International Journal of
Emerging Research in Management &Technology
ISSN: 2278-9359 (Volume-3, Issue-5)

Research Article | **May 2014**

An Efficient Low Power L2 Cache Architecture Using Pre-Computation Logic and Bloom Filter

M. Gopinath,, **L. Praveen**,
PG Scholar,
Department of ECE,
Ranganathan Engineering College,
Coimbatore, Tamil Nadu, India – 641109.

P. Ramalakshmi,
Assistant Professor,
Department of ECE,
Ranganathan Engineering College,
Coimbatore, Tamil Nadu, India – 641109

Abstract—

Caches consume a significant amount of energy in modern microprocessors. To design an energy-efficient microprocessor, it is important to optimize cache energy consumption. 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. Write-through policy also consumes large power due to the increased access to caches in different level during write operation. In this paper, we propose an efficient low power cache design referred to as way-tagged cache using Pre-Computation Logic. The cache architecture is designed using a Pre-Computational technique in place of the comparators. This helps to achieve low power consumption than existing technique.

Keywords— ***Cache, low power, Pre-computation logic, way-tagged cache, write-through policy.***

I. INTRODUCTION

The cache is a smaller, faster memory which stores copies of the data from frequently used main memory locations. As long as most memory accesses are cached memory locations, the average latency of memory accesses will be closer to the cache latency than to the latency of main memory. When the processor needs to read from or write to a location in main memory, it first checks whether a copy of that data is in the cache. If so, the processor immediately reads from or writes to the cache, which is much faster than reading from or writing to main memory. The use of cache memory, however, has often aggravated the bandwidth problem rather than reduce it. Optimizing the design has four general aspects:

(1) Maximizing the hit ratio,
(2) Minimizing the access time to data in the cache,
(3) Minimizing the delay due to a miss, and
(4) Minimizing the overheads of updating main memory, maintaining multi-cache consistency, etc.

Multi-Level on-chip cache systems have been widely adopted in high- performance architectures. Two techniques commonly available to keep data throughout the memory levels, write-through and write –back polices. In write-back policy initially, writing is done only to the cache. The write to the backing store is postponed until the cache blocks containing the data are about to be modified/replaced by new content. While under the write-through policy, Write is done synchronously both to the cache and to the backing store all copies of a cache block are updated immediately after the cache block is modified at the current cache. 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.

This feature is important as CMOS technology is scaled into the nano meter range, where soft errors have emerged as a major reliability issue in on-chip cache systems. Due to this feature, many high-performance microprocessor designs have adopted the write-through policy while enabling better tolerance to soft errors, the write-through policy also incurs large energy overhead. This is because under the write-through policy, caches at the lower level experience more accesses during write operations. Consider a two-level (i.e., Level-1 and Level-2) cache system for example. If the L1 data cache implements the write-back policy, a write hit in the L1 cache does not need to access the L2 cache. In contrast, if the L1 cache is write-through, then both L1 and L2 caches need to be accessed for every write operation. Obviously, the write -through policy incurs more write accesses in the L2 cache, which in turn increases the energy consumption of the cache system. Power dissipation is now considered as one of the critical issues in cache design. Studies have shown that on-chip caches can consume about 50% of the total power in high-performance microprocessors. In this project, we propose new cache architecture, referred to as efficient low power L2 way-tagged cache architecture using pre-computation logic, to improve the energy efficiency of write-through cache systems with minimal area overhead and no performance degradation

II. RELATED WORK

Many techniques have been developed to reduce cache power dissipation. In this section, we briefly review some existing work related to the proposed technique.

Koji Inoue, Tohru Ishihara, and Kazuaki Murakami, proposed a Way-Predicting Set-Associative Cache for High Performance and Low Energy Consumption. This paper proposes a new approach using way prediction for achieving high performance and low energy consumption of set-associative caches. By accessing only a single cache way predicted, instead of accessing all the ways in a set, the energy consumption can be reduced. This paper shows that the way-predicting set-associative cache improves the ED (energy-delay) product by 60–70% compared to a conventional set-associative cache.

The way-predicting cache speculatively chooses one way before starting the normal cache-access process, and then accesses the predicted way. If the prediction is correct, the cache access has been completed successfully. Otherwise, the cache then searches the other remaining ways. On a prediction-hit, the way-predicting cache consumes only energy for activating the predicted way. In addition, the cache access can be completed in one cycle.

On prediction-misses (or cache misses), however, the cache-access time of the way-predicting cache increases due to the successive process of two phases. Since all the remaining ways are activated in the same manner as a conventional set-associative cache, the way-predicting cache could not reduce energy consumption in this scenario.

Albert Ma, Michael Zhang, and Krste Asanovic, proposed a Way Memoization to Reduce Fetch Energy in Instruction Caches. Way memoization is an alternative to way prediction. As in way prediction schemes, way memoization stores way information (links) within the instruction cache, but in addition maintains a valid bit per link that when set guarantees that the way link is valid. In contrast, way prediction schemes must always read one tag to verify that the prediction is correct.

Way memoization stores tag lookup results (links) within the instruction cache in a manner similar to some way prediction schemes. However, way memoization also associates a valid bit with each link. These valid bits indicate, prior to instruction access, whether the link is correct. This is in contrast to way prediction where the access needs to be verified afterward. This is the crucial difference between the two schemes, and allows way-memoization to work better in CAM-tagged caches. If the link is valid, we simply follow the link to fetch the next instruction and no tag checks are performed. Otherwise, we fall back on a regular tag search to find the location of the next instruction and update the link for future use.

The main complexity in our technique is caused by the need to invalidate all links to a line when that line is evicted. The coherence of all the links is maintained through an invalidation scheme.

Rui Min, Wenben Jone and Yiming Hu, proposed a Phased Tag Cache: An efficient Low Power Cache System. P*hased tag cache* proposed for reducing the power consumption of set-associative caches. In the phased tag cache, the tag is compared in two phases. A small part of the tag is compared in the first phase to determine the data way which a memory reference falls into. The remaining bits of the tag are compared in the second phase to verify if the result from the first phase is valid, by doing so we can eliminate most of the unnecessary activities on the entire tag. By comparing a small part of the tag, it is possible that we find several hits in phase 1. This introduces both time and energy overhead. To alleviate the problem, we propose two circuit designs. The Enforced design requires enforcing the unique property on the least significant bits compared in the first phase, which are called identity bits. The Non-enforced design does not require that. A new cache replacement algorithm associated to the enforced design in phase 2, the enforced phased tag design can reduce the number of sense amplifiers and comparators, compared with non-enforced and regular tag designs. However, the structure of non-enforce phased tag design is similar to that of a conventional tag. We can easily implement a tag system that can be accessed both as a regular tag and a non-enforced phased tag. Simulation results based on Spec2000 benchmark applications suggest that the phased tag cache design has small impact on the cache performance.

III. PRE- COMPUTATION LOGIC

In this section, we propose a pre- computation logic that exploits the way information in L2 cache and reduce switching while comparison to improve energy efficiency. In the proposed architecture the inputs to the block A have been partitioned into two sets, corresponding to the register R1 and R2. The output of the logic block feeds the register R3.

A way-tagged cache consider a conventional set-associative cache system when the L1 data cache loads/writes data from/into the L2 cache; all ways in the L2 cache are activated simultaneously for performance consideration at the cost of energy overhead. The way-tag arrays are very small and the involved energy overhead can be easily compensated. For L1 read operations, neither read hits nor misses need to access the way-tag arrays. This is because read hits do not need to access the L2 cache; while for read misses, the corresponding way tag information is not available in the way-tag arrays. As a result, all ways in the L2 cache are activated simultaneously under read misses.

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.

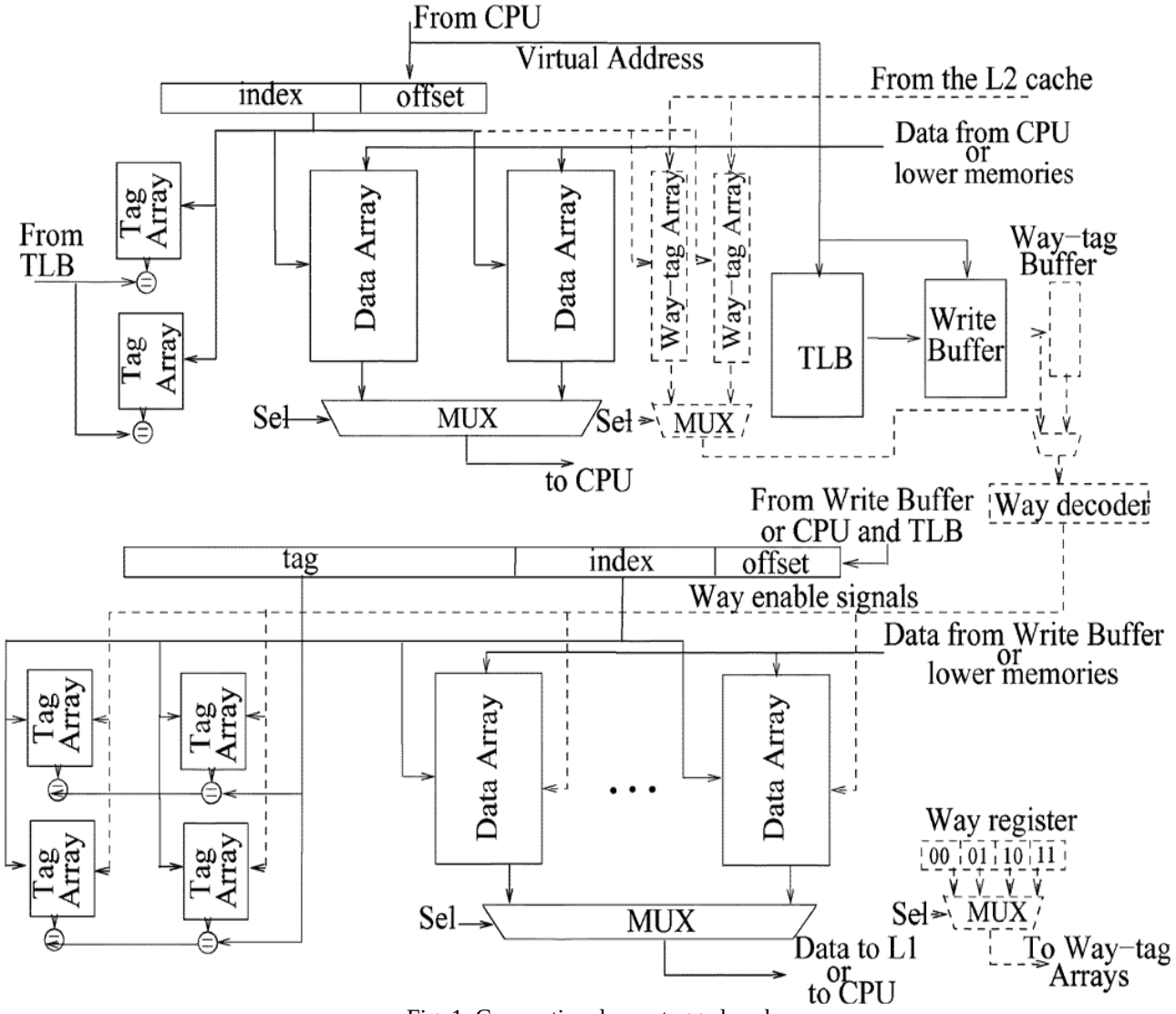

Fig. 1. Conventional way-tagged cache.

This avoids write stalls when the processor waits for write operations to be completed in the L2 cache. 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 ways in the L2 cache. Each way in the L2 cache is encoded into a way tag. A way register stores way tags and provides this information to the way-tag arrays.

TABLE 1
EQUIVALENT L2 ACCESS MODES UNDER DIFFERENT
OPERATIONS IN THE L1 CACHE

	Operation in the L1 cache			
	Read hit	Read miss	Write hit	Write miss
L2	No access	Set-associative	Direct-mapping	Set-associative

In general, both write and read accesses in the L1 cache may need to access the L2 cache. Under the write-through policy, all write operations of the L1 cache need to access the L2 cache. In the case of a write hit in the L1 cache, only one way in the L2 cache will be activated write accesses account for the majority of L2 cache accesses in most applications. In addition, write hits are dominant among all write operations which is shown in Table 1.

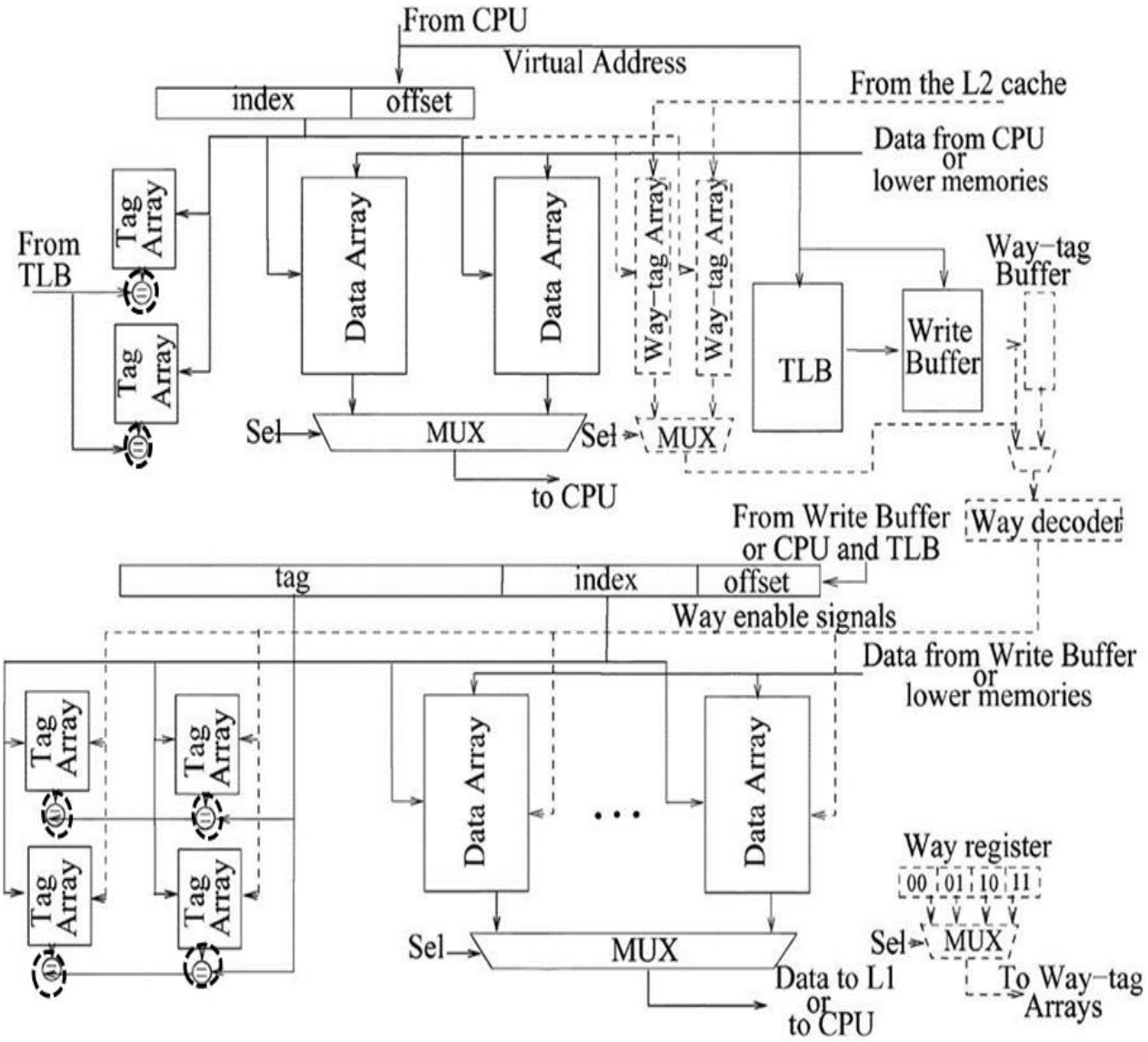

fig. 2 pre-computation logic way tag array.

A. *Pre-Computation Logic Comparator*

In the proposed architecture pre- computation logic is implemented in the place of comparators. In conventional way-tagged cache architecture, when the CPU request for a read/write access the virtual address is converted to physical address by the TLB. The address from the TLB will be compare with the TAG arrays once the comparison result is true then cache hits the data will be accessed, if it is false then the comparison will check all the TAGS which are stored in TAG array. Each address bit in the tag are compared with the address from the TLB. The proposed pre- computation logic reduces this effect by loading the address from TLB to one register R1 and the address from tag array will be loaded one by one in the another register R2 shown in fig. 3. In comparator instead of comparing all the bits together at a time it will just take one bit from each register at a time, if the result is true then the feedback is given to the register R2 which sends the next bit for comparison this continues up to all bits are compared. If the comparison is false the feedback will send a false condition to the register R2 which load the another address from the TAG array for the comparison. This reduces the power consumption due to switching also no impact on cache performance.

B. *Way-Tag Arrays*

In the way-tagged cache, each cache line in the L1 cache keeps its L2 way tag information in the corresponding entry of the way-tag arrays, as shown in Fig. 4, where only one L1 data array and the associated way-tag array are shown for simplicity. When a data is loaded from the L2 cache to the L1 cache, the way tag of the data is written into the way-tag array. At a later time when updating this data

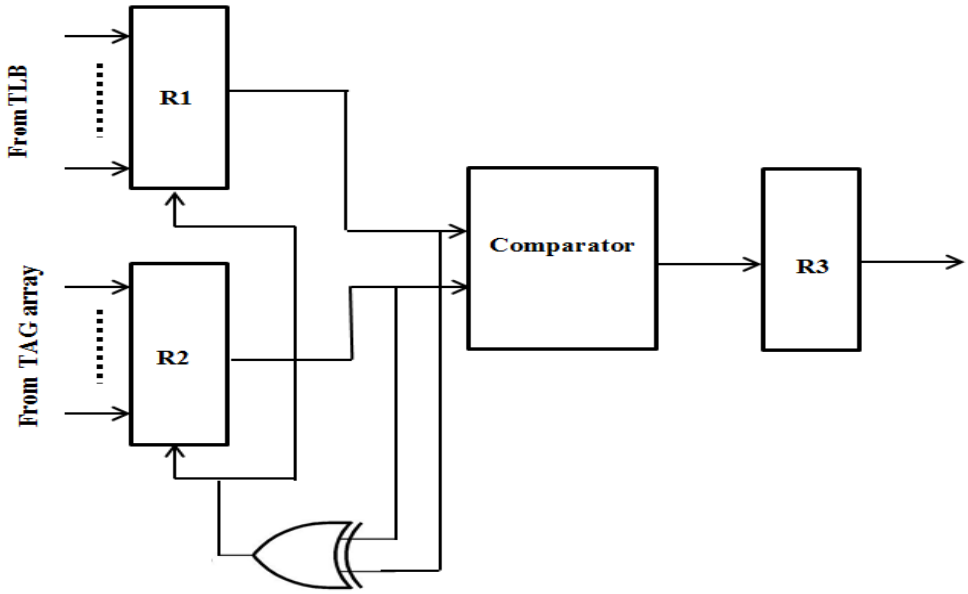

Fig. 3 pre- computation logic comparator.

in the L1 data cache, the corresponding copy in the L2 cache needs to be updated as well under the write-through policy. The way tag stored in the way-tag array is read out and forwarded to the way-tag buffer together with the data from the L1 data cache. Note that the data arrays in the L1 data cache and the way-tag arrays share the same address as the mapping between the two is exclusive.

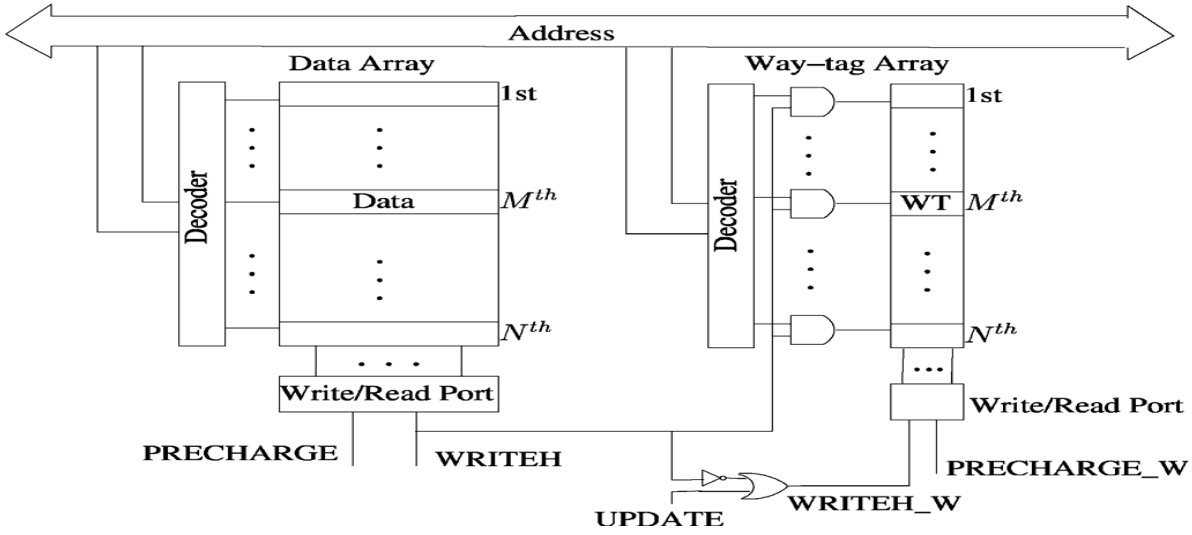

Fig. 4 Way-tag arrays.

The read/write signal of way-tag arrays, WRITEH_W, is generated from the write/read signal of the data arrays in the L1 data cache as shown in Fig. 3.3. A control signal referred to as UPDATE is obtained from the cache controller. When the write access to the L1 data cache is caused by a L1 cache miss, UPDATE will be asserted and allow WRITEH_W to enable the write operation to the way-tag arrays (WRITEH = 1, UPDATE = 1, see fig. 3.4). If a STORE instruction accesses the L1 data cache, UPDATE keeps invalid and WRITE_W indicates a read operation to the way-tag arrays (WRITEH =1, UPDATE =0). During the read operations of the L1 cache, the way-tag arrays do not need to be accessed and thus are deactivated to reduce energy overhead. To achieve this, the word line selection signals generated by the decoder are disabled by WRITEH(WRITEH = 0, UPDATE = 0/1) through AND gates. The above operations are summarized in TABLE II.

When a cache line is evicted from the L2 cache, the status of the cache line changes to “invalid” to avoid future fetching and thus prevent cache coherence issues. A read or write operation to this cache line will lead to a miss, which can be handled by the proposed way-tagged cache. Since way-tag arrays will be accessed only when a data is written into the L1 data cache (either when CPU updates a data in the L1 data cache or when a data is loaded from the L2 cache), they are not affected by cache misses.

IV. BLOOM FILTER

Partial Tag Comparison is comparing a small part of two different tags, instead of comparing the entire tag address for cache hit. The Bloom filter is utilized to check the approximate non membership of a set. When applied to reducing tag comparisons, each cache way is equipped with a Bloom filter. A query to the Bloom filter (e.g., "is address 0×100 in the cache way?") gives either of two results: negative (definite nonexistence) and positive (likely existence). Note that a negative result from the Bloom filter guarantees nonexistence, i.e., a cache way miss. Thus, before the tag structure in each cache way is accessed, first the Bloom filter per cache way is looked up. If the Bloom filter indicates nonexistence, then tag comparison for the cache way is avoided, thereby saving the energy that would have been consumed in tag comparison.

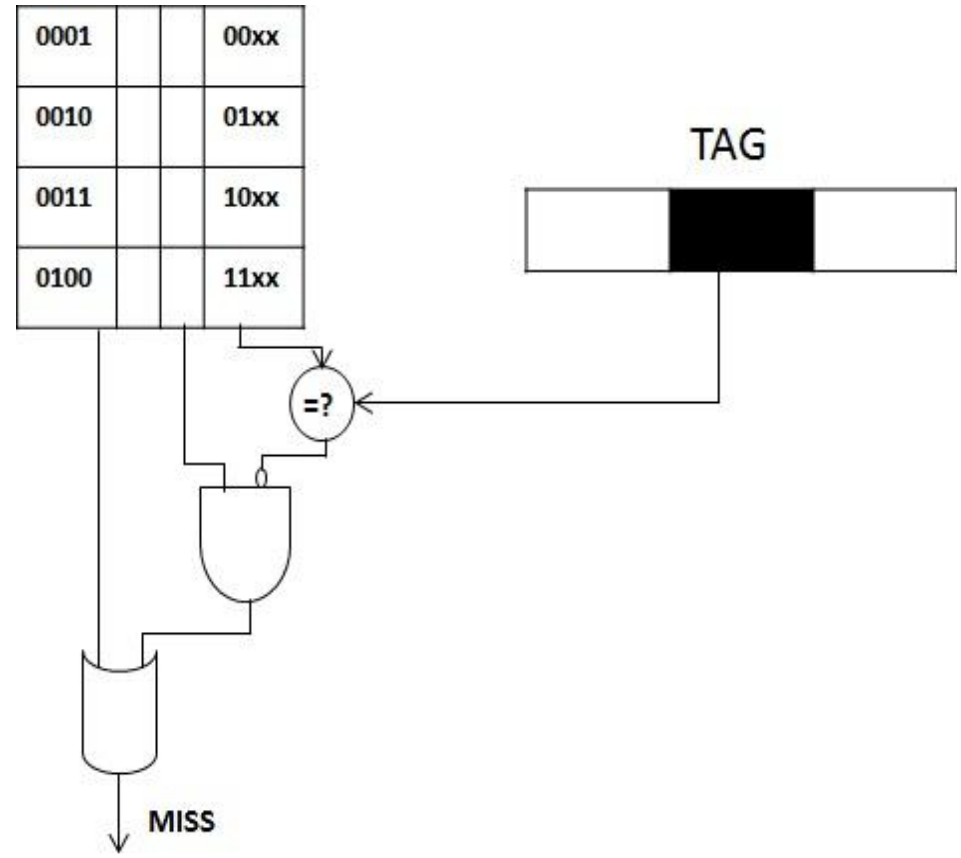

Fig. 5 Bloom Filter Technique.

A. *Bloom Filter Technique*

We introduce a tag comparison technique using bloom filter. In Bloom filter the tag address is grouped according to the unchanged MSB Bits which have different combinations depends on the LSB bits and given an address in the tag array. When a Tag address is given the filter divides the MSB and LSB bits, first the MSB is compared and activate a particular address in the bloom filter TAG array then the comparison is done with the LSB bit in the array once the Cache hit occurs particular data from the L2 cache is given to the CPU and L1 cache. If the cache Miss occurs the Tag address is then given to the higher level of cache.

A BF entry has the tuple <C, Z, S, P>, where C is the counter, Z is the zero flag, S is the singleton flag, and P is the partial tag. The size of the partial tag is small, 3 bits. Thus, compared to the original Bloom filter, the partial tag-enhanced Bloom filter has an overhead of 4 bits (including the S flag) per entry.

The Bloom filter is utilized to check the approximate non-membership of a set. When applied to reducing tag comparisons, each cache way is equipped with a Bloom filter.

B. *L2 Cache Architecture Using Bloom Filter*

In this paper, we show that the partial comparison method can filter out most of the unmatched tag comparisons. In the first step, the partial tag is compared with that of the incoming address. If the comparison does not yield a match, the remainder of the tag does not need to be compared; this saves energy because the corresponding cache way does not include the incoming address. If the partial tag comparison yields a match, then the remainder of the tag is compared. In the proposed architecture the virtual address is given to the TLB, it generates the corresponding Tag address. The TAG address is compared with the TAG array in L1 cache using Pre computational logic. If the Cache hit occurs corresponding Data array is selected and the data is given to the CPU. If the cache Miss occurs then the tag address is given to the L2 cache.

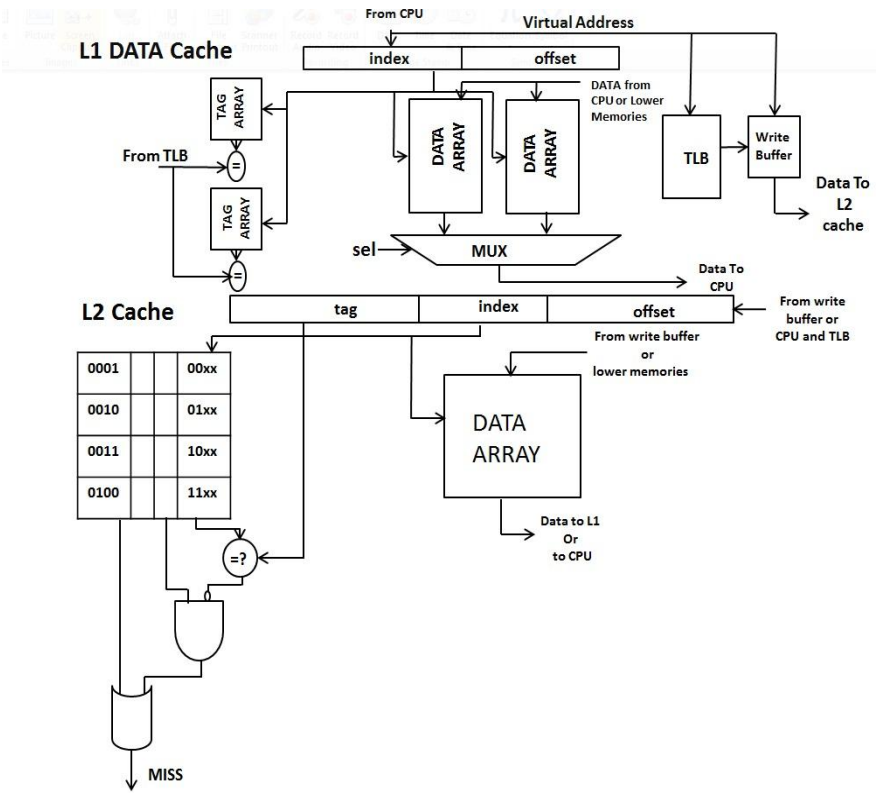

Fig. 6. L2 Cache Architecture Using Bloom Filter Technique

V. EVALUATION AND DISCUSSION

In this section, we evaluate the proposed technique by comparing energy savings with existing cache design techniques. The Pre-computation logic discussed before is designed using VHDL (VHSIC Hardware Description Language). Then the coding is checked and synthesized using the XILINX tool.

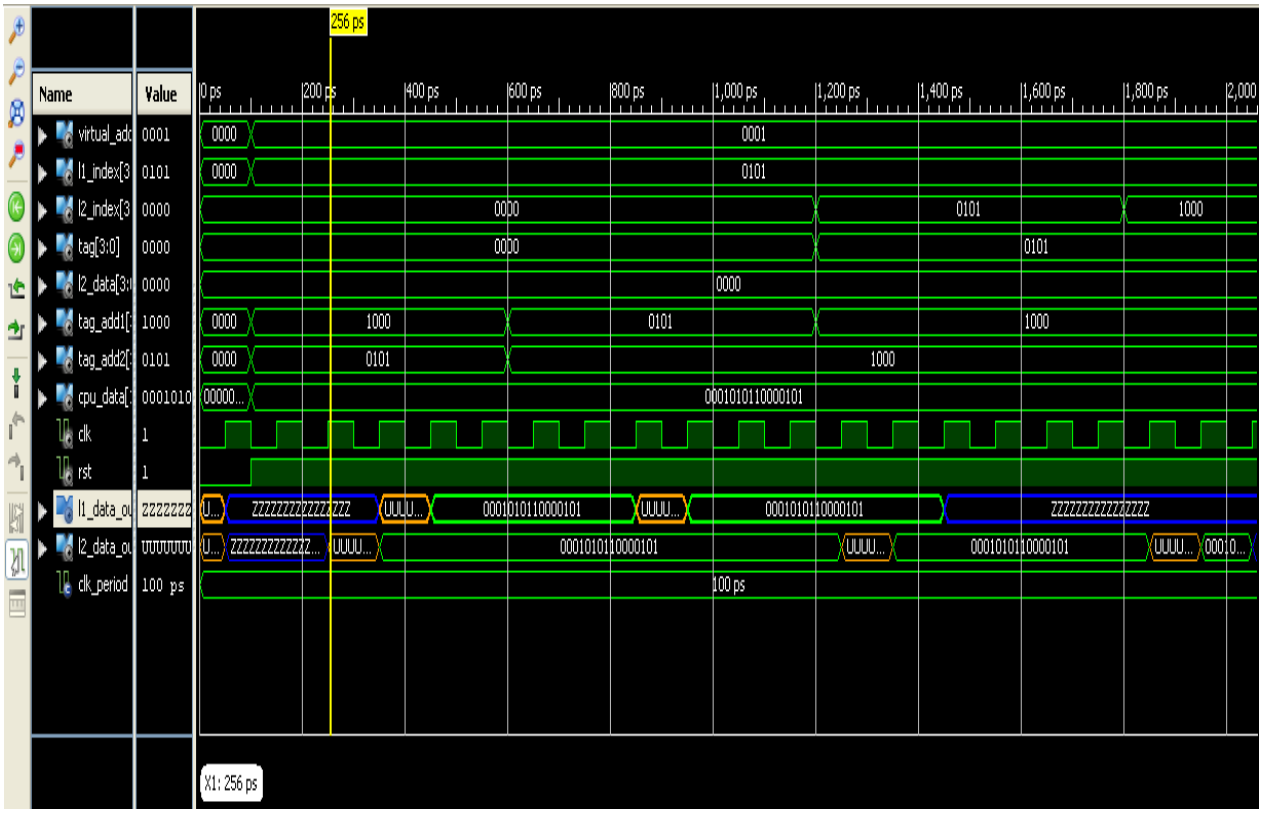

Fig. 7. Output Waveform of L2 Cache Architecture Using Bloom Filter Technique

The simulation is done using MODELSIM tool. The way-tagged cache architecture discussed in this project uses a newly modified pre-computation logic in order to reduce the power consumption. The simulated results using MODELSIM software are given for the proposed pre-computation logic. Then the synthesis report which shows the total power consumption is presented next. Finally a comparison is made with conventional way tagged cache architecture.

Table 2
Power Comparison between cache architecture using bloom filter technique, Conventional way-tagged cache and pre-computation logic cache

Parameter	Conventional way-tagged array	Pre-computation logic	cache architecture using bloom filter technique
NO.OF SLICES	143	157	71
NO. OF I/P LUTs	207	305	126
Total power(w)	**0.802**	**0.408**	**0.346**

Using Xilinx xpower analyzer the total power of L2 Cache Architecture Using Bloom Filter is calculated and shown in fig. 8.

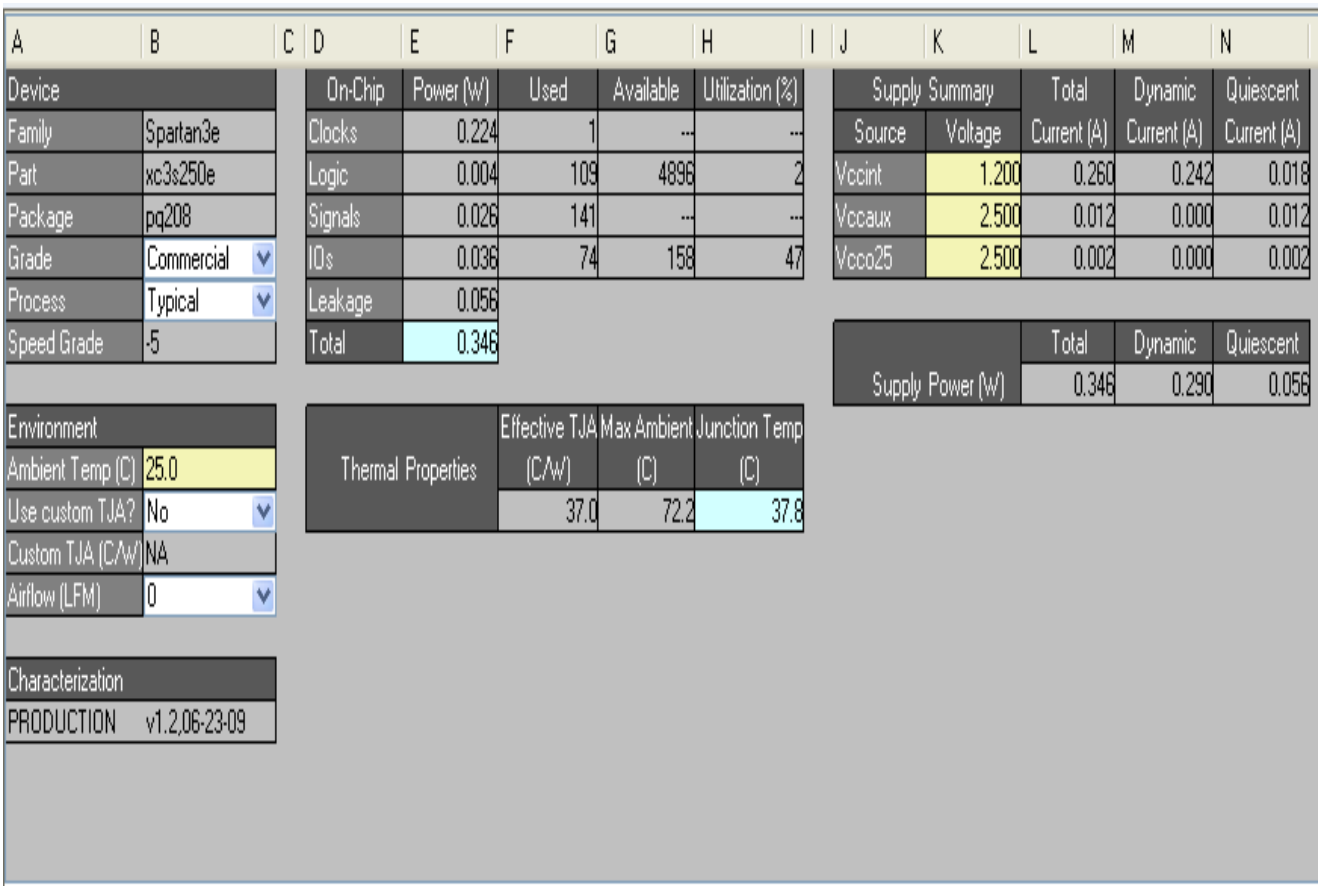

Fig. 8. power analysis of L2 Cache Architecture Using Bloom Filter Technique

From the power analysis it shows that the total power consumption by the proposed L2 Cache architecture using Bloom filter is reduced. Total power in way-tagged architecture using Pre-computation logic is 0.408(w) and conventional architecture 0.802(w), were as in L2 Cache architecture using Bloom Filter is 0.346(w) so the proposed system has reduced 50% power consumption than conventional technique. The proposed technique achieves 50% energy saving with no area over head and no impact on cache performance.

The cache architecture with Bloom filter is designed for reduce Area, power, and is simulated using Xilinx tool. The power analysis is obtained for both way tagged cache and modified L2 cache using Bloom Filter. It is shown that the

proposed L2 cache architecture using Bloom filter provides less power consumption than existing way tag cache. The waveforms are also obtained using the Xilinx tool.

REFERENCES

[1] K. Inoue, T. Ishihara, and K. Murakami, "*Way-predicting set-associative cache for high performance and low energy consumption*," in Proc.Int. Symp. Low Power Electron. Design, 1999, pp. 273–275.
[2] 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.
[3] 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.
[4] 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.
[5] 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.
[6] M. K. Qureshi, D. Thompson, and Y. N. Patt, "*The V-way cache: Demand based associativity via global replacement,"* in Proc. Int. Symp. Comput. Arch., 2005, pp. 544–555.
[7] S. Segars, *"Low power design techniques for microprocessors,"* in Proc. Int. Solid-State Circuits Conf. Tutorial, 2001, pp. 268–273.
[8] T. N. Vijaykumar, *"Reactive-associative caches,"* in Proc. Int. Conf. Parallel Arch. Compiler Tech., 2011, p. 4961.
[9] 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.
[10] 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.
[11] N. Quach, *"High availability and reliability in the Itanium processor,"* IEEE Micro, pp. 61–69, 2000.