PERFORMANCE ANALYSIS OF TAG BIT BASED LOW POWER CACHE MEMORY

Mrs.P.Francis antony selvi, S.Mathavi, N.Meenakshi, S.Lakshmi

1,2,3,4 Assistant Professor/ECE Department, R.V.S. College of Engineering, Dindigul

Abstract—A new cache design technique, referred to as early tag access (ETA) cache is proposed to improve the energy efficiency of data caches in embedded processors. The proposed technique performs ETAs to determine the destination ways of memory instructions before the actual cache accesses. It, thus, enables only the destination way to be accessed if a hit occurs during the ETA. The proposed ETA cache can be configured under two operation modes to exploit the tradeoffs between energy efficiency and performance. It is shown that our technique is very effective in reducing the number of ways accessed during cache accesses. This enables significant energy reduction with negligible performance overheads.

Keywords—cache memory, ETA, reliability, tag bits, transient errors.

I. INTRODUCTION

Multi-level on-chip cache systems have been widely adopted in high performance microprocessors. To keep data consistence throughout the memory hierarchy, write-through and write-back policies are commonly employed. 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. This feature is important as CMOS technology is scaled into the nanometer range, where soft errors have emerged as a major reliability issue in on-chip cache systems. 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. 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 paper, I propose a new cache technique, referred to as early tag access (ETA) cache, to improve the energy efficiency of L1 data caches. In a physical tag and virtual index cache, a part of the
physical address is stored in the tag arrays while the conversion between the virtual address and the physical address is performed by the TLB. By accessing tag arrays and TLB during the LSQ stage, the destination ways of most memory instructions can be determined before accessing the L1 data cache. As a result, only one way in the L1 data cache needs to be accessed for these instructions, thereby reducing the energy consumption significantly. Note that the physical addresses generated from the TLB at the LSQ stage can also be used for subsequent cache accesses. Therefore, for most memory instructions, the energy overhead of way determination at the LSQ stage can be compensated for by skipping the TLB accesses during the cache access stage. For memory instructions whose destination ways cannot be determined at the LSQ stage, an enhanced mode of the ETA cache is proposed to reduce the number of ways accessed at the cache access stage. Note that in many high-end processors, accessing L2 tags is done in parallel with the accesses to the L1 cache. Our technique is fundamentally different as ETAs are performed at the L1 cache.

II. BACKGROUND

Multi-level on-chip cache systems have been widely adopted in high performance microprocessors. To keep data consistency throughout the memory hierarchy, write-through and write-back policies are commonly employed. 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. This feature is important as CMOS technology is scaled into the nanometer range, where soft errors have emerged as a major reliability issue in on-chip cache systems. 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.

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.

I propose new cache architecture, referred to as way-tagged cache, to improve the energy efficiency of write-through cache systems with minimal area overhead and no performance degradation. Consider a two-level cache hierarchy, where the L1 data cache is write-through and the L2 cache is inclusive for high performance. It is observed that all the data residing in the L1 cache will have copies in the L2
cache. In addition, the locations of these copies in the L2 cache will not change until they are evicted from the L2 cache. Thus, we can attach a tag to each way in the L2 cache and send this tag information to the L1 cache when the data is loaded to the L1 cache. By doing so, for all the data in the L1 cache, we will know exactly the locations (i.e., ways) of their copies in the L2 cache. During the subsequent accesses when there is a write hit in the L1 cache (which also initiates a write access to the L2 cache under the write-through policy), we can access the L2 cache in an equivalent direct-mapping manner because the way tag of the data copy in the L2 cache is available. As this operation accounts for the majority of L2 cache accesses in most applications, the energy consumption of L2 cache can be reduced significantly.

A. USAGE:
The basic idea of way-tagged cache was initially proposed in our past work [26] with some preliminary results. In this paper, we extend this work by making the following contributions.

First, a detailed VLSI architecture of the proposed way-tagged cache is developed, where various design issues regarding timing, control logic, operating mechanisms, and area overhead have been studied.

Second, we demonstrate that the idea of way tagging can be extended to many existing low-power cache design techniques so that better tradeoffs of performance and energy efficiency can be achieved.

Third, a detailed energy model is developed to quantify the effectiveness of the proposed technique. Finally, a comprehensive suite of simulations is performed with new results covering the effectiveness of the proposed technique under different cache configurations. It is also shown that the proposed technique can be integrated with existing low-power cache design techniques to further improve energy efficiency.

B. CACHE
1. CPU CACHE
A CPU cache is a cache used by the central processing unit of a computer to reduce the average time to access memory. 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.

1.1 CACHE ENTRIES
Data is transferred between memory and cache in blocks of fixed size, called cache lines. When a cache line is copied from memory into the cache.

A cache entry is created. The cache entry will include the copied data as well as the requested memory location (now called a tag). When the processor needs to read or write a location in main memory, it first checks for a corresponding entry in the cache. The cache checks for the contents of the requested memory location in any cache lines that might contain that address. If the processor finds that the
memory location is in the cache, a cache hit has occurred. However, if the processor does not find the memory location in the cache, a cache miss has occurred.

In the case of:

- A cache hit, the processor immediately reads or writes the data in the cache line
- A cache miss, the cache allocates a new entry, and copies in data from main memory; then, the request is fulfilled from the contents of the cache.

1.2 CACHE PERFORMANCE

The proportion of accesses that result in a cache hit is known as the hit rate, and can be a measure of the effectiveness of the cache for a given program or algorithm.

Read misses delay execution because they require data to be transferred from memory much more slowly than the cache itself. Write misses may occur without such penalty, since the processor can continue execution while data is copied to main memory in the background.

1.3 WRITE POLICIES

If data is written to the cache, at some point it must also be written to main memory. The timing of this write is known as the write policy.

- In a write-through cache, every write to the cache causes a write to main memory.
- Alternatively, in a write-back or copy-back cache, writes are not immediately mirrored to the main memory. Instead, the cache tracks which locations have been written over (these locations are marked dirty). The data in these locations are written back to the main memory only when that data is evicted from the cache.

For this reason, a read miss in a write-back cache may sometimes require two memory accesses to service: one to first write the dirty location to memory and then another to read the new location from memory.

There are intermediate policies as well. The cache may be write-through, but the writes may be held in a store data queue temporarily, usually so that multiple stores can be processed together (which can reduce bus turnarounds and improve bus utilization).

The data in main memory being cached may be changed by other entities (e.g. peripherals using direct memory access or multi-core processor), in which case the copy in the cache may become out-of-date or stale. Alternatively, when the CPU in a multi-core processor updates the data in the cache, copies of data in caches associated with other cores will become stale. Communication protocols between the cache managers which keep the data consistent are known as cache coherence protocols.

1.4 CACHE MISS

A cache miss refers to a failed attempt to read or write a piece of data in the cache, which results in a main memory access with much longer latency. There are three kinds of cache misses: instruction read miss, data read miss, and data write miss.

A cache read miss from an instruction cache generally causes the most delay, because the processor, or at least the thread of execution, has to wait (stall) until the instruction is fetched from main memory.
A cache read miss from a data cache usually causes less delay, because instructions not dependent on the cache read can be issued and continue execution until the data is returned from main memory, and the dependent instructions can resume execution.

A cache write miss to a data cache generally causes the least delay, because the write can be queued and there are few limitations on the execution of subsequent instructions. The processor can continue until the queue is full.

If requested data is contained in the cache (cache hit), this request can be served by simply reading the cache, which is comparatively faster.

Otherwise cache miss the data has to be recomputed or fetched from its original storage location, which is comparatively slower. Hence, the greater the number of requests that can be served from the cache, the faster the overall system performance becomes.

Most modern desktop and server CPUs have at least three independent caches:
- An instruction cache to speed up executable instruction fetch,
- A data cache to speed up data fetch and store, and
- A Translation Look aside Buffer (TLB) used to speed up virtual-to-physical address translation for both executable instructions and data
- The data cache is usually organized as a hierarchy of more cache levels (L1, L2, etc.).

2. TYPES OF CACHE:

2.1 CACHE LEVEL 1
The Level 1 cache, or primary cache, is on the CPU and is used for temporary storage of instructions and data organized in blocks of 32 bytes. Primary cache is the fastest form of storage. Because it’s built into the chip with a zero wait-state (delay) interface to the processor’s execution unit, it is limited in size.

L1 Cache is divided into two parts. Instruction cache and data cache. Instruction cache stores the set of instructions that are required by the CPU for computing. While the data cache stores the values that are required for current execution.

Cache is implemented using Static RAM (SRAM) and until recently was traditionally 16KB in size. SRAM uses two transistors per bit and can hold data without external assistance, for as long as power is supplied to the circuit. The second transistor controls the output of the first: a circuit known as a flip-flop – so-called because it has two stable states which it can flip between. This is contrasted to dynamic RAM (DRAM), which must be refreshed many times per second in order to hold its data contents.

2.2 CACHE LEVEL 2
Level 2 cache, also called secondary cache, is a memory that is used to store recently accessed information. The goal of having the level 2 cache is to reduce data access time in cases when the same data was already accessed before.

In modern microprocessors that incorporate data prefetching feature the level 2 cache may also be used to buffer program instructions and data that the processor is about to request from memory. This also
reduces data access time. Please note that the level 2 cache is secondary to the CPU - it is not as fast as the level 1 cache, although it is usually much larger. All data that is requested from level 2 cache is copied to level 1 cache.

Requested data stays in the secondary cache if it's an inclusive cache, and is removed from secondary cache if it's an exclusive cache. Secondary cache is usually unified, i.e. it is used to store both program instructions and program data.

Level 2 cache is often abbreviated as "L2 cache".

L2 cache may be placed.
- On the processor core - integrated or on-die cache.
- In the same package/cartridge as the processor, but separate from the processor core - backside cache. This type of L2 cache was used in Pentium Pro, Pentium II, early Pentium III and slot A Athlon processors.
- Separate from the core and processor package. In this case L2 cache memory is usually located on the motherboard.

Level 2 caches typically comes in two sizes, 256KB or 512KB, and can be found, or soldered onto the motherboard, in a Card Edge Low Profile (CELP) socket or, more recently, on a COAST ("cache on a stick") module. The latter resembles a SIMM but is a little shorter and plugs into a COAST socket, which is normally located close to the processor and resembles a PCI expansion slot. The Pentium Pro deviated from this arrangement, siting the Level 2 cache on the processor chip itself.

The aim of the Level 2 cache is to supply stored information to the processor without any delay (wait-state). For this purpose, the bus interface of the processor has a special transfer protocol called burst mode. A burst cycle consists of four data transfers where only the address of the first 64 are output on the address bus. The most common Level 2 cache is synchronous pipeline burst.

To have a synchronous cache a chipset, such as Triton, is required to support it. It can provide a 3-5% increase in PC performance because it is timed to a clock cycle. This is achieved by use of specialized SRAM technology which has been developed to allow zero wait-state access for consecutive burst read cycles. Pipelined Burst Static RAM (PB SRAM) has an access time in the range 4.5 to 8 nanoseconds (ns) and allows a transfer timing of 3-1-1-1 for bus speeds up to 133MHz. These numbers refer to the number of clock cycles for each access of a burst mode memory read. For example, 3-1-1-1 refers to three clock cycles for the first word and one cycle for each subsequent word.

III PROPOSED ETA CACHE

In a conventional set-associative cache, all ways in the tag and data arrays are accessed simultaneously. The requested data, however, only resides in one way under a cache hit. The extra way accesses incur unnecessary energy consumption. In this section, a new cache architecture referred to as ETA cache will be developed. The ETA cache reduces the number of unnecessary way accesses, thereby reducing the unnecessary energy consumption. To accommodate different energy and performance requirements in embedded processors, the ETAcache can be operated under two different modes: the basic mode and the advanced mode.
To avoid the data contention with the L1 data cache, the LSQ tag arrays and LSQ TLB are implemented as a copy of the tag arrays and TLB of the L1 data cache, respectively. There are two types of operations in the LSQ tag arrays and LSQ TLB: lookup and update. Each time a memory address reaches the LSQ, the LSQ tag arrays and LSQ TLB will be searched for the early destination way. In case of a hit, the early destination way will be available; otherwise, the instruction will cause either an early tag miss. For update operations, the contents of LSQ tag arrays and LSQ TLB are updated with the tag arrays and TLB of the L1 cache, so that they are identical to avoid cache coherence problems. The update logic of LSQ tag arrays and LSQ TLB is the same as that of the tag arrays and TLB of the L1 cache.

Fig. 2 shows the implementation of the LSQ tag arrays, where only one way is shown as the other ways are the same. Consider that generally at most \( N \) instructions can enter the LSQ while the L1 data cache allows \( M \) replacements to occur at the same time. Therefore, there might be at most \( N \) lookup operations and \( M \) update operations occurring at the LSQ tag arrays and LSQ TLB at the same time. In order to perform these operations simultaneously, the LSQ tag arrays and LSQ TLB have \( N \) read ports and \( M \) write ports. In the simulations in Section V, both \( M \) and \( N \) are chosen to be two for the purpose of demonstration. Write/read conflicts occur when the lookup and update operations target
the same location of the LSQ tag arrays at the same time. To address this issue, we disable the lookup operation if an update operation is currently performed. This is achieved by the control signal lookup-disable, which is generated by the way enabling signals from the cache controller for cache replacements. Consider a two-way set-associative cache for example. Assume that there is a replacement occurring at the way 1 of the L1 data cache. As a result, the way enabling signal is set to “1” and then sent to the NAND gates in the way 1 of the LSQ tag arrays. If the write decoder outputs a “0,” i.e., no update operation on this entry of the tag array, the lookup-disable signal will be set to “1” and the activating circuit will not block the lookup operation on this entry. Otherwise, the lookup-disable signal will be “0,” and the activating circuit will block possible lookup operations to avoid write/read conflicts.

A. Information Buffer.

![Fig. 3. Information Buffer.](image)

The information buffer has separate write and read ports to support parallel write and read operations. The write operations of the information buffer always start one clock cycle later than the corresponding write operations in the LSQ. This is because the accesses to the LSQ, LSQ tag arrays, and LSQ TLB occur simultaneously. Since the way information is available after the write operations in the LSQ, this information will be written into the information buffer one clock cycle later than the corresponding write operation in the LSQ.

B. Way Hit/Miss Decoder

![Fig. 4. Way Hit/Miss Decoder](image)
If a cache coherence problem is detected, an additional access to the L1 data cache is required. Here, we introduce a way hit/miss decoder to determine whether the additional access is necessary. Fig shows the implementation of this decoder with dotted lines. A conventional cache hit/miss decoder is also shown with solid lines. The configuration bit is used to set the ETA cache for the basic mode or the advanced mode. As shown in Fig. 13, if both the cache hit/miss and way hit/miss signals indicate a hit (e.g., “1”), the cache access is considered a hit.

C. Way Decoder

In the proposed ETA cache, way enabling signals are needed to control the access to the ways in the data arrays. Fig. 14 shows the implementation of the way decoder that generates these signals. When the instruction is associated with an early hit (e.g., “1”), the data arrays need to be accessed according to the early destination way. If the instruction experiences an early tag miss or an early TLB miss, the configuration bit shown in Fig. 14 determines which way in the data arrays of the L1 data cache needs to be accessed. Specifically, by setting the configuration bit to “1,” the ETA cache will operate under the basic mode.

IV. RESULTS AND DISCUSSION

A. RTL Schematic
B. Technology Schematic

C. PERFORMANCE EVALUATION RESULTS

<table>
<thead>
<tr>
<th>Performance Parameters</th>
<th>Achieved Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power Consumption</td>
<td>203mW</td>
</tr>
<tr>
<td>Memory Usage</td>
<td>155996KB</td>
</tr>
<tr>
<td>Latency</td>
<td>5.077ns</td>
</tr>
<tr>
<td>Gate Counts</td>
<td>1728</td>
</tr>
</tbody>
</table>

Table 1. performance evaluation results

V. CONCLUSION

A new energy-efficient cache technique for high performance microprocessor employing the write-through policy is implemented in this project. The proposed technique attaches a tag to each way in the L2 cache. The 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. 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. Future work is being directed towards extending this technique to other levels of cache hierarchy and reducing the energy consumption of other cache operations.
REFERENCES