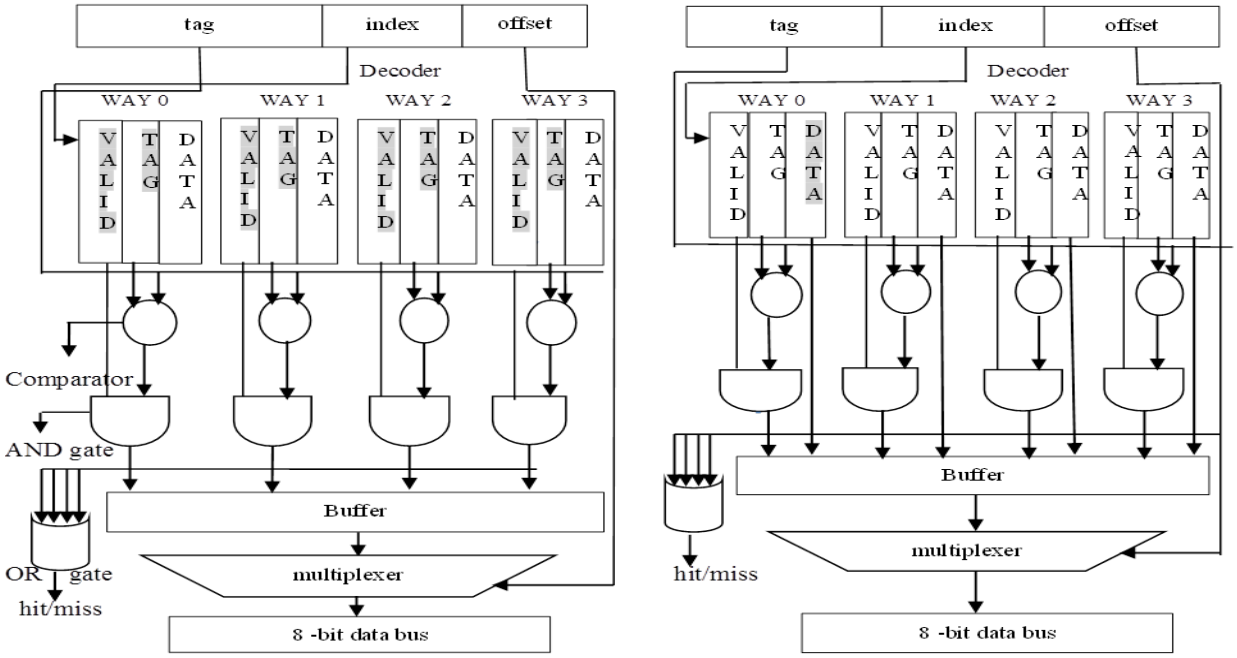


Figure 4.7 Phased Parallel-access set-associative cache architecture: (a) Clock cycle 1 (b) Clock cycle 2.

4.3.2 CONVENTIONAL PARALLEL SET-ASSOCIATIVE ARCHITECTURE WITH VALID BIT PRE-CHECK TECHNIQUE

In the set-associative cache architectural scheme, each block of data brought from the main memory and residing inside the cache has a valid bit associated with it which keeps the track that whether the information in the data block is valid or not. Therefore, when the cache is accessed, in additional to tag checking of all ways, the corresponding valid bits of all ways must be checked. In the proposed architectural scheme, the valid bits of all the ways are checked prior to the tag bits comparison. Each way has an AND gate associated with it for the prior checking of the valid bits unlike the previous architectures. After the cache's set

isolation when the clock is enabled, the valid bits of all the ways are fed to the two inputs AND gates along with the output bit generated after decoding the index bits. The tag and data sub-arrays of the respective ways are accessed when their valid bits associated with the corresponding valid bit banks are set i.e. when the valid bit is 1. The corresponding ways of the sub-cache whose valid bits are not set are disabled and hence are not accessed. Therefore, the set-associative architecture can be configured as 1-way, 2-way, 3-way or 4-way depending upon the number of tag sub-arrays and data sub-arrays to be accessed which in turn depends on the status of valid bits associated with the respective ways. The technique prevents the unnecessary comparison and accessing of tag and data sub-arrays of all the ways unlike the conventional parallel set-associative structure. This cache architectural scheme needs only one clock cycle for performing the desired operation, thus giving reduced power consumption without compromising the speed.

CHAPTER 5

RESULTS AND DISCUSSION

5.1 LANGUAGE USED

Verilog HDL (Hardware Description Language) is used as the designing language for the modelling of various cache architectures described above. Behavioral approach is adopted for designing of the cache architectures.

5.2 VERIFICATION TOOL

The verification of the functional correctness of the cache architectures is done through behavioural simulation of the respective designs in Xilinx 8.2i.

5.3 RESULTS

Although any specifications for cache memory can be considered for the sake of implementation. In this dissertation, the following specifications are considered for all the implemented architectures.

Size of cache memory = 16Kbyte

Cache Line size = 32 byte

Bit-length for Main Memory address = 32 bits

Size of tag field = 20 bits

Size of index field = 7 bits

Size of offset field = 5 bits

Word size for the processor = 8 bits

Associativity of the cache architecture = 4

5.3.1 CONVENTIONAL PARALLEL-ACCESS SET-ASSOCIATIVE CACHE ARCHITECTURE

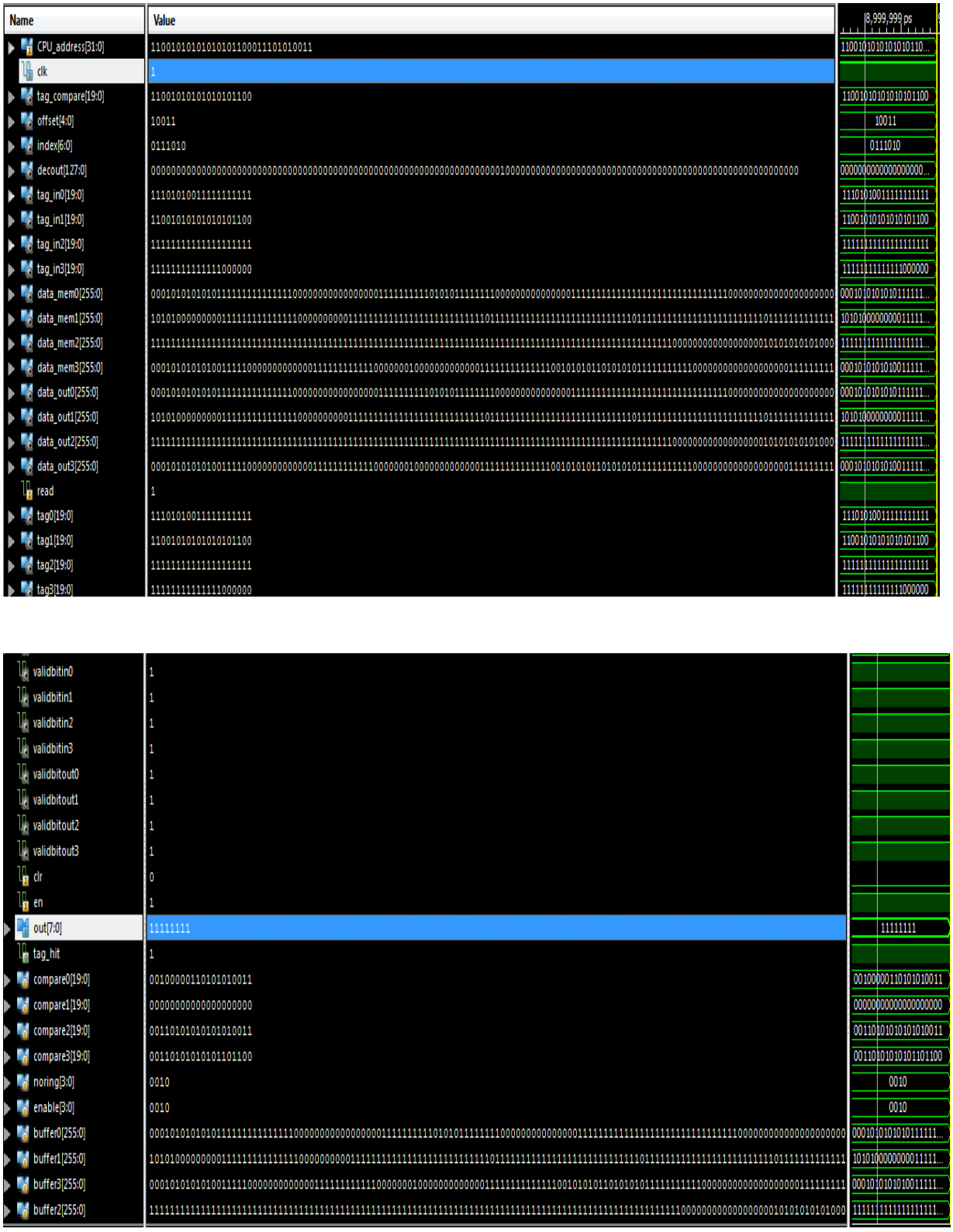


Figure 5.1 Simulation results for Conventional Parallel-access set-associative cache architecture.

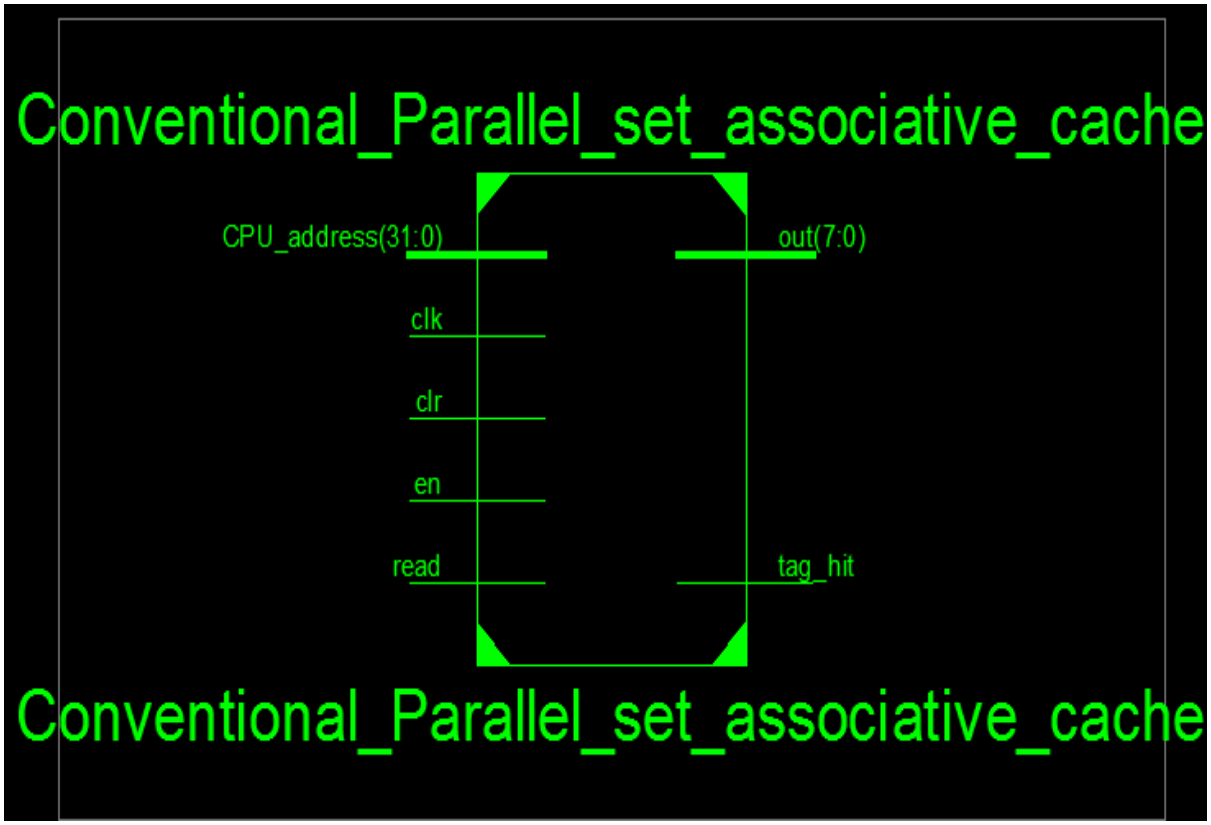


Figure 5.2 Block diagram for Conventional Parallel-access set-associative cache architecture.

Table 5.1 Design summary of Conventional Parallel-access set-associative cache architecture.

```
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 3s500eft256-4

 Number of Slices:                      579  out of   4656    12%
 Number of Slice Flip Flops:            269  out of   9312     2%
 Number of 4 input LUTs:               1298  out of   9312    13%
 Number of IOs:                          45
 Number of bonded IOBs:                  45  out of    190    23%
 Number of GCLKs:                         1  out of     24     4%

---------------------------
```

5.3.2 CONVENTIONAL SEQUENTIAL-ACCESS SET-ASSOCIATIVE CACHE ARCHITECTURE

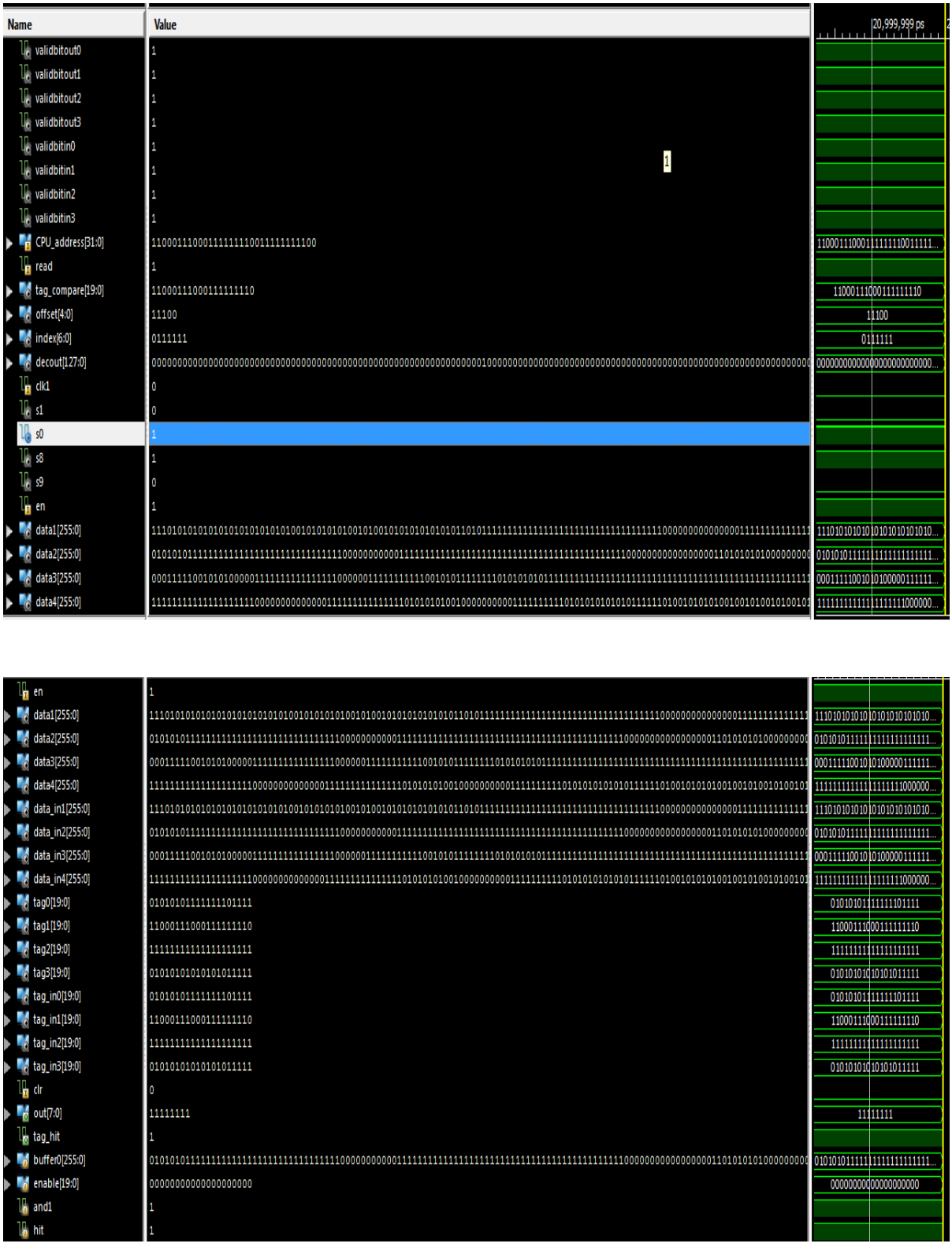


Figure 5.3 Simulation results for Conventional Sequential-access set-associative cache architecture.

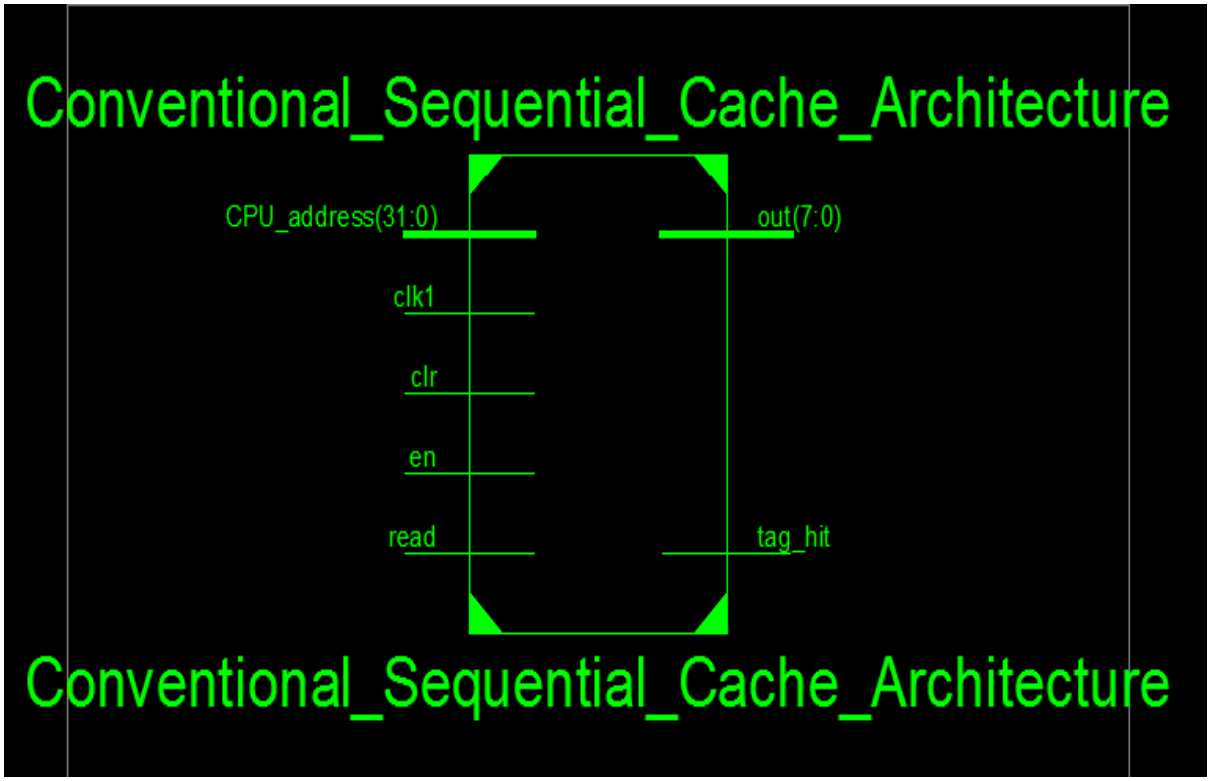


Figure 5.4 Block diagram of Conventional Sequential-access set-associative cache architecture.

Table 5.2 Design summary of Conventional Sequential-access set-associative cache architecture.

```
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 3s500eft256-4

 Number of Slices:                      597  out of   4656    12%
 Number of Slice Flip Flops:            266  out of   9312     2%
 Number of 4 input LUTs:               1063  out of   9312    11%
 Number of IOs:                          45
 Number of bonded IOBs:                  45  out of    190    23%
 Number of GCLKs:                         1  out of     24     4%

---------------------------
```

5.3.3 PHASED PARALLEL-ACCESS SET-ASSOCIATIVE CACHE ARCHITECTURE

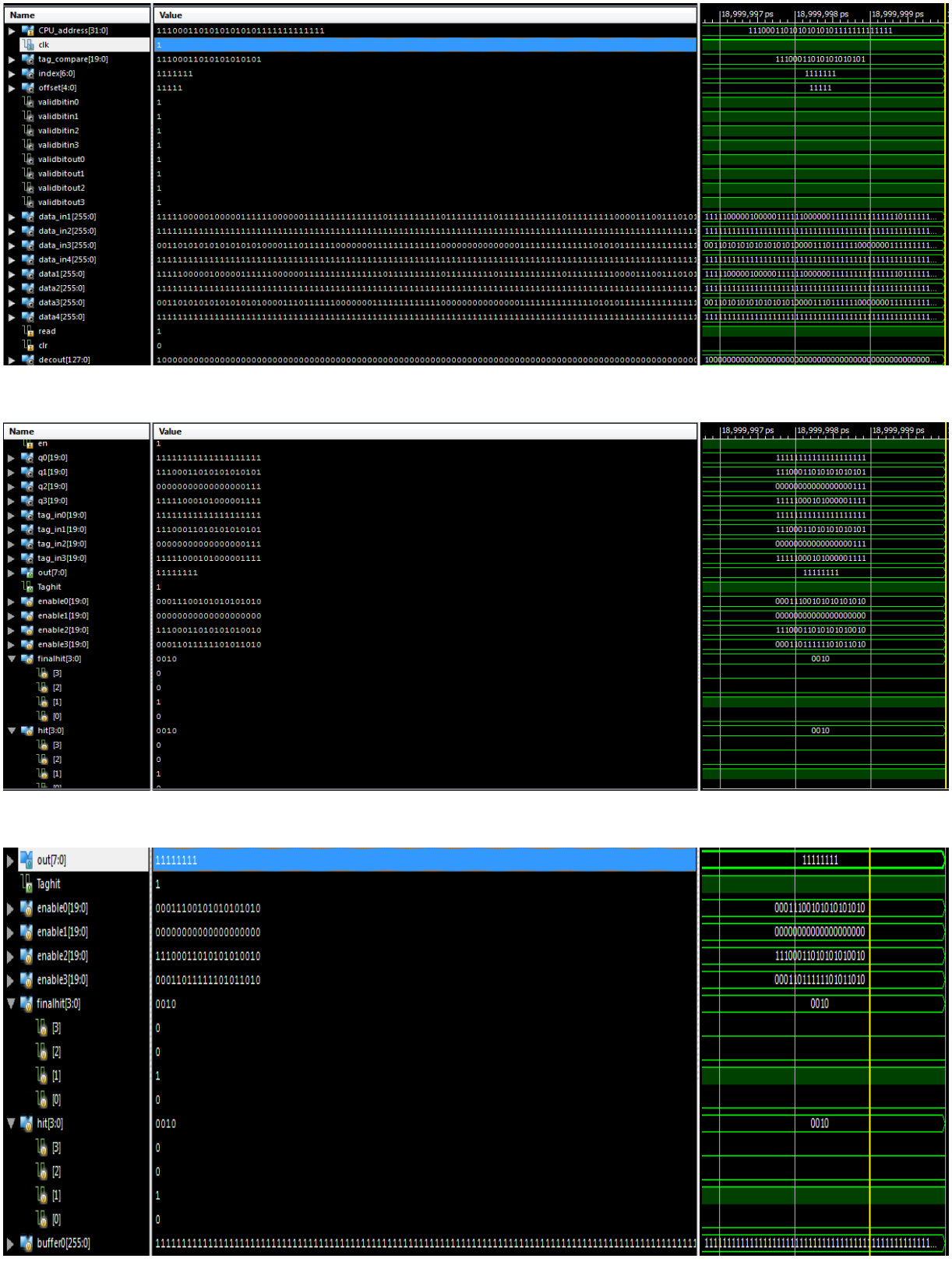


Figure 5.5 Simulation results for Phased Parallel-access set-associative cache architecture.

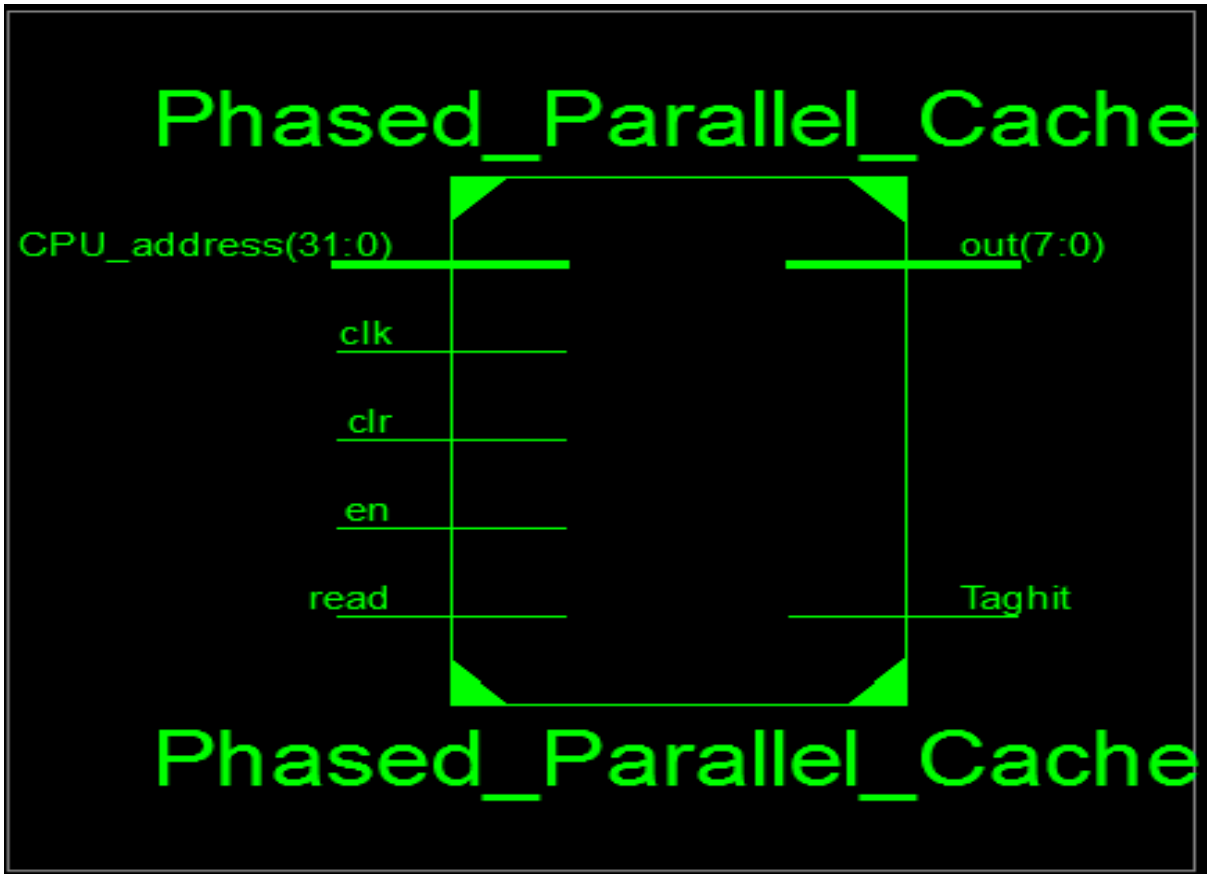


Figure 5.6 Block diagram for Phased Parallel-access set-associative cache architecture

Table 5.3 Design summary of Phased Parallel-access set-associative cache architecture.

```
--------------------------------------------------------------------------

Device utilization summary:
---------------------------

Selected Device : 3s500eft256-4

 Number of Slices:                      576  out of   4656    12%
 Number of Slice Flip Flops:             10  out of   9312     0%
 Number of 4 input LUTs:               1011  out of   9312    10%
 Number of IOs:                          45
 Number of bonded IOBs:                  45  out of    190    23%
 Number of GCLKs:                         1  out of     24     4%

---------------------------
```

5.3.4 PHASED SEQUENTIAL-ACCESS SET-ASSOCIATIVE CACHE ARCHITECTURE

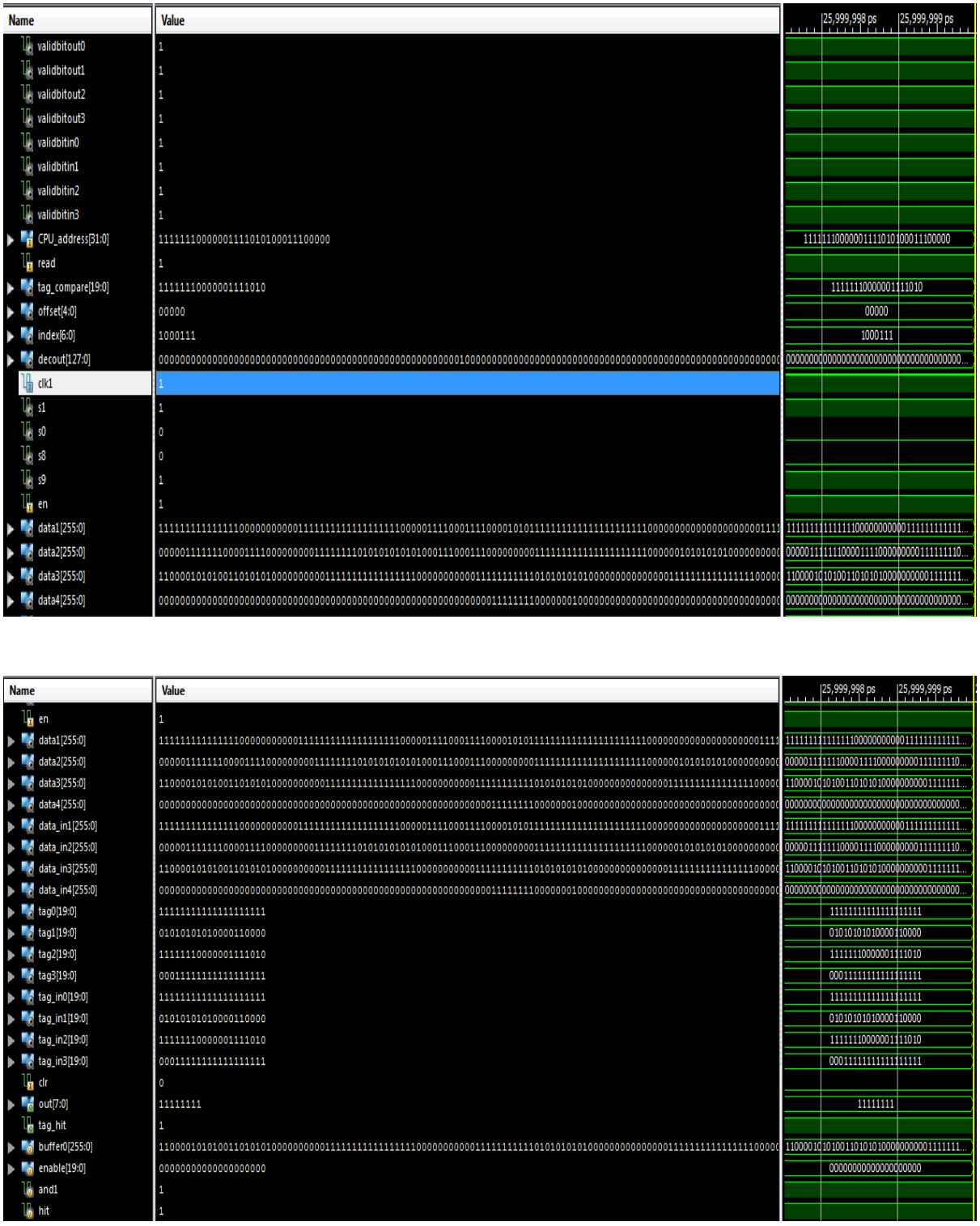


Figure 5.7 Simulation results for Phased Sequential-access set-associative cache architecture.

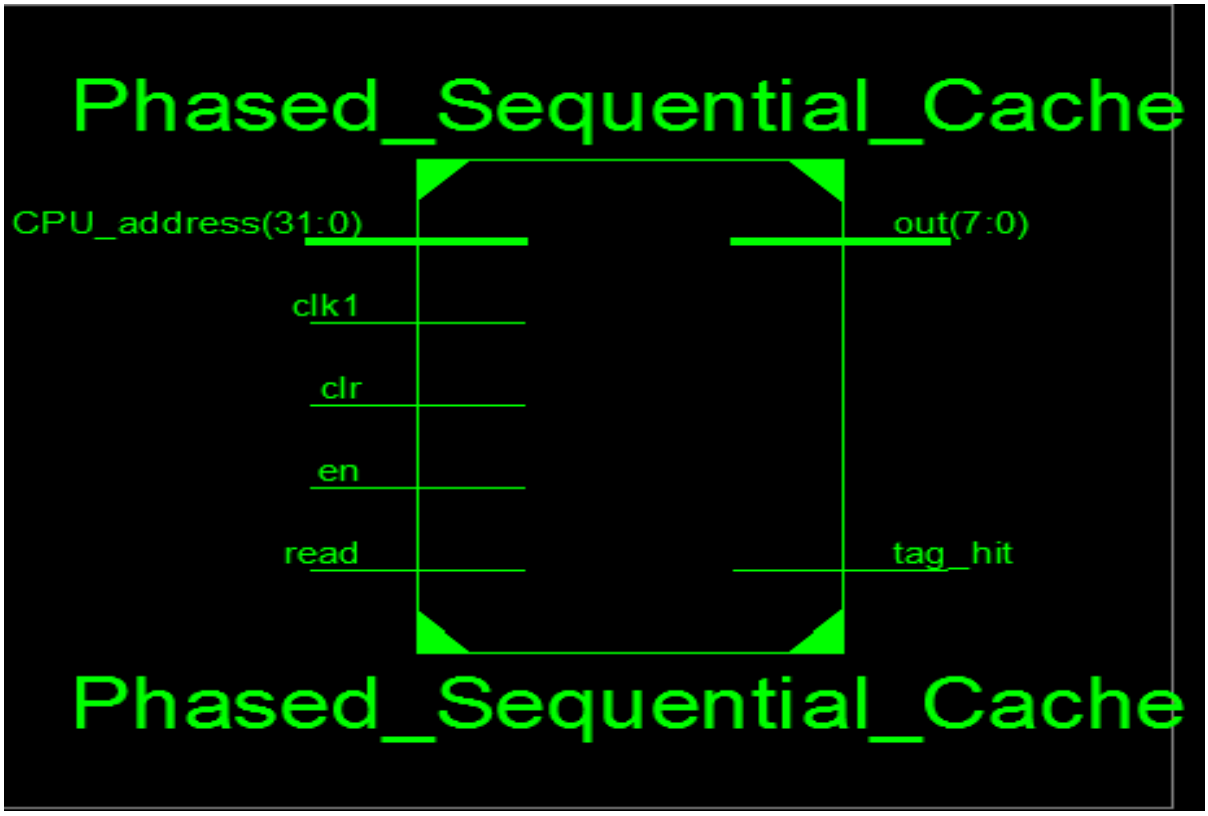


Figure 5.8 Block diagram for Phased Sequential-access set-associative cache architecture

Table 5.4 Design summary for Phased Sequential-access set-associative cache architecture.

```
Device utilization summary:
---------------------------

Selected Device : 3s500eft256-4

 Number of Slices:                      520  out of   4656    11%
 Number of Slice Flip Flops:             10  out of   9312     0%
 Number of 4 input LUTs:                930  out of   9312     9%
 Number of IOs:                          45
 Number of bonded IOBs:                  45  out of    190    23%
 Number of GCLKs:                         1  out of     24     4%

---------------------------
```

5.3.5 CONVENTIONAL PARALLEL WITH VALID BIT PRE-CHECK SET-ASSOCIATIVE ARCHITECTURE

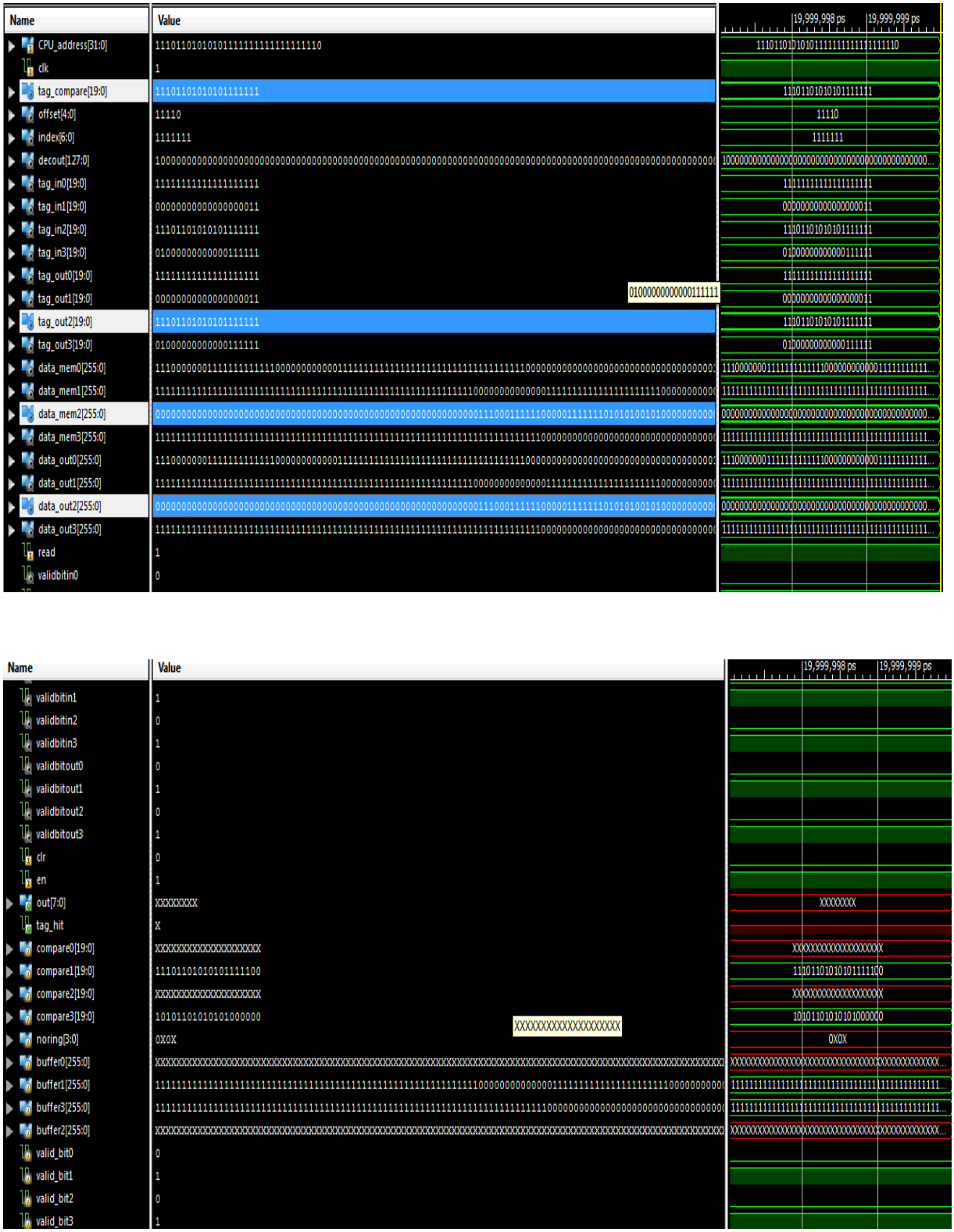


Figure 5.9 Simulation results for Conventional Parallel with valid bit pre-check set-associative cache architecture.

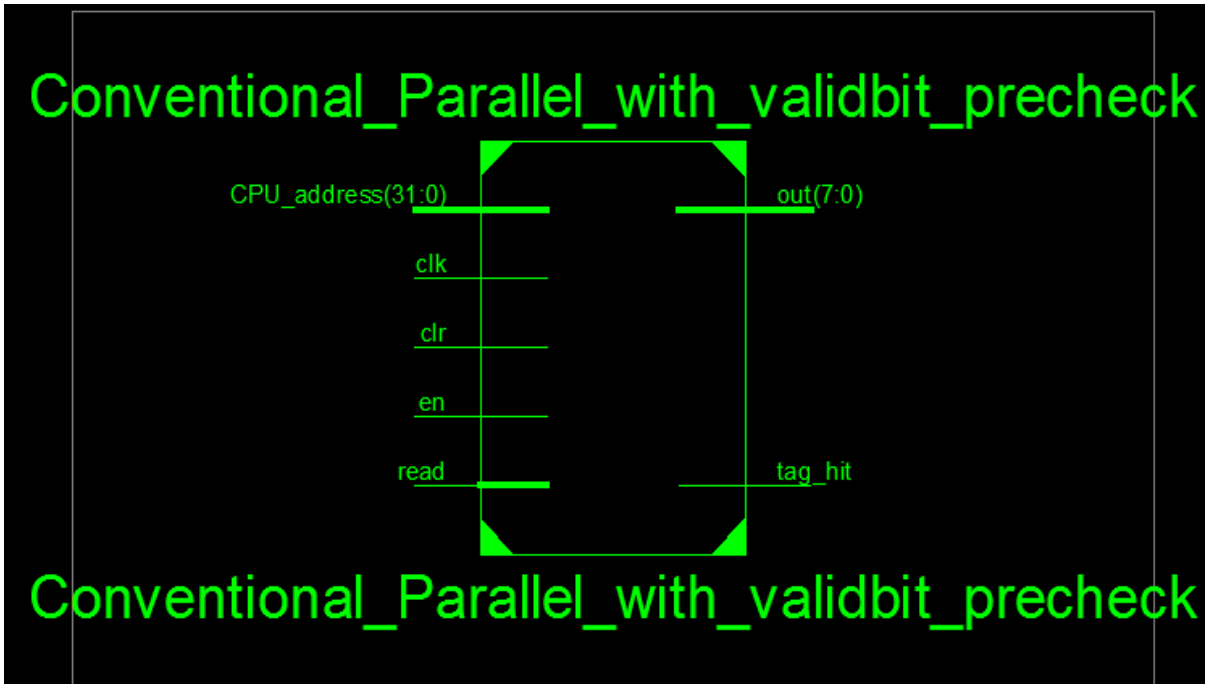


Figure 5.10 Block diagram for Conventional Parallel with valid bit pre-check set-associative cache architecture

Table 5.5 Design summary for Conventional Parallel with valid bit pre-check set-associative cache architecture.

```
--------------------------------------------------------------------------------

Device utilization summary:
---------------------------

Selected Device : 3s500eft256-4

 Number of Slices:                      718  out of   4656    15%
 Number of Slice Flip Flops:            269  out of   9312     2%
 Number of 4 input LUTs:               1276  out of   9312    13%
 Number of IOs:                          45
 Number of bonded IOBs:                  45  out of    190    23%
 Number of GCLKs:                         1  out of     24     4%

---------------------------
```

5.3.6 POWER CONSUMPTION ANALYSIS OF BASE AND PROPOSED SET-ASSOCIATIVE ARCHITECTURES

Table 5.6 Power consumption analysis of Conventional Parallel-access and Phased Parallel-access set-associative cache architectures using Xilinx8.2i

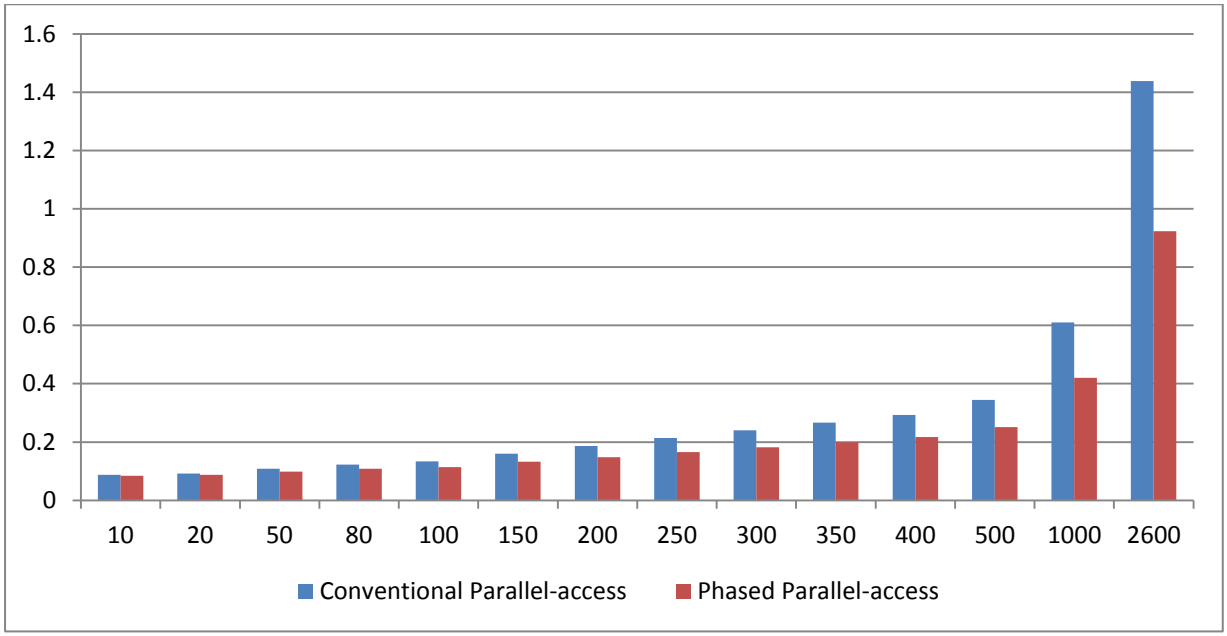
Clock frequency (in MHz)	Power Consumption (in watts)		
	Conventional Parallel-access	Phased Parallel-access	% Reduction in Power Consumption
10	0.087	0.084	3.44
20	0.092	0.087	5.43
50	0.108	0.098	9.25
80	0.123	0.108	12.19
100	0.134	0.114	14.92
150	0.160	0.132	17.5
200	0.186	0.148	20.43
250	0.214	0.166	22.42
300	0.240	0.182	24.17
350	0.266	0.200	24.81
400	0.293	0.217	25.93
500	0.345	0.251	27.25
1000	0.610	0.420	31.15
2600	1.438	0.923	35.87

Table 5.7 Power consumption analysis of Conventional Sequential-access and Phased Sequential-access set-associative cache architectures using Xilinx 8.2i

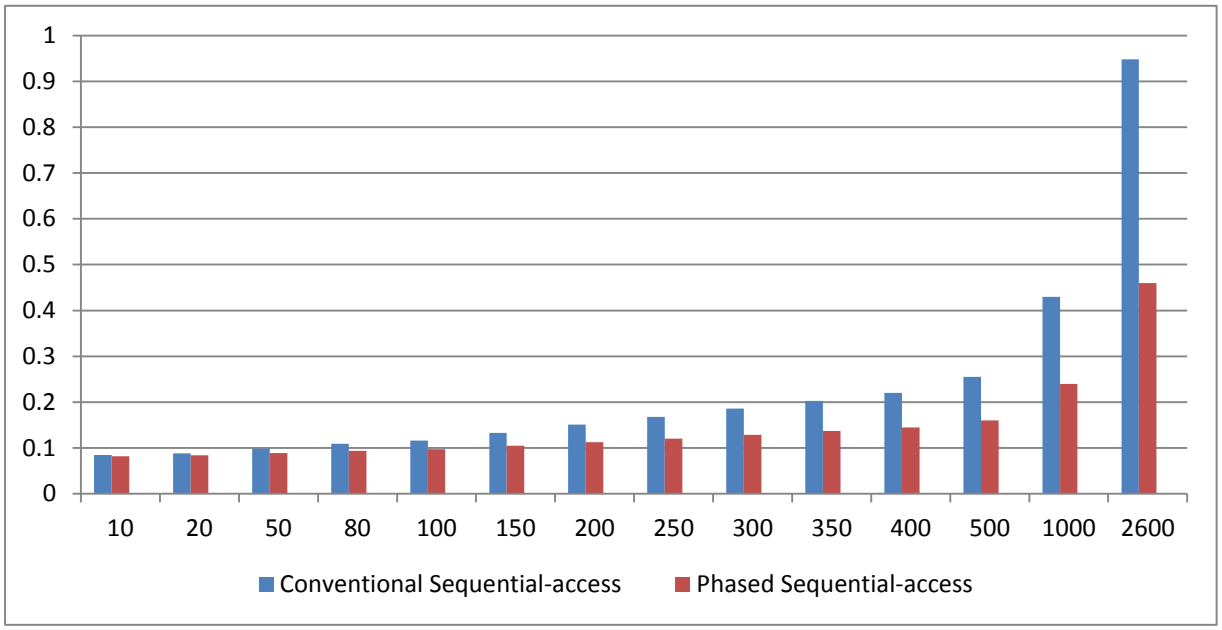
Clock frequency (in MHz)	Power Consumption (in watts)		
	Conventional Sequential-access	Phased Sequential-access	% Reduction in Power Consumption
10	0.085	0.082	3.53
20	0.088	0.084	4.54
50	0.099	0.089	10.10
80	0.109	0.094	13.76
100	0.116	0.097	16.38
150	0.133	0.105	21.05
200	0.151	0.113	25.17
250	0.168	0.120	28.57
300	0.186	0.129	30.61
350	0.203	0.137	32.51
400	0.220	0.145	34.09
500	0.255	0.160	37.25
1000	0.430	0.240	44.19
2600	0.948	0.460	51.48

Table 5.8 Power consumption analysis of Conventional parallel and Conventional Parallel with valid bit pre-check technique set-associative cache architectures using Xilinx 8.2i

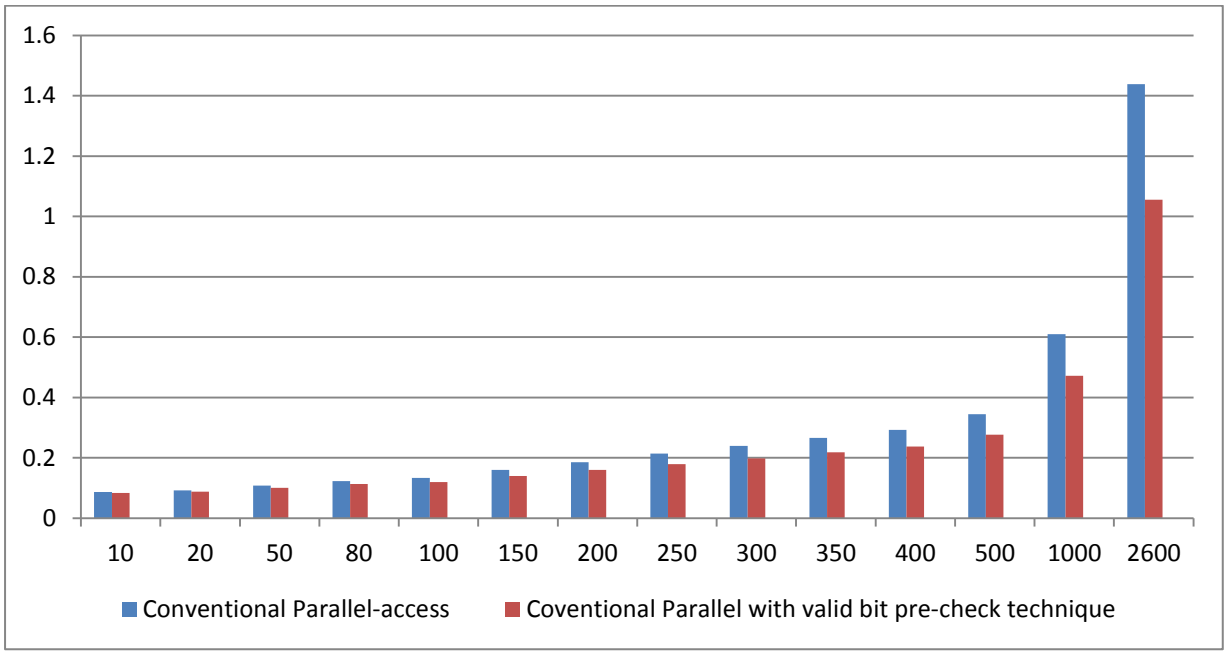
Clock frequency (in MHz)	**Power Consumption (in watts)**		
	Conventional Parallel	**Conventional Parallel with valid bit pre-checking**	**% Reduction in Power Consumption**
10	0.087	0.084	3.45
20	0.092	0.088	4.35
50	0.108	0.101	6.48
80	0.123	0.113	8.13
100	0.134	0.120	10.45
150	0.160	0.140	12.50
200	0.186	0.160	13.98
250	0.214	0.179	16.55
300	0.240	0.198	17.50
350	0.266	0.218	18.04
400	0.293	0.238	18.77
500	0.345	0.277	19.71
1000	0.610	0.472	22.62
2600	1.438	1.055	26.63



(a).



(b).



(c).

x axis: clock frequency(MHz) y axis: power dissipation(watts)

Figure 5.11 Comparison of Power dissipation of set-associative Cache architectures : (a). Conventional parallel-access and Phased parallel-access. (b). Conventional sequential-access and phased sequential-access. (c). Conventional Parallel-access and Conventional Parallel with valid bit pre-check technique.

5.4 DISCUSSION

The access time required by conventional parallel-access cache architecture to deliver the requested data is much less when compared to the conventional sequential-access cache. However, the architecture requires more hardware for its functionality and hence the power consumed will be more. The same is the case with the phased parallel-access and phased sequential-access cache architectural schemes. Phased parallel-access cache architecture consumes more power than phased sequential-access architecture cache as the requirement for hardware is more in parallel scheme of comparison of tags and accessing of data. The proposed conventional parallel with valid bit pre-check cache architectural shows power reduction without increasing the cache's access time.

Tables 5.1 to 5.5 illustrate the device utilization summaries of each cache architectural scheme with respect to the number of four inputs LUTs (Look up Tables) used by the respective architectures. From the tables it can be inferred that the amount of power consumed by a particular cache architecture is directly related to its device utilization count. The increase in the device utilization count results in an increased area hence resulting in an increased power consumption. It can be observed from the table that conventional parallel-access architecture has highest device utilization count (in terms of LUTs), thus dissipating more power in comparison to other architectures. On the contrary, phased sequential-access architecture requires least number of LUTs for its functionality and therefore, it has lowest power consumption.

The power dissipated by the proposed architectural schemes are synthesized and checked using Xilinx 8.2i. Tables 5.6, 5.7 and 5.8 depict the values of the power consumed (in watts) by the cache architectures discussed above at different values of clock frequencies (in MHz). Figures 5.11(a) and 5.11(b) are the graphs showing the variation of power with respect to different clock frequencies for sequential and parallel cache architectures. From the figures it can be observed that the power consumed by conventional sequential-access architecture is much higher as compared to phased sequential-access cache. At low clock frequencies the difference is not significant but the difference in power consumption increases with increasing clock frequencies. The same observations can be made for the conventional parallel-access and phased parallel-access set-associative architectures. The conventional parallel with valid bit pre-check technique also shows reduction in power consumption in comparison to conventional parallel cache architecture as observed from figure 5.11(c).

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

6.1 CONCLUSION

Cache memories are employed in embedded architectures to bridge the performance gap which arises due to latency between the processor cycle time and memory access delay. However, cache memory takes a large portion of the chip area and consumes significant power. In this dissertation, phased set-associative cache architectures are proposed that aim at achieving low power consumption by efficient handling of tag and data bits. As the size of the data sub-array is large, the key idea is to access the data sub-array related to a particular sub-cache in the set-associative arrangement only when the tag bits comparison comes out be a hit. Since only one way has the data required to be referenced by the processor on a cache hit, phased cache architectures avoid the associative lookup to the data store by first accessing the tag store and only accessing the desired way after the tag access completes, thus returning the correct way for the data store. This technique will avoid unnecessary access to large data sub-arrays and hence reduction in power consumption is obtained. Though the phased architectural scheme has a disadvantage of slower execution but it can be employed in cases especially in embedded systems where a little sacrifice in performance is acceptable for reduced power. These cache architectures can be employed at high frequencies as the reduction in power is much higher at high frequencies. The conventional and phased set-associative architectures are simulated and synthesized using Xilinx 8.2i. The results obtained show that phased parallel-access and phased sequential-access architectures provide an average power reduction of 28 % and 35.36% respectively, when compared to conventional parallel-access and conventional sequential-access set-associative cache architectures. The implementation of conventional parallel set-associative architecture with valid bit pre-checking technique also results in 19.88% average power reduction.

6.2 FUTURE SCOPE

The cache performance depends on how fast the memory reference can be performed, while the power consumed by the cache depends on how many unnecessary operations can be eliminated. Considering the above facts the future scope of this work includes the following:

- The design can be further optimized by selecting methods leading to lower miss rate of the cache architecture. The miss rate reduction not only reduces execution time but also optimize power return.
- Prediction mechanisms and re-sizing low power schemes for the cache architecture can be employed to further optimize the work.

List of Publications

Ankita Pandey, Sanjay Sharma "Different Access Mechanisms for Set-Associative Cache Architecture for Reduced Power Consumption", Taylor & Francis :International Journal of Electronics Letters, 2013.

REFERENCES

[1] W. Stallings, "Computer Organization and Architecture" 6th edition, New Jersey, NJ: Pearson Education, 2003.

[2] V.P. Heuring and H.F. Jordan, "Computer Systems Design and Architecture" 3rdedition, India, Pearson Education, 2004.

[3] C.C. Liu, I. Ganusov, M. Burtscher and S. Tiwari, "Bridging the processor-memory performance gap with 3D IC technology," Design & Test of Computers, IEEE , vol.22, no.6, pp.556-564, 2005.

[4] C. Carvalho, "The Gap between Processor and Memory Speeds," Proc. IEEE International Conference on Control and Automation, 2002.

[5] C. Srilatha and 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, Vol.1 (5), pp.581-587, 2009.

[6] A.J. Smith," Cache memories," ACM Computing Surveys (CSUR), 1982.

[7] A. Malik, B. Moyer and D. Cermak, "A Low Power Unified Cache Architecture Providing Power and Performance Flexibility," Int. Symp. on Low Power Electronics and Design, pp. 241-243, 2000.

[8] D.H. Albonesi,"Selective cache ways: on-demand cache resource allocation," Proc. Int. Symp. on Microarchitecture, pp. 70–75, 1999.

[9] M.D. Powell, A. Agarwal, T. N. Vijaykumar, B. Falsafi and K. Roy, "Reducing set-associative cache energy via way-prediction and selective direct-mapping," Proc. 34th annu. ACM/IEEE int. Symp. on Microarchitecture, pp. 54 – 65, 2001.

[10] J. Yang and R. Gupta, "Energy efficient Frequent Value data Cache design," Proc. 35th Annu. IEEE/ACM Int. Symp. on Microarchitecture, pp.197-207, 2002.

[11] C. Zhang, F. Vahid and W. Najjar, "Energy benefits of a configurable line size cache for embedded systems," Proc. IEEE Computer Soc. Annu. Symp., no. 20-21, pp.87-91, 2003.

[12] R. Min, W. Jone and Y. Hu, "Phased tag cache: an efficient low power cache system," Proc. Int. Symp. on Circuits and Systems, vol.2, pp.23-26, 2004.

[13] C.H. Chen and J.S. Chiang, "Low-power way-predicting cache using valid-bit pre-decision for parallel architectures," 19th International Conference on Advanced Information Networking and Applications, vol.2, no. 28-30, pp.203-206, 2005.

[14] C.H. Kim, S.W. Chung and C.S. Jhon, “PP-cache: A partitioned power-aware instruction cache architecture,” Microprocessors and Microsystems, vol. 30, pp. 268-279, 2006.

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

[16] C.Y. Tseng and H.C. Chen, “The Design of Way-Prediction Scheme in Set-Associative Cache for Energy Efficient Embedded System,” WRI International Conference on Communications and Mobile Computing, vol.3, no. 6-8, pp.3-7, 2009.

[17] C. Xu, G. Zhang and S. Hao, “Fast Way-Prediction Instruction Cache for Energy Efficiency and High Performance,” IEEE International Conference on Networking, Architecture, and Storage, NAS, pp.235-238, 9-11, 2009.

[18] C. Zhang, F. Vahid and W. Najjar, “A highly configurable cache for low energy embedded systems,” in Journal ACM Trans. on Embedded Computing Systems, vol. 4, pp.363 – 387, 2005.

[19] M. Ghosh, E. Ozer, S. Ford, S. Biles and H.S. Lee, “Way guard: a segmented counting bloom filter approach to reducing energy for set-associative caches,” in Proc. 14th ACM/IEEE int. symp. on Low power Electronics and Design, pp. 165-170, 2009.

[20] C. Zhang, F. Vahid, J. Yang and W. Najjar, “A Way-Halting Cache for Low-Energy High-Performance Systems,” in Journal ACM Trans. on Architecture and Code Optimization vol.2, no.1, pp.34-54, 2005.

[21] C. J. Janraj, T.V. Kalyan, T. Warrier and M. Mutyam,“Way Sharing Set Associative Cache Architecture,” 25th International Conference on VLSI Design (VLSID), no.7-11, pp.251-256, 2012.

[22] T. Ishihara and F. Fallah, “A non-uniform cache architecture for low power system design,” Proc. Int. Symp. on Low Power Electronics and Design, pp.363 - 368, 2005.

[23] G. Ji, G. Hui and P. Li, “ROBTIC: An On-chip Instruction Cache Design for Low Power Embedded Systems,” 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, pp.419-424, 2009.

[24] M. Soryani, M. Sharifi and M.H. Rezvani, “Performance Evaluation of Cache Memory Organizations in Embedded Systems,” Fourth International Conference on Information Technology, pp.1045-1050, 2007.

[25] M. Zahran, “Cache replacement policy revisited,” Proc. 6th Workshop on Duplicating, 2007.

[26] J.A. Rivers and E.S. Davidson, “Reducing conflicts in direct-mapped caches with a temporality-based design,” Proc. of the International Conference on Parallel Processing Software, vol.3, 1996.

[27] S. Dandamudi, “Fundamentals of Computer Organization and Design,” Springer, 2003.

[28] M.D. Hill and A.J. Smith, “Evaluating associativity in CPU caches,” IEEE Trans. Computers, vol.38, no.12, pp.1612-1630, 1989.

[29] B. Calder, D. Grunwald and J. Emer, “Predictive sequential associative cache,” Proc. of Second Int. Symp. on High-Performance Computer Architecture, no. 3-7, pp.244-253, 1996.

[30] R.E. Aly, B.R. Nallamilli and M.A. Bayoumi, “Variable-way set associative cache design for embedded system applications,” IEEE 46th Midwest Symp. on Circuits and Systems, vol.3, no.27-30, pp.1435-1438, 2003.

[31] M.Y. Qadri, H.S. Gujarathi and K.D. McDonald-Maier, “Low Power Processor Architectures and Contemporary Techniques for Power Optimization – A Review,” Journal of Computers, vol. 4, no 10, pp.927-942, 2009.