International Journal of Engineering Science Invention Research & Development; Vol. III, Issue X, April 2017
www.ijesird.com, e-ISSN: 2349-6185

ENCODING CACHE MEMORY DATA BITS FOR REMOVAL OF SAME TAG BITS

[1]SK.NAGUR BASHA, [2] P.L.S NARASIMHARAO
PG Scholar[1], Assistant Professor[2], Dept of ECE, DVR & Dr HS MIC College of Technology, Kanchikacherla, AP, India

ABSTRACT: Many high-performance microprocessors employ cache write-through policy for performance improvement and at the same time achieving good tolerance to soft errors in on chip caches. However, write-through policy also incurs large energy overhead due to the increased accesses to caches at the lower level (e.g., L2 caches) during write operations. In this paper, we propose a new cache architecture referred to as way-tagged cache to improve the energy efficiency of write-through caches. By maintaining the way tags of L2 cache in the L1 cache during read operations, the proposed technique enables L2 cache to work in an equivalent direct-mapping manner during write hits, which account for the majority of L2 cache accesses.

I. INTRODUCTION

We now give an overview of RAM – Random Access Memory. This is the memory called “primary memory” or “core memory”. The term “core” is a reference to an earlier memory technology in which magnetic cores were used for the computer’s memory. This discussion will pull material from a number of chapters in the textbook.

Primary computer memory is best considered as an array of addressable units. Addressable units are the smallest units of memory that have independent addresses. In a byte-addressable memory unit, each byte (8 bits) has an independent address, although the computer often groups the bytes into larger units (words, long words, etc.) and retrieves that group. Most modern computers manipulate integers as 32-bit (4-byte) entities, so retrieve the integers four bytes at a time.

In this author’s opinion, byte addressing in computers became important as the result of the use of 8–bit character codes. Many applications involve the movement of large numbers of characters (coded as ASCII or EBCDIC) and thus profit from the ability to address single characters. Some computers, such as the CDC–6400, CDC–7600, and all Cray models, use word addressing. This is a result of a design decision made when considering the main goal of such computers – large computations involving integers and floating point numbers. The word size in these computers is 60 bits (why not 64? – I don’t know), yielding good precision for numeric simulations such as fluid flow and weather prediction.

Although this is not the definition, virtual memory has always been implemented by pairing a fast DRAM Main Memory with a bigger, slower “backing store”. Originally, this was magnetic drum memory, but it soon became magnetic disk memory. Here again is the generic two–stage memory diagram, this time focusing on virtual memory.

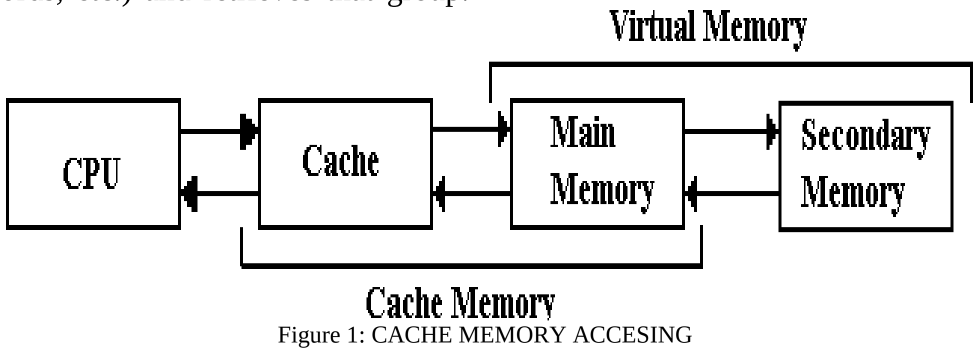


Figure 1: CACHE MEMORY ACCESING

The invention of time–sharing operating systems introduced another variant of VM, now part of the common definition. A program and its data could be “swapped out” to the disk to allow another program to run, and then “swapped in” later to resume.

Virtual memory allows the program to have a logical address space much larger than the computers physical address space. It maps logical addresses onto physical addresses and moves “pages” of memory between disk and main memory to keep the program running.

An address space is the range of addresses, considered as unsigned integers that can be generated. An N–bit address can access 2^N items, with addresses $0 \dots 2^N - 1$.

16–bit address 2^{16} items	0 to	65535
20–bit address 2^{20} items	0 to	1,048,575
32–bit address 2^{32} items	0 to	4,294,967,295

The cache systems are divided into *three* categories, to implement cache system. As shown in figure, the lower order 4-bits from 16 words in a block constitute a word field. The second field is known as block field used to distinguish a block from other blocks. Its length is 7-bits, when a new block enters the cache; the 7-bit cache block field determines the cache position in which this block must be stored. The third field is a Tag field, used to store higher order 5-bits of the memory address of the block, and to identify which of the 32blocks are mapped into the cache.

Tag	Block	Word
5	7	4

Figure 2: Main Memory Address

It is the simplest mapping technique, in which each block from the main memory has only one possible location in the cache organization. For example, the block I of the main memory maps on to block i module128 of the cache. Therefore, whenever one of the main memory blocks 0, 128, 256, is loaded in the cache, it is stored in the block 0. Block 1, 129, 257, are stored in block 1 of the cache and so on.

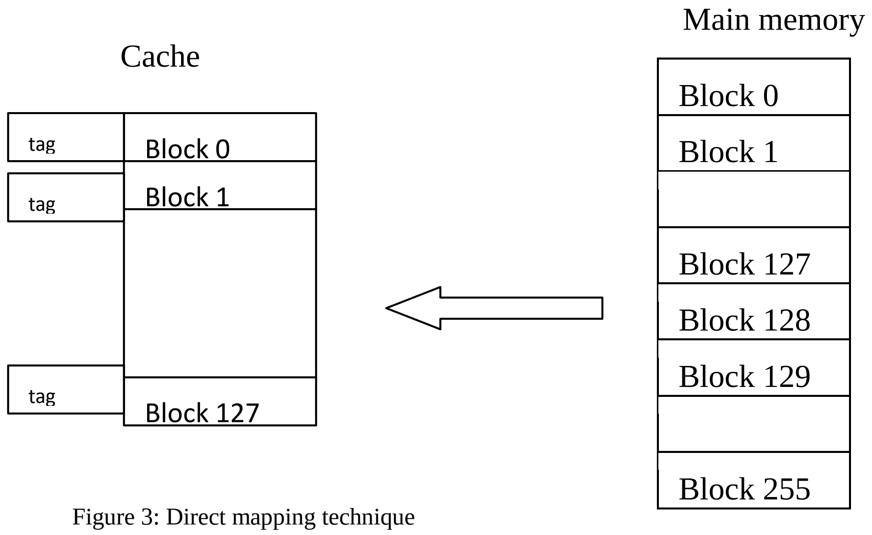


Figure 3: Direct mapping technique

It is a combination of the direct and associative-mapping techniques can be used. Blocks of the cache are grouped into sets and the mapping allows a block of main memory to reside in any block of the specific set. In this case memory blocks 0, 64,128……4032 mapped into cache set 0, and they can occupy either of the two block positions within this set. The cache might contain the desired block. The tag field of the address must then be associatively compared to the tags of the two blocks of the set to check if the desired block is present this two associative search is simple to implement

Many of the modern developments in memory technology involve **Synchronous Dynamic Random Access Memory**, SDRAM for short. Although we have not mentioned it, earlier memory was asynchronous, in that the memory speed was not related to any external speed. In SDRAM, the memory is synchronized to the system bus and can deliver data at the bus speed. The earlier SDRAM chips could deliver one data item for every clock pulse; later designs called DDR SDRAM (for Double Data Rate SDRAM) can deliver two data items per clock pulse. Double Data Rate SDRAM (DDR–SDRAM) doubles the bandwidth available from SDRAM by transferring data at both edges of the clock.

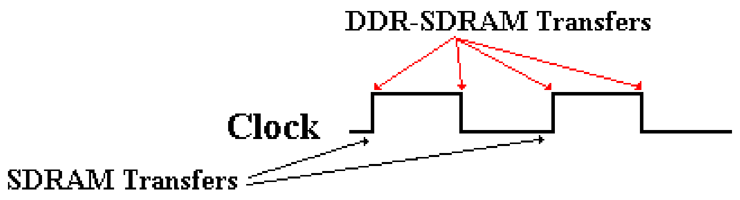
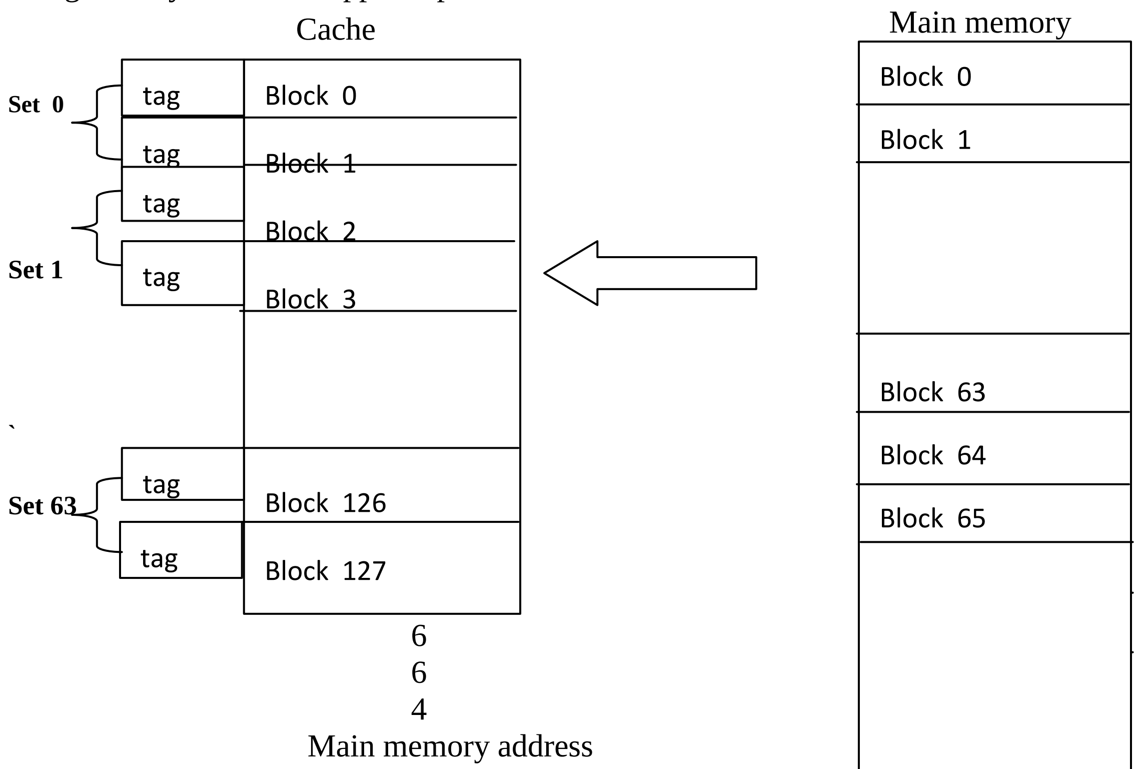


Figure 4: DDR-SDRAM Transfers Twice as Fast

As an example, we quote from the Dell Precision T7500 advertisement of June 30, 2011. The machine supports dual processors, each with six cores. Each of the twelve cores has two 16 KB L1 caches (an Instruction Cache and a Data Cache) and a 256 KB (?) L2 cache. The processor pair shares a 12 MB Level 3 cache. The standard memory configuration calls for 4GB or DDR3 memory, though the system will support up to 192 GB. The memory bus operates at 1333MHz (2666 million transfers per second). If it has 64 data lines to the L3 cache (following the design of the Dell Dimension 4700 of 2004), this corresponds to $2.666 \bullet 10^9$ transfers/second • 8 bytes/transfer ≈ $2.13 \bullet 10^{10}$ bytes per second. This is a peak transfer rate of 19.9 GB/sec.



II. LITERATURE SURVEY

A. CACHE MAPPING SCHEME

If the CPU generates an address for a particular word in main memory, and that data happens to be in the cache, the same main memory address cannot be used to access the cache. Hence a mapping scheme is required that converts the generated main memory address into a cache location. Mapping Scheme also determines where the block will placed when it originally copied into the cache.

B. CACHE OPERATION

Most of the time the cache is busy filling cache lines (reading from memory) But the processor doesn't write a cache line which can be up to 128 bytes - it only writes between 1 and 8 bytes. Therefore it must perform a read-modify-write sequence on the cache line. Also, the cache uses one of two write operations: Write-through, where data is updated both on the cache and in the main memory Write-back, where data is written to the cache, and updated in the main memory only when the cache line is replaced.

C. MAPPING SCHEME 1:
DIRECT MAPPED CACHE:

The cache consists of normal high speed random access memory, and each location in the cache holds the data, at an address in the cache given by the lower significant bits of the main memory address. This enables the block to be selected directly from the lower significant bits of the memory address. The remaining higher significant bits of the address are stored in the cache with the data to complete the identification of the cached data.

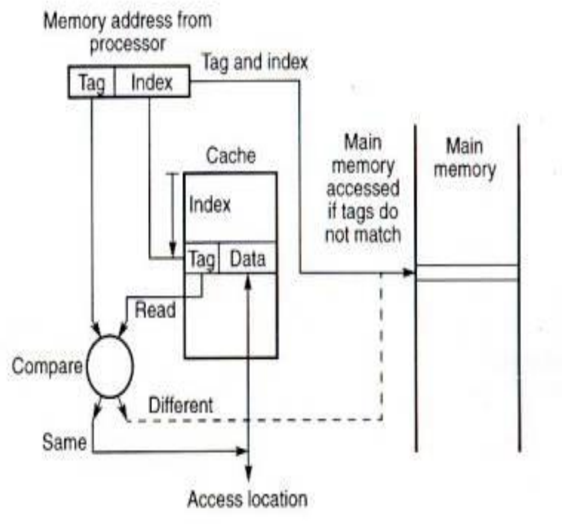


Figure 5: cache direct mapping

The address from the processor is divided into two fields, a tag and an index. The tag consists of the higher significant bits of the address, which are stored with the data. The index is the lower significant bits of the address used to address the cache. When the memory is referenced, the index is first used to access a word in the cache. Then the tag stored in the accessed word is read and compared with the tag in the address. If the two tags are the same, indicating that the word is the one required, access is made to the addressed cache word. However, if the tags are not the same, indicating that the required word is not in the cache, reference is made to the main memory to find it. For a memory read operation, the word is then transferred into the cache where it is accessed. It is possible to pass the information to the cache and the processor simultaneously, i.e., to read-through the cache, on a miss. The cache location is altered for a write operation. The main memory may be altered at the same time (write-through) or later. The main memory address is compared of a tag, an index, and a word with in a line. All the words within a line in the cache have the same stored tag. The index part to the address is used to access the cache and the stored tag is compared with required tag address. For a read operation, if the tags are the same the word within the block is selected for transfer to the processor. If the tags are not the same, the block containing the required word is first transferred to the cache.

III. METHODOLOGY

A. READ AND WRITE ADDRESS CHANNELS:

Read and write transactions each have their own address channel. The appropriate address channel carries all of the required address and control information for a transaction.

B. READ DATA CHANNEL

The read data channel conveys both the read data and any read response information from the slave back to the master. The read data channel includes:

- The data bus, which can be 8, 16, 32, 64, 128, 256, 512, or 1024 bits wide
- A read response indicating the completion status of the read transaction.

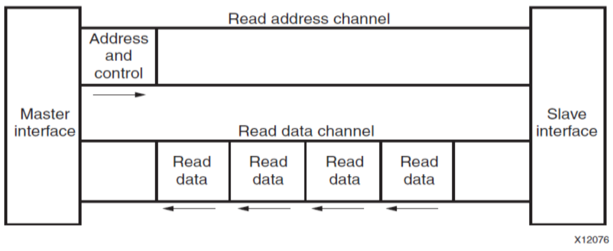


Figure 6: Channel Architecture of Reads

C. WRITE DATA CHANNEL

The write data channel conveys the write data from the master to the slave and includes:

- The data bus, which can be 8, 16, 32, 64, 128, 256, 512, or 1024 bits wide
- One byte lane strobe for every eight data bits, indicating which bytes of the data bus are valid.

Write data channel information is always treated as buffered, so that the master can perform write transactions without slave acknowledgement of previous write transactions.

D. WRITE RESPONSE CHANNEL:

The write response channel provides a way for the slave to respond to write transactions. All write transactions use completion signaling. The completion signal occurs once for each burst, not for each individual data transfer within the burst.

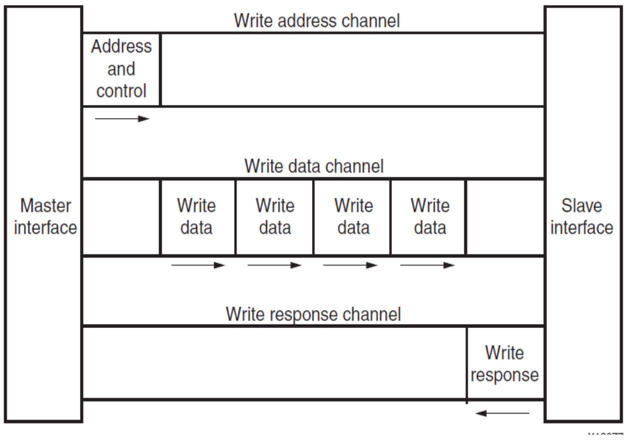


Figure 7: Channel Architecture of Writes

E. READ BURST EXAMPLE:

A read burst of four transfers. In this example, the master drives the address, and the slave accepts it one cycle later. After the address appears on the address bus, the data transfer occurs on the read data channel. The slave keeps the VALID signal LOW until the read data is available. For the final data transfer of the burst, the slave asserts the RLAST signal to show that the last data item is being transferred.

The master also drives a set of control signals showing the length and type of the burst, but these signals are omitted from the figure for clarity.

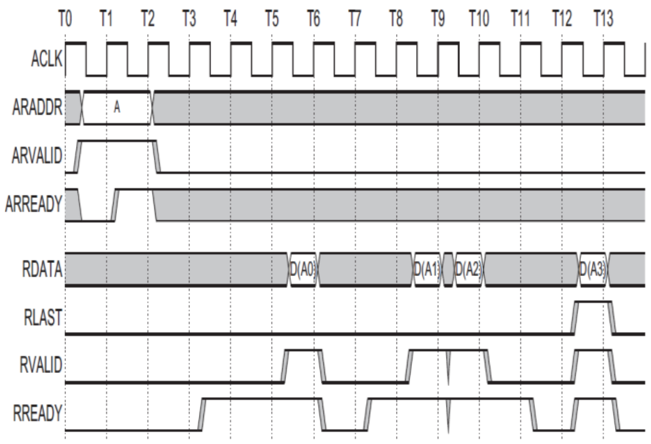


Figure 8: Read burst

F. WRITE BURST EXAMPLE

The process starts when the master sends an address and control information on the write address channel. The master then sends each item of write data over the write data channel. When the master sends the last data item, the **WLAST** signal goes HIGH.

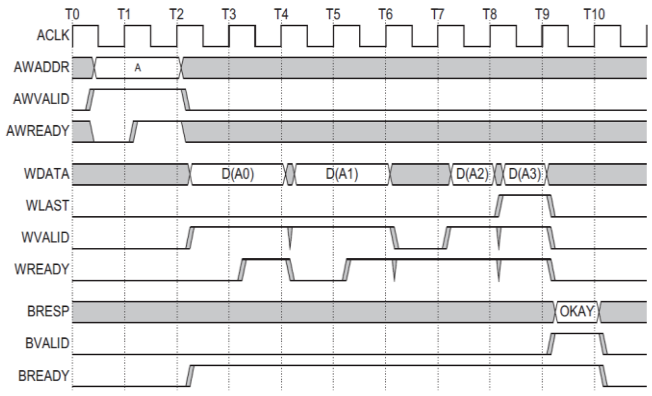


Figure 9: Write burst

G. TRANSACTION ORDERING

The protocol enables out-of-order transaction completion. It gives an ID tag to every transaction across the interface. The protocol requires that transactions with the same ID tag are completed in order, but transactions with different ID tags can be completed out of order.

If a master requires that transactions are completed in the same order that they are issued, then they must all have the same ID tag. If, however, a master does not require in-order transaction completion, it can supply the transactions with different ID tags, enabling them to be completed in any order. In a multi master system, the interconnect is responsible for appending additional information to the ID tag to ensure that ID tags from all masters are unique.

The ID tag is similar to a master number, but with the extension that each master can implement multiple virtual masters within the same port by supplying an ID tag to indicate the virtual master number. Although complex devices can make use of the out-of-order facility, simple devices are not required to use it. Simple masters can issue every transaction with the same ID tag, and simple slaves can respond to every transaction in order, irrespective of the ID tag.

IV. SIMULATION RESULTS

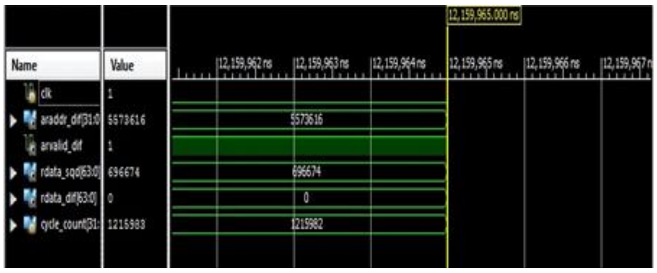


Figure 10: Cache Output

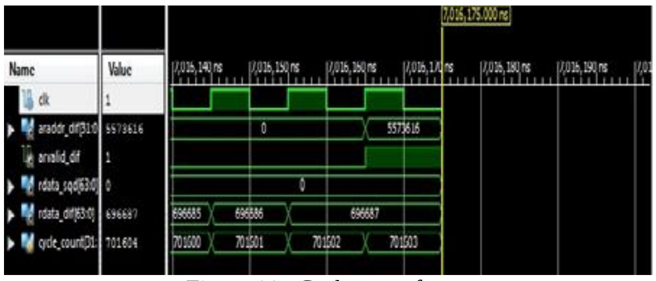
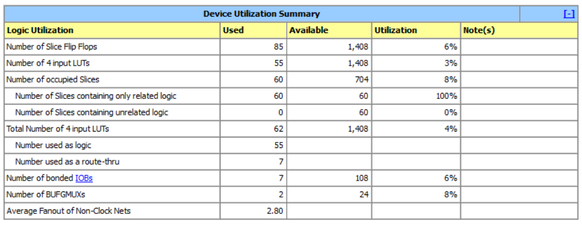


Figure 11: Cache waveforms

POWER ANALYSIS:



Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	85	1,408	6%	
Number of 4 input LUTs	55	1,408	3%	
Number of occupied Slices	60	704	8%	
Number of Slices containing only related logic	60	60	100%	
Number of Slices containing unrelated logic	0	60	0%	
Total Number of 4 input LUTs	62	1,408	4%	
Number used as logic	55			
Number used as a route-thru	7			
Number of bonded IOBs	7	108	6%	
Number of BUFGMUXs	2	24	8%	
Average Fanout of Non-Clock Nets	2.80			

V. 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.

In this proposed system using AXI protocol for data transmission and here developed the FIFO architecture. Main advantages for this project modification, reduce the power consumption up to 30%, reduce the execution time and no data lose if Cache enable or Disable.

REFERENCES

1. G. Konstadinidis, K. Normoyle, S. Wong, S. Bhutani, H. Stuimer, T. Johnson, A. Smith, D. Cheung, F. Romano, S. Yu, S. Oh, V.Melamed, S. Narayanan, D. Bunsey, C. Khieu, K. J. Wu, R. Schmitt, A. Dumlao, M. Sutera, J. Chau, andK. J. Lin, “Implementation of a third-generation 1.1-GHz 64-bit microprocessor,” *IEEE J. Solid-State Circuits*, vol. 37, no. 11, pp. 1461–1469, Nov. 2002.
2. S. Rusu, J. Stinson, S. Tam, J. Leung, H. Muljono, and B. Cherkauer, “A 1.5-GHz 130-nm itanium 2 processor with 6-MB on-die L3 cache,” *IEEE J. Solid-State Circuits*, vol. 38, no. 11, pp. 1887–1895, Nov. 2003.
3. D. Wendell, J. Lin, P. Kaushik, S. Seshadri, A. Wang, V. Sundararaman, P. Wang, H. McIntyre, S. Kim, W. Hsu, H. Park, G. Levinsky, J. Lu, M. Chirania, R. Heald, and P. Lazar, “A 4 MB on-chip L2 cache for a 90 nm 1.6 GHz 64 bit SPARC microprocessor,” in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, 2004, pp. 66–67.
4. S. Segars, “Low power design techniques for microprocessors,” in *Proc. Int. Solid-State Circuits Conf. Tutorial*, 2001, pp. 268–273. [5] A. Malik, B. Moyer, and D. Cermak, “A low power unified cache architecture providing power and performance flexibility,” in *Proc. Int. Symp. Low Power Electron. Design*, 2000, pp. 241–243.
5. D. Brooks, V. Tiwari, and M. Martonosi, “Wattch: A framework for architectural- level power analysis and optimizations,” in *Proc. Int. Symp. Comput. Arch.*, 2000, pp. 83–94.
6. J. Maiz, S. hareland, K. Zhang, and P.Armstrong, “Characterization of multi-bit soft error events in advanced SRAMs,” in *Proc. Int. Electron Devices Meeting*, 2003, pp. 21.4.1–21.4.4.
7. K. Osada, K. Yamaguchi, and Y. Saitoh, “SRAM immunity to cosmic-ray-induced multierrors based on analysis of an induced parasitic bipolar effect,” *IEEE J. Solid-State Circuits*, pp. 827–833, 2004.
8. F. X. Ruckerbauer and G. Georgakos, “Soft error rates in 65 nm SRAMs: Analysis of new phenomena,” in *Proc. IEEE Int. On-Line Test. Symp.*, 2007, pp. 203–204.
9. 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.
10. L. Li, V. Degalahal, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, “Soft error and energy consumption interactions: A data cache perspective,” in *Proc. Int. Symp. Low Power Electron. Design*, 2004, pp. 132–137.
11. http://www.cs.ucr.edu/~junyang/teach/161_W06/slides/L15-cache2.pdf
12. http://www.authorstream.com/Presentation/rajendra.raju-1199306-cache-memory/
13. http://news.softpedia.com/newsPDF/How-Cache-Memory-Works-83803.pdf
14. http://homepage.cs.uiowa.edu/~ghosh/4-8-08.pdf
15. http://www.spec.org/cpu2000/research/umn/a