



# Reduction of Dynamic Power in L2 Cache Architecture Using Way Tag Information under Write-Through Policy

Mr. Mirza Farz Ali<sup>1</sup>

M. Tech Scholar

Electronics & Communication Engg  
Aurora's Scientific, Technological and  
Research Academy  
Hyderabad, India.

Mrs. Surya Mukhi<sup>2</sup>

Sr. Assistant professor

Electronics & Communication Engg  
Aurora's Scientific, Technological and  
Research Academy  
Hyderabad, India.

Mrs. D. kavitha<sup>3</sup>

Associate professor

Electronics & Communication Engg  
Aurora's Scientific, Technological and  
Research Academy  
Hyderabad, India.

**Abstract**— To perform operations the processor has to fetch the instructions from the memory in this process the time taken to fetch the instructions from the larger memory block (main memory) is more i.e., it does not reach the processor speed of execution, to decrease the gap between the processor speed of execution and data fetching of the processor we go for the cache memory. The cache memory performance is the most significant factor in achieving high processor performance because cache memory is the very small in size than the main memory it will be helpful in fetching the data very fast which increases the performance of the processor if the data is not present in the cache memory then it fetches the data from the main memory and stores it in the cache memory.

In this paper we are going to increase the performance of the processor by a new policy called write-through and a new cache architecture referred to as way-tagged. In this way-tagged process we are having L1 cache and L2 cache and the address at which the data have to be stored is divided into three parts tag, index and offset address and the data which is going to be stored in the L1 & L2 caches are stored with reference with the tag address and the copy of the tag address is stored in the way-tag array. Way-tag array is an array where the way-tag address of the data is stored.

When the processor requires the data to perform the required operations first it checks the L1 cache and if the data is present in the L1 cache it fetches the data otherwise it checks the L2 cache for the data and similarly if the data is not present in L2 cache the processor checks the data in the main memory. While the processor fetches the data from the main memory it stores the data in the L2 & L1 cache respectively and stores the way-tag address in the respective L1 and L2 way-tag arrays.

By this process of way-tag we are going to increase the performance of the processor than the previous cache process. Simulation results on the ModelSim and synthesis results on Xilinx demonstrate that the proposed technique achieves total power saving of 56.42% and dynamic power saving of 41.31% in L2 caches on average with small area overhead and no performance degradation.

**Index Terms**— Cache, dynamic power, write-through policy.

## I. INTRODUCTION

A general-purpose processor is a finite-state automaton that executes instructions held in a memory. The state of the system is defined by the values held in the memory locations together with the values held in certain registers within the processor itself (see Fig. 1). Each instruction defines a particular way the total state should change and it also defines which instruction should be executed next.

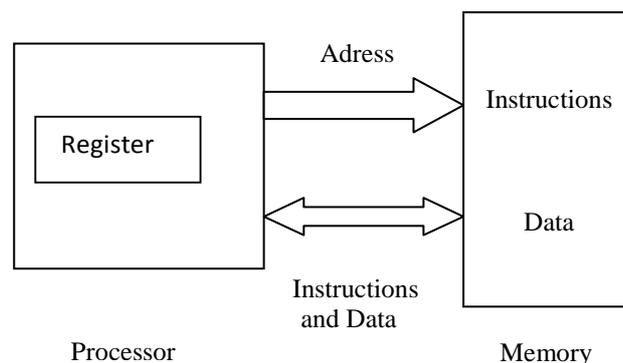


Fig.1: The state in a stored-program digital computer

If we want to make a processor go fast, we must first understand what it spends its time doing. It is a common misconception that computers spend their time computing, that is, carrying out arithmetic operations on user data. In practice they spend very little time 'computing' in this sense. Although they do a fair amount of arithmetic, most of this is with addresses in order to locate the relevant data items and program routines. Then, having found the user's data, most of the work is in moving it around rather than processing it in any transformational sense.

At the instruction set level, it is possible to measure the frequency of use of the various different instructions. It is very important to obtain dynamic measurements, that is, to measure the frequency of instructions that are executed, rather than the static frequency, which is just a count of the various instruction types in the binary image.

A typical set of statistics is shown in Table. these statistics were gathered running a print preview program on an

instruction emulator, but are broadly typical of what may be expected from other programs and instruction sets.

Instruction type	Dynamic Usage
Data movement	43%
Control Flow	23%
Arithmetic Operations	15%
Comparisons	13%
Logical Operations	5%
Others	1%

Table 1: Typical Dynamic Instruction usage.

These sample statistics suggest that the most important instructions to optimize are those concerned with data movement, either between the processor registers and memory or from register to register. These account for almost half of all instructions executed. Second most frequent are the control flow instructions such as branches and procedure calls, which account for another quarter. Arithmetic operations are down at 15%, as are comparisons.

Now we have a feel for what processors spend their time doing, we can look at ways of making them go faster. The most important of these is pipelining. Another important technique is the use of a cache memory, which will be covered in Section 10.3 on page 272. A third technique, super-scalar instruction execution, is very complex, has not been used on processors.

In this paper we are concerned mainly about the Data movement by keeping the data very near to the processor by using the L1 & L2 Cache and by employing Write-through and Write-back policies. Under the W-B policy, when the block is about to be replaced a modified cache block is copied back to its corresponding lower level cache. While under the W-T policy, at the current cache all copies of a cache block are updated immediately after the cache block is modified, even though the block might not be activated. As a result, the W-T policy maintains equal data copies at all levels of the cache hierarchy throughout most of their life time of process.

It has been reported that single-event multi bit upsets are getting lower standard in on-chip memories. Presently, this problem has been addressed at different stages of the design the quality of dealing. At the architecture level, a correct solution is to keep data consistent among different levels of the memory stages is to prevent the system from collapse due to soft problems.

## II. RELATED WORK

To improve the performance of the processor we use Way-tagged cache this will help in storing the data in the L1 and L2 caches in a order that the data with same address tag are stored in the same location and the data with different address tag are stored in the different locations.

We consider a conventional set-associative cache system when the L1 data cache loads/writes data from/into the L2cache, all ways in the L2 cache are activated simultaneously for performance consideration at the cost of energy overhead.

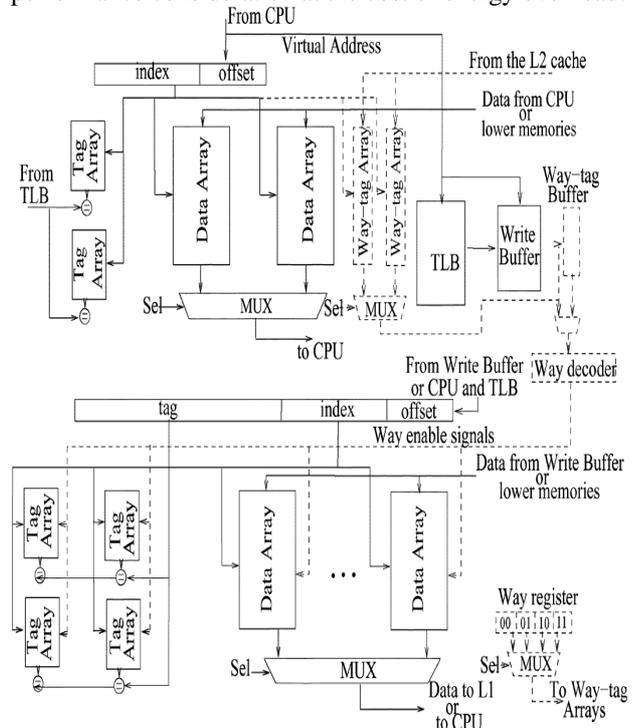


Fig. 2: Block diagram of way-tag cache

By this process when the processor required the data for the execution of the instructions it directly check the any one of the way-tag array instead of checking all the memory location(all the way-tag array).This is shown in the Fig .2.

### Example:

Consider the different data with different address

Data	Address
12342	00110011
12343	11001111
12344	01001100
12345	10100010
12346	00101010
12347	11001100
12348	10001111
12349	01010101

In this the data with address starting with 00 are stored in the same way-tag array in 00 location i.e., the data 12342 and 12346 are stored in same location. Similarly the data with address starting with 01,10,11 are stored in the same way-tag array in 01,10,11 location respectively.

### Write-Back:

In this write-back policy data is stored in memory when it is about to replace.

In this write-back policy there is a chance of data missing because it does not store the data instantly in the memory location when the data is modified. To overcome this drawback we use write-through policy.

### Write-Through:

In this write-through policy when ever the data is modified by the processor it stores the modified data in the memory location.

Different Operations perform in L1, L2 and Main Memory					
L1 Cache	Read_hit		Write_hit		
	Yes	No	Yes (If same data is not present)	No	
			Need to update	Need to copy	
L2 Cache	No need to check	Need to check (set-associative)		Need to update (direct mapping)	Need to copy (set-associative)
		yes	No		
Main Memory	No need to check	No need to check	Need to check	Need to update	Need to copy

Fig. 3: Different Operations perform in L1, L2 and Main Memory

Under the write-through policy, the L2 cache always maintains the most recent copy of the data. Thus, whenever a data is updated in the L1 cache, the L2 cache is updated with the same data as well. This results in an increase in the write accesses to the L2 cache and consequently more energy consumption.

For read process initially the processor check the L1 cache memory for the data, if the required data is present read\_hit signal is set to Yes other wise No. If read\_hit signal in L1 is Yes then no-need to check the L2 and Main memory. If it is No it has to check L2 cache in that if the required data is found in L2 cache then the read\_hit signal is set to Yes other wise No. If the data found in the L2 Cache no need to go for Main memory otherwise it has to checks the main memory.

For write process the processor initially checks the address location in the L1 cache if the address location is found the it compare the data. If the same data is present then no need write other wise it has to update. Same update operation is carried out in L2 and main Memory. If the required address is not found in the L1 cache then it has to copy the data in the L1, L2 and Main Memory.

### WAY-TAGGED CACHE

Fig. 2 Block diagram of way-tagged cache. We introduce several new components: way-tag arrays, way-tag buffer, way decoder, and way register, all shown in the dotted line. The

way tags of each cache line in the L2 cache are maintained in the way-tag arrays, located with the L1 data cache. Note that write buffers are commonly employed in write-through caches (and even in many write-back caches) to improve the performance. With a write buffer, the data to be written into the L1 cache is also sent to the write buffer.

The operations stored in the write buffer are then sent to the L2 cache in sequence. This avoids write stalls when the processor waits for write operations to be completed in the L2 cache. In this technique, we also need to send the W-Tags stored in the W-Tag arrays to the L2 cache along with the operations in the write buffer. Thus, a small W-Tag buffer is introduced to buffer the way tags read from the way-tag arrays. A way decoder is employed to decode way tags and generate the enable signals for the L2 cache, which activate only the desired way in the L2 cache. Each way in the L2 cache is encoded into a waytag. A way register stores way tags and provides this information to the way-tag arrays.

### III. IMPLEMENTATION OF WAY-TAG CACHE

**Way-Tag Array:** Way-tag array stores the tag information of the data in different location depending on the starting address location. And when ever the data is needed by the processor, the processor check address in the way-tag and if the tag address is found then the data is fetched from the corresponding address location. If not then there is no need in check the memory location.

WRITEH	UPDATE	OPERATION
0	0	No access
0	1	No access
1	1	Write way-tag arrays
1	0	Read way-tag arrays

Table 2: Operations of Way-Tag array

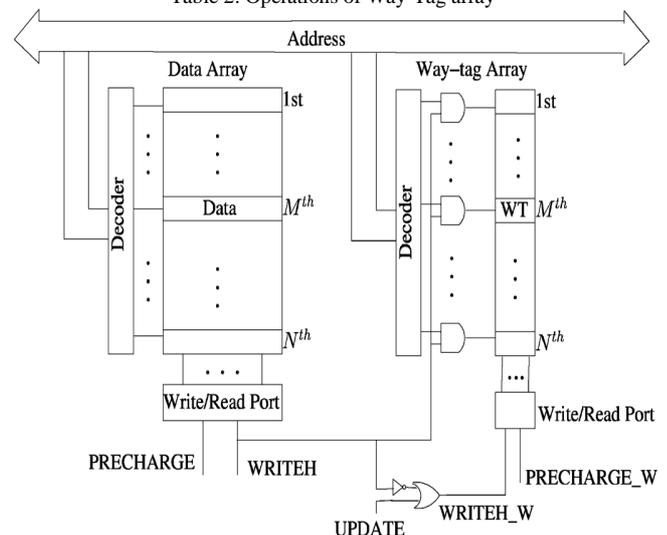


Fig. 5: Way-Tag Array

**Way-Tag buffer:** Way-Tag buffer is used to mirroring the information from one place to another place. Way-tag buffer temporarily stores the way tags read from the way-tag arrays. Implementation is shown in below fig. 6.

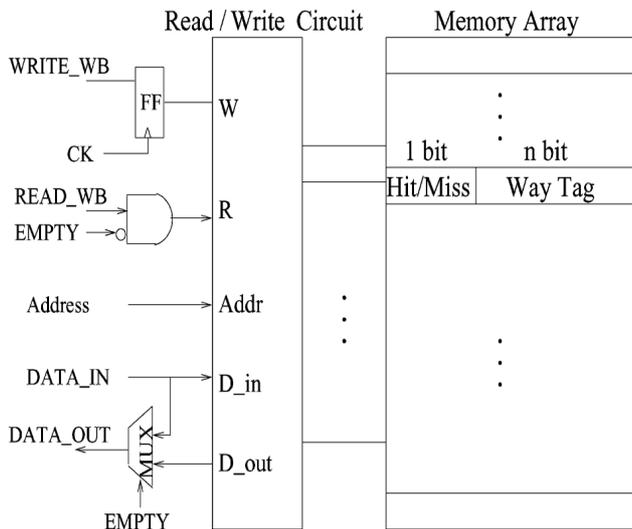


Fig. 6: Way-tag buffer.

**Way-Decoder:** Way decoders are used to select the way. And activate only the desired ways in the L2 cache. Below fig. 7 shows the block diagram of the Way-Decoder.

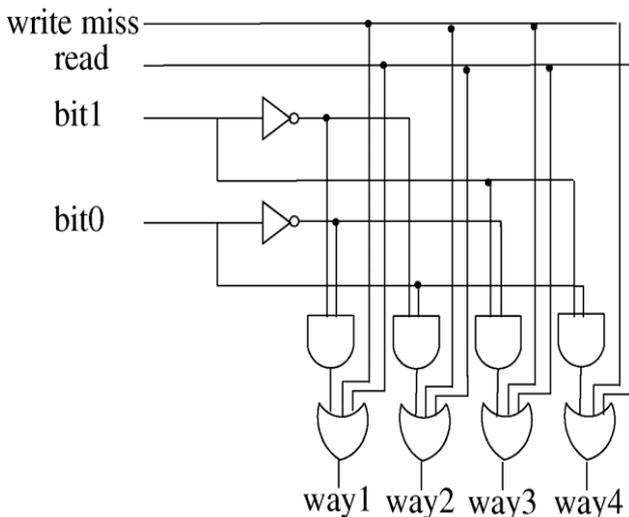


Fig. 7: Block diagram of Way-Decoder

**Way Register:** The way register are used to provide way tags for the way-tag arrays. When the data is carried from main memory to L2 cache and also from L2 to L1 cache i.e., when the data is carried the way-tag address is carried by way-register.

## IV. EVALUATION AND DISCUSSION

### simulation results of top module:

The below fig. 9 shows the simulation results of top module with Way-Tag of this project in that at the starting of the simulation we set CLK (clock) signal and initially we set RST(reset) signal as logic1 at that instant all the other signal values are at logic0.

In the next state we set the RST(reset) signal as logic0 then the operation of the cache controller take place in that process the processor will do two operation read and write.

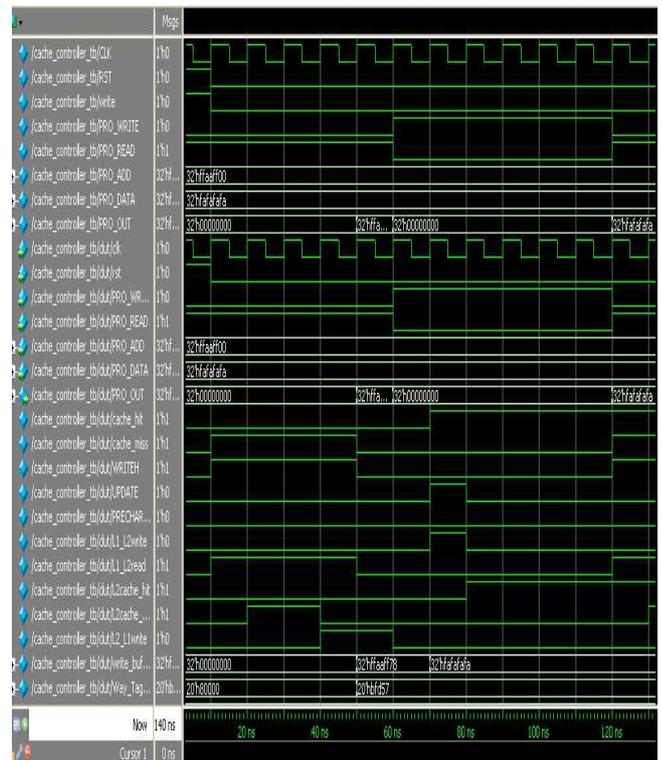


Fig.9: Simulation Result of Top Module With Way-Tag(capture 1)

Depending on the instructions of the programmer at the starting read operation is take place in that process the processor checks for the required data in the nearest and the smallest memory location (L1cache) if the data present in the L1cache(i.e, L1way\_tag array is having the required way-tag address and data) then the cache-hit signal is set to logic1 otherwise cache-miss signal is set to logic1. If cache-hit is logic1 then the processor fetch the data from the L1cache memory and carry out its operations. If the cache-miss signal is logic1 then the processor check for the data in the L2 cache memory in the similar way if the data found in the L2cache(i.e, L2way\_tag array is having the required way \_tag address and

data) then the L2cache\_hit signal is set to logic1 otherwise the L2cache\_mis signal is set to logic1. If L2cache\_hit signal is logic1 then the processor fetch the data from the L2cache memory and carry out its operations.





# International Journal of Ethics in Engineering & Management Education

Website: [www.ijeee.in](http://www.ijeee.in) (ISSN: 2348-4748, Volume 1, Issue 9, September 2014)

- [9] About Cache Basics by Gene Cooperman Available at:  
<http://www.ccs.neu.edu/course/com3200/parent/NOTES/cache-basics.html> (Accessed: 17 April 2014)
- [10] About Set Associative Cache Available at:  
<http://www.cs.umd.edu/class/sum2003/cmsc311/Notes/Memory/set.html>  
(Accessed: 03 May 2014)
- [11] A brief tutorial on Xilinx. Available at:  
<https://www.digilentinc.com/Data/Documents/Tutorials/Xilinx%20ISE%20WebPACK%20VHDL%20Tutorial.pdf> (Accessed: 19 May 2014)
- [12] About Predictions for Low-Power Cache Design Available at:  
<http://web.cse.ohio-state.edu/hpcs/WWW/HTML/publications/papers/TR-02-7.pdf>  
(Accessed: 27 May 2014)
- [13] About Literature Survey and Analysis of Low-Power Techniques for Memory and Microprocessors Available at:  
[http://users.ece.utexas.edu/~bevans/courses/ee382c/lectures/00\\_welcome/project2.html](http://users.ece.utexas.edu/~bevans/courses/ee382c/lectures/00_welcome/project2.html) (Accessed: 1 June 2014)

## About the authors



**Mr. Mirza Farz Ali<sup>1</sup>** received B.Tech degree in ECE from JNTUK in 2012 , pursuing M.Tech (2012-2014) in the stream of VLSI at Aurora's Scientific, Technological and Research Academy, (Affiliated to JNTUH) Hyderabad.



**Mrs. Surya Mukhi<sup>2</sup>** received M.Tech degree in ES from JNTUH in 2010, B.Tech degree in ECE from JNTUH in 2005, Diploma in ECE from SBTET 2002.

Presently working as Sr. Assistant professor in ASTRA, Hyderabad.



**Mrs. D.kavitha<sup>3</sup>** received M.Tech degree in VLSI from JNTUH in 2010, B.E degree in ECE from OU in 2003, Diploma in ECE from SBTET in 1998.

Presently working as Associate professor in ASTRA, Hyderabad.