Performance and Power Solutions for Caches Using 8T SRAM Cells

Mostafa Farahani  Amirali Baniasadi
Department of Electrical and Computer Engineering
University of Victoria, BC, Canada
{mostafa, amirali}@ece.uvic.ca

Abstract

Voltage scaling can reduce power dissipation significantly. SRAM cells (which are traditionally implemented using six-transistor cells) can limit voltage scaling due to stability concerns. Eight-transistor (8T) cells were proposed to enhance cell stability under voltage scaling. 8T cells, however, suffer from costly write operations caused by the column selection issue. Previous work has proposed Read-Modify-Write (RMW) to address this issue at the expense of an increase in cache access frequency.

In this work, we introduce two microarchitectural solutions to address this inefficiency. Our solutions rely on grouping write accesses and bypassing read accesses made to the same cache set. We reduce cache access frequency up to 55%.

1. Introduction

Power dissipation is a major concern in designing mobile and high performance computing devices. Voltage scaling is commonly used to reduce power dissipation in high-performance processors. On the negative side, reducing voltage can result in an increase in transistor switching latency which in turn requires a longer time to operate correctly.

Dynamic Voltage and Frequency Scaling (DVFS) has been introduced to scale voltage and frequency adaptively to meet power and performance requirements. DVFS is able to deliver the required performance while minimizing power dissipation. DVFS switches between predefined voltage levels dynamically according to the required performance and power demand. The more the number of voltage levels the higher the chances of operating at the optimal voltage and frequency level. Among the different levels, the minimum voltage level (V_{min}) assuring correct operation limits the lowest operating voltage.

One of the system components likely to serve as the bottleneck in deciding V_{min} is the cache, which is traditionally implemented using six-transistor cells (referred to as 6T cells) [10]. 6T cells can suffer from poor stability under technology scaling. Stability is also aggravated more by voltage scaling (often required for near threshold computing) [12]. To overcome the stability issue, Chang et al. [1] introduced the 8T cell.

Figure 1 shows an 8T cell. An 8T cell consists of a 6T cell at its core and two extra transistors. These two extra transistors improve stability by enhancing read stability and write ability simultaneously [2]. Stability improvement allows scaling voltage even below threshold [12]. In addition to stability enhancement, 8T cells can improve performance by separating read and write ports. Using separate ports allows issuing one read and one write request simultaneously.

Although 8T cells improve cell stability and enable scaling down supply voltage, avoiding soft errors is still important especially in lower voltage levels. Hence, bit interleaving is commonly used to spread out bits belonging to one word across one SRAM array row and prevent multi-bit upsets in one word [4]. However, interleaved SRAM array induces column selection issue. Column selection refers to selecting the desired bits in an SRAM array row without affecting unselected columns. This is due to the fact that all columns in one SRAM array row share word lines. Therefore, by asserting a word line all cells in that row are selected. This issue is important during write operations as only the selected columns are intended to be written while unselected columns have to be unaffected.

Morita et al. [8] studied the column selection issue in SRAM arrays using 8T cells. They proposed the read-modify-write (also known as write-back) scheme to solve this issue. In Read-Modify-Write (RMW), the addressed row in an SRAM array has to be read and buffered before being written. This imposes an extra read for every write operation.

The extra read per each write operation by RMW has negative impact on cache traffic, performance, and power dissipation. Our simulation results show that RMW increases cache access frequency by more than 32% on average (max 47%). Furthermore, RMW occupies the read port (in addition to a write port) for each write operation and makes servicing one read and one write operation simultaneously impossible. Moreover, RMW is power inefficient as it requires an extra read operation per each write.

In this paper, we propose two performance and power efficient solutions, Write Grouping (WG) and Write Grouping and Read By-passing (WG+RB). WG reduces cache access frequency by grouping write requests made to the same cache set during short intervals. Once this group is formed, WG writes the entire group performing a single cache access. WG stores the group in a buffer referred to as the Set-Buffer. WG+RB further enhances WG by bypassing a subset of read accesses made to the data already stored in the Set-Buffer.

In summary, we make the following contributions:

1. We show that cache traffic increases significantly by exploiting RMW. We also show that a significant share of RMW operations are unnecessary. Further, we analyze the sequence of cache accesses and show that a considerable share of RMW operations could be reduced by grouping them.

2. We propose WG as a novel microarchitecture solution. WG reduces the frequency of RMW write operations. In addition, WG...
elaborates RMW reads associated with the eliminated writes.

3. We improve WG’s effectiveness in reducing cache access frequency further by proposing WG+RB. WG+RB, in addition to reducing RMW operations frequency, eliminates a considerable share of read accesses.

The rest of our paper is organized as follow. In Section 2, we briefly overview background and related work. We present motivation in Section 3. We explain our proposed techniques in Section 4. Section 5 presents our simulation results. Finally, in Section 6 we offer concluding remarks.

2. Background and Related Work

We show an 8T cell in Figure 1. A cross-coupled inverter is formed by M1 through M4 transistors to keep data. M5 and M6 are write access transistors. By rising Write Word Line (WWL), the value on write bit lines (WBL and WBLB) is written to the cell. For read operations, Read Bit Line (RBL) is precharged initially. Then, Read Word Line (RWL) rises to turn on M8. If the cell holds zero (i.e., Q=0), M7 turns on and RBL discharges through M7 and M8. On the contrary, if the cell value is one (i.e., Q=1), M7 is off and RBL keeps the charge. Although the associated overhead (extra transistors and corresponding word line and bit line) could increase area, 8T cells are more compact in technology nodes beyond 45nm compared to 6T cells [8].

Figure 2 shows one SRAM array row implemented using 8T cells. In this array, word lines (WWL and RWL) are shared among cells in one row and bit lines (WBLs, WBLBs, and RBLs) are shared among cells in one column. Note that WWL and RWL are driven independently.

In order to optimize word and bit lines’ latency, power, and area, SRAM arrays are broken vertically and horizontally into interleaved sub-arrays [9, 2]. In addition, bit-interleaving is used to reduce the probability of upsetting two bits in one word making using simple and low cost one bit correction techniques possible [4]. Accordingly, two adjacent cells in one row belong to two different words. Hence, an activated row (by WWL or RWL) selects all cells in one row. However, only a subset of cells are intended to be selected. Column selection refers to this issue. The cells that are not intended to be selected are called half-selected.

Column selection issue is solved by using multiplexing in read operations. It is worth to mention that before each read operation, RBLs are precharged by precharge circuits. After the precharge phase, the read word line driver rises one of the RWLs to initiate read. All cells in the selected row perform a read operation as explained earlier. However, only the selected cells are routed to the output by multiplexers (not shown in Figure 2) at the end of RBLs. Note how multiplexers ignore half-selected cells.

Despite using multiplexing, column selection issue still arises for write operations. In 6T cells, half-selected cells are biased for read operations [11] while selected cells are written. In 8T cells, however, cells are optimized for write operations and biasing cells for read operations can corrupt data [11]. Morita et al. [8] proposed RMW as an efficient write mechanism in 8T cells. In RMW, a read operation is required before each write. In Figure 2 we show the sequence of steps needed in RMW. The block diagram at the bottom of figure shows write-back and write driver circuits that are involved in writes.

The sequence of steps for RMW is as follows:
1. Precharge circuit charges RBLs.
2. Read word line driver rises RWL to initiate a read operation. In this phase of RMW, multiplexers do not route data to the output.
3. After performing read, latches at the bottom of columns store the data.
4. The multiplexer controlled by Write-back signal loads write drivers. Write drivers in selected columns are loaded by Data-in path while write drivers in half-selected columns are loaded by latches. Then, Write Drivers drive write bit lines (WBLs and WBLBs) with new values.
5. Write is finalized by rising WWL and writing the value on bit lines to the cell.

In summary, RMW operations consist of reading row, partially modifying row and finally writing back to SRAM arrays. Since a read before each write is required in RMW, the read port is occupied and servicing one read and one write in parallel is not possible [11]. In addition to performance loss, RMW dissipates higher power due to extra reads.

Previous studies offered solutions to address the column selection issue. Park et al. [11] exploited the hierarchical structure of RBLs to perform RMW locally. They isolated the sub-array performing write-back from RBLs to service another request simultaneously. However, in this technique the sub-array performing write-back is not available to any other cache access. Kim et al. [5] proposed adaptive WWL pulse width and voltage modulation to address dynamic write failure. They modulated WWL pulse width and voltage level to ensure that all cells are written.

Chang et al. [2] proposed to address the cells in one row at word granularity in non-interleaved SRAM arrays. Therefore, they were able to select only the desired bits in one row. This technique requires multi-bit correction techniques and larger write word line drivers, which could increase area and power dissipation.

3. Motivation

Memory instructions (reads and writes) contribute to a significant share of executed instructions. In Figure 3 we show the frequency of read and write accesses to the L1 data cache (methodology detail in Section 5.1). As presented, on average 40% of executed instructions
are memory requests (26% reads and 14% writes). Write frequency increases to more than 22% for write-intensive applications (e.g., bwaves). As explained earlier, write requests are costly in SRAM arrays using 8T cells.

In this work, we propose two solutions to reduce the frequency of RMW operations. Our proposed techniques build on two observations:

1. **Set Access Locality**: Our study shows that a considerable share of cache accesses (on average 27%) are made to the same cache set. Since there are two types of cache accesses (i.e., read and write) there are four possible scenarios. These scenarios include Read-Read (RR), Write-Write (WW), Read-Write (RW), and Write-Read (WR). RR refers to the scenario where both consecutive accesses to the same cache set are reads. Similarly in WW, both cache accesses are writes. Two other scenarios are RW and WR, which refer to read then write and write then read, respectively. In Figure 4 we show how often these scenarios occur. As presented, RR and WW account for the largest share of consecutive accesses in almost all benchmarks.

2. **High Silent Write Frequency**: “Silent store instructions write values that exactly match the values that are already stored at the memory address that is being written” [6]. Since silent writes do not change the memory state, avoiding executing them has no impact on program correctness execution. In Figure 5 we show the frequency of silent writes in SEPC 2006 benchmarks. As presented, on average more than 42% of writes are silent.

We explain our proposed techniques in the following section in more detail.

4. **Solutions**

4.1. **Write Grouping (WG)**

We propose WG to group consecutive writes to the same cache set (i.e., WW) and perform one RMW operation for them. Figure 6a shows WG’s block diagram. In this technique, we add a buffer referred to as the Set-Buffer between the multiplexer and the write driver. The Set-Buffer stores the last accessed cache set’s data due to write requests. To keep track of addresses, we also store cache set number and all cache block tags in a second buffer referred to as the Tag-Buffer in the cache controller as shown in Figure 6b. The cache controller probes the Tag-Buffer for each cache request.

For each write request, after reading and modifying a set, we do not write-back immediately. The Set-Buffer keeps data at least till the next cache request. If the next cache request is a write and misses in the Tag-Buffer, we evict the previously stored data from the Set-Buffer and write the stored data to the cache. We move the new set to the Set-Buffer and update the Tag-Buffer accordingly. Under this scenario, we do not reduce cache access frequency. However, if the new write access is made to the same set already stored in the Set-Buffer, we update the Set-Buffer without performing write-back. Note that, in this case we eliminate two accesses, one read for reading row and one write-back.

On the other hand, read operations have to read the most recently written value. However, WG does not update cache immediately. In order to guaranty correct execution and retrieving the most recently written value, we check if the read operation’s address is already stored in the Tag-Buffer. If so, we write-back the Set-Buffer (prematurely) to update the cache before reading data. This is unnecessary if the read request’s address does not exist in the Tag-Buffer.
In addition to grouping consecutive writes to reduce RMW operations frequency, WG further detects silent writes in order to eliminate unnecessary RMW operations. WG detects silent writes after updating the Set-Buffer by comparing the prior value and the updated value. We keep track of silent writes by using the Dirty bit shown in Figure 6b. The Dirty bit is set by the cache controller upon encountering non-silent writes to indicate that the Set-Buffer has been modified and has to be written back to cache. When the cache controller initiates the write-back, it first checks the Dirty bit and eliminates the write-back if the Dirty bit is zero. The cache controller also clears the Dirty bit after write-back to indicate that cache is updated. Algorithm 1 summarizes WG's steps.

Besides RMW operation frequency reduction, WG increases read port availability. In conventional RMW, the read port is busy for reading row and filling latches. However in WG, we reduce the frequency of write-backs and consequently reading rows. The higher reduction in write-back frequency, the higher read port availability.

Read operations can degrade WG’s effectiveness as they can force premature RMW upon a Tag-Buffer hit. We propose WG+RB, to address this limitation in Section 4.2.

4.2. Write Grouping + Read Bypassing (WG+RB)

Figure 7 shows WG+RB’s block diagram. We add a multiplexer to route the output data from two sources, the Set-Buffer or the RBL. This multiplexer is controlled by the Bypass signal.

In WG+RB, if the current read request hits in the Tag-Buffer, a write-back operation is not required to update the cache. Instead, the Bypass signal routes data from the Set-Buffer to the output. Thus, WG+RB reduces not only premature write-back frequency, it also eliminates read operations that hit in the Tag-Buffer. WG+RB can improve both performance and power. Performance is expected to improve as access latency to the Set-Buffer is less than the cache latency. In addition, read port availability increases improving performance further. Power is reduced as we eliminate activities in RWLs and RBLs in SRAM arrays.

As we show in Figure 4, the RR share is considerable in all benchmarks. WG+RB takes advantage of this behavior to reduce the cache access frequency. It is worth to mention that WG+RB also eliminates WR accesses.

4.3. Example

To provide better understanding, we present an example in Figure 8. To simplify the example, we consider only two cache sets, set a
and set b. For example, R_{a} and W_{b} indicate reading from set a and writing to set b, respectively. In this figure, the first row shows the stream of requests issued by a processor. Requests arrive at the L1 data cache from right to left (the cache receives a read request from set a, then a write request to set b, and so on).

The second row shows the cache access stream under RWM, where each write request is preceded by a read request. In the third row, we show how WG groups write operations and eliminates unnecessary cache accesses. In this example, we assume that the last write request (W_{b}) is silent. Initially, the Tag-Buffer is empty. Hence, the first read request (R_{a}) misses in the Tag-Buffer and cache is accessed. The next cache access is W_{b} and therefore the cache controller issues a read (the first R_{b}). The Set-Buffer in the selected columns are filled with data from Data-in and the half selected columns are filled with data read from set b. Unlike RMW, WG does not write back immediately.

The next request (the second W_{b}) hits in the Tag-Buffer and the Set-Buffer in the selected columns is updated. In addition, WG compares the previous value against the new value to detect silent writes. In this case, the Dirty bit is set as W_{b} is not silent. The following requests are read from set b. Hence, the cache controller issues a write back to update cache and clears the Dirty bit. Thereafter, the two read operations are performed. The next request (the third W_{b}) hits in the Tag-Buffer and updates the Set-Buffer. The cache controller sets the Dirty bit to indicate that the Set-Buffer is modified.

However, the next request (the first W_{a}) misses in the Tag-Buffer and the cache controller updates the cache by issuing another write back operation and a read operation to fill the Set-Buffer with set a’s data. Since this write operation is silent, the Dirty bit is not set by the cache controller. Upon the next request (the last R_{b}), a Tag-Buffer miss occurs and since this is a read operation a cache access is performed. Although the last request is read and hits in the Tag-Buffer, the cache controller does not initiate a write back to update cache since the Dirty bit is clear (the cache and the Set-Buffer are consistent).

We show how WG+RB deals with the same request stream in the forth row in Figure 8. After reading set b and filling the Set-Buffer due to the first W_{b} all subsequent requests to set b are serviced without accessing the cache. The write back to the cache (W_{b}) happens before W_{a}. Note that the last request (R_{b}) is eliminated as it hits in the Tag-Buffer and is bypassed by WG+RB.

5. Evaluation

5.1. Methodology

We use Pin [7] and develop a Pin tool to simulate an L1 data cache. We consider a baseline cache with 64KB cache size, 4-way, 32B block size, and LRU replacement policy. We also run 25 out of 29 SPEC 2006 benchmarks [3] for 10 billions instructions (1 billion fast forward to warm-up cache). Since Pin simulations are not repeatable, we run all evaluations and techniques in one run.

5.2. Cache Access

As presented, our proposed techniques reduce the frequency of cache access due to RMW operations. We present cache access frequency reduction in Figure 9. In this figure, the left bar shows the access reduction achieved by WG. The right bar shows the achieved access reduction by WG+RB. On average cache access frequency is reduced by 27% and 33% by employing WG and WG+RB, respectively. In addition, WG+RB outperforms WG in all benchmarks.

We achieve a significant cache access frequency reduction (47%) in bwaves by employing WG. This could be explained by referring to Figures 4 and 5. As presented, the WW share is highest (24%) for bwaves. Hence, a large share of writes are consecutive in this benchmark. In addition, silent write frequency is high (77%) in bwaves. Similar conclusions can be made for wrf and lbm.

Our simulation results show that WG+RB improves WG’s effectiveness in all benchmarks. However, some benchmarks (e.g., games and cactusADM) benefit more as their RR share is higher compared to others.

5.3. Sensitivity Analysis

In this section, we report how variations in block size and cache size impact cache access reduction in WG and WG+RB. In Figure 10 we report cache access reduction for a 32KB cache with 64B block size. As presented cache access frequency is reduced by 29% and 37% for WG and WG+RB, respectively. This is due to the fact that, by increasing block size the Set-Buffer hit rate improves resulting in grouping a larger number of write operations. Similarly, more read operations can be bypassed by WG+RB. In Figure 11 we present how
variations in cache size impact WG and WG+RB. We report cache access reduction for 32KB and 128KB caches. As presented cache access reduction is 26.9% and 26.6% for 32KB and 128KB caches for WG, respectively. Cache access reduction is 32.6% and 32.1% for 32KB and 128KB caches for WG+RB, respectively.

5.4. Area Overhead

WG and WG+RB impose the extra area overhead associated with the Set-Buffer and the Tag-Buffer. Set-Buffer size is equal to the size of one cache set. For example, in our baseline cache (64KB, 4-way, and 32B block size), the size of a cache set is 128B. Accordingly, the Set-Buffer imposes less than 0.2% area overhead compared to the overall cache size. In this case, the Tag-Buffer size is negligible (less than 150 bits assuming 48 bits physical address). In addition to storage overhead, WG and WG+RB require comparators to detect silent writes. WG+RB needs extra multiplexers to route data from the Set-Buffer to the output.

5.5. Performance and Power

Evaluating performance and power dissipation under our solutions is part of our ongoing research. However, in this section we comment on both issues. Both WG and WG+RB can impact performance. WG can potentially increases write latency. Write operations, however, are not on the processor critical path. Therefore, we expect WG’s performance cost to be negligible. WG+RB reduces average cache read latency as it replaces a subset of cache accesses with accessing a smaller buffer. Since read operations are on the processor’s critical path we expect WG+RB to come with performance improvements. As for power measurements, we expect both our solutions to reduce the overall power dissipation as they replace power hungry cache accesses with accessing a smaller and hence more power efficient structure.

6. Conclusion

6T cells stability limits the potential power saving achievable by voltage scaling. While proposed 8T cells enhance stability, they suffer from column selection issue during write operations. RMW addresses this issue at the cost of an increase in the number of cache accesses resulting in higher power dissipation and performance loss.

In this work, we propose two microarchitectural solutions, WG and WG+RB. WG reduces RMW operation’s frequency. It groups consecutive write operations made to one cache set during short intervals. WG performs one RMW operation for the entire group. We further enhance WG’s effectiveness in cache access frequency reduction by bypassing a subset of read operations in WG+RB. Our evaluations show that WG and WG+RB reduce cache access frequency by 27% and 33%, respectively.

References

Figure 11: Cache access frequency reduction for different cache sizes


