A NEW N-WAY RECONFIGURABLE DATA CACHE ARCHITECTURE FOR EMBEDDED SYSTEMS

Ruchi Rastogi Bani

Thesis Prepared for the Degree of

MASTER OF SCIENCE

UNIVERSITY OF NORTH TEXAS

December 2009

APPROVED:

Saraju P. Mohanty, Major Professor
Elias Kougianos, Co-Major Professor
Armin R. Mikler, Committee Member
Ian Parberry, Interim Chair of the
Department of Computer Science
and Engineering
Costas Tsatsoulis, Dean of the College of
Engineering
Michael Monticino, Dean of the Robert B.
Toulouse School of Graduate
Studies

Bani, Ruchi Rastogi. A New N-way Reconfigurable Data Cache Architecture for Embedded Systems. Master of Science (Computer Engineering), December 2009, 76 pp, 15 tables, 28 figures, references, 47 titles.

Performance and power consumption are most important issues while designing embedded systems. Several studies have shown that cache memory consumes about 50% of the total power in these systems. Thus, the architecture of the cache governs both performance and power usage of embedded systems. A new N-way reconfigurable data cache is proposed especially for embedded systems. This thesis explores the issues and design considerations involved in designing a reconfigurable cache. The proposed reconfigurable data cache architecture can be configured as direct-mapped, two-way, or four-way set associative using a mode selector. The module has been designed and simulated in Xilinx ISE 9.1i and ModelSim SE 6.3e using the Verilog hardware description language.

ACKNOWLEDGEMENTS

I would like to extend my gratitude to my major professor Dr. Saraju Mohanty, who has motivated and guided me throughout my studies and thesis work. I would also like to express my unconditional thanks to Dr. Elias Kougianos, my co-major professor for his support, advice and suggestions. I am also grateful to Dr. Armin Mikler for being my thesis committee member and sparing his time to review my work.

I would like to thank my husband, Nishith Bani, and my family for their encouragement and patience throughout my research work. I could not have made it without them. They give me a reason to work very hard. I would also like to thank Dr. Dhruva Ghai and Oluwayomi Bamidele Adamo, for their advice and spending time to guide me. Finally, I would like to thank friendly staff and my colleagues in computer science and engineering department who encouraged me throughout my research work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The need for mobile systems, portable devices, and many other appliances used in our modern life results in a growing demand for embedded computing systems. As this growth occurs at tremendous rate, it reduces the window for time-to-market, which in turn is motivating us to design new core-based system-on-chip (SOC) architectures. In such systems' design, embedded processors (microprocessor cores) are playing a vital role [30]. Several programming languages and electronic design automation (EDA) tools are available currently for embedded processors, which make programming straightforward and offer different solutions to overcome design challenges and simultaneously optimize design metrics. Designers have to maintain a balance of these metrics and compromise between power, cost, performance and time-to-market. Cache memory, a crucial part of embedded systems, is responsible for consuming approx. half of the total power consumption by these systems. Research in design of optimal cache architectures for portable and mobile devices is being actively pursued. According to certain studies, the use of separate data and instruction cache is one way of improving the performance of today's microprocessors. Proper cache architecture can bring down the time overhead of accessing data and instructions from off-chip main memory, thereby reducing power consumption. High gains in performance have been achieved by tuning appropriate cache architectures to the application set of embedded systems. Cache sizes, degree of associativity, block replacement algorithms, write policies, block size (cache line) are the core parameters for optimizing the cache architecture. Suitable selection of these design parameters can enhance cache performance in terms of power consumption, hit ratio, access time and hardware requirements.

1.1 Motivation

Embedded systems have always been cost sensitive. Cache occupies approx. fifty percent of the total area and also accounts for approximately fifty percent of a processor's total power in embedded systems, including both static and dynamic components [5][6]. Thus, cache governs the performance and cost of application specific embedded systems. The direct-mapped (DN) cache architecture is very popular in embedded systems because of its simplicity, faster access time and low power consumption. A DM cache is more energy efficient and uses less power than the same sized two-way or four-way set associative cache since it accesses only one location of tag and data arrays per access [2]. Moreover, a direct-mapped cache has faster access time as it does not require a multiplexer to select the requested data from multiple accessed data items in different sets. Although direct mapped cache has the advantage of consuming less area and power, it suffers from poor performance. One way to improve the performance of such systems is to use set associative cache at the expense of larger area and higher power consumption compared to direct mapped cache. Other ways of improving the performance of direct-mapped cache are discussed in section 2.1. The data cache architectures of most commonly used embedded microprocessors are summarized in Table 1.1.

Table 1-1 Data Cache Associativities of Popular Embedded Microprocessors [2].

Processor	Associativity	Processor	Associativity
Hitachi SH7750S (SH4)	1	Hitachi SH7727	4
NEC VR4181	1	IBM PPC 7603	4
NEC VR4181A	1	Motorola MPC8240	4
NEC VR4121	1	Motorola MPC823E	4
Sun Ultra SPARC lie	1	Motorola MPC8540	4
AMD-K6- IIIE	2	PMC Sierra RM9000X2	4
IBM403GCX	2	PMC Sierra RM7000A	4
IBM Power PC 405CR	2	SandCraft sr71000	4
Intel 960IT	2	SuperH	4
TI TMS320C6414	2	Triscend A7	4
Xilinx Virtex IIPro	2	IBM PPC 750CX	8
Alchemy AU1000	4	IBM750FX	8
ARM 7	4	Motorola MPC7455	8

From Table 1.1, it is clear that the associativity requirement of data cache for almost all the embedded systems is one way, two-way or four-way. In order to match the cost, performance, and power goals with targeted time-to-market, a new N-way reconfigurable data cache is proposed. The proposed design can be configured as direct-mapped, two-way or four-way set associative according to the system's requirement.

1.2 History

Recent advances in chip technology, such as the ability to place more transistors on the same die together with increased operating speeds, has led to a tremendous gap between processor's and main memory's speed. According to Moore's law, there is an annual increase in performance of processor and main memory approximately by 60%

and 10%, respectively [4]. Thus, this gap between processor and main memory speed is growing at a rate of 50% per year as shown in Fig. 1.1.

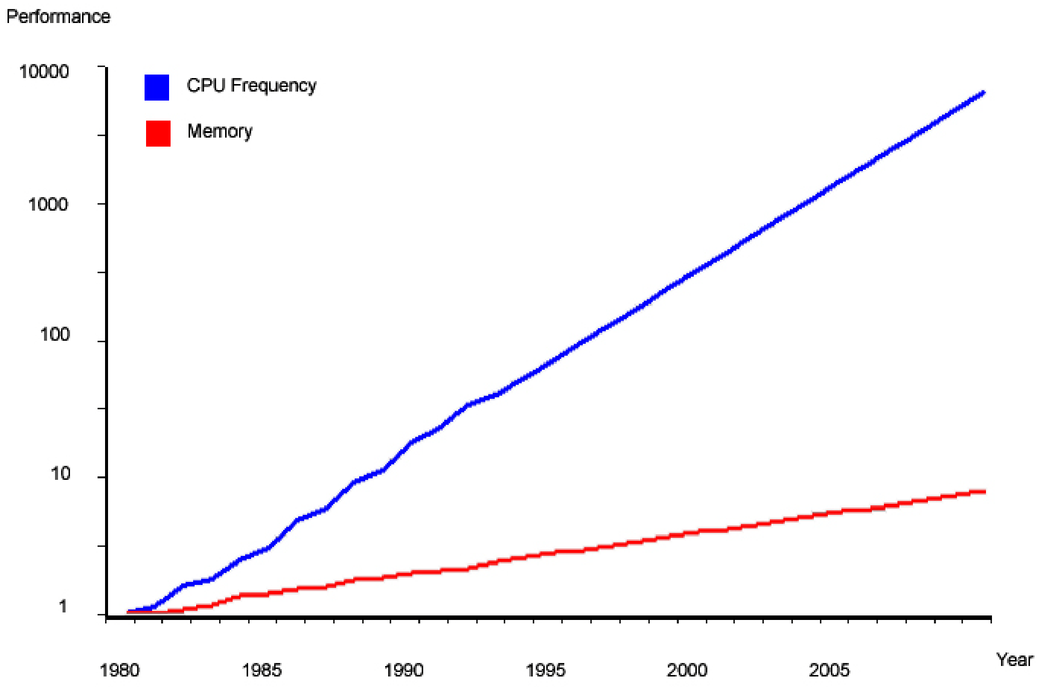


Figure 1-1 CPU-Memory Speed Gap [31]

With advancements in technology, this speed gap is gradually reduced as the processor has to access on-chip cache memory instead of off-chip main memory. Cache is a small on-chip memory situated between a high speed processor and low speed main memory. A cache is implemented using SRAM (Static Random Access Memory) which makes the cache fast, unlike the main memory which comprises of DRAM (Dynamic Random Access Memory). Fig. 1.2 illustrates the memory hierarchy for embedded microprocessors.

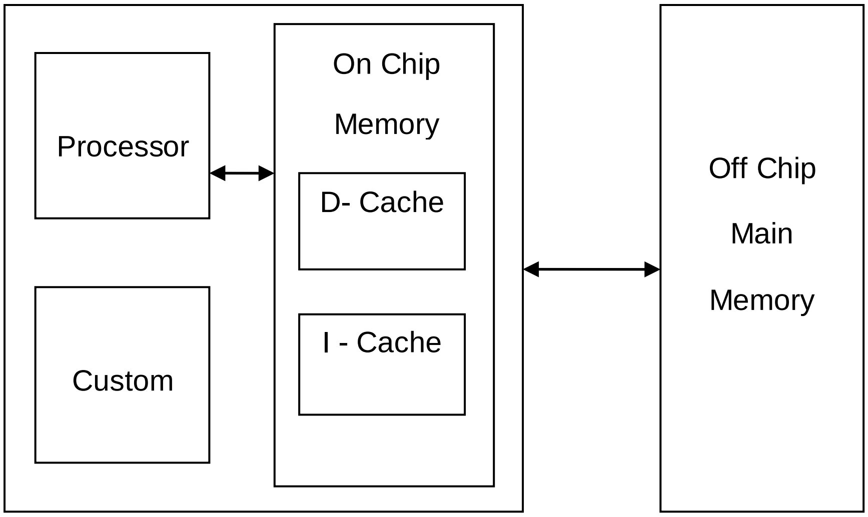


Figure 1-2 Memory Hierarchy for Embedded Microprocessor [35]

As cache is very fast and on-chip, the processor can access it more quickly than main memory. A cache is local memory in a computing system that stores a copy of data and instructions currently used by the processor. The architecture of cache memory is largely determined by the behavior of the application using that cache. Single functioned embedded systems like scanners, fax machines, digital cameras, etc. are designed to execute a small range of well defined tasks in the system's lifetime, requiring a small, high performance, low power cache. In contrast, a desktop computer has to support various applications, like word processors, spreadsheets, CAD software, etc. which need large amounts of cache [14]. Desktop systems afford greater flexibility for the design of cache memories in terms of cache size, associativity, block size, line size, and multi-level cache. Due to limitations on the physical size and energy budget, design of cache for embedded applications is more stringent than those for desktop applications.

1.3 Principles of Cache Memory

At any given time, the processor needs only small amount of data [28]. The cache memory tries to predict the range of memory locations, which the processor will need in the near future and copies the content of these locations in advance. Whenever the processor needs any data, first it attempts to retrieve it from cache and if data is not available there, it has to wait until the data is loaded from main memory to cache. At this time data from nearby locations of the requested address are also copied to cache.

The basic principle behind cache operation is *locality of reference,* also known as *principle of locality*. According to this principle, any program or application running on a processor needs to access only a small portion of their total available address space at a given point of time [26]. There are two basic types of reference locality:

1. *Temporal locality* (locality in time): At a given instant of time, if a particular data item is used, then there is a high probability that it will be required again in near future [33].
2. *Spatial locality* (locality in space): If a data item from a particular memory address is used at a given instant of time, then there is a high probability that data items from nearby addresses will be required soon [33].

1.4 Operation of Cache Memory

Main memory with n-bit address lines has a total space of 2^n words. It is divided into numbers of blocks containing k words each. Thus, it has total $2^n/k$ = MB (main memory blocks). Cache memory is also divided into a number of lines containing k words each. The total number of cache lines (CL) is considerably smaller than the number of main memory blocks (MB). Thus, only few main memory blocks are mapped to cache at any point of time. When a read request is initiated by the processor not present in the cache, then the whole block containing that requested data item is mapped to one of the cache lines [24]. A cache line cannot be uniquely assigned to individual main memory block because of the large number of blocks compared to cache lines. Hence a tag is associated with each cache line to identify the physical address corresponding to that particular line. Fig. 1.3 illustrates the structure of cache and main memory.

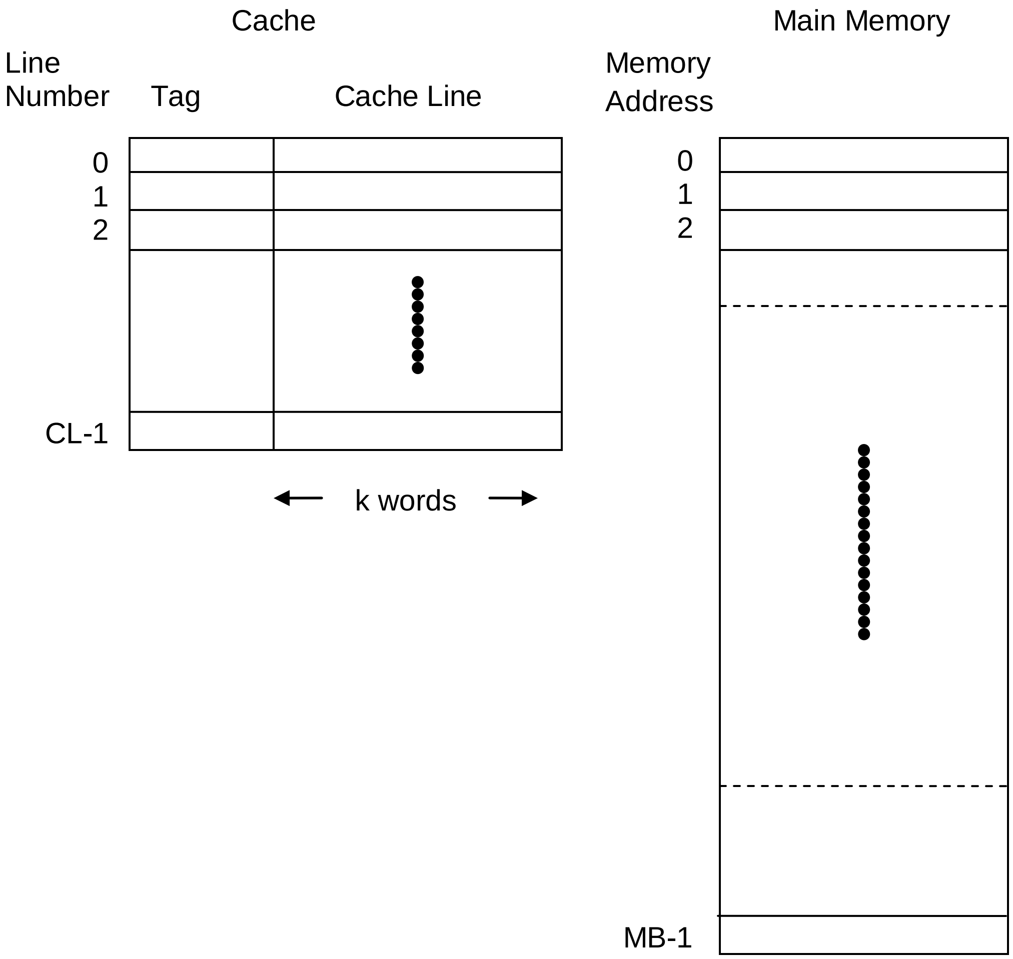


Figure 1-3 Cache and Main Memory Structure [24]

1.5 Cache Memory Performance

The performance of the cache is based on how efficiently it provides the requested data. It is measured in terms of the hit or miss ratio, and access time. When the requested data is found in the cache then it is called a cache hit, otherwise a cache miss. Hit ratio is defined as the number of memory accesses found in cache with respect to the total requested memory accesses. Miss ratio is given as (1 - hit ratio) [28]. The time taken by the cache to provide the requested data in case of a hit is called hit time. When there is a cache miss, then the requested data is fetched from main memory and mapped to cache. The time required for fetching and mapping of the data is called miss penalty. Conventional approaches to improve the performance of cache (by increasing the hit ratio or by decreasing the miss ratio) are generally categorized as: (1) increasing block size and cache size, (2) increasing associativity, (3) cache probing, (4) supplementing the regular cache with victim cache, (5) hardware prefetching of data, and (6) including additional cache hierarchy [25].

1.6 Cache Architecture

Cache memory is responsible for half of the total power and area usage in embedded systems. Effectiveness of the cache memory is determined by its architecture, which means how the cache is mapped to the system's main memory. Mapping reduces the chance that a moved-out block will be used again in the near future. There are three types of mapping: direct, fully associative, and n-way set associative.

1.6.1 Direct-Mapped Cache

In direct-mapped cache (also known as one way-set associative cache), each block from main memory is assigned to one particular cache line. Mapping is based on the following relation [24], C_L = (M_B mod CL), where M_B is the main memory block which is mapped to the cache line number C_L and CL is the total number of lines in the cache. Fig. 1.4 shows how the position of block number 15 from main memory can be placed in three different cache organizations.

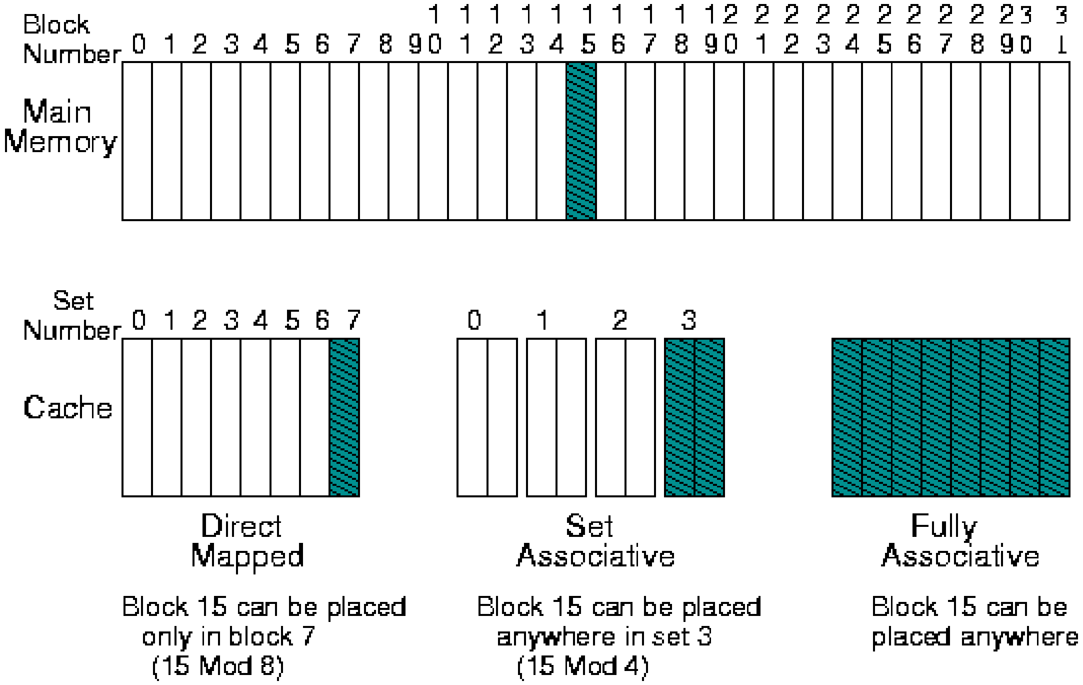


Figure 1-4 Mapping of Main Memory Block 15 in Three Different Cache Architectures [25].

Simple design and comparatively easy hardware implementation are the two benefits of direct mapped cache [29]. However, the only problem associated with this design is the mapping of each block from main memory to one specific cache line. In spite of the fact that cache follows the principle of locality, for some programs or applications, there is a possibility of requiring few data items very frequently which are mapped to the same cache line. These data items will be moved in and out of the cache continuously causing a low hit ratio. In single tasking embedded systems, this situation is unusual but in multi-tasking systems it can arise fairly often and hence deteriorate the performance of direct-mapped cache.

1.6.2 Fully Associative Cache

In fully associative cache each block from main memory can be assigned to any of the cache lines, as shown in Fig. 1.4 and thus provides best hit ratio. At the same time, it suffers from the overhead of cost, complexity, hardware and access time involved in the search of requested address within the cache. All cache blocks are simultaneously compared with the requested address to determine whether a requested memory address is in the cache. In addition to this, extra logic is required to find out which cache line should be replaced when requested data is not available in the cache. Generally, fully associative cache is not used due to its high cost, complexity, and access time.

1.6.3 Set Associative Cache

In set associative cache architecture, the cache memory is divided into a number of small, direct mapped modules, where each module is called a set. A cache comprising of N sets is called N-way set-associative. A particular main memory block can be assigned to any cache line within the set according to the relation [24], $C_{SN} = (M_B \bmod S)$, where M_B is the main memory block which is mapped to the cache set number C_{SN} and S is the number of sets in the cache. In order to determine whether a requested memory address is in the cache, locations within the set indicated by an index field are compared with the desired address.

Set associative cache is basically a compromise between direct mapped and fully associative cache, thus its performance lies between these two cache architectures. Two-way and four-way set associative caches give the best performance in terms of hit ratio and access time for embedded systems [24].

Table 1-2 Comparisons of Mapping Functions

Cache Type	Access Time per Access	Hit Ratio
Direct Mapped	Less	Poor comparatively
N-way Set Associative	Moderate (worsen as N increases)	Good, (improves further as N increases)
Fully Associative	Highest	Best

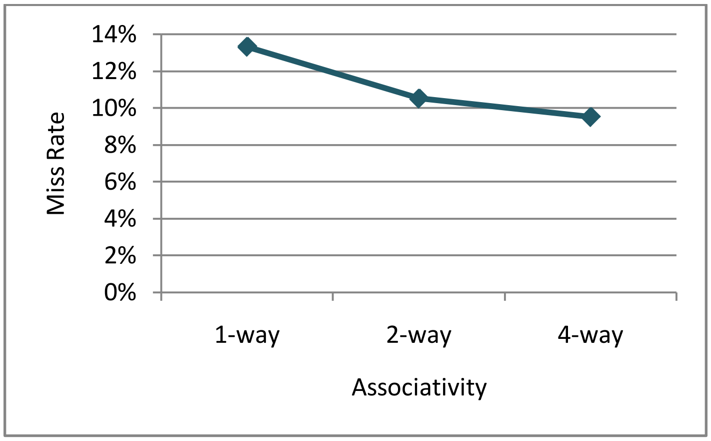
1.6.4 Impact of Cache Associativity

The associativity of cache memory not only affects its performance but also greatly impacts the overall energy consumed. A direct mapped cache consumes less energy per access than a two-way or four-way set-associative cache, because only one tag and one data array are read during an access, rather than two or four arrays. However, for some applications direct mapped cache has a higher miss rate and access time, consuming higher energy for accessing the off-chip main memory. In such cases, increasing the cache associativity is one way to reduce the miss rate and access time, which in turn reduces the overall energy consumed by the cache [2]. Fig. 1.5(a) shows the miss rate for the SPEC92 benchmark and Fig. 1.5(b) shows average memory access time for these miss rates under the assumption that higher associativity will increase the clock cycle [25]. Impact of cache associativity on miss rate and access time is described in Table 1.3.

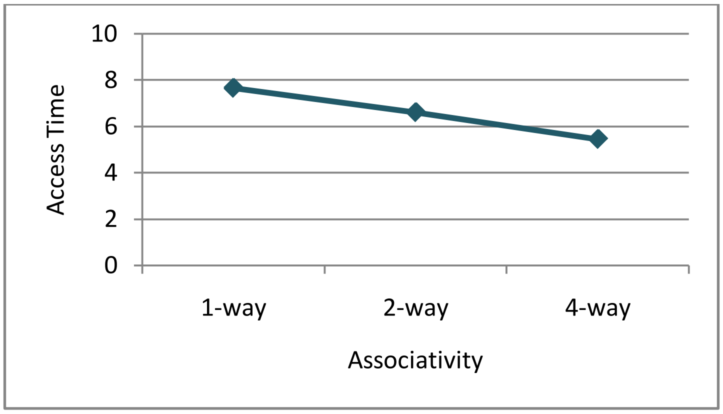
Table 1-3 Impact of Cache Associativity on Miss Rate and Access Time

Associativity	Miss Rate	Access Time (Clock Cycles)
1-way	13.3%	7.65
2-way	10.5%	6.60
4-way	9.5%	5.44

As shown in Fig. 1.5(a), the total miss rate for a one-way 1 KB cache is 13.3%, for a two-way cache is 10.5% and for four-way cache is only 9.5%. It is clear from Fig. 1.5(b) that the average memory access time decreases with increase in associativity.



(a) Miss Rate vs. Associativity



(b) Access Time vs. Associativity

Figure 1-5 Miss Rate (a) and Access Time (b) of SPEC92 Benchmarks on 1KB Data Caches of Different Associativities.

Although more energy per access is required for accessing a four-way set associative cache, due to the additional hardware required to support line replacement, the extra energy may be compensated by reduction in access time and energy that would have been caused by misses. Thus, choosing the appropriate associativity to a particular application is very essential to reduce energy, which motivates the need for an N-way set associative cache.

1.7 Cache Line Replacement Policy

When a cache miss occurs, the requested address line must be placed into the cache. To load a new line into the cache, one of the existing cache lines must be replaced. Cache line replacement policy is a technique for selecting the line which should be replaced when all the lines in set associative or fully associative cache are full [14]. As discussed in section 1.6, there is no choice in a direct-mapped cache, as the requested line can go to exactly one location in the cache. In set associative cache the requested line can go in one of the fixed number of cache locations. Thus, we have a choice as to where to place the requested line and hence a choice of which line to replace. In a fully associative cache, the requested line can go to any location in the cache. Thus all cache lines are candidates for replacement. The proposed design can work either as direct-mapped, which does not need any replacement policy or as set associative, which requires some replacement policy for line replacement in case of cache miss. There are four most common replacement policies [24].

1. *Random* – A random replacement policy selects the cache line to be replaced randomly.

2. *Least Recently Used (LRU)* – A LRU replacement policy replaces a cache line which has not been used for a long time.
3. *First in - First out (FIFO)* – A FIFO replacement policy uses a queue of size N, to keep track of the sequence in which cache locations are being accessed, and replaces the cache line that was loaded in the queue in the most distant time.
4. *Least Frequently Used (LFU)* – A LFU replacement policy replaces the cache line that has been referenced the fewest number of times.

The random replacement policy is simple to implement in hardware but it has been found that this policy would be a poor replacement line selection method. In reality, it performs worse when compared to any of the other three replacement methods mentioned above. The FIFO replacement policy is also easy to implement in hardware via a queue of size N for cache lines. The most commonly used replacement policy is LRU. It is implemented by maintaining a record of access of each cache line within a set, relative to the other lines. According to the principle of locality, a recently accessed cache line is more likely to be referenced again in the near future. Thus, LRU tends to give the best performance among other methods. This policy provides an excellent hit rate but relatively expensive hardware is required for its implementation.

1.8 Cache Write Policies

It is necessary to check whether the cache line has been modified or not, before replacement. If the content of the cache line has not been updated since its arrival in the cache, there is no need to modify the main memory corresponding to this cache line prior to its replacement. When we write to a particular cache line, the data contents of the cache will be modified after the write operation therefore it would have a different value from the corresponding main memory location. In such a case, there is a need to update the main memory before replacement of that cache line. The cache line and corresponding main memory location should hold the same data. There are three different write policies which can be used to ensure that the cache and main memory contents are the same: write-through, write-through with write buffer, and write-back [26].

1. *Write-through:* It is the simplest method to simultaneously update the main memory with modifications in the cache. In this technique the information is immediately written to both cache and main memory during write operations. Easy implementation and consistency among main memory and cache are two major benefits of this policy. On the other hand, the write through policy introduces a significant amount of delay as the processor has to wait until the write operation to the main memory is completed. In spite of this delay, most Intel microprocessors use a write-through cache design.
2. *Write-through with write buffer*: It is one of the solutions to reduce processor delay during write operation. This approach uses a write buffer queue that holds data which is waiting for it to be written into the main memory. As the cache controller writes the data into the cache and into the write buffer queue, it immediately returns a ready signal to the processor, so that it can continue execution. Thus, the processor does not have to wait to write data into main memory and saves valuable clock cycles. If the write buffer queue is

completely full, with pending data to be written on main memory, and a write request comes from processor, then the processor has to wait until it gets an empty location in the write buffer queue. One way to solve this problem is to increase the size of the write buffer queue.

3. *Write-back:* It is another alternative to write through, sometimes also called a posted write or copy back policy. During a write operation the data is written only to the line in cache. This takes less time and allows the processor to continue execution immediately. The updated cache line is written to the main memory only when it is replaced. In order to keep track of a cache line which has been updated and written to main memory before replacement, an extra bit called dirty bit is associated with each line. The status of the cache line is indicated by this bit, whether the line is updated (dirty) while in the cache or, not updated (clean). If the cache line is not updated, there is no need to copy its content before replacement on a cache miss**.** The advantage of the write-back policy is that it can improve system performance when the processor requests write operations faster than writes, which can be handled by main memory. Although write-back is a faster alternative to the write-through policy, it suffers from the major drawback of content inconsistency between main memory and cache, which is called cache coherency problem. Moreover, it is more complicated to implement in hardware when compared to the other two polices.

1.9 Organization of the Thesis

This thesis is organized as follows; Chapter 1 provided an introduction and motivating factors of this research. Related research is discussed in Chapter 2. An explanation of design elements of proposed N-way reconfigurable data cache is given in Chapter 3. Chapter 4 discusses architectures of reconfigurable data cache. In Chapter 5, the prototype of the direct-mapped, two-way and four-way associative data cache along with proposed reconfigurable data cache is explained. Chapter 6 concludes the work and gives suggestions for future research.

CHAPTER 2

RELATED RESEARCH

Cache plays a vital role in any processor based embedded system in order to achieve high performance. Several cache designs have been proposed by researchers either to improve the performance of direct-mapped cache or to reduce the access time and power consumption of set associative cache. In this chapter, a review of the research work done in the field of cache design is presented. Classification of related research is given in Fig. 2.1. Section 2.1 reviews various research works on improving the performance of direct-mapped cache by reducing cache misses. Cache misses incur accessing to off chip main memory which is both power costly and time consuming. Sections 2.2 and 2.3 discuss in brief various research works done to reduce the access time and power consumption in set associative caches. Section 2.4 covers the work done in the area of reconfigurable caches. FPGA technology overview is described in Section 2.5.

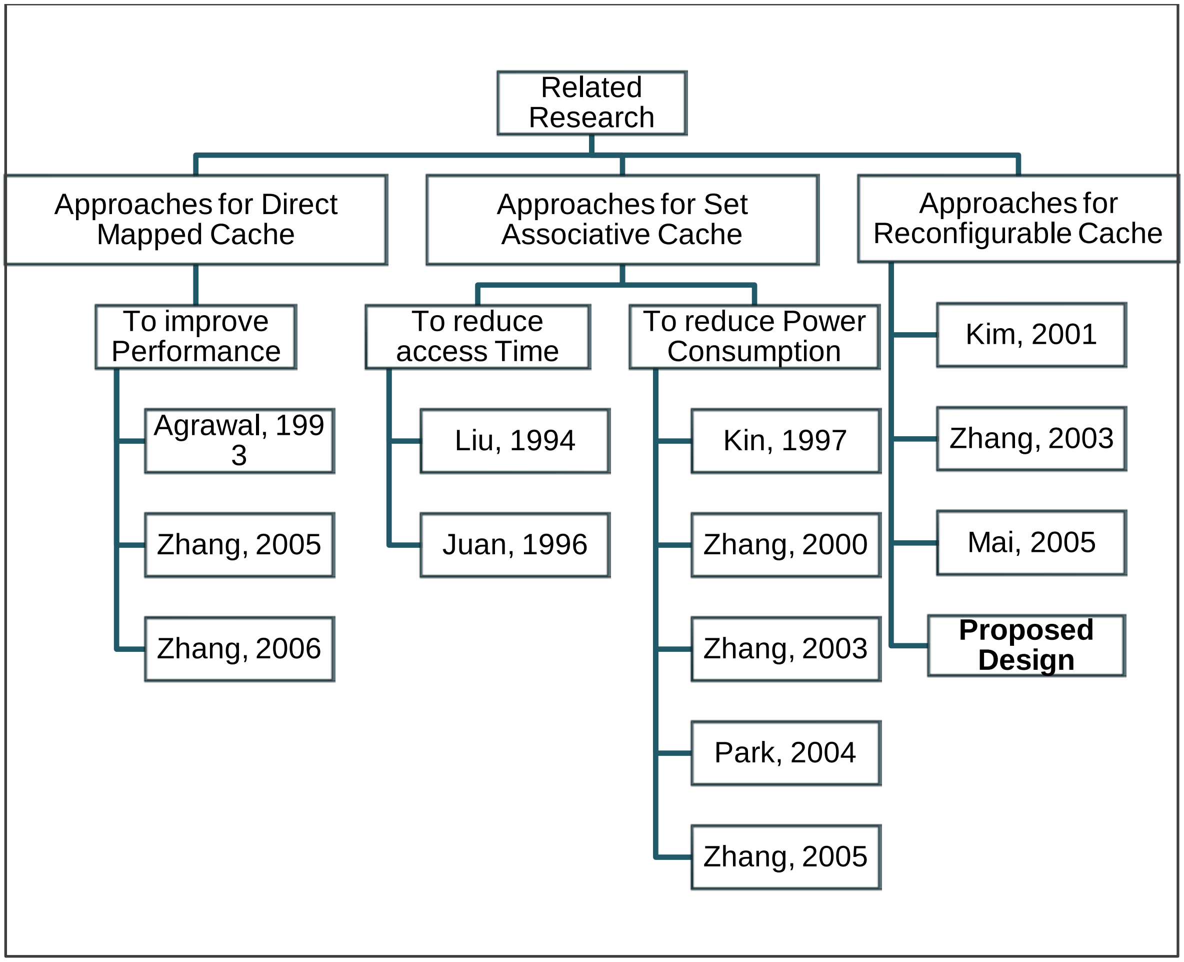


Figure 2-1 Classification of Related Research

2.1 Improving Performance of Direct-Mapped Caches by Reducing Miss Rate

The ever growing need for high performance processors, especially for embedded applications, has motivated the computer designers to study and work on various techniques for improving the performance of direct-mapped caches. As discussed in chapter 1, the most commonly and efficiently used method for increasing performance of embedded processors is to use an efficient cache memory which accounts for approx. half of the total power consumed by the system. Direct-mapped cache is very popular in embedded systems due to its small size and high speed. The key to improving the performance of direct-mapped cache is to reduce the miss rate or to increase the hit rate.

A technique called Column Associative Cache for reducing the miss rate of direct mapped caches has been proposed by Agarwal and Pudar [12]. This design uses an extra bit for dynamically choosing alternate hashing functions and a multiplexer for the address generation to achieve almost the same miss rate as that of a two way set associative cache at a cost of extra hardware. The extension of column associative cache has multiple alternative locations as described in [13].

Efficient direct mapped cache [3] architectures reduce the accesses to overused cache blocks and increase the accesses to underused blocks without increasing the cache access time, associativity, and area. Cache memory is divided into sub arrays in order to achieve the best trade-off among power consumption, area, and performance. In addition to this, conventional decoders have been replaced by SRAM, CAM (Content Addressable Memory), and XCAM (CAM with ‘don’t care’) based configurable decoders to maintain the same access time as that of the original direct mapped design. The small amount of extra power consumption comes from the fact that instead of the original four AND gate decoders, eight rows of five-bit long configurable decoders have been used in this design. Total power overhead due to replacing the original decoder with modified configurable decoders is 0.18% per access. Compared with a conventional two-way set associative cache, efficient cache consumes less energy but achieves almost the same hit rate. The design of this cache suffers from the drawback that it requires simulation of applications in advance, to find out the best decoding scheme for the configurable decoder.

Balanced Cache [10] introduces a programmable decoder and block replacement policy to increase the access to the underutilized cache blocks in direct-mapped cache architectures. The decoder length of a direct-mapped cache is increased by three bits, which is further divided into a programmable decoder (PD) and a conventional non-programmable decoder (NDP). This increase in decoder length potentially reduces access to heavily used blocks to one eighth as compared to the original design. While maintaining the same access time as of a traditional direct-mapped cache, balanced cache design provides the advantage of block replacement at the expense of 10.5% power and 7.98% area overhead.

The design features of the approaches discussed above are tabulated in Table 2.1.

Table 2-1 Table Showing Prior Research Done to Improve the Performance of Direct-Mapped Cache

Name	Column Associative Cache [12]	Efficient Direct – Mapped Cache [3]	Balanced Cache [10]
Year	1993	2005	2006
Miss Rate	As two-way	As two-way	As four-way
Av. Memory Access Time	As two-way	As direct-mapped	As direct-mapped
Hardware & Area Overhead	Due to rehash bit and MUX	Due to SRAM, CAM, XCAM based configurable decoder	7.98% due to programmable decoder
Power Overhead Compared to DM		1.8%	10.5%
Energy Savings Compared to 2 Way-Set Associative		25%	2%
Simulation Approach	Trace driven	Simpler scalar	Simpler scalar

2.2 Reducing Access Time of Set associative Caches

As discussed in chapter 1, set associative caches offer good performance in terms of hit ratio but have relatively higher access time and power consumption than same size direct mapped cache. Several approaches have been proposed to improve the performance of set associative cache by reducing access time.

Cache design with partial address matching [20] uses a hit way predicting technique to reduce the access time. The tag field is divided into two arrays: Main Directory (MD) and Partial Address Directory (PAD). Both Main Directory and Partial Address Directory arrays possess the same parameters such as number of sets and associativity, as the cache. The Main Directory array contains the thirteen most significant bits of the original tag field while the remaining least significant five bits are moved out to the Partial Address Directory. Only the PAD array is compared to predict the hit instead of comparing the full tag field, and this hit is verified by MD comparison. This design is faster because initially only the five bit PAD comparison is required to predict the hit. If the PAD comparison is not correct then only the MD comparison is required to rectify it.

The Difference-bit Cache [18] is a two-way set-associative cache with an access time almost the same as that of a conventional one way set associative cache. This cache design is based on the fact that there is a difference of at least one bit among two tags of a set. This bit is called difference bit and the corresponding bit position in which these two tags differ is called the difference-index. These difference indices and difference bits are key design features and are used for way selection. Area overhead depends on the size of the cache; it is 2% in the case of 8K and 1% in the case of 16K.

The design features of the approaches discussed above are tabulated in Table 2.2.

Table 2-2 Table Showing Prior Research Done to Reduce Access Time Set of Associative Cache

Name	Partial Address Matching Cache Design [20]	Difference Bit Cache [18]
Year	1996	1994
Memory Access Time		As DM Cache
Hardware & Area Overhead	Due to way prediction technique	Due to special decoder and depends on size of cache 2% for 8K 1% for 16K
Associativity	N-way	2-way
Simulation Approach	Benchmark based	Cacti Software

2.3 Reducing Power Consumption of Associative Caches

Filter cache [9] is an unusually small low power direct mapped cache, which is positioned in front of the regular cache. In embedded systems most of the processor's time is spent in executing just a few tasks, so most hits would take place in the small filter cache. Therefore, the power hungry regular cache would not be accessed frequently. Filter cache designs achieve overall power reduction, at the cost of performance loss and extra hardware.

A CAM based cache [22] is designed to reduce the power consumption of set associative caches. Each standard CAM cell consists of ten transistors compare to the standard SRAM cell which consists of six transistors, hence occupying about twice the area of an SRAM cell. CAM-tag caches have comparable access latency, but give lower hit energy and higher hit rates than RAM-tag set-associative caches at the expense of area overhead. CAM-tag caches can provide lower total memory access energy by reducing the hit energy cost of the high associativities required to avoid costly misses. There is no significant performance overhead associated with CAM designs except for a 10% area overhead.

Selective-way-access skewed associative cache [17] is a software controlled low power two-way set associative cache for embedded applications. The design consists of a modified two-way set associative cache, a decoder for the skewing function, and a skewing function controlled way-selecting mechanism. In this approach, both sets cooperate logically with each other: one behaves as a main cache and the other behaves as a shadow cache to reduce conflict misses. Specialized replacement policy in combination with differentiated function based skewing mechanism gives perfect prediction for way selection, thereby obtaining higher hit rate. Moreover, this design has the ability to be converted into one-way set associative for special applications. This cache structure saves up to 55% power over conventional set associative cache.

Way halting cache [19] is a four way set associative cache with a halt tag array. Halt tag array is a small fully associative memory, which stores the lowest four bits of all ways tags. There is simultaneous comparison of the halt tag array with the requested address tag, in parallel with address decoding of data and tag. This design consumes less power as it uses static logic only instead of CAM based dynamic logic which is used in modern highly associative cache. The halting tag array is the key component of the design, which is responsible for overall power savings. Way halting cache achieves about 55% of energy savings over conventional four-way set-associative cache at the expense of area overhead of 2%.

Another method of reducing memory access power of cache is presented in [21]. The design of configurable line size cache is based on the fact that reducing the number of switches per access can reduce overall power per access. Configurable line size is achieved by a configurable counter which is placed in the cache controller and specifies how many words to read at a time from the off chip main memory.

The design features of the approaches discussed above are tabulated in Table 2.3.

Table 2-3 Table Showing Prior Research Done to Reduce Power Consumption of Associative Cache

Name	Filter Cache [9]	CAM based Cache [22]	Configurable Line Size Cache [21]	Selective-way-access Skewed Associative[17]	Way Halting Cache[19]
Year	1997	2005	2003	2004	2005
Performance / Memory Access Time	Reduction by 21%			As conventi-onal set asso-ciative cache	As conventi-onal 4-way
Hardware & Area Overhead		10% due to CAM based tag array	Due to configurable counter	Due to decoder for skewing function and way selection mechanism	2% due to halting tag array
Associativity	Fully		4-way	Set	4-way
Power Savings	Up to 58%		Up to 50%	Up to 55%	Up to 55%
Simulation Approach	IMPACT Lsim	HSPICE		Simple Scalar	Simple Scalar

2.4 Reconfigurable Cache Architectures

In recent years, a lot of research has been conducted towards reconfigurable cache design for embedded applications. In [11] the authors propose a reconfigurable multi-function computing cache architecture based on the fact that some computing related applications use complete cache storage. This reconfigurable cache structure is divided into a regular cache and a configurable cache. The configurable part of the cache architecture can be converted into a functional unit for either of the two computations, Finite Impulse Response (FIR) and Discrete Cosine Transform/Inverse Discrete Cosine Transform (DCT/IDCT). Additional logic is embedded into the cache structure to convert the cache memory into a functional unit. This cache design achieves performance improvement up to a factor of 50 for computational applications with about 10-20% area overhead.

Zhang [2] proposed a highly configurable cache architecture which can be tuned to direct mapped, set associative cache according to the requirements of embedded systems. This architecture aids the cache system to be configured at the optimal configuration. A software controlled technique called way concatenation is used to configure the cache into direct mapped two-way or four-way associative architecture. This structure achieves static and dynamic power savings with very little size and performance overhead. But applications should be profiled before executing to achieve this energy efficiency, since there is no simple method to obtain the optimal configuration dynamically.

A reconfigurable memory architecture is proposed in [16], which can emulate many memory structures including cache, a FIFO and a simple scratchpad memory. This design comprises of configurable memory mats as a fundamental building block. These memory mats consist of a memory array with metadata bits and a small peripheral circuitry. In addition to this, extra functional blocks and flexible status bits have been used to support reconfigurability. This additional logic accounts for 32% area and 23% power overhead.

The design features of the approaches discussed above are tabulated in Table 2.4.

Table 2-4 Table Showing Prior Research Done for Reconfigurable Cache

Name	Multi-function Computing Cache [11]	Way Concatenation Cache [2]	Reconfigurable Memory [16]
Year	2001	2003	2005
Performance Improvement	Up to factor of 50-60 for computational applications	Due to way concatenation and way shutdown circuitry	
Hardware & Area Overhead	~10-20% of base cache memory	Negligible	~15%
Access Time	Increased by 1.6%	As 4-way	Increased by ~10%
Associativity/ Configuration		1,2,4-way	Cache, FIFO, Scratchpad
Power		Dynamic Power savings up to 40% compared to conventional 4-way	Power Overhead ~10%
Simulation Approach		Simple Scalar	

2.5 Field Programmable Gate Array Technology

FPGA is a programmable device that can be configured by the designer after manufacturing. A typical FPGA consists of combinational logic, interconnect, and I/O blocks [38]. Fig. 2.2 shows the basic structure of an FPGA that incorporates all these elements. The combinational logic is further divided into small units called combinational logic blocks (CLBs) or logic elements (LEs). Fig. 2.3 shows the architecture of the combinational logic block (CLB). A typical CLB consists of a look-up table (LUT), which can be configured to a specific type of logic function and a flip-flop, thus providing combinational as well as sequential logic [39]. A typical FPGA contains hundreds or thousands of CLBs.

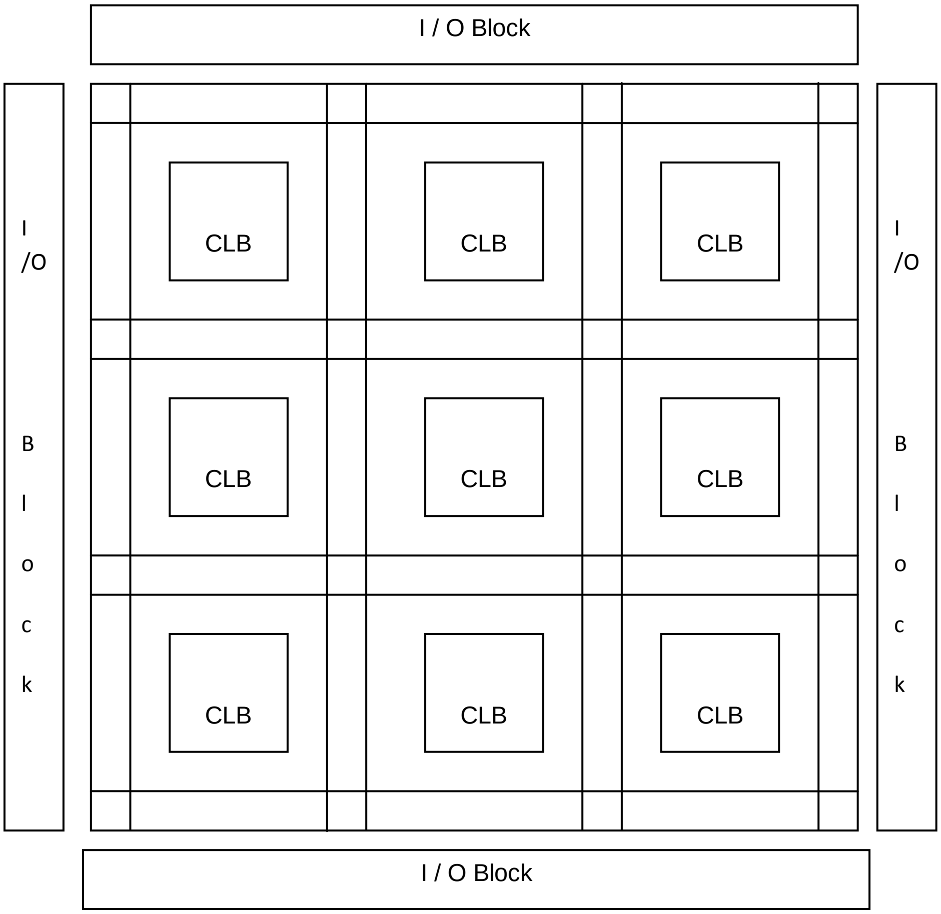


Figure 2-2 Basic Structure of an FPGA

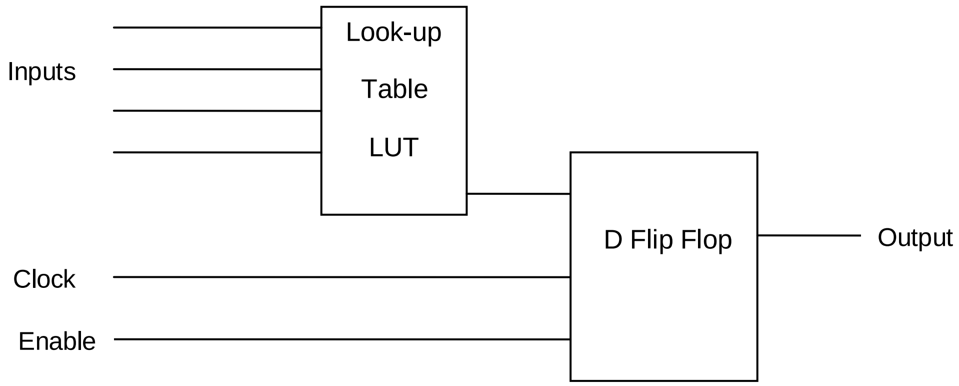


Figure 2-3 FPGA CLB [36]

The interconnections between these CLBs are made using programmable interconnect. FPGAs usually offer various types of interconnect based on the distance between the CLBs which are to be connected. Interconnects are organized in routing/wiring channels that run horizontally and vertically through the chip. I/O blocks are used as an interface between package pins and the internal configurable logic and often provide other features such as high-speed or low power connections. FPGAs are slower than ASICs, and cannot accommodate complex designs, but are ideal to check the functionality of designs. Once the logic design of any module is written using a hardware description language, it needs to be mapped onto the low-level logic blocks of the FPGA. When a program is loaded onto an FPGA, three specific functions are carried out:

1. Mapping: The logic design is placed onto CLBs.
2. Placement: CLBs are placed on the FPGA.
3. Routing: Interconnections between CLBs and I/O blocks.

Verilog and VHDL are common hardware description languages. A single FPGA can be reprogrammed by various HDL or schematic-driven flows [44].

2.6 Contributions of This Thesis

This thesis proposes a new N-way reconfigurable data cache architecture. The proposed design is capable of operating in three configurations: (i) one-way associative (ii) two-way associative, and (iii) four-way associative. We have designed a Mode Selector module within the data cache which allows the data cache to work in any of these three configurations. This N-way reconfigurable data cache has been designed to target embedded systems, as most of the popular embedded systems use direct-mapped, two-way or four-way data cache, as given in Table 1.1. The proposed architecture has been prototyped using Verilog in Xilinx ISE 9.1i and simulated through the ModelSim SE 6.3e simulator.

CHAPTER 3

ELEMENTS OF CACHE DESIGN

The overall performance of cache is determined by the basic elements of cache design, such as cache size, mapping function, replacement policy, write policy, and block size. Table 3.1 gives the design features of the proposed architecture.

3.1 Cache Size

Performance of the cache depends on its size. Table 3.2 shows the effect of cache size on miss rate and access time. The performance of the cache increases with increase in cache size, but at the same time there is an increase in power consumption and per bit storage cost.

Table 3-1 Effect of Cache Size on Miss Rate and Access Time [25]

Cache Size	Miss Rate	Av. Access Time (Clock cycles)
1 KB	13.3%	7.65
2 KB	9.8%	5.90
4 KB	7.2%	4.60
8 KB	4.6%	3.30
16 KB	2.9%	2.45
32 KB	2.0%	2.00

The size of the cache should be small enough to reduce the per bit storage cost and it should be large enough to provide better hit rate. Currently, a range of cache sizes from a few words to several mega words are available. For simplicity we have chosen a cache size of 256 bytes.

Table 3-2 Elements Available for Cache Design and Possessed by the Proposed Reconfigurable Cache Architecture [24].

Elements	Existing	Proposed Architecture
Cache Size	Few bytes to several Kbytes	256 bytes
Mapping Function	Direct Fully Associative N-way Set Associative	Direct Mapped Two-way Set-associative Four-way Set-associative
Replacement Policy	Least Recently Used (LRU) First-in-first-out (FIFO) Least Frequently Used (LFU) Random	Least Recently Used (LRU)
Write Policy	Write through Write Buffer Write Back	Write through
Block Size	Multiple Words	1 Word

3.2 Mapping Functions

3.2.1 Direct-Mapped Cache

In direct mapping, illustrated in Fig. 3.1 [14], the physical address is divided into three fields: tag, index, and offset. The tag bits (most significant bits) are unique identifiers for each memory block, which are currently mapped to the cache. Index (middle) bits specify the location of the requested address within the cache and also determine the size of the cache. Offset (least significant) bits represent a particular word within the cache line. Whenever the processor initiates a request to access a particular memory address, the tag comparator compares the contents of the indexed tag with the tag of the desired address and simultaneously reads the content of the indexed data array. If the requested address is currently in the cache, it generates a high *Match* signal. The cache produces a high on the hit line if the requested address is in cache and valid.

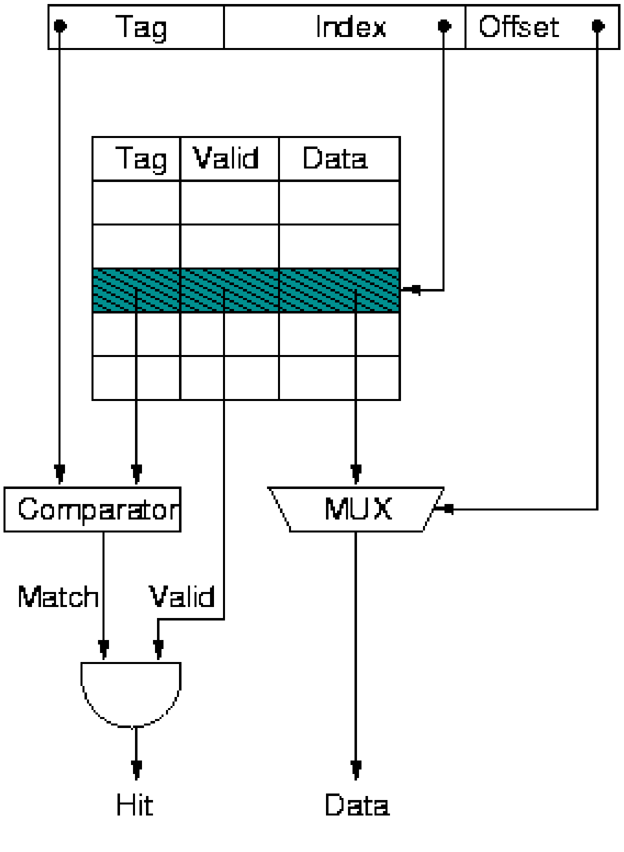


Figure 3-1 Direct Mapped Cache Architecture

3.2.2 Two-Way Set-Associative Cache

In a two-way set associative configuration, the full capacity of the cache is divided into two sets of direct-mapped cache. Each set has its separate tag, and valid and data arrays as shown in Fig. 3.2. Whenever the processor initiates a request to access a particular memory address, both tag comparators compare the contents of the indexed tag with the tag of desired address and simultaneously the cache read the contents of indexed data array from both sets. In case of a case hit, the multiplexer with the help of the encoder routes the corresponding data to the cache output.

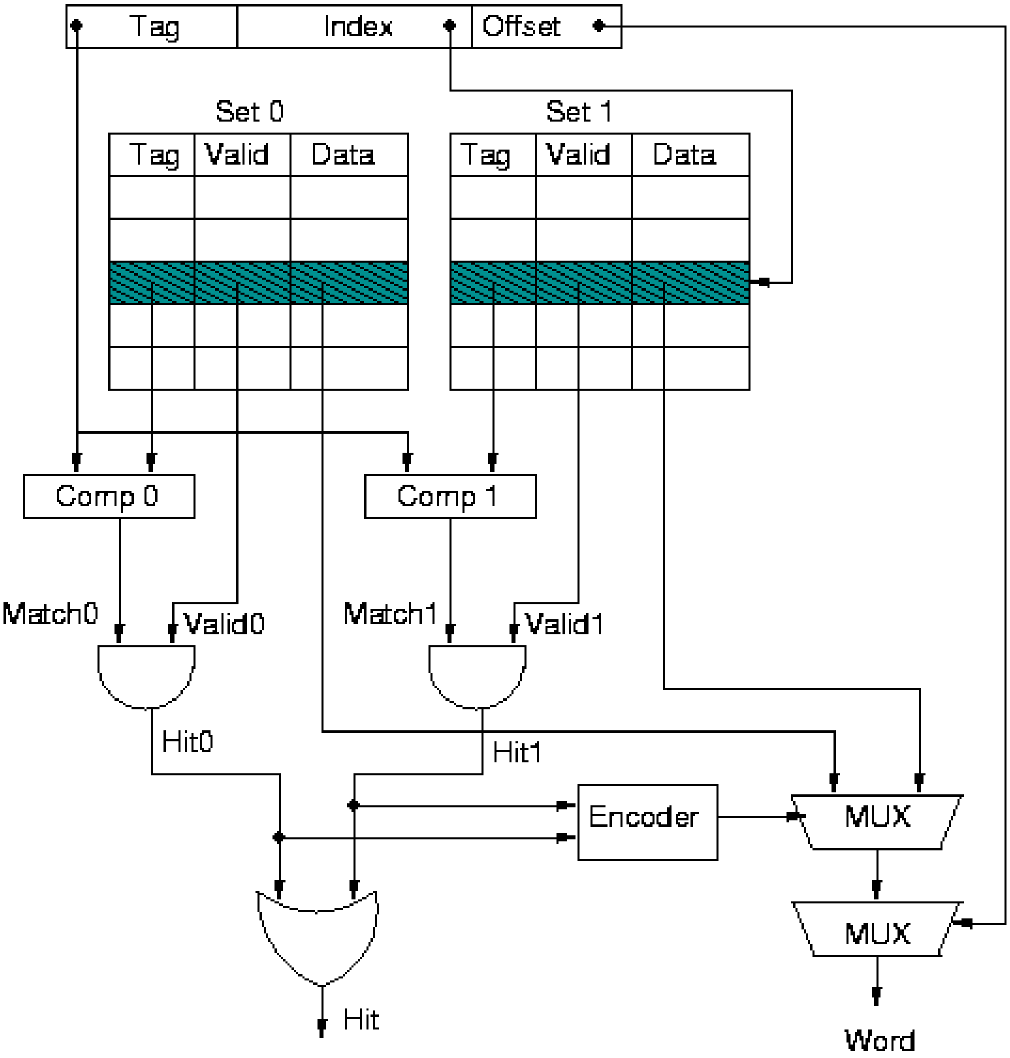


Figure 3-2 Two-Way Set-Associative Cache Architecture

3.2.3 Four-Way Set-Associative Cache

Fig. 3.3 depicts the architecture of a four-way set-associative cache where the physical address is divided into tag field, index field, and line offset field. Being four-way set associative, the cache consists of four sets of tag, valid and data arrays. During an access initiated by the processor, the cache first decodes the address bits of the index field and then concurrently read out the contents from appropriate locations of all sets of tag, valid and data arrays. The cache simultaneously compares the content of all four selected tag locations with the content of physical address' tag field. If any of the selected tags matches the requested address's tag, the corresponding comparator generates a high match signal.

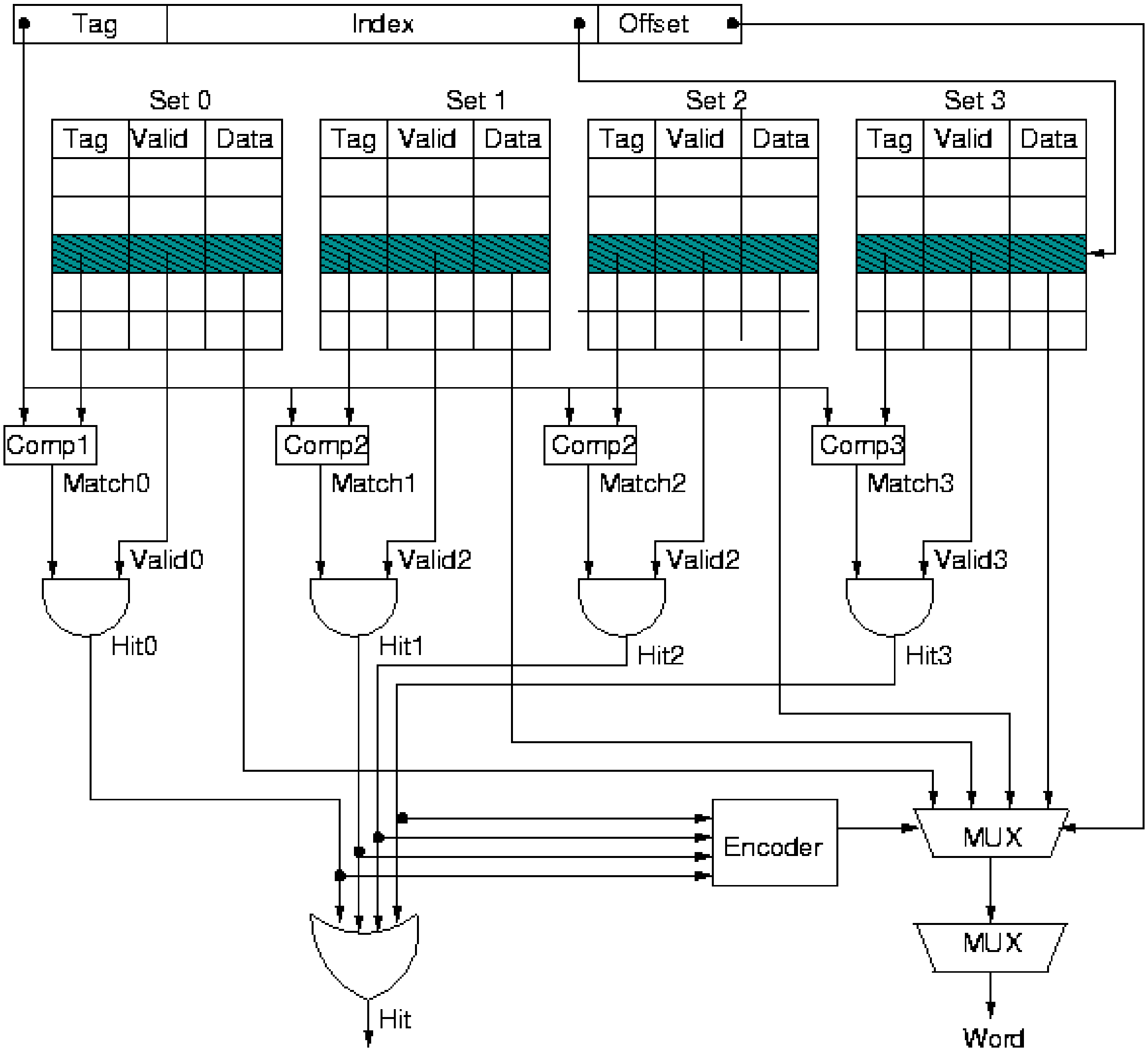


Figure 3-3 Four-Way Set-Associative Cache Architecture.

Valid data contents in the requested location with high match signal produces a high hit corresponding to that set. If there is a hit in any of the sets, the multiplexer in combination with the encoder passes the corresponding data to the cache output.

3.3 Replacement Policy

An efficient replacement policy can reduce the power consumption for set associative cache. The replacement policy basically reduces the number of cache misses, which in turn reduces power consumption [43]. The performance of the replacement policy depends on the accuracy of prediction of future access locations by the cache based on past accesses. LRU is the most popular replacement policy which gives higher performance for set associative and fully associative cache design. There are various ways to implement LRU policy in hardware. The complexity of implementation increases with increase in cache associativity, which additionally increases the delay associated with reaching the best candidate for replacement. Even though a highly associative cache with LRU policy is designed, its performance can be degraded because of inappropriate implementation.

Various ways of implementing LRU policy for an N-way set associative cache are as follows [8]:

1. Square Matrix Implementation
2. Skewed Matrix Implementation
3. Counter Implementation
4. Phase Implementation
5. Link List Implementation
6. Systolic Array Implementation

In the proposed design of reconfigurable data cache, we have implemented the LRU policy using the counter implementation method. In this method, each set has its own LRU unit which has a $\log_2 N$ bits down counter corresponding to each cache line within a set. The value of the counter shows the order in which the associated cache line is used within the set. The larger value indicates the most recently used line, while the smallest value shows the least recently used cache line within a set. Whenever a cache line is accessed, the corresponding counter value is compared with other counters' values within set. The counter values which are greater than the counter value of the currently accessed line, are decremented by one and the counter value of the currently accessed line is set to highest value (N-1). Initially all counters are set to zero.

The proposed design can work either as two-way set associative or as four-way set-associative. In a two-way set associative configuration the realization of LRU policy is straightforward. Only a single bit ($\log_2 2 = 1$) is required corresponding to each line within the set. Whenever a particular cache line within a set is referenced, the corresponding LRU bit is set to 1, to indicate the most recently used line and the LRU bit of the other cache line is set to zero to indicate the least recently used line. The cache line whose LRU bit is currently 0, is selected for replacement whenever a miss is occurred. To implement the LRU policy for a four way set associative configuration, a 2-bit ($\log_2 4$) counter is needed corresponding to each cache line within the set. The counter implementation method provides good performance only for low associativity (two or four) caches. Complexity of implementation increases with increase in associativity [8].

3.4 Write Policy

A write operation can create discrepancy between cache and main memory if only the cache is updated after a write operation [42]. In our design, to prevent this discrepancy we have employed a write-through policy, which ensures synchronization of cache and main memory during write operation.

3.5 Block/Line Size

Another element of cache design is the size of cache line or main memory block. It indicates the number of words per cache line. Whenever there is a cache miss, the whole block of main memory containing the requested word is mapped to cache. According to the reference of locality, the hit ratio increases with increase in line size. At the same time, if block size is increased further, the hit ratio will start to decrease because a chance of using newly fetched block is less than the one which has been replaced.

CHAPTER 4

PROPOSED DESIGN OF RECONFIGURABLE DATA CACHE FOR EMBEDDED SYSTEMS

4.1 Overview of the Proposed Design

A data cache memory lies between the processor and the main memory [7]. Fig. 4.1 shows the high level block diagram of the proposed design along with the signals that the cache needs to communicate with the processor and main memory interface. The processor interface consists of address bus (PAddress), data bus (PData), and three control signals ($\mathrm{PR}\overline{\mathrm{W}}$, PStrobe, PReady). The processor starts a bus transaction when PStrobe is high and a requested address is placed on the address bus. The cache sends a PReady signal to the processor when the bus transaction is completed. The $\mathrm{PR}\overline{\mathrm{W}}$ signal is low for write operation and high for read operation. The main memory interface consists of address bus (MAddress), data bus (MData), and three control signals ($\mathrm{MR}\overline{\mathrm{W}}$, MStrobe, MReady).

In order to access the main memory, the requested address is first placed on MAddress bus along with MStrobe and $\mathrm{MR}\overline{\mathrm{W}}$ control signals. The $\mathrm{MR}\overline{\mathrm{W}}$ signal is low for write operation and high for read operation. MReady is used to signal the cache memory that the bus transaction is completed by main memory. Two global signals, Clk and Reset are used to synchronize the cache operation with the processor. There is an additional signal called mode, which makes the cache reconfigurable. Depending upon the value of mode signal, the proposed design of reconfigurable data cache can work in any of the following three configurations:

1. Direct-Mapped Cache

2. Two-Way Set Associative Cache

3. Four-Way Set Associative Cache

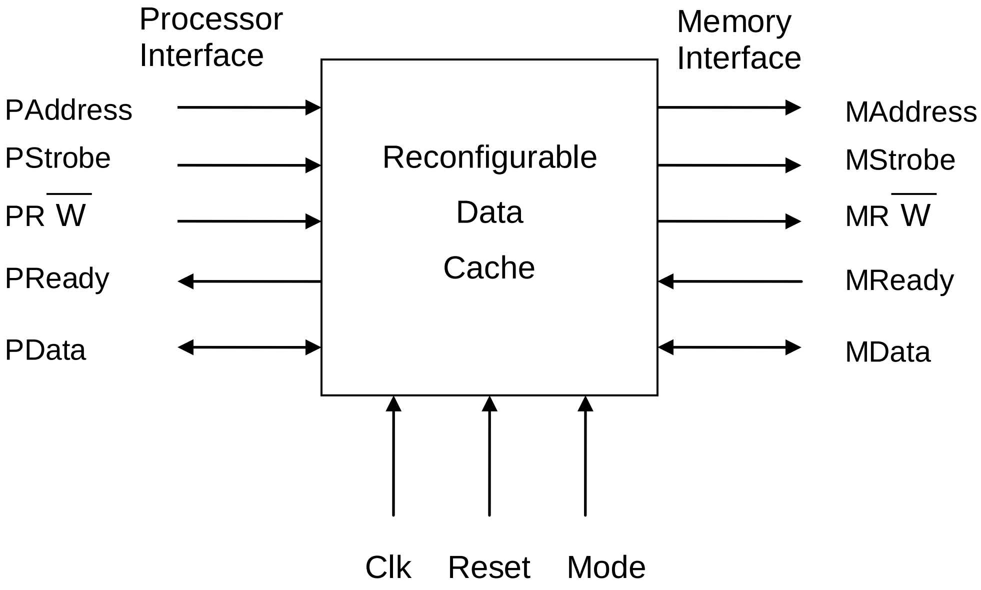


Figure 4-1 High Level View of Proposed Architecture of Reconfigurable Data Cache.

4.2 Architecture of the Reconfigurable Data Cache

We have used top down approach to design our module. The top-level design module shown is further divided into sub modules to build the main module as shown in Fig. 4.2. The proposed reconfigurable cache consists of six sub modules:Tag RAMs, Valid RAMs, Data RAMs, Line Replacement unit, Mode Selector Unit, Cache Controller.

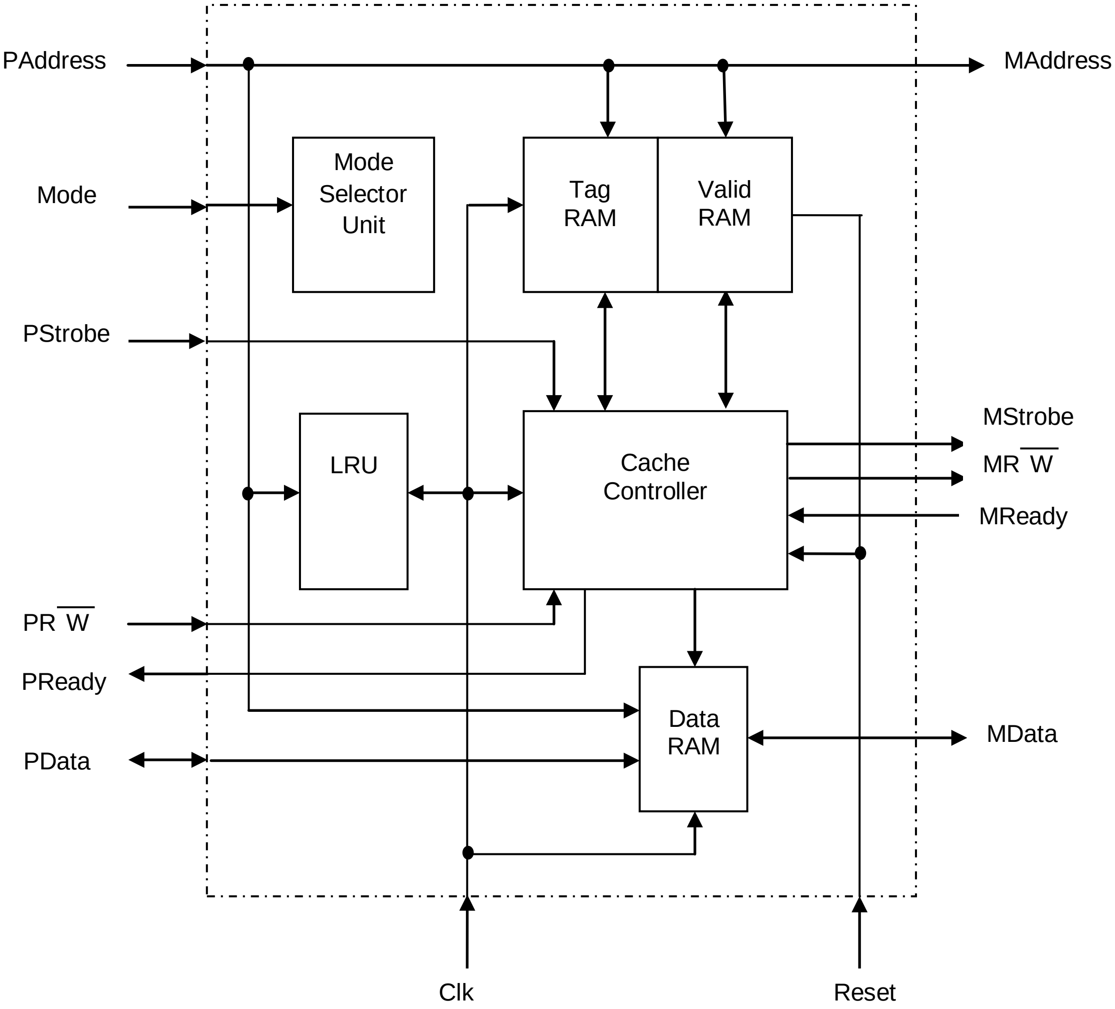


Figure 4-2 Sub-Modules of Proposed Design

The detailed architecture of proposed design is shown in Fig. 4.3, which consists of sub modules along with connecting logics like Encoder, Tag Comparators, and several Multiplexers. For simplicity we have chosen the following:

1. 16-bit address bus, which gives us a total address space of 64K as main memory
2. 256 bytes of cache, that means only 8 least significant bits are required to address the cache and
3. Data bus of 8 bit

The main memory is divided up into 256 blocks of 256 bytes each, where each block is mapped to the cache. Because only 8 address bits are needed to identify the address in cache, the 16 bit physical address is divided into 8 bit tag field and 8 bit index field. Due to reconfigurable architecture, total cache of 256 bytes is divided into four sets of 64 locations each. Thus, we have used four sets of Tag, Valid, and Data RAMs in this design.

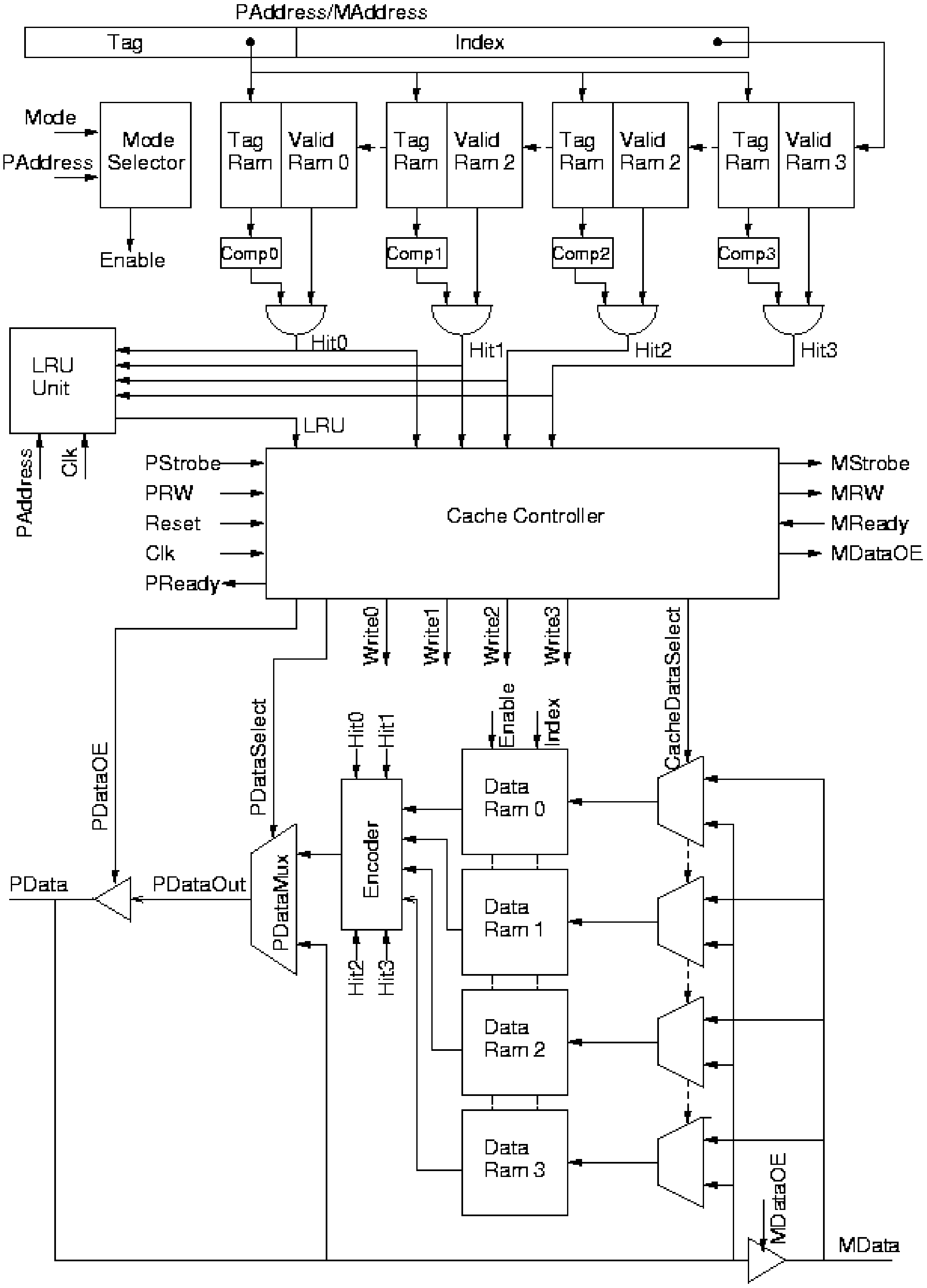


Figure 4-3 Architecture of Reconfigurable Data Cache

1. *Tag RAM* - Contains the TAG fields of the physical addresses which are currently mapped into the cache.
2. *Valid RAM* - Contains a valid bit associated with each location of cache memory. This valid bit indicates whether the cache entry is valid or not. Initially, all entries are set to invalid. As the data contents are moved from main memory to cache, the valid bit corresponding to them is set to valid.
3. *Data RAM* - Contains the data contents of physical addresses which are currently mapped to cache.
4. *Tag Comparators* – When a data access request is initiated by the processor, then all tag comparators simultaneously compare the content of tag array indicated by index field with the requested address's tag field. If any of the tag arrays hold the requested address, the corresponding tag comparator generates the active high match signal.
5. *Line Replacement Unit* – When a cache miss occurs, the line replacement unit determines which line should be removed from the cache. According to the mode selection signals, this unit will replace the least recently used (LRU) line from the requested address in case of cache misses.
6. *Mode Selector* - The data cache can be configured as one, two and four way set associative according to input value at two bit mode terminal. In order to generate the enable control signal for each of the four set, these two mode bits are combined with A9 and A8 bits of physical address. Operation of the mode selector unit is summarized in Table 4.1. Partition of the total cache memory space under three modes is shown in Fig. 4.3

Table 4-1 Mode Selector Operation

A_9 A_8	Direct Mapped	2 Way	4 Way
	Mode – 00	Mode - 01	Mode - 11
0 0	S_0	S_0, S_2	S_0, $S_{1,}$ S_2, S_3
0 1	S_1	S_1, S_3	S_0, $S_{1,}$ S_2, S_3
1 0	S_2	S_0, S_2	S_0, $S_{1,}$ S_2, S_3
1 1	S_3	S_1, S_3	S_0, $S_{1,}$ S_2, S_3

(i) When mode = “00”, only one of the four output signal is set to high for a particular address. Thus, only one set of data, tag and valid array will be active and cache behaves as one way set associative or direct-mapped.

(ii) With mode = “11”, all the four output signals will set to high which in turn activate the associated data and tag array. As all four sets are active in this case, cache works as a four way set associative.

(iii) When mode = “01”, cache works as a two-way set associative, because two sets of data, tag, and valid arrays will be active for any given address.

(iv) Mode “10” reserved for future use.

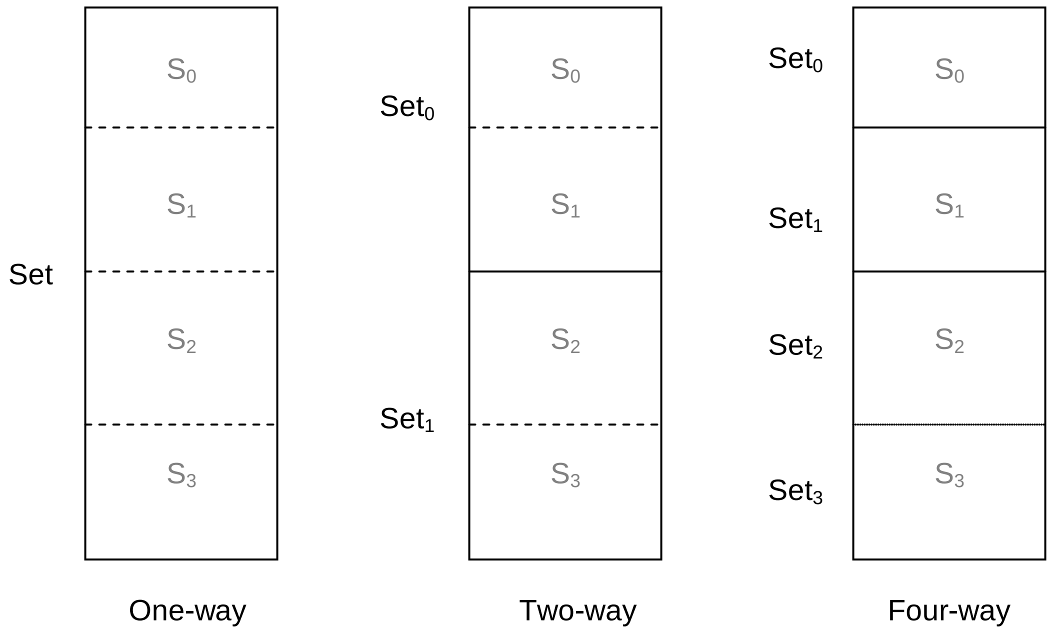


Figure 4-4 Partition of Available Cache into Three Modes

7. *Cache Controller* – It controls all operations within the cache and is implemented using a finite state machine as shown in Fig. 4.4. Control signals asserted during each state are summarized in Table 4.2.

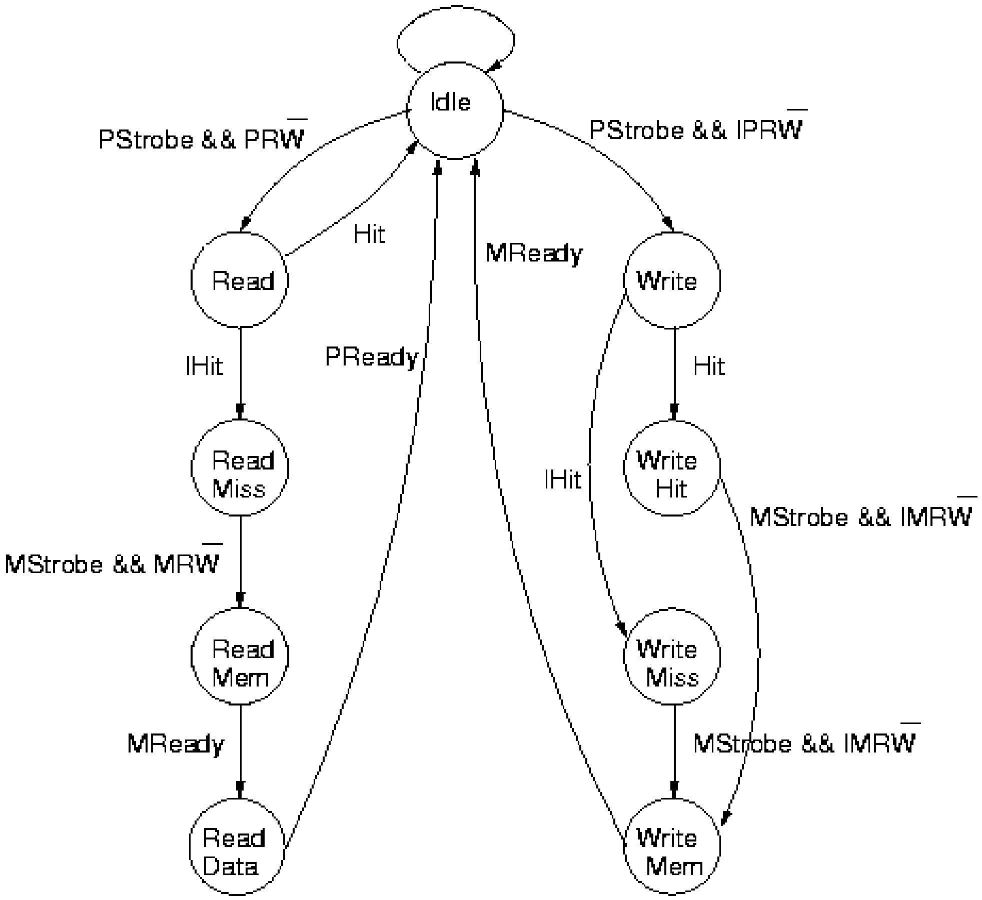

Idle
PStrobe && PR$\overline{W}$
PStrobe && !PR$\overline{W}$
Hit
MReady
Read
Write
!Hit
PReady
Hit
Read Miss
!Hit
Write Hit
MStrobe && !MR$\overline{W}$
MStrobe && MR$\overline{W}$
Read Mem
Write Miss
MReady
MStrobe && !MR$\overline{W}$
Read Data
Write Mem

Figure 4-5 Cache Controller State Machine

(i) *Idle State:* No memory access and processor is idle in this state. Controller will remain in idle state until some read or write operation is requested by the processor. If a read request is initiated by processor, control transfers to *read* state or if a write request is initiated by the processor, control transfers to *write* state.

(ii) *Read State:* In this state, the cache is checked for availability of the requested address, as the processor initiates a read operation**.** If the requested address is currently in cache, a cache hit occurs and control returns to *idle state*

during next active clock. Otherwise, a cache miss occurs and control transfers to *readmiss* state to initiate main memory access.

(iii) ReadMiss State: Main memory read access is initiated by the cache controller and control handed over to *readmemory* state.

(iv) ReadMemory State: Controller has to wait until main memory finds the requested address and reads the data from that location. Once main memory loads the data contents on data bus, it asserts a ready signal to controller and control transfers to *readdata* state.

(v) ReadData State: Now requested data is available on data bus. Write this data to least recently used cache line and simultaneously load it to processor's data bus to complete the read request. After the completion of processor's initiated read operation control go back to *idle state.*

(vi) Write State: In this state, cache is checked for availability of the requested address, as processor initiates a write operation**.** If the requested address is currently in cache, a cache hit occurs and control returns to *writehit state* during next active clock. Otherwise, a cache miss occurs and control transfers to *writemiss* state. Whether it is a write hit or miss, data must be written to cache and simultaneously updated to main memory also.

(vii) WriteHit Data: On a cache hit for write operation**,** controller stimulates the write control signal of data cache to write the data contents sent by the processor and also initiates write through for main memory. Control transfers to *writedata* state on next active clock.

(viii) *WriteMiss State:* On a cache miss for write operation, data contents sent by the processor are written to least recently used cache line and associated tag and valid rams are also updated. Controller initiates write through policy for main memory and transfers control to *writedata* state on next active clock.

(ix) *WriteData State:* Controller has to wait until main memory completes the write operation and sends back a ready signal to controller. After completing the requested write operation, controller will come back to *idle* state.

Table 4-2 Active High Control Signal During Each State

State	Active High Control Signals
Idle	None
Read	PStrobe, PR$\overline{W}$, PDataOE, PReadyEnable, Hit (in case of read hit)
ReadMiss	PR$\overline{W}$, PDataOE, Miss, Mstrobe,
ReadMemory	PR$\overline{W}$, PDataOE, MR$\overline{W}$
ReadData	PR$\overline{W}$, MR$\overline{W}$, PDataOE, PDataSelect, CacheDataSelect, Ready
Write	PStrobe
WriteHit	Hit, MStrobe, MDataOE
WriteMiss	Miss, MStrobe, MDataOE
WriteMemory	MStrobe, Ready

CHAPTER 5

PROTOTYPING AND SIMULATION RESULTS

This chapter presents the simulation and synthesis results of the reconfigurable data cache. The mode selector unit has been considered as the basis of the proposed architecture, which provides the reconfigurability of the design. A trace file of 20 different test cases has been used for functional simulation of the design. In order to study the comparative performance of the proposed design, direct mapped, two-way and four-way set-associative caches are also implemented on the same target device.

5.1 Basic Design Considerations

A typical design flow for designing VLSI systems with the Verilog HDL is shown in Fig. 5.1. The design flow usually begins with design specification and then moves to behavioral and structural level (gates and registers). Finally, the physical description of the system is obtained [34].

There are two basic approaches to design VLSI circuits: a top-down approach and a bottom-up approach [34]. In top-down approach, we first define top-level design module and then find the sub-modules required to build this main module. The bottom-up approach is reverse of top-down approach, where we first define the basic building blocks and then build main design module using these blocks. A digital designer can chose any of these two approaches. However, a top-down approach is always preferred as system level design can be done through HDL design, which can be converted into RTL, gate and physical level with the help of EDA tools. It is easier to decode and fix any error or fault at the system level. This thesis has performed the system level implementation of the reconfigurable data cache in Xilinx ISE 9.1i and simulation using ModelSim. The code is written in Verilog on windows XP platform.

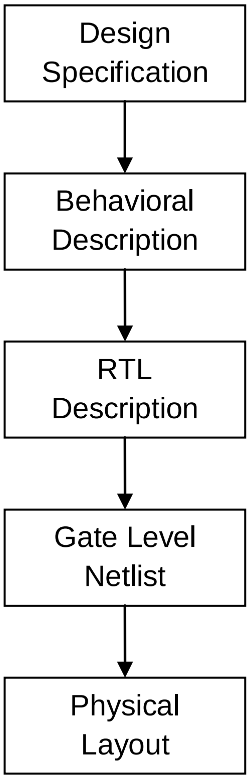


Figure 5-1Typical Design Flow

Verilog is a hardware description language which can be used to model a digital module, which can be as simple as a logic gate or as complex as a complete electronic system. Once the Verilog code is written, it can be loaded onto an FPGA board and implemented, based on its complexity.

5.2 Elements of Verilog

Constructs of the Verilog language are designed for describing hardware modules and primitives [23]. The basic building block in Verilog is module. Module consists of keyword ***module*****,** which is the name of design unit, list of I/O ports, statements to describe design functionality, and keyword *endmodule*. Design within each module can be defined at four levels of abstraction, in order to fulfill design specifications. The four levels of abstractions are as follows [34].

1. *Behavioral or Algorithmic level:* It is the highest level of abstraction in Verilog HDL design. At this level, module is described in terms of algorithm.
2. *Dataflow level*: At this level, the module is implemented using hardware registers.
3. *Gate level*: At this level, the design in implemented using logic gates.
4. *Switch level*: It is the lowest level of abstraction in Verilog HDL design. At this level, module can be implemented using switches, storage nodes and interconnections between them.

Verilog provide the flexibility to mix and match all four levels of abstractions in a single module. Generally the design is technology independent and more flexible at higher level of abstraction. It becomes inflexible and technology dependent at lower levels of abstraction.

Once Verilog code is written to run on an FPGA, a stimulus or testbench is required for functional verification. A stimulus block serves two main purposes

1. It generates test cases for simulation

2. It applies test cases to the design under test (DUT) and also collects the output responses, which is compared with the expected output.

5.3 FPGA Prototyping

The architecture of reconfigurable data cache was modeled using Verilog and the functional simulation was carried out using Modelsim SE 6.3 e. The code is written in a hierarchical fashion and is a combination of structural and behavioral type of coding. This Verilog code was compiled using Xilinx ISE 9.1i. The implementations are targeted for Xilinx's Virtex-5 family of FPGAs. The Virtex-5 family of FPGA from Xilinx is built in a 65-nm copper CMOS technology. The Virtex-5 Configurable Logic Blocks (CLBs) are based on 6-input look-up tables and a flip-flop. The CLBs are the main logic resources for implementing sequential as well as combinatorial circuits. In Virtex-5 FPGA family, each CLB contains a pair of bit slices; each bit slice further consists of four 6-input look-up tables and four flip-flops, for a total of eight 6-input look-up tables and eight flip-flops per CLB [45]. Complete synthesis of all Verilog modules is performed along with its mapping, placement, and routing.

The RTL schematic and functional simulation result of the direct mapped cache is shown in Fig. 5.2 and 5.3 respectively. For functional verification of the designed cache module in direct mapped mode, a trace file of 20 different test cases has been applied through a driver. The simulation waveform shows only one data access, which is a cache read miss.

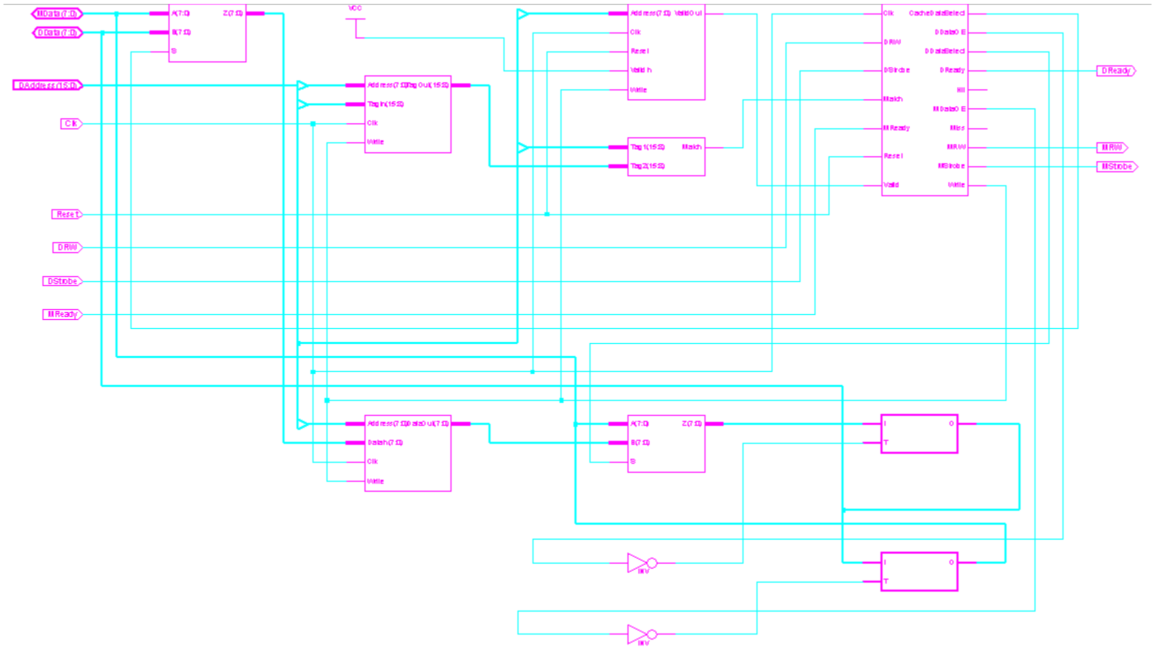
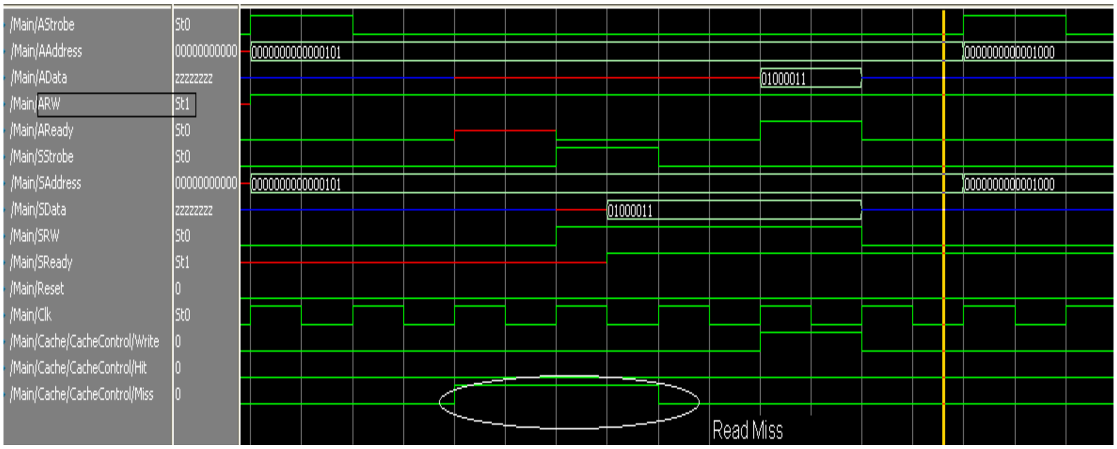


Figure 5-2 RTL Schematic of Direct Mapped Cache



```
/Main/AStrobe                    St0
/Main/AAddress                   00000000000
/Main/AData                      zzzzzzzz
/Main/ARW                        St1
/Main/AReady                     St0
/Main/SStrobe                    St0
/Main/SAddress                   00000000000
/Main/SData                      zzzzzzzz
/Main/SRW                        St0
/Main/SReady                     St1
/Main/Reset                      0
/Main/Clk                        St0
/Main/Cache/CacheControl/Write   0
/Main/Cache/CacheControl/Hit     0
/Main/Cache/CacheControl/Miss    0
```

Figure 5-3 Simulation Waveform of Direct Mapped Cache Showing Read Miss

Fig. 5.4 shows the RTL schematic of two-way set-associative cache. The same trace file is used for functional verification. Fig. 5.5 depicts the data access which is a read hit, during functional simulation.

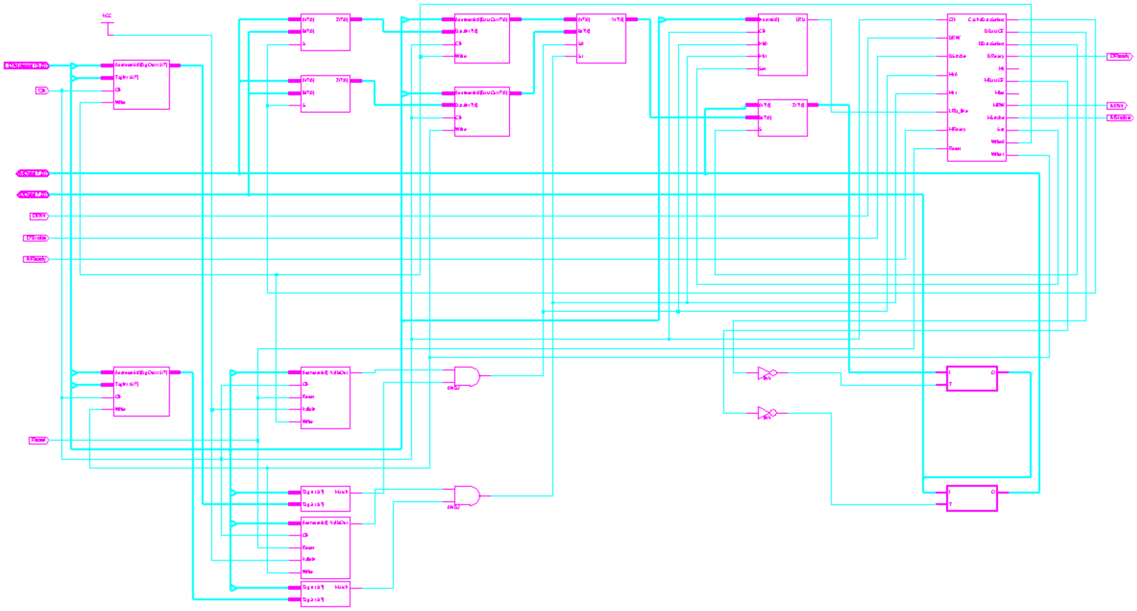
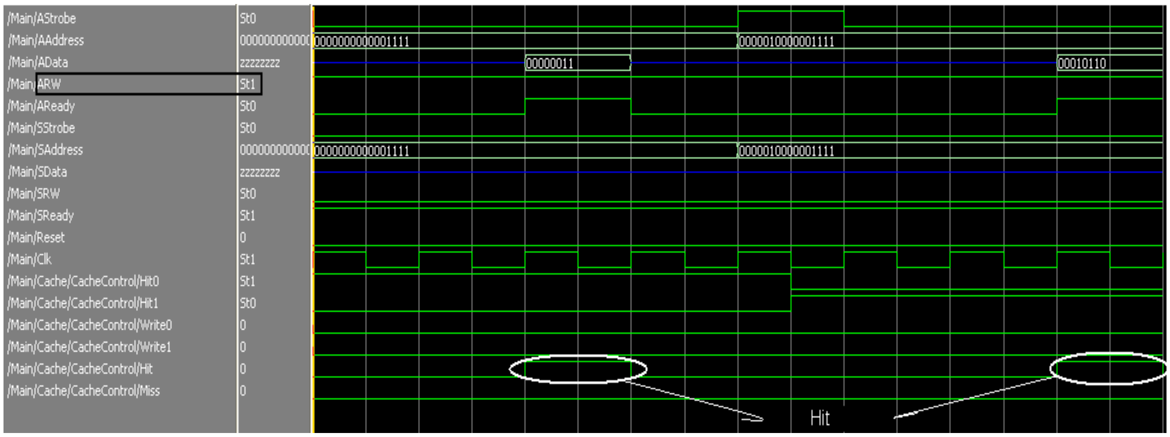


Figure 5-4 RTL Schematic of Two-Way Set-associative Cache



/Main/AStrobe	St0
/Main/AAddress	000000000000
/Main/AData	zzzzzzzz
/Main/ARW	St1
/Main/AReady	St0
/Main/SStrobe	St0
/Main/SAddress	000000000000
/Main/SData	zzzzzzzz
/Main/SRW	St0
/Main/SReady	St1
/Main/Reset	0
/Main/Clk	St1
/Main/Cache/CacheControl/Hit0	St1
/Main/Cache/CacheControl/Hit1	St0
/Main/Cache/CacheControl/Write0	0
/Main/Cache/CacheControl/Write1	0
/Main/Cache/CacheControl/Hit	0
/Main/Cache/CacheControl/Miss	0

Figure 5-5 Simulation waveform of Two-Way Associative Cache Showing Read Hit

The RTL schematic and functional simulation result of the four-way set-associative cache is shown in Fig. 5.6 and 5.7 respectively. For functional verification of the design, the same trace file has been applied through a driver. The simulation waveform shows only one data access, which is a write miss.

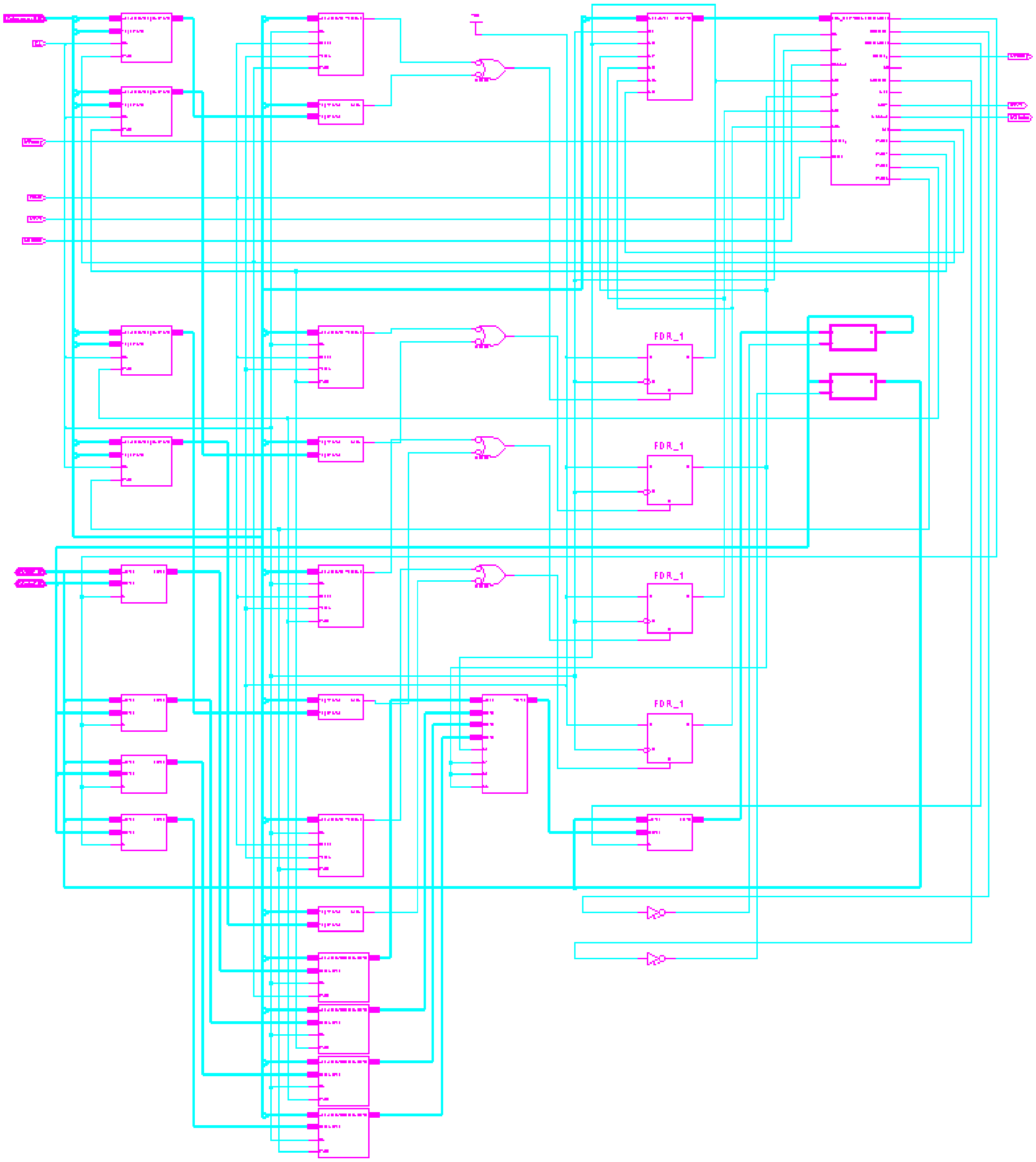


Figure 5-6 RTL Schematic of Four-Way Set-associative Cache

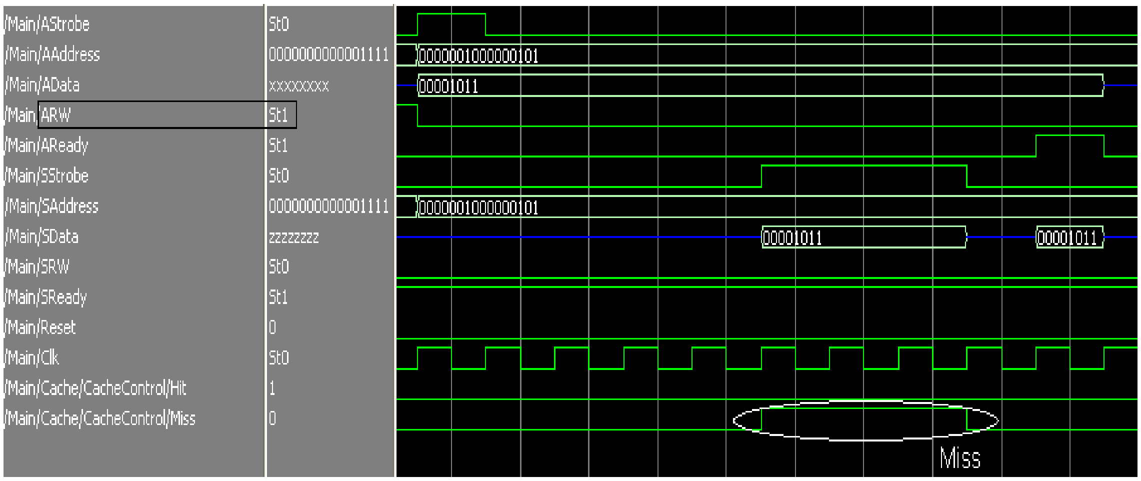


Figure 5-7 Simulation Waveform of Four-Way Set-Associative Cache Showing Read Miss

The RTL schematic of the reconfigurable cache is shown in Fig. 5.8. For functional verification of the designed cache module in all three different modes, the same trace file of 20 test cases has been applied through a driver. The simulation waveforms of the reconfigurable cache in direct mapped, two-way and four-way mode are shown in Fig. 5.9, Fig 5.10 and Fig 5.11.

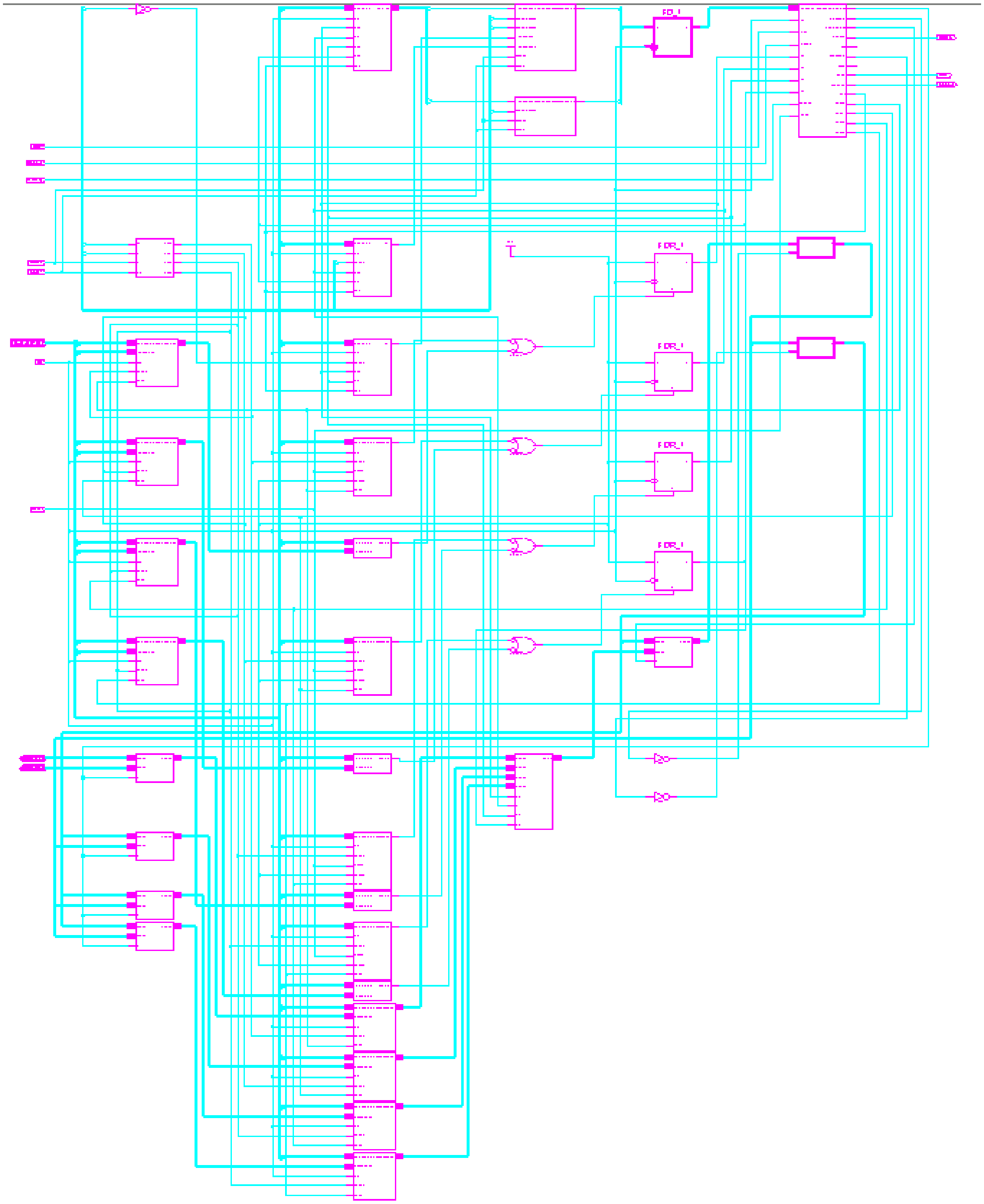


Figure 5-8 RTL Schematic of Reconfigurable Cache

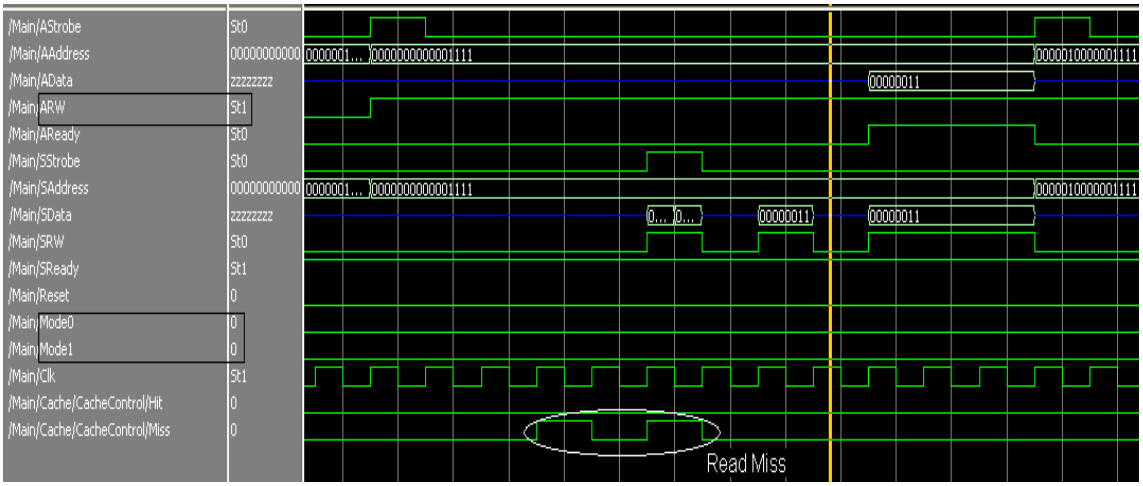

/Main/AStrobe St0
/Main/AAddress 00000000000 0000001... 0000000000001111 000001000001111
/Main/AData zzzzzzzz 00000011
/Main/ARW St1
/Main/AReady St0
/Main/SStrobe St0
/Main/SAddress 00000000000 0000001... 0000000000001111 000001000001111
/Main/SData zzzzzzzz 0... 0... 00000011 00000011
/Main/SRW St0
/Main/SReady St1
/Main/Reset 0
/Main/Mode0 0
/Main/Mode1 0
/Main/Clk St1
/Main/Cache/CacheControl/Hit 0
/Main/Cache/CacheControl/Miss 0
Read Miss

Figure 5-9 Simulation Waveform of Reconfigurable Cache in Mode ‘00’ (Direct Mapped) Showing Read Miss

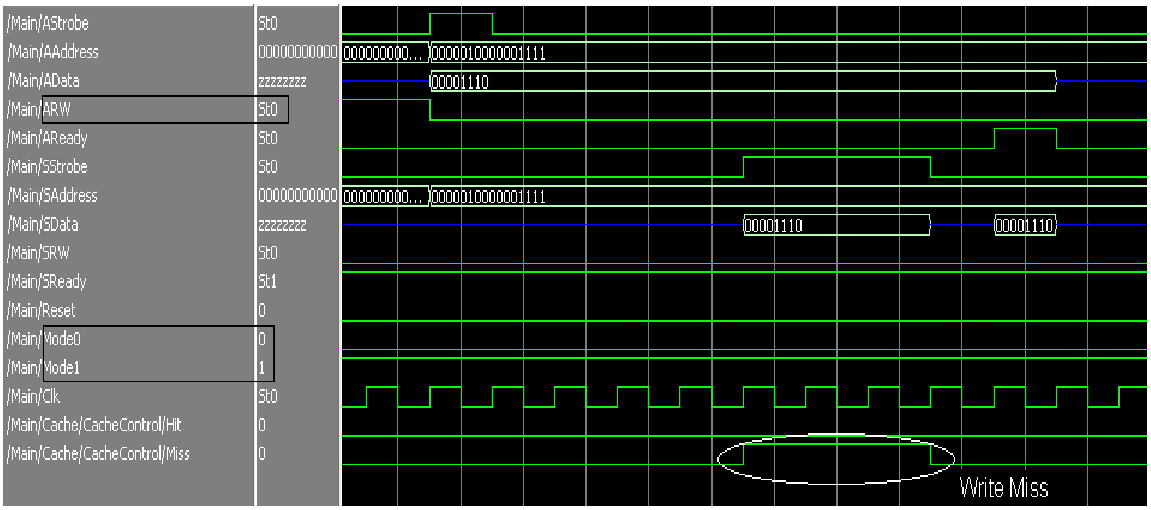

/Main/AStrobe St0
/Main/AAddress 00000000000 000000000... 000001000001111
/Main/AData zzzzzzzz 00001110
/Main/ARW St0
/Main/AReady St0
/Main/SStrobe St0
/Main/SAddress 00000000000 000000000... 000001000001111
/Main/SData zzzzzzzz 00001110 00001110
/Main/SRW St0
/Main/SReady St1
/Main/Reset 0
/Main/Mode0 0
/Main/Mode1 1
/Main/Clk St0
/Main/Cache/CacheControl/Hit 0
/Main/Cache/CacheControl/Miss 0
Write Miss

Figure 5-10 Simulation Waveform of Reconfigurable Cache in Mode ‘01’ (Two-Way Set-Associative) Showing Write Miss

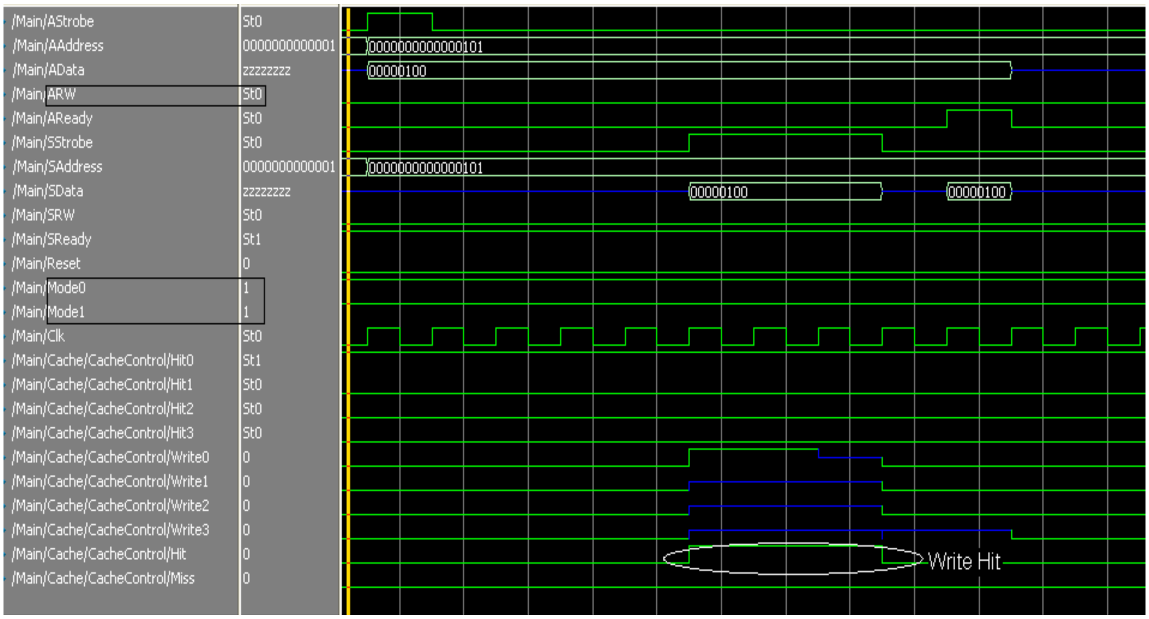

/Main/AStrobe St0
/Main/AAddress 0000000000001
/Main/AData zzzzzzzz
/Main/ARW St0
/Main/AReady St0
/Main/SStrobe St0
/Main/SAddress 0000000000001
/Main/SData zzzzzzzz
/Main/SRW St0
/Main/SReady St1
/Main/Reset 0
/Main/Mode0 1
/Main/Mode1 1
/Main/Clk St0
/Main/Cache/CacheControl/Hit0 St1
/Main/Cache/CacheControl/Hit1 St0
/Main/Cache/CacheControl/Hit2 St0
/Main/Cache/CacheControl/Hit3 St0
/Main/Cache/CacheControl/Write0 0
/Main/Cache/CacheControl/Write1 0
/Main/Cache/CacheControl/Write2 0
/Main/Cache/CacheControl/Write3 0
/Main/Cache/CacheControl/Hit 0
/Main/Cache/CacheControl/Miss 0
0000000000000101
00000100
0000000000000101
00000100
00000100
Write Hit

Figure 5-11 Simulation Waveform of Reconfigurable Cache in Mode '11' (Four-Way Set-Associative) Showing Write Hit.

Complete results obtained from trace file in three different modes are summarized in Table 5.1. The memory write time for all three configurations (1-way, 2-way, and 4-way) is same because we have used *write through* policy for write operations. In this policy, main memory is simultaneously updated with data cache for every write access. Total number of hits and misses are same for 2-way and 4-way configurations, as we have considered a very small trace file for functional simulation of our design. We can obtain a significant difference in number of hits and misses for these two configurations by considering a large trace file (which contains approximately 20-30 thousands of test cases).

Table 5-1 Summary of Reconfigurable Cache Operation in Three Different Modes

Design Metrics	Direct Mapped	2-Way	4-Way
Mode	00	01/10	11
No. of Access	20	20	20
Read Access	11	11	11
Write Access	9	9	9
Read Hits	2	7	7
Read Miss	9	4	4
Write Hits	2	4	4
Write Miss	7	5	5
Total Hits	4	11	11
Total Miss	16	9	9
Memory Read Time	1030	730	730
Memory Write Time	945	945	945

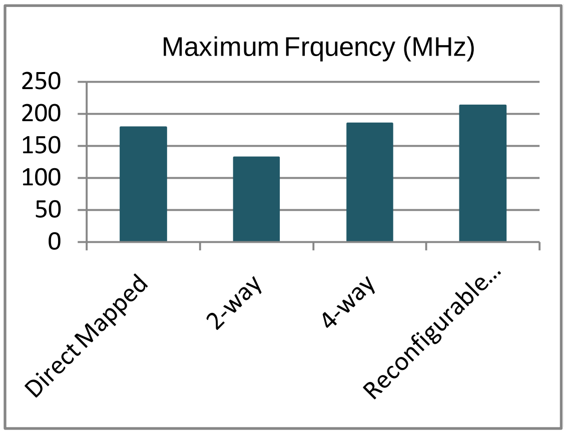
Various design metrics of the proposed design in direct mapped, two-way and four-way set-associative modes are summarized in Table 5.2. The timing path from a clock to any other clock in the design indicates the *minimum period* [47]. The proposed design possesses smallest *minimum period* and also highest *maximum operating frequency* with respect to 1-way, 2-way, and 4-way set-associative caches*.* The *maximum path delay* is an indicator of maximum path from inputs to outputs. This delay is smallest for the proposed design. The proposed design has obtained performance improvement in terms of *minimum period, maximum operating frequency* and *maximum path delay* due to distribution of clock in target device (Virtex-5). In virtex-5 family of FPGAs, there are total 32 global clocks and each device is divided into regions for distribution of clock. The design of proposed module is mapped into the target device in a very compact manner as compare to other three configurations, thus producing a minimum timing path from one synchronous element to other. Since the proposed design is mapped compactly, maximum delay from any one node to any other node is also small compare to other designs. The *Cell Usage* expressed in BELS, reports the count of all the logical cells that are basic elements of the Virtex technology, for example, LUTs, MUXCY, MUXF5, MUXF6, MUXF7, MUXF8 [46]. *Flip-flops* or *slice register* count indicates the total number of latches and flip flop used by the design. Reconfigurable cache design occupies more number of cells, slice registers and LUTs in order to provide reconfigurability.

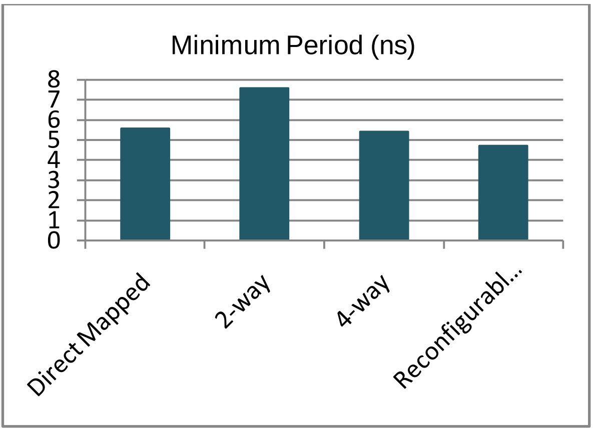
Xilinx gives us the flexibility of implementing the HDL code into all the leading devices of Virtex, Spartan, and Cool Runner families. The reconfigurable cache also has been targeted to two other FPGA boards. Table 5.3 summarizes design metrics of reconfigurable data cache for various three different FPGA technologies. The power consumption has been calculated under ambient temperature of 25ºC. Comparison of proposed design with other reconfigurable memory is given in Table 5.4.

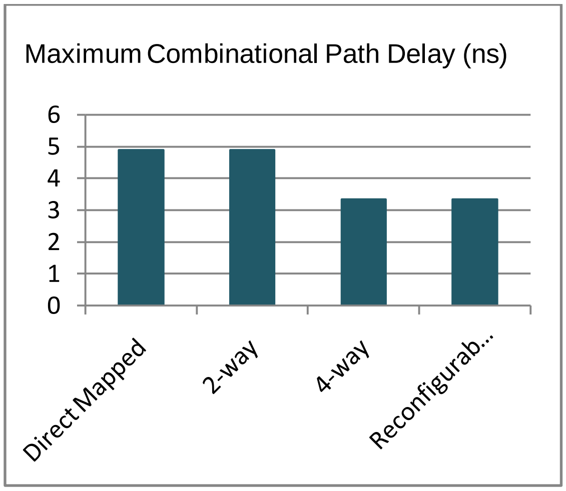
Table 5-2 Comparison of Various Design Metrics of Proposed Design with Direct Mapped, 2-Way, and 4-Way Set Associative Caches.

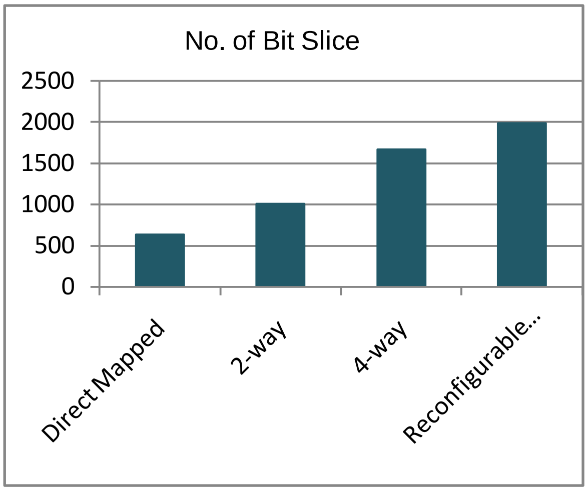
Design Metrics	Direct Mapped	2 Way	4 Way	Reconfigurable Cache
Maximum Frequency (MHz)	154.036	131.683	184.706	212.513
Minimum Period (ns)	6.492	7.594	5.41	4.706
Maximum Combinational Path Delay (ns)	4.901	4.897	3.338	3.342
Cell Usage (BELS)	399	782	1468	1888
FlipFlops/Slice Reg	267	530	946	1228
Slice LUTs	395	776	1446	1881
IO Utilization	56	56	56	58
No of Bit Slices	638	1006	1666	1991
Power (mW)*	-	-	1033	1362
Gate Count	135,952	271,795	280,781	288986

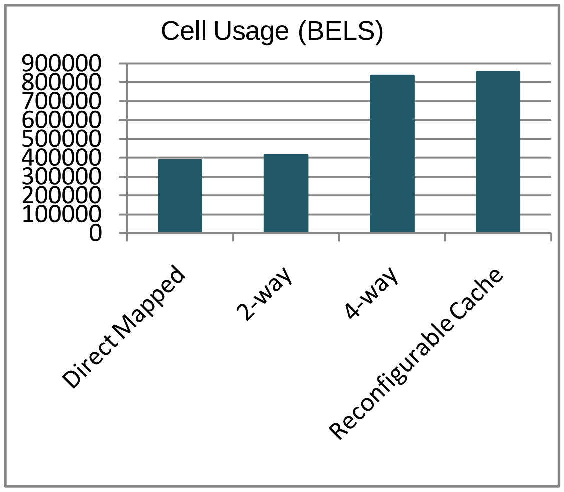
*****Power calculated incorrectly for small designs (1, 2-way) due to software (Xilinx ISE 9.1) bugs.

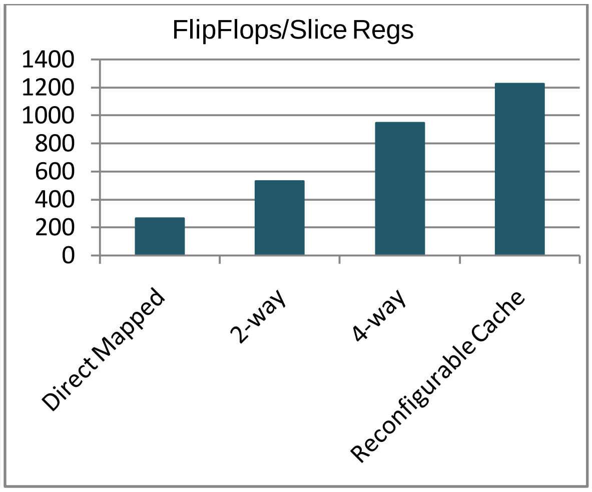

Maximum Frquency (MHz)
250
200
150
100
50
0
Direct Mapped
2-way
4-way
Reconfigurable...


Minimum Period (ns)
8
7
6
5
4
3
2
1
0
Direct Mapped
2-way
4-way
Reconfigurabl...


Maximum Combinational Path Delay (ns)
6
5
4
3
2
1
0
Direct Mapped
2-way
4-way
Reconfigurab...


No. of Bit Slice
2500
2000
1500
1000
500
0
Direct Mapped
2-way
4-way
Reconfigurable...


Cell Usage (BELS)
900000
800000
700000
600000
500000
400000
300000
200000
100000
0
Direct Mapped
2-way
4-way
Reconfigurable Cache


FlipFlops/Slice Regs
1400
1200
1000
800
600
400
200
0
Direct Mapped
2-way
4-way
Reconfigurable Cache

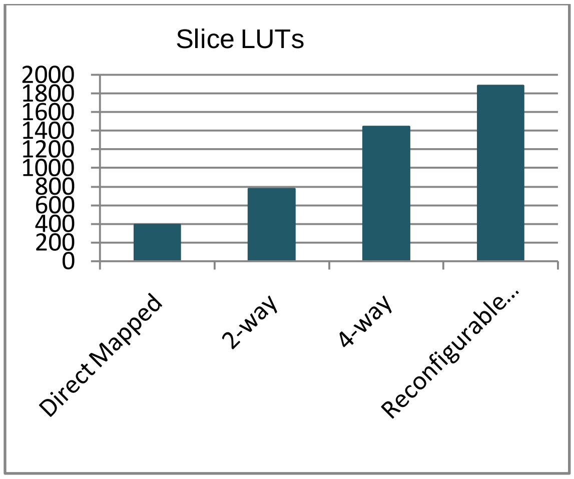


Figure 5-12 Performance Comparison of Reconfigurable Cache with Direct Mapped, Two-Way and Four-Way Set-Associative Caches.

Table 5-3 Comparison of Design Metrics of Reconfigurable Data Cache for Various FPGA Technologies.

Design Metrics	QPro Virtex Hi-Rel XQV1000	Automotive Spartan 3E	Virtex 5
Maximum Frequency (MHz)	37.111	76.429	212.513
Minimum Period (ns)	26.946	13.084	4.706
Maximum Combinational Path Delay (ns)	21.074	6.280	3.342
Cell Usage (BELS)	7596	4726	1888
FlipFlops/Slice Reg	1311	1230	1228
Slice LUTs	6361	4014	1881
IO Utilization	58	58	58
No of Bit Slices	3335	2090	1991
Power (mW)	-	102	1362

Table 5-4 Comparison of Proposed Design with Existing Reconfigurable Memories.

Name	Multi-Function Computing Cache [11]	Way Concatenation Cache [2]	Reconfigurable Memory [16]	Proposed Design
Year	2001	2003	2005	2009
Performance Improvement	Up to factor of 50-60 for computational applications	Due to way concatenation and way shutdown circuitry		In terms of Maximum Frequency, Combinational delay
Hardware & Area Overhead	~10-20% of base cache memory	Negligible	~15%	Due to Mode Selector Unit
Access Time	Increased by 1.6%	As 4-way	Increased by ~10%	
Associativity/ Configuration		1,2,4-way	Cache, FIFO, Scratchpad	1,2,4-way
Power		Dynamic Power savings up to 40% compared to conventional 4-way	Power Overhead ~10%	Static Power Overhead ~ 20%
Simulation Approach		Simple Scalar		ModelSim

CHAPTER 6

CONCLUSIONS AND FUTURE RESEARCH

This thesis presented the architecture and design of a new N-way reconfigurable data cache for embedded systems. The proposed data cache can be configured as direct-mapped, two-way and four-way set-associative cache to fulfill the systems' requirements. We have achieved this reconfigurability with the help of a mode selector while utilizing the full capacity of cache. The FPGA implementations of the reconfigurable cache along with direct-mapped, two-way and four-way set-associative caches have been implemented. The performance of the proposed design is compared with the direct-mapped, two-way and four-way cache architectures.

The proposed design can be further optimized in terms of speed, area and power consumption.

REFERENCES

[1] C. Zhang, “Reducing Cache Misses Through Programmable Decoders”, *ACM Transactions on Architecture and Code Optimization,* Vol. 4, No.4, Article 24, Jan 2008.

[2] C. Zhang, F. Vahid, and W. Najjar, “A Highly-Configurable Cache Architecture for Embedded Systems”, in *Proceedings of the 30th Annual International Symposium on Computer Architecture*, 2003.

[3] C. Zhang,” An Efficient Direct Mapped Instruction Cache for Application-Specific Embedded Systems”, in *Proceedings of the 3rd IEEE/ACM/IFIP International Conference on Hardware/software Codesign and System Synthesis,* Sep 2005.

[4] P. Grun, N. Dutt, and Alexandru Nicolau, “Memory Architecture Exploration for Programmable Enbedded Systems”, Boston*:* Kluwer Academic Publishers, 2003

[5] A. Malik, B. Moyer and D. Cermak, “A Low Power Unified Cache Architecture Providing Power and Performance Flexibility,” *International Symposium on Low Power Electronics and Design*, June 2000.

[6] S. Segars, “Low Power Design Techniques for Microprocessors,” International Solid-State Circuits Conference Tutorial, 2001

[7] Crisu, D., “An Architectural Survey and Modeling of Data Cache Memories in Verilog HDL”, in *Proceedings of International Semiconductor Conference*, Vol. 1, pp.139-142, 1999.

[8] T.S.B. Sudarshan, Rahil Abbas Mir, and S.Vijayalakshmi,”Highly Efficient LRU Implementations for High Associativity Cache Memory”, in *Proceedings of 12th IEEE International Conference on Advanced Computing and Communications,* pp.24-35, Dec 2004.

[9] J. Kin, M. Gupta and W. Mangione-Smith, “The Filter Cache: An Energy Efficient Memory Structure,” *International Symposium on Microarchitecture*, Dec 1997, pp. 184-193.

[10] C. Zhang, “Balanced Cache: Reducing Conflict Misses of Direct-Mapped Caches Through Programmable Decoders”, in *Proceedings of the 33rd International Symposium on Computer Architecture (ISCA)*, 2006.

[11] H. Kim, A.K. Somani, and A. Tyagi, “A Reconfigurable Multi-Function Computing Cache Architecture,” *IEEE Transactions on VLSI*, Vol. 9, No. 4, pp. 509-523, Aug. 2001.

[12] A. Agarwal and S. D. Pudar, “Column-Associative Caches: A Technique for Reducing the Miss Rate of Direct-Mapped Caches.” in *Proceedings of the International Symposium on Computer Architecture*, pp. 179-180, 1993.

[13] C. Zhang, X. Zhang, and Y. Yan, “Two Fast and High- Associativity Cache Schemes,” *IEEE Micro*, Vol. 17(5), pp. 40-49, Sep/Oct 1997.

[14] Frank Vahid, and Tony Givargis, “Embedded System Design: A Unified Hardware/Software Introduction”, John Wiley & Sons, Inc., 2002.

[15] N. H. E. Weste and D. Harris, “CMOS VLSI Design: A Circuit and Systems Perspective”, Addison Wesley, 2005.

[16] K. Mai, R. Ho, E. Alon, D. Liu, Y. Kim, D. Patil, and M.A. Horowitz, “ Architecture and Circuit Techniques for a 1.1-GHz 16-kb Reconfigurable Memory in 0.18-µm CMOS” , *IEEE Journal of Solid-State Circuits*, Vol.40, No.1, pp. 261-275, Jan 2005

[17] J.W. Park, C.G. Kim, J.H. Lee, and S.D. Kim, “An Energy Efficient Cache Memory Architecture for Embedded Systems”, in *Proceedings of the ACM symposium on Applied Computing*, pp. 884-890, 2004.

[18] T. Juan, T. Lang, and J. Navarro. “The Difference-Bit Cache.” in *Proceedings of the 27th International Symposium on Computer Architecture*, 1996.

[19] C. Zhang, F. Vahid, J. Yang, and W. Najjar, “A Way-Halting Cache for Low-Energy High-Performance Systems,” *ACM Transactions on Architecture and Code Optimization,* Vol. 2, Issue 1, pp. 34-54, March 2005.

[20] L. Liu, “Cache Design with Partial Address Matching,” in *Proceedings of the International Symposium on Microarchitecture*, 1994.

[21] C. Zhang, F. Vahid, and W. Najjar, “Energy Benefits of a Configurable Line Size Cache for Embedded Systems,” *International Symposium on VLSI Design*, 2003.

[22] M. Zhang and K.Asanovic,"Highly-Associative Caches for Low-Power Processors," *Kool Chips Workshop, in conjunction with International Symposium on Microarchitecture*, Dec. 2000.

[23] Z. Navabi, "Verilog Digital System Design", New York: McGraw-Hill*,* 1999.

[24] http://www.faculty.iu-bremen.de/birk/lectures/PC101 2003/07cache/cache%20memory.htm

[25] J. L. Hennessy and D .A. Patterson, "Computer Architecture Quantitative Approach," 2nd edition, Morgan-Kaufmann Publishing Co., 1996.

[26] J. L. Hennessy and D .A. Patterson, "Computer Organization and Design: The Hardware/Software Interface," 4th edition, Morgan-Kaufmann Publishing Co., 2009.

[27] http://www.pcguide.com/ref/mbsys/cache/funcComparison-c.html

[28] http://webster.cs.ucr.edu/AoA/Windows/HTML/MemoryArchitecturea2.html

[29] www.cse.uconn.edu/~huang/spring08_340/May_5/Wei_Zeng.ppt

[30] T. Givargis, "Improved Indexing for Cache Miss Reduction in Embedded Systems," in *Proceedings of Design Automation Conference*, 2003.

[31] http://blogs.sun.com/toddjobson/resource/Memory_Latency.png

[32] B, Jacob, "Cache Design for Embedded Real-Time Systems", Embedded Systems Conference, Danvers MA, June 30, 1999.

[33] A. J. Smith, "Cache memories", *ACM Computing Surveys*, Vol.14, Issue 3, pp. 473-530, Sep.1982.

[34] S. Palnitkar, "Verilog HDL: A Guide to Digital Design and Synthesis", SunSoft Press, 1996.

[35] P.R. Panda, N. Dutt, and Alexandra Nicolau, "Memory Issues in Embedded Systems-On-Chip: Optimizations and Exploration", Kluwer Academic Publishers, 1999.

[36] Wlison Peter R., “Design Recipes for FPGAs”, Elsevier/Newnes*,* 2007.

[37] Wolf, Wayne Hendrix, “FPGA-based System Design”, Prentice Hall PTR, 2004.

[38] S. Brown, J. Rose; “Architecture of FPGAs and CPLDs: A tutorial,” Department of Electrical and Computer Engineering, Toronto University.

[39] Christoph Scholl, “Functional Decomposition with Application to FPGA Synthesis”, Kluwer Academic Publishers, 2001.

[40] S. Tarigopula, “A CAM-based, High-Performance Classifier-Scheduler for a Video Network Processor”, *Thesis*, Department of Computer Science and Engineering, University of North Texas, May2008.

[41] O. B Adamo, “VLSI Architecture and FPGA Prototyping of a Secure Digital Camera for Biometric Application*”, Thesis*, Department of Computer Science and Engineering, University of North Texas, Aug2006.

[42] P. Yiannacouras and J. Rose, “A Parameterized Automatic Cache Generator for FPGAs”, in *Proceedings of IEEE International Conference on Field-Programmable Technology,* pp.324-327, Dec 2003.

[43] B.K. Raveendran, T.B.S. Sudarshan, P.D. Kumar, P. Tangudu, and S. Gurunaravanan, “LLRU: Late LRU Replacement Strategy for Power Efficient Embedded Cache”, *International Conference on Advanced Computing and Communications*, pp.339-344, Dec 2007.

[44] K. Coffman, “Real World FPGA Design with Verilog”, Prentice Hall Modern Semiconductor Design Series, 2000.

[45] www.xilinx.com

[46] S. P. Mohanty, R. Kumara C., and S. Nayak, "FPGA Based Implementation of an Invisible-Robust Image Watermarking Encoder", in *Lecture Notes in Computer Science (LNCS)*, *CIT 2004,Springer-Verlag*, Vol. 3356, pp. 344-353, 2004.

[47] S. P. Mohanty, E. Kougianos, and N. Ranganathan, "VLSI Architecture and Chip for Combined Invisible Robust and Fragile Watermarking", *IET Computers & Digital Techniques (CDT)*, Vol. 1, Issue 5, pp. 600-611, Sep 2007.