Vandana Publications
IJEMR

Volume-4, Issue-6, December-2014, ISSN No.: 2250-0758
International Journal of Engineering and Management Research
Available at: www.ijemr.net
Page Number: 151-157

Energy Reduction under Different Cache Configurations by using Way Tag Architecture

Harika[1], D.VaraPrasada Rao[2]
M. Tech Scholar, VLSI and Embedded Systems, Turbo Machinery Institute of Technology and Sciences, INDIA
Associate Professor & HOD Dept of ECE, Turbo Machinery Institute of Technology and Sciences, 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 helpful in fetching the data very fastly which increase the performance of the processor if the data is not present in the cache memory then it fetch the data from the main memory and stored in the cache memory.

The cache memory work on the principle of locality. Cache works by storing a small subset of the external memory contents, typically out of its original order. Data and instructions that are being used frequently, such as a data array or a small instruction loop, are stored in the cache and can be read quickly without having to access the main memory. Cache runs at the same speed as the rest of the processor, which is typically much faster than the external RAM operates at. This means that if data is in the cache, accessing it is faster than accessing 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 required the data to perform the required operations first it check the L1 cache and if the data is present in the L1 cache it fetches the data otherwise it check 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 processor fetching the data from the main memory it stores the data in the L2 & L1 cache respectively and stores the way-tag address in the respectively 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.

Furthermore, the idea of way tagging can be applied to existing low-power cache design techniques to further improve energy efficiency.

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.

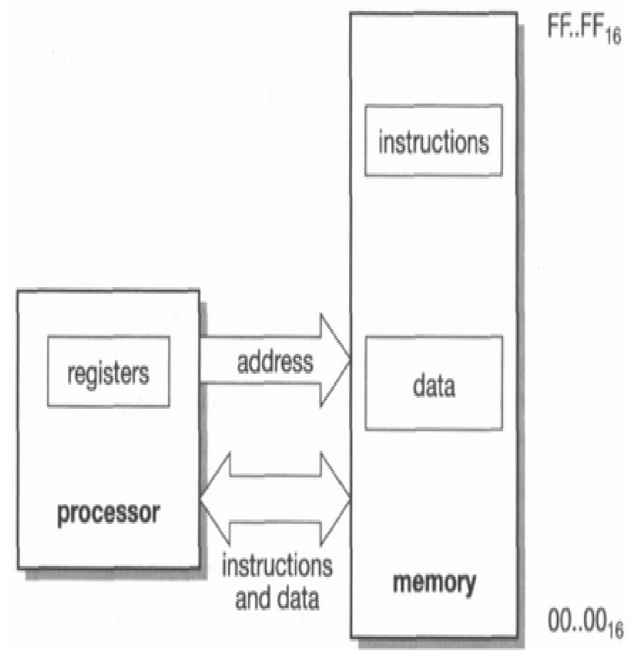
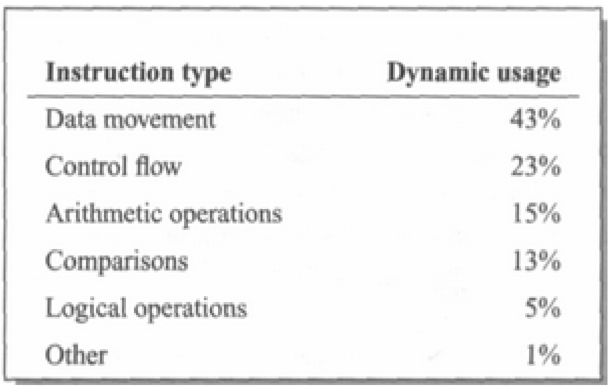


Fig.1: The state in a stored-program digital computer

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%
Other	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 polices. Under the write-back policy, a modified cache block is copied back to its corresponding lower level cache only when the block is about to be replaced. While under the write-through policy, all copies of a cache block are updated immediately after the cache block is modified at the current cache, even though the block might not be evicted. As a result, the write-through policy maintains identical data copies at all levels of the cache hierarchy throughout most of their life time of execution.

It has been reported that single-event multi bit upsets are getting worse in on-chip memories. Currently, this problem has been addressed at different levels of the design abstraction. At the architecture level, an effective solution is to keep data consistent among different levels of the memory hierarchy to prevent the system from collapse due to soft errors. Benefited from immediate update, cache write-through policy is inherently tolerant to soft errors because the data at all related levels of the cache hierarchy are always kept consistent. Due to this feature, many high-performance microprocessor designs have adopted the write-through policy.

II. Type 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.

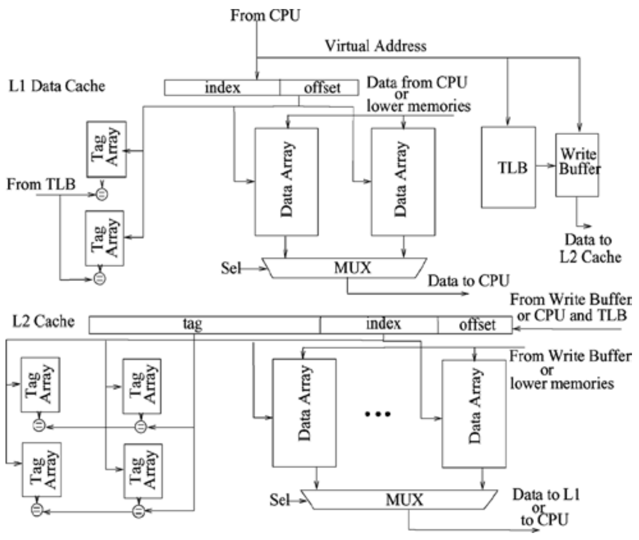
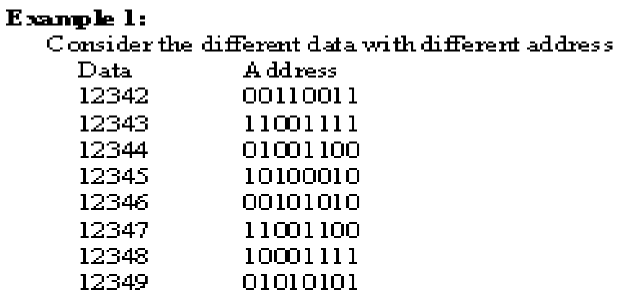


Fig. 2: Block diagram of way-tag cache

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.

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 1:

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.

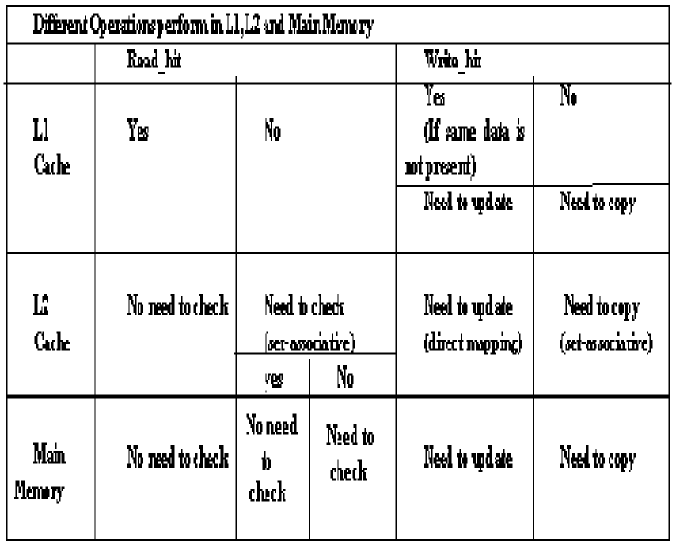
Write-Back:

In this write-back police 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 over come this draw back 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					
	Read_hit			Write_hit	
L1 Cache	Yes	No		Yes (If same data is not present) Need to update	No 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.

III. WAY-TAGGED CACHE

Fig. 4 shows the system diagram of proposed 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.

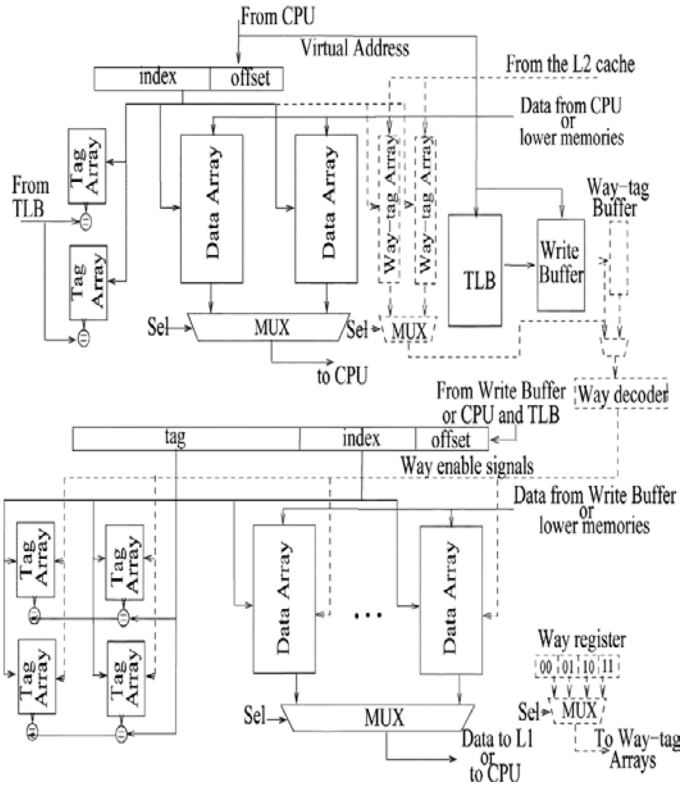


Fig. 4: Proposed way-tagged cache

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 the pro- posed technique, we also need to send the way tags stored in the way-tag arrays to the L2 cache along with the operations in the write buffer. Thus, a small way-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 sin 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.

IV. 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.

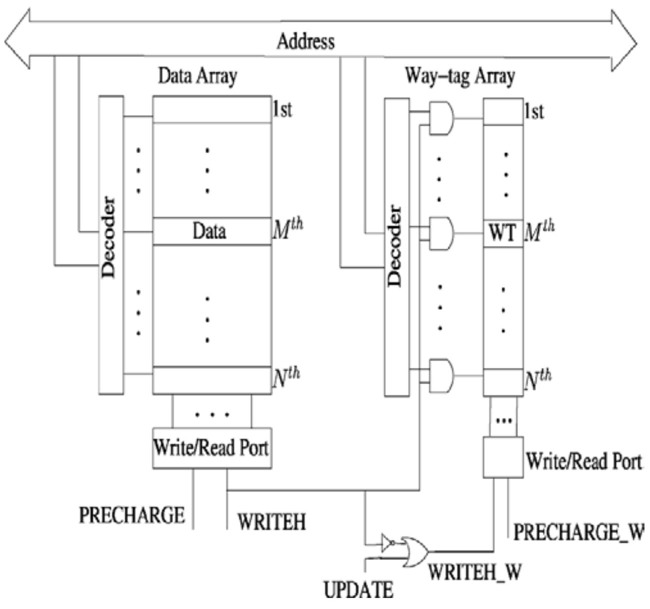


Fig. 5: Way-Tag Array

WRITEH	UPDATE	OPERATION
1	1	write way-tag arrays
1	0	read way-tag arrays
0	0	no access
0	1	no access

Table 2: Operations of 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.

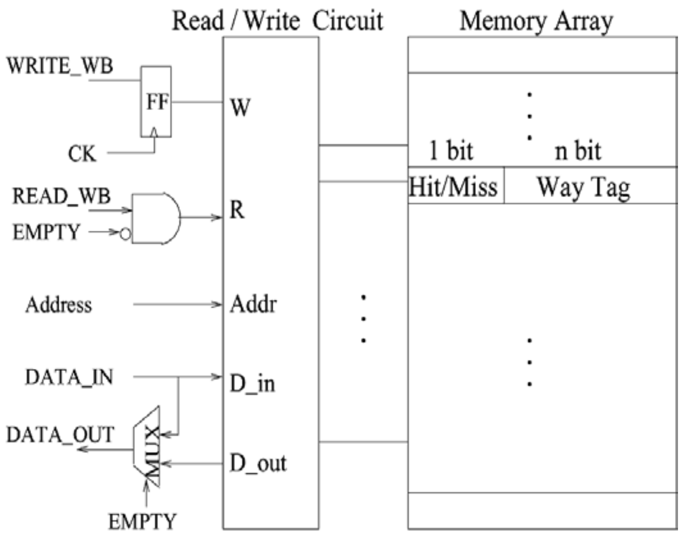


Fig. 6: Way-tag buffer.

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.

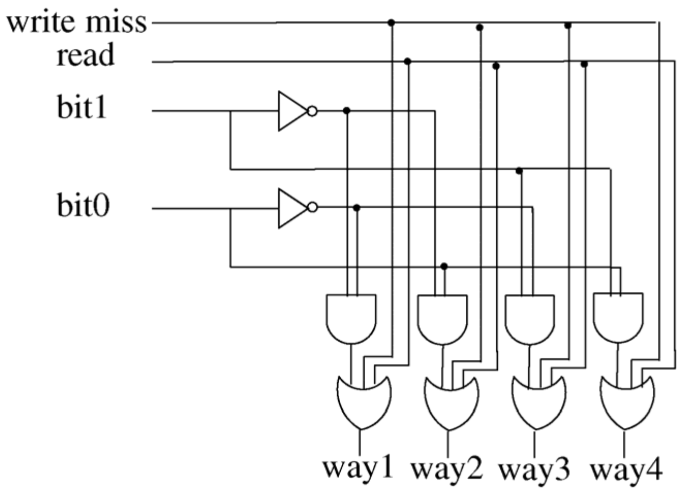


Fig. 7: Block diagram of Way-Decoder

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 form L2 to L1 cache i.e., when the data is carried the way-tag address is carried by way-register.

V. FURTHER EXTENSION

We can extend this 2-way tag L2 cache architecture to 4-way tag L2 architecture for more high performance. The below fig.8 shows the architecture of the 4-way tag L2Cache.

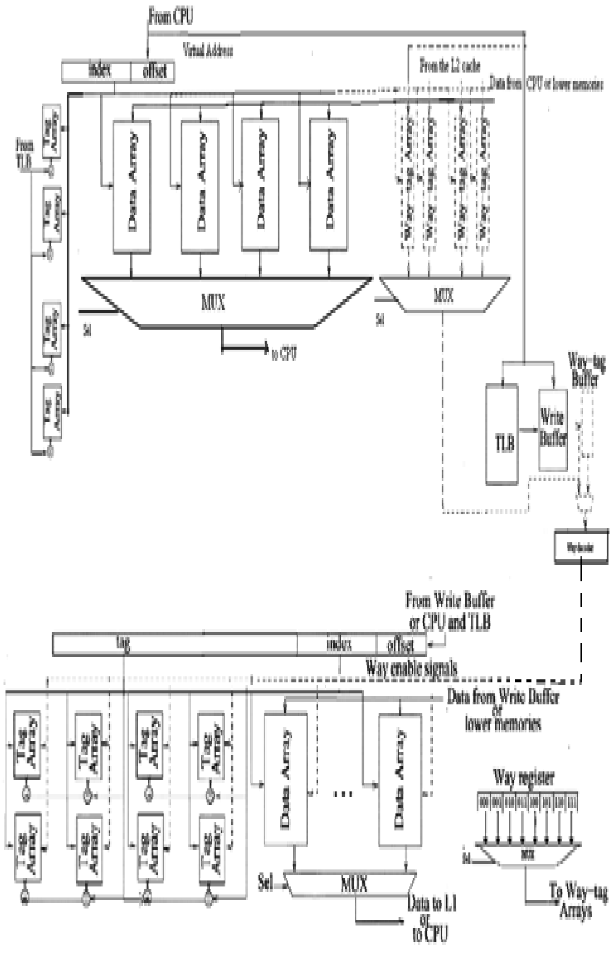


Fig. 8: The Architecture of the 4-way tag L2 Cache

VI. 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.

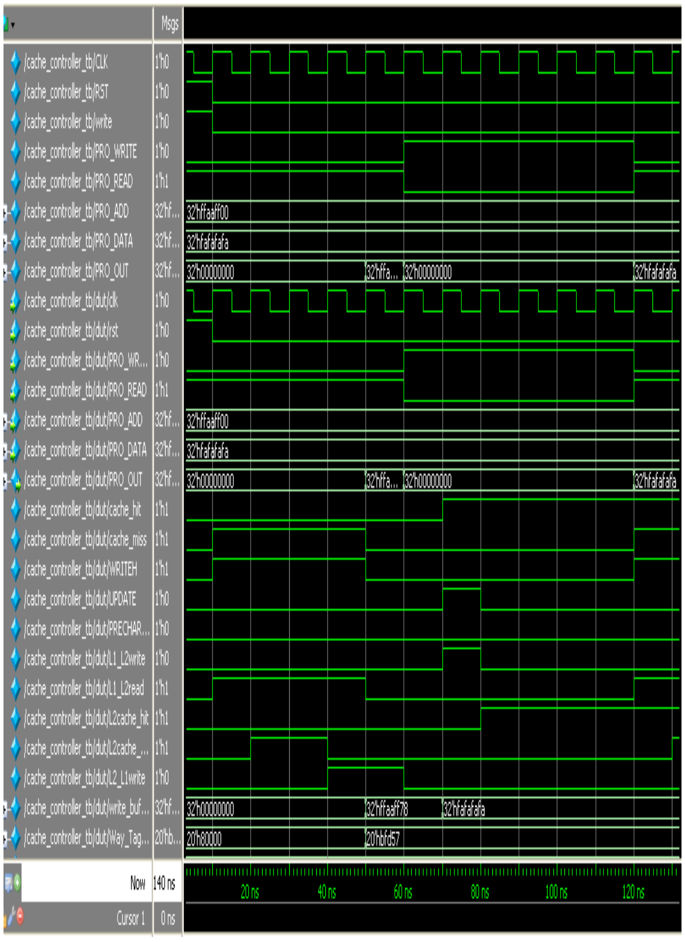

Msgs
/cache_controller_tb/CLK 1'h0
/cache_controller_tb/RST 1'h0
/cache_controller_tb/write 1'h0
/cache_controller_tb/PRO_WRITE 1'h0
/cache_controller_tb/PRO_READ 1'h1
/cache_controller_tb/PRO_ADD 32'hf... 32'hffaaff00
/cache_controller_tb/PRO_DATA 32'hf... 32'hfafafafa
/cache_controller_tb/PRO_OUT 32'hf... 32'h00000000 32'hffa... 32'h00000000 32'hfafafafa
/cache_controller_tb/dut/clk 1'h0
/cache_controller_tb/dut/rst 1'h0
/cache_controller_tb/dut/PRO_WR... 1'h0
/cache_controller_tb/dut/PRO_READ 1'h1
/cache_controller_tb/dut/PRO_ADD 32'hf... 32'hffaaff00
/cache_controller_tb/dut/PRO_DATA 32'hf... 32'hfafafafa
/cache_controller_tb/dut/PRO_OUT 32'hf... 32'h00000000 32'hffa... 32'h00000000 32'hfafafafa
/cache_controller_tb/dut/cache_hit 1'h1
/cache_controller_tb/dut/cache_miss 1'h1
/cache_controller_tb/dut/WRITEH 1'h1
/cache_controller_tb/dut/UPDATE 1'h0
/cache_controller_tb/dut/PRECHAR... 1'h0
/cache_controller_tb/dut/L1_L2write 1'h0
/cache_controller_tb/dut/L1_L2read 1'h1
/cache_controller_tb/dut/L2cache_hit 1'h1
/cache_controller_tb/dut/L2cache_... 1'h1
/cache_controller_tb/dut/L2_L1write 1'h0
/cache_controller_tb/dut/write_buf... 32'hf... 32'h00000000 32'hffaaff78 32'hfafafafa
/cache_controller_tb/dut/Way_Tag... 20'hb... 20'h80000 20'hbfd57
Now 140 ns
20 ns
40 ns
60 ns
80 ns
100 ns
120 ns
Cursor 1 0 ns

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.

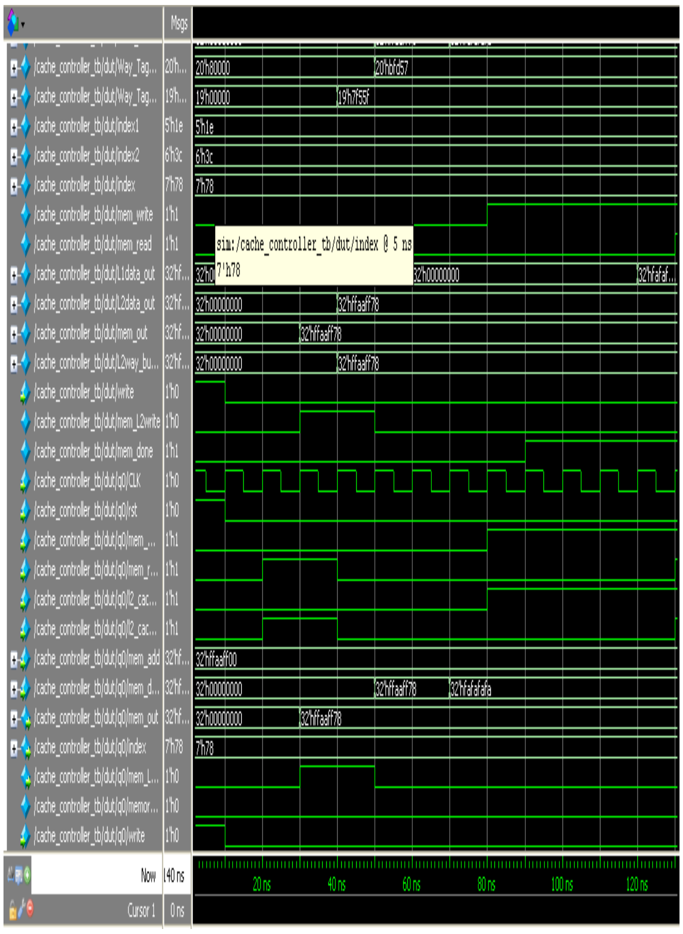

Msgs
/cache_controller_tb/dut/Way_Tag...
/cache_controller_tb/dut/Way_Tag...
/cache_controller_tb/dut/index1
/cache_controller_tb/dut/index2
/cache_controller_tb/dut/index
/cache_controller_tb/dut/mem_write
/cache_controller_tb/dut/mem_read
/cache_controller_tb/dut/L1data_out
/cache_controller_tb/dut/L2data_out
/cache_controller_tb/dut/mem_out
/cache_controller_tb/dut/L2way_bu...
/cache_controller_tb/dut/write
/cache_controller_tb/dut/mem_L2write
/cache_controller_tb/dut/mem_done
/cache_controller_tb/dut/q0/CLK
/cache_controller_tb/dut/q0/rst
/cache_controller_tb/dut/q0/mem_...
/cache_controller_tb/dut/q0/mem_r...
/cache_controller_tb/dut/q0/l2_cac...
/cache_controller_tb/dut/q0/l2_cac...
/cache_controller_tb/dut/q0/mem_add
/cache_controller_tb/dut/q0/mem_d...
/cache_controller_tb/dut/q0/mem_out
/cache_controller_tb/dut/q0/index
/cache_controller_tb/dut/q0/mem_L...
/cache_controller_tb/dut/q0/memor...
/cache_controller_tb/dut/q0/write
sim:/cache_controller_tb/dut/index @ 5 ns
7'h78
Now 140 ns
Cursor 1 0 ns
20 ns
40 ns
60 ns
80 ns
100 ns
120 ns

Fig. 10: Simulation Result of Top Module with Way-Tag (capture 2)

If the L2cache_mis signal is logic1 then the processor check for the data in the main memory in the similar way if the data found in the main memory then the processor fetch the data from the main memory and in the processor the data is stored in the L1cache memory and L2cache memory and the way-tag address is stored in the way-tag array as shown in fig.10.

The below fig. 11 shows the simulation results of the top module without Way-Tag in this we don't have any tag-array signal if the processor required the data from the memory location it has to check the entire L1cache memory if the required data found it is fetched by the processor otherwise it has to check the L2cache memory if the data is found in the L2cache memory it is fetched by the processor otherwise it checks the main memory and fetch the data from the main memory and in the same process the copies of the data is stored in the L1 and L2 cache memories.

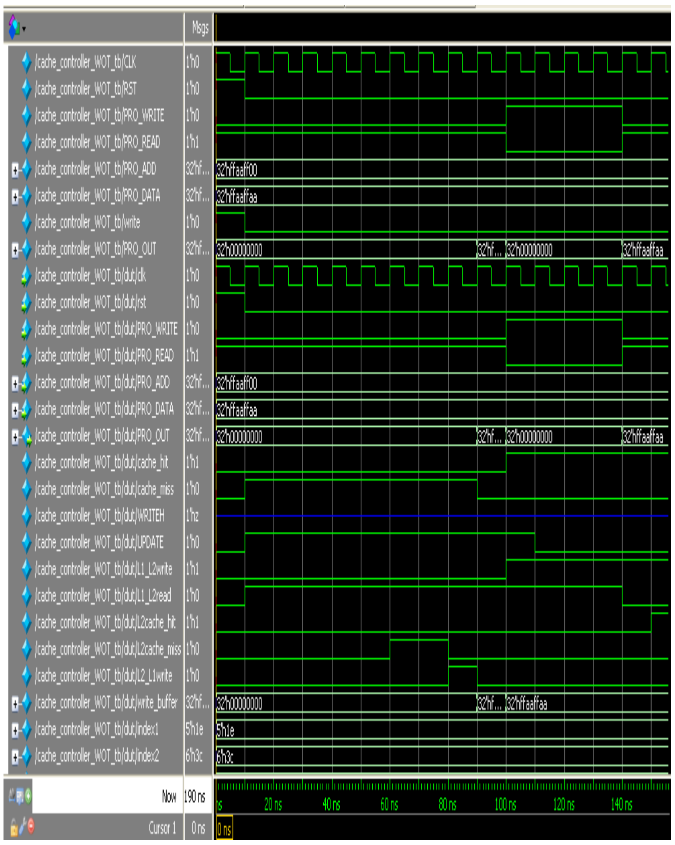

Msgs
/cache_controller_WOT_tb/CLK
/cache_controller_WOT_tb/RST
/cache_controller_WOT_tb/PRO_WRITE
/cache_controller_WOT_tb/PRO_READ
/cache_controller_WOT_tb/PRO_ADD
/cache_controller_WOT_tb/PRO_DATA
/cache_controller_WOT_tb/write
/cache_controller_WOT_tb/PRO_OUT
/cache_controller_WOT_tb/dut/clk
/cache_controller_WOT_tb/dut/rst
/cache_controller_WOT_tb/dut/PRO_WRITE
/cache_controller_WOT_tb/dut/PRO_READ
/cache_controller_WOT_tb/dut/PRO_ADD
/cache_controller_WOT_tb/dut/PRO_DATA
/cache_controller_WOT_tb/dut/PRO_OUT
/cache_controller_WOT_tb/dut/cache_hit
/cache_controller_WOT_tb/dut/cache_miss
/cache_controller_WOT_tb/dut/WRITEH
/cache_controller_WOT_tb/dut/UPDATE
/cache_controller_WOT_tb/dut/L1_L2write
/cache_controller_WOT_tb/dut/L1_L2read
/cache_controller_WOT_tb/dut/L2cache_hit
/cache_controller_WOT_tb/dut/L2cache_miss
/cache_controller_WOT_tb/dut/L2_L1write
/cache_controller_WOT_tb/dut/write_buffer
/cache_controller_WOT_tb/dut/index1
/cache_controller_WOT_tb/dut/index2
Now 190 ns
Cursor 1 0 ns
20 ns
40 ns
60 ns
80 ns
100 ns
120 ns
140 ns

Fig. 11: Simulation Result of Top Module without Way-Tag

Synthesis Report:

Synthesis report without way-tag array

Power Supply Summary			
	Total	Dynamic	Quiescent
Supply Power (mW)	1597.00	355.56	1241.45

Synthesis report with way-tag array

Power Supply Summary			
	Total	Dynamic	Quiescent
Supply Power (mW)	901.15	146.89	1164.67

VII. CONCLUSION

This paper presents a new energy-efficient cache technique for high-performance microprocessors employing the write-through policy. The proposed technique attaches a tag to each way in the L2 cache. This way tag is sent to the way-tag arrays in the L1 cache when the data is loaded from the L2 cache to the L1 cache. Utilizing the way tags stored in the way-tag arrays, the L2 cache can be accessed as a direct-mapping cache during the subsequent write hits, thereby reducing cache energy consumption. Simulation results demonstrate significantly reduction in cache energy consumption with minimal area overhead and no performance degradation. Furthermore, the idea of way tagging can be applied to many existing low-power cache techniques such as the phased access cache to further reduce cache energy consumption.

REFERENCES

[1] G. H.Asadi,V. Sridharan, M. B. Tahoori, Andd.Kaeli, "Balancing Performance And Reliability In The Memory Hierarchy," In Proc. Int. Symp.Perform. Anal. Syst. Softw., 2005, Pp. 269–279.
[2] C. Su And A. Despain, "Cache Design Tradeoffs For Power And Performance Optimization: A Case Study," In Proc. Int. Symp. Low Power Electron. Design, 1997, Pp. 63–68.
[3] C. Zhang, F. Vahid, And W. Najjar, "A Highly-Configurable Cache Architecture For Embedded Systems," In Proc. Int. Symp. Comput. Arch., 2003, Pp. 136–146.
[4] B. Brock And M. Exerman, "Cache Latencies Of The Powerpc Mpc7451," Freescale Semiconductor, Austin, Tx, 2006. [Online]. Available: Cache.Freescale.Com
[5] A.Ma, M. Zhang, And K.Asanovi, "Way Memoization To Reduce Fetch Energy In Instruction Caches," In Proc. Isca Workshop Complexity Effective Design, 2001, Pp. 1–9.
[6] T. Ishihara And F. Fallah, "A Way Memoization Technique For Reducing Power Consumption Of Caches In Application Specific Integrated Processors," In Proc. Design Autom. Test Euro. Conf., 2005, Pp. 358–363.
[7] R. Min, W. Jone, And Y. Hu, "Location Cache: A Low-Power L2 Cache System," In Proc. Int. Symp. Low Power Electron. Design, 2004, Pp. 120–125.
[8] B. Calder, D. Grunwald, And J Emer, "Predictive Sequential Associative Cache," In Proc. 2nd Ieee Symp. High-Perform. Comput. Arch., 1996, Pp. 244–254.
[9] T. N. Vijaykumar, "Reactive-Associative Caches," In Proc. Int. Conf. Parallel Arch. Compiler Tech., 2011, P. 4961.
[10] J. Dai And L. Wang, "Way-Tagged Cache: An Energy Efficient L2 Cache Architecture Under Write Through Policy," In Proc. Int. Symp. Low Power Electron. Design, 2009, Pp. 159 164.
[11] R.Min,W. Jone, And Y. Hu, "Phased Tag Cache: An Efficient Low Power Cache System," In Proc. Int. Symp. Circuits Syst., 2004, Pp. 23–26.
[12] About Cache Available at:
http://www.tfinley.net/notes/cps104/cache.html#direct
(Accessed: 20 Aprial 2014)
[13] About Cache Basics by Gene Cooperman Available at:
http://www.ccs.neu.edu/course/com3200/parent/NOTES/cache-basics.html
(Accessed: 17 April 2014)
[14] About Set Associative Cache Available at:
http://www.cs.umd.edu/class/sum2003/cmsc311/Notes/Memory/set.html
(Accessed: 03 May 2014)
[15] A brief tutorial on Xilinx. Available at:
https://www.digilentinc.com/Data/Documents/Tutorials/Xilinx%20ISE%20WebPACK%20VHDL%20Tutorial.pdf
(Accessed: 19 May 2014)
[16] 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)
[17] 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
(Accessed: 1 June 2014)