

Portland State University

PDXScholar

Dissertations and Theses

Dissertations and Theses

1-1-2011

Spare Block Cache Architecture to Enable Low-Voltage Operation

Nafiul Alam Siddique
Portland State University

Follow this and additional works at: https://pdxscholar.library.pdx.edu/open_access_etds

Let us know how access to this document benefits you.

Recommended Citation

Siddique, Nafiul Alam, "Spare Block Cache Architecture to Enable Low-Voltage Operation" (2011).
Dissertations and Theses. Paper 216.
<https://doi.org/10.15760/etd.216>

This Thesis is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: pdxscholar@pdx.edu.

Spare Block Cache Architecture to Enable Low-Voltage Operation

by

Nafiul Alam Siddique

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science
in
Electrical and Computer Engineering

Thesis Committee:
Douglas V. Hall, Chair
Alaa Alameldeen
Mark Faust

Portland State University
©2011

ABSTRACT

Power consumption is a major concern for modern processors. Voltage scaling is one of the most effective mechanisms to reduce power consumption. However, voltage scaling is limited by large memory structures, such as caches, where many cells can fail at low voltage operation. As a result, voltage scaling is limited by a minimum voltage (V_{ccmin}), below which the processor may not operate reliably. Researchers have proposed architectural mechanisms, error detection and correction techniques, and circuit solutions to allow the cache to operate reliably at low voltages. Architectural solutions reduce cache capacity at low voltages at the expense of logic complexity. Circuit solutions change the SRAM cell organization and have the disadvantage of reducing the cache capacity (for the same area) even when the system runs at a high voltage. Error detection and correction mechanisms use Error Correction Codes (ECC) codes to keep the cache operation reliable at low voltage, but have the disadvantage of increasing cache access time. In this thesis, we propose a novel architectural technique that uses spare cache blocks to back up a set-associative cache at low voltage. In our mechanism, we perform memory tests at low voltage to detect errors in all cache lines and tag them as faulty or fault-free. We have designed shifter and adder circuits for our architecture, and evaluated our design using the SimpleScalar simulator. We constructed a fault model for our design to find the cache set failure probability at low voltage. Our evaluation shows that, at 485mV, our designed cache operates with an equivalent bit failure probability to a

conventional cache operating at 782mV. We have compared instructions per cycle (IPC), miss rates, and cache accesses of our design with a conventional cache operating at nominal voltage. We have also compared our cache performance with a cache using the previously proposed Bit-Fix mechanism. Our result show that our designed spare cache mechanism is 15% more area efficient compared to Bit-Fix. Our proposed approach provides a significant improvement in power and EPI (energy per instruction) over a conventional cache and Bit-Fix, at the expense of having lower performance at high voltage.

DEDICATION

To my parents and my brother Nahian

ACKNOWLEDGMENTS

I would like to thank my thesis advisors, Dr. Alaa Alameldeen and Prof. Douglas V. Hall for helping to make this project successful. I would also like to thank Prof. Mark Faust for his invaluable service as a member of my advisory committee. I am grateful to have had all three of these professors as teachers, advisors, mentors, and friends.

TABLE OF CONTENTS

ABSTRACT	i
DEDICATION	iii
ACKNOWLEDGMENTS	iv
LIST OF TABLES	vii
LIST OF FIGURES	viii
1. INTRODUCTION	1
1.1 Conventional 6T SRAM Cell	4
1.2 Memory Architecture	5
1.3 Set-associative Caches and Critical Path	6
1.4 Domino logic	8
1.5 Cache Failure Probability Models	10
1.6 Thesis Organization	11
2. RELATED WORK	12
2.1 Architectural Techniques	12
2.2 Circuit-level Solutions	16
2.3 Coding Techniques	17
3. SPARE BLOCK CACHE ARCHITECTURE	22
3.1. Cache Failure Probability Model	26
3.2. Cache Controller Architecture	29
3.2.1 Serial Design for Cache Controller	31
3.2.2 Controller Implementation	33
3.2.3 SRAM cells with control gates	45
3.2.4 Critical Path for Cache Access	47
4. OVERHEAD AND FAULT COVERAGE	52

4.1 Hardware overhead	52
4.2 Cache Failure Probability Model	53
5. PERFORMANCE AND POWER EVALUATION	59
5.1 Simulation Methodology	59
5.2 Performance	61
6. CONCLUSION	68
REFERENCES	71

LIST OF TABLES

Table 3.1: Truth table for 3 rd level Shifter Circuit	41
Table 4.1: Transistors and Overhead of proposed technique in L1 and L2 cache	52
Table 4.2: Cache set failure probability for the spare block cache architecture	54
Table 5.1: Baseline processor configuration	60
Table 5.2: Conventional cache, Bit-Fix mechanism, and our proposed architecture in L1 and L2 cache	66

LIST OF FIGURES

Fig 1.1: 6T SRAM Cell.....	3
Fig 1.2: An n-way set-associative cache.....	7
Fig 1.3: Critical Path of a set associative cache.....	8
Fig 1.4: Domino gates.....	9
Fig 3.1: A 4-way set associative cache	22
Fig 3.2: Faulty words and disable bits	23
Fig 3.3: word shifting.....	24
Fig 3.4: Four consecutive words with faulty word w_1	25
Fig 3.5: Proposed cache architecture with spare blocks	26
Fig 3.6: Cache set.....	26
Fig 3.7: Cache controller and pMOS switches between words and data word line.....	29
Fig 3.8: Serial Access for Cache Controller	32
Fig 3.9: Cache access controller with three types of shifters.....	34
Fig 3.10 Shifter design in cache controller with 64B lines and 16 spare words.....	37
Fig 3.11: Three bit 3 rd level shifter circuits.....	40
Fig 3.12: Small spare block cache architecture with 3 rd level shifters.....	42
Fig 3.13: SRAM cell access control across multiple cache sets	44
Fig 3.14: Critical path of Spare block cache.....	46
Fig 3.15: Critical path of the cache controller	49
Fig 4.1: Voltage vs. Cache Set Failure Probability.....	55
Fig 4.2: 32 KB cache failure probabilities at different voltages	56
Fig 4.3: 2 MB cache failure probabilities at different voltages	57
Fig 5.1: Normalized IPC comparison in different benchmarks	62

Fig 5.2: Normalized L1 data cache miss rate in different benchmarks	63
Fig 5.3: Normalized L2 unified cache miss rate in different benchmarks	63
Fig 5.4: Normalized L1 data cache access in different benchmark	64
Fig 5.5: Normalized L2 unified cache access in different benchmark	65

1. INTRODUCTION

Reducing supply voltage is the most effective method to reduce power consumption in modern processors. However, manufacturing-induced parameter variations cause failures of many memory cells at lower voltages. A minimum value of supply voltage, V_{ccmin} , is needed for reliable operation [1]. Caches and large memory structures constitute a significant fraction of die area and, therefore, are the largest inhibitors of V_{ccmin} scaling. The memory hierarchy of a processor contains different levels of data and instruction caches. For each of these caches, the bit with the highest operating voltage determines the V_{ccmin} of that cache, and the highest V_{ccmin} of all caches determines V_{ccmin} of the whole processor. As the defective cells are distributed randomly throughout the die, it is likely that the largest cache would determine the V_{ccmin} of the processor as a whole [1].

The scaling of transistor dimensions in each technology generation increases transistor density and improves device performance. This geometric shrinkage allows an increasing number of transistors in each new CMOS process generation, which leads to higher activity and power density per unit chip area. Therefore, the electric field density per unit area increases [2]. Thus, if the operating voltage is not scaled down, the performance of the smaller transistors would degrade faster. Therefore, the operating voltage must be decreased to keep the power demand and electric field density within reasonable limits, and must not exceed the maximum degradation level during product

lifetime. In the L2 cache of the Intel Core 2 Duo processor for the 130nm process, V_{ccmin} is 1.2V, and, for the 65nm process the V_{ccmin} decreases to 825mv [1]. A few factors, such as sub-wavelength lithography, line edge roughness, and random doping fluctuations result in a wide distribution of transistor characteristics, which is the main cause of bit failures at lower voltages [3].

Though lowering supply voltage helps to reduce dynamic power consumption and increase lifetime reliability, the consequent decrease in the threshold voltage increases the leakage power. In the 90 nm process, leakage represents 21% of the total power, but below 45 nm, process leakage power increases to about 50% [4]. A microprocessor is composed of billions of transistors, and more than 70% of all transistors are devoted to cache in some designs [5]. It has been estimated that total cache leakage energy is 30% of L1 cache energy and 70% of L2 cache energy in the 130nm process [4]. Thus, half of the total power consumption of memory cells is wasted as leakage power. Moreover, bit cell storage capacitance decreases with geometric scaling. Voltage scaling further reduces the stored charge. Lower operating voltages cause an increasing level of noise and instability of SRAM bit cells. It may also result in flipping of their contents and results in bit cell operation failures (e.g., read failures, hold failures, access time failures, and write failures) [6].

In addition, processor operation at low voltages is susceptible to soft errors. Soft errors occur when an alpha particle or a cosmic ray strikes a memory node and causes data loss. Soft Error Rate (SER) increases in the sub-threshold voltage region. The combination of growing cache capacity, shrinking SRAM cell dimensions, low operating

voltages, and increasing fabrication variability leads to a higher soft error rate (SER) [7, 8].

Many architectural, circuit, and device solutions have been proposed to mitigate the impact of cache cell failures at low voltages. Most improvements of cache reliability are achieved at the expense of reducing cache capacity. While circuit solutions decrease cache capacity in both high and low voltage mode, architectural solutions sacrifice cache capacity only at low voltages. Our mechanism builds architectural solutions and attempts to decrease the capacity reduction at low voltages.

In the following sections, we introduce some of the design concepts we use for our method, and present an outline for the remainder of this thesis.

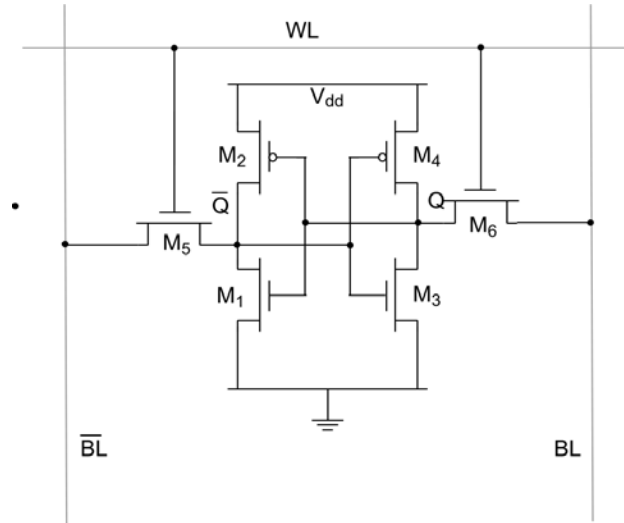


Fig 1.1: 6T SRAM Cell

1.1 Conventional 6T SRAM Cell

An SRAM cell consists of two identical CMOS inverters connected in a positive loop and two NMOS access transistors. The access writing transistors, M5 and M6, control reads and writes in memory nodes Q and Q-bar by WL (Figure 1.1). Initially, bit lines (BL and its complement) are pre-charged to read cell data. Then the access transistor is turned on by WL to allow a differential voltage (50mV-100mV) between the bit lines. A differential voltage amplifier is used to measure the cell value. During a write operation, bit lines are first charged to the desired value, and then access transistors give access to the cell to store that value.

Reducing voltage can cause many types of bit cell failures. Bit cells can fail in the following four ways:

1. Read failure: A read failure occurs when the stored value flips during a read operation. In a SRAM cell either node Q or Q-bar stores '0'. When a noise in the stored node, sometimes logic '0' becomes high enough to trip the inverter, and the data flips. At low voltage, a cell's noise margin decreases, and the difference between the trip voltage and logic '0' also decreases. Thus the probability of a read failure increases at low voltages.

2. Hold failure: This failure occurs when the stored value in the cell is lost during standby. A sudden voltage drop is the main cause of this failure [1].

3. Access failure: An access failure occurs when the differential voltage across the bit line is not sufficient for the sense amplifier to identify the correct value. It usually

occurs during a read operation. Increasing the pulse width of a word line reduces this failure.

4. Write failure: A write failure occurs when the cell cannot overwrite the existing value of the node. Geometric reduction of transistor size and low operating voltage are the main causes of write failures [1].

1.2 Memory Architecture

In a modern microprocessor, there are many memory components that contain program instructions and data. There are three kinds of physical memory: registers, caches, and main memory. *Registers* are the temporary memory that store data to be used in subsequent computations. Each register usually holds one word. There are some specialized registers that hold specific types of data, such as floating point numbers, addresses, etc. The registers are much faster compared to caches and main memory. However, the fast register memory is very expensive. Therefore, the number of registers available is quite small compared to cache and main memory sizes. The *cache* consists of a small, fast memory that acts as a buffer for main memory. A cache can be designed using six transistors SRAM or 6T SRAM. There are different levels of cache memory depending on size and access time. A higher level cache is bigger and slower compared to a lower level cache. The lowest level cache (L1) is typically split into two parts: the data cache (DL1), and the instruction cache (IL1). The *main memory* contains both instructions and data. It is typically built using DRAM cells [2].

Processor performance depends on the performance of its memory system. The latency of fetching, executing, and storing of instructions and data from the memory to the processor determines how fast the processor can execute a program. As all memory operations (i.e., loads and stores) involve cache accesses, it is very critical to improve cache performance.

1.3 Set-associative Caches and Critical Path

An n -way set-associative cache memory is divided into sets; where each set consist of ' n ' cache lines. A block from main memory is first mapped onto a specific cache set, and then it can be placed anywhere within that set. Address bits are divided into tag, index, and offset bits. When the data (instruction) address is available, the index bits are used to activate the appropriate set that should contain the cache line. The tag array is also divided into separate way banks to hold the tag information for the cache blocks in corresponding cache lines of the data array. All tag arrays are probed in parallel to produce inputs to the n tag comparators to compare these stored tags to the tag bits of the address. The results of these tag comparisons are used to generate the select lines for the output way multiplexer on the data side. Once these select lines are available, the output way multiplexer will output the correct cache block onto the output data bus. If the tag doesn't match, then the CPU has to bring the data in from memory.

The main memory data can go into any of the ' n ' cache lines during a cache fill, and the controller picks one of the lines to store the main memory data. The controller has a cache replacement policy to select a victim cache line. For example, the least

recently used (LRU) policy selects the line that has been accessed earlier than the other lines in the set.

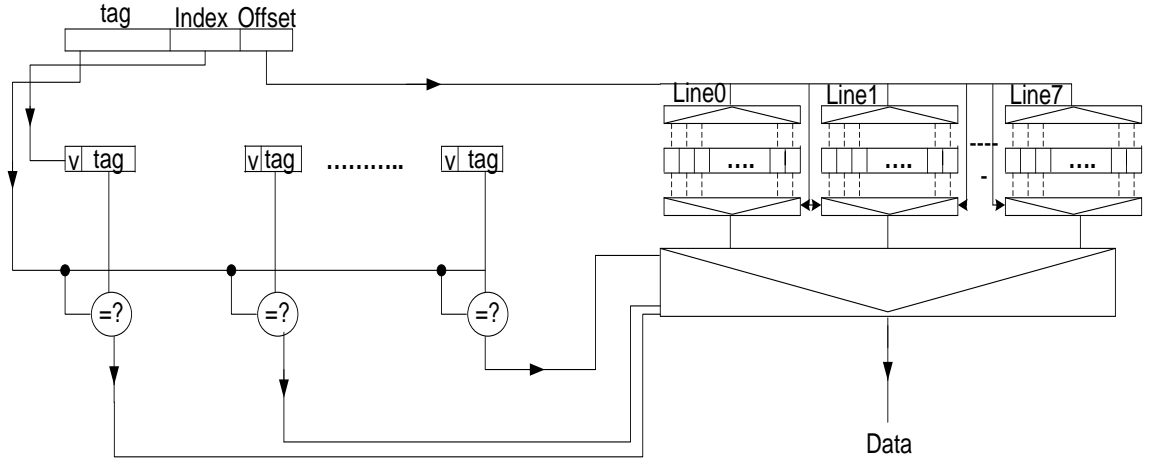


Fig 1.2: An n-way set-associative cache

In a set associative cache, the path of the tag comparison logic is the critical path of the overall cache access. The critical path delay for the whole structure of a conventional set-associative cache is the cumulative delay time to decode the index ($T_{\text{tag_index_decode}}$), read the tag array ($T_{\text{tag_array_w/b_line}}$), compare the tag ($T_{\text{tag_comparison}}$), access to the MUX driver ($T_{\text{MUX_driver}}$), and access the output driver ($T_{\text{output_driver}}$). Thus, the critical path includes selecting lines from the tag comparators to the output multiplexor, switching the multiplexers and providing the results to the data bus. The delay time can be expressed as,

$$T_{\text{cache_access}} = T_{\text{tag_index_decode}} + T_{\text{tag_array_w/b_line}} + T_{\text{tag_comparison}} + T_{\text{MUX_driver}} + T_{\text{output_driver}}$$

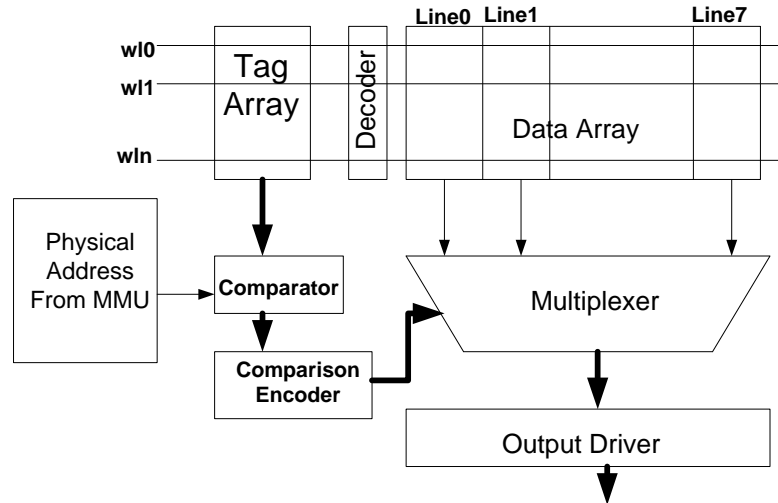


Fig 1.3: Critical Path of a set associative cache

1.4 Domino logic

In our proposal, we used shifters to correct bit failures that are based on domino logic. We introduce some of the domino logic concepts in this section. Most combinational gates have been designed using static CMOS gates. Advantages of using CMOS gates are rail-to-rail switching and simple sizing. However, an N-input CMOS gate requires 2N gates (N pMOS and N nMOS). The width of the pMOS is large due to the slow response of hole carriers. A Dynamic gate is an alternative to a static gate that reduces the number of pMOS transistors. In a static ‘OR’ gate design, a NOR gate is connected in series with an inverter. Here, in a NOR gate pMOS transistors are connected in series which results in a large pMOS width.

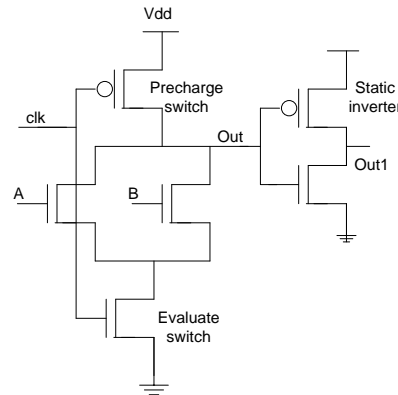


Fig 1.4: Domino gates

Figure 1.4 presents an OR domino gate. The large-width pMOS problem can be solved by domino logic. Domino logic is a CMOS-based evolution of the dynamic logic technique, which is based on either pMOS or nMOS. Domino logic is driven by clocked logic, where every single logic gate has a clock signal. When the clock signal turns low, output node “Out” (i.e., the dynamic node in Figure 1.4) turns high, causing the output of the gate (Out1) to turn low. This is the only path for the gate output to turn low. The operating period of the cell is when its input clock is low. This period is called the *precharge phase*. The next phase, when the clock is high, is called the *evaluate phase*. The evaluate phase is the functional operating phase in domino cells. Since the domino cell only switches from logic low to logic high, there is no need for the inputs A and B to drive any pull-up pMOS transistors. The lack of pMOS transistors eventually reduces the effective transistor width. Therefore, for a particular current drive, domino logic outperforms static logic. The effect of using domino gates in logic gates with a larger

number of inputs is significant. There are also advantages of domino logic over dynamic logic, e.g., cascading and rail-to-rail logic switching.

1.5 Cache Failure Probability Models

At low-voltages, many SRAM cells fail to operate reliably due to process variation. The failing cells are randomly distributed throughout a memory array. Each cell has a probability of failure ($P_{fail}(b)$). A die containing even a single cell failure must be discarded. In our analysis, we assume that the $P_{fail}(b)$ for each memory array must be kept at less than 1 out of 1000 for reliable operation [1]. Achieving reliable operation for a cache requires V_{ccmin} of nearly 782mV. One possible mechanism to decrease cache V_{ccmin} is to selectively disable defective data in the cache. Such disabling can be carried out at different granularities, ranging from cache ways, entire cache lines (coarse), to individual bits (fine). As the number of cells in the cache is very large, it is not practical to correct and isolate the faulty bit at the granularity of a bit [7]. At low voltage at least one bit may fail to operate properly in many lines [1, 3, 9]. It is easier to disable a whole line for a faulty cell. Disabling faulty lines degrades performance significantly by increasing cache miss rate. For this reason, it is efficient to disable the faulty bit at a smaller granularity, such as half line, double word, or word. There are many existing techniques that correct the value stored in a failing cache cell. In our proposed technique, we have disabled the faulty bit at a word granularity.

1.6 Thesis Organization

The remainder of this thesis is organized as follows. In chapter 2, we review prior related work that was proposed to enable low voltage operation. In chapter 3, we describe our proposed cache architecture, operation, and the fault model of our design in detail. In chapter 4, we explain how we estimated the performance and overhead of our design. We present our results in chapter 5 and conclude in chapter 6.

2. RELATED WORK

In this chapter, we describe previously proposed mechanisms to allow a cache to operate at low voltages. These techniques include architectural, circuit, and code-based error detection and correction mechanisms.

2.1 Architectural Techniques

The architectural techniques emphasize disabling the erroneous SRAM cells in the cache. These techniques modify the existing 6T SRAM cache architectures by disabling bits at different granularities such as word, double-word, or line granularity.

Wilkerson, et al. [1], have proposed two cache mechanisms (Word-disable and Bit-Fix) that can operate below 500mV reliably in a 65nm process by sacrificing 25% to 50% of cache size at low voltage. Both mechanisms perform a memory test at low voltage when the processor boots to discover faulty bits. When the system switches to the low voltage mode, it flushes the existing cache data. The tag array is designed with more fault-tolerant SRAM cells such as 10T Schmitt Trigger (ST) [6]. It is also possible to use SECDED ECC (Single-bit Error Correction, Double-bit Error Detection) code instead of the big 10T SRAM in tag array to get reliable performance. Using ST SRAM in the tag array requires 2.5X- 4X more area compared to 6T SRAM tag area.

The Word-disable mechanism [1] isolates defects at a 32-bit word granularity and then disables words that contain defective bits. Every cache line keeps a defect map array

of one tag bit per word indicating whether cache word is defective. As two physical lines store the data of one logical line of memory, tags of the two consecutive ways in a cache set are the same. The sixteen words in a 64B cache line are divided into two halves of 8 words, each with a maximum of four defective words, and each storing four of the eight required words. Two four-stage shifters were used in the 16 words line to remove defective words. A line with more than four defective words in either half renders the whole cache as defective. In this mechanism, the two consecutive ways of physical lines were combined to form a single logical line. For example, an L1 cache that is, at high voltage, a 32KB 8-way set associative with 64B per line becomes a 16KB 4-way set associative cache with 64B per line at low voltage. This mechanism increases the cache size by ~15% to store the defect map and 50% of the cache area can be effectively used.

Another technique described by Wilkerson, et al. [1], is the cache Bit-Fix mechanism. It disables any adjacent two bits that contain a faulty bit, and uses 2-bit patches to correct the defective bit pair. Bit-Fix, unlike Word-disable, does not store defect map in every line. Instead, it stores its repair pattern using one cache line for every three data lines. In the high voltage mode, repair patterns are saved in main memory. During a read or a write operation, the repair line is fetched in parallel with the data line. For example, Bit-Fix organizes an 8-way cache at low voltage into two banks, each with three data lines and one repair line. On a cache hit, both the data line from bank A and repair pattern line from bank B are read. The data line passes through 'n' bit shift stages, where 'n' represents the number of defective bit pairs. Each stage removes a defective pair, replacing it with the fixed pair. In a 512-bit line, 256, 2-bit multiplexers were used.

A 140-bit repair pattern can repair a single data cache line (of 512 bits) with ten or fewer defects. The main advantage of Bit-Fix over Word disable is that it provides bigger effective cache capacity in the low-voltage mode, and therefore increases memory accesses by only 7.5% on average (compared to 28% in Word-disable). However, the access logic is more complex and needs a higher latency compared to Word-disable, making it unsuitable for level-1 caches.

Another technique was proposed by Abella, et al. [9]. In this technique, the cache line is divided into sub-blocks. Each of the cache lines was extended with a few bits as VS (Valid Sub-block bits) to track the faultiness of each sub-block. The number of extended bits VS is equal to the number of sub-blocks. The sub-block is considered to be faulty if the corresponding sub-block has more faulty bits than allowed by the underlying protection scheme. If SECDED protection scheme is used, VS will be reset (0) if two or more faulty bits are found in a sub-block; otherwise VS bits are set (1). The VS bit is protected in the same way as data and tag by parity or SECDED. When a cache access is performed, the address offset bits are used to pick the VS bit corresponding to the sub-block of desired cache line. If the tags indicate a miss and the VS bit is reset (0), then a false hit signal is generated but a miss is reported. In a write back cache, data is updated in the higher cache level in valid sub-blocks. In write through caches, higher cache levels are updated for stores, and the request is treated as a regular miss. If the tag indicates a hit and the VS bit is reset (0), a false miss arises. In this case, the higher cache level provides data directly to the requester, and the data cache line is filled in a different, fault-free cache line. The main advantage of this mechanism is that data, tags, and VS bits are all

protected using the same code. The VS bit can be obtained in BIOS initialization and during operation. It is possible to store the VS bit configuration for different voltages in main memory. But switching to a different voltage requires flushing the existing cache data. This scheme is advantageous to Word-disable as the position of a fault is only needed to fetch data directly from a higher-level cache, not to skip the sub-block [1]. But it also puts pressure on L1-to-L2 communication bus due to high failures in sub-blocks.

Ansari, et al. [5], proposed Zereh Cache (ZC) that remaps the cache structure by intelligent arrangement of data in cache lines. ZC partitions the complete cache array into sets of equally sized logical groups, where each logical group is allocated with one spare cache word line. Each data or spare line is divided into equally sized data chunks to allow smaller granularities of spare substitution. The physical cache lines are shuffled to form logical groups in order to optimize the utilization of an externally added single spare cache line. A self-testing module BIST (Built in Self-Test) has been used. The BIST module creates the fault map when the system boots. The fault map array and spare cache access are performed in parallel during the time of cache array access. The fault map access determines whether the spare data chunk should be routed to the output instead of the main cache content. A non-blocking routing is provided by a back-to-back connection between the row decoder of the main cache and the cache word-lines through Benes Network [14]. The main advantage of this scheme is that effective cache size in the low voltage mode is the same as that in high voltage mode, whereas all other schemes sacrifice cache capacity in low voltage. However, ZerehCache increases the cache access latency, and significantly increases cache design complexity.

2.2 Circuit-level Solutions

In circuits and device-level techniques, the organization of 6T SRAM cell is modified. Calhoun, et al. [7], proposed a bit cell architecture named sub-threshold SRAM. It can operate below 400mV in the sub-threshold region in a 65nm process. A buffer is used for reading. Read access is single-ended and occurs on a separate bit line, which is pre-charged prior to read access. The structural difference between sub-threshold SRAM and 6T SRAM is that the read word line is distinct from the write word line. This 10T sub-threshold SRAM technique keeps the static noise margin (SNM) constant below threshold voltage in read access. SNM is the maximum amount of voltage noise that can be tolerated at the cross-inverters output nodes. The main disadvantage of this technique is the 66% area overhead compared to 6T SRAM. But due to the constant SNM, the design has more bit cells in a bit line.

Kulkarni, et al. [6], proposed modified Schmitt-Trigger-based 10T SRAM with feedback mechanism. The proposed SRAM cell focused on making the basic inverter pair memory cells robust. In the low voltage mode, the cross-coupled inverter pair loses its write and read stability. The Schmitt-trigger increases or decreases the switching threshold of an inverter depending on the direction of the input transition using feedback mechanism. Therefore, in the low voltage mode, it is expected that the threshold voltage will decrease. The basic structural difference between Schmitt-trigger and a normal inverter is the feedback loop in both pull-down and pull-up network, which requires six transistors instead of two transistors in the inverter circuit. They used feedback only in pull-down network, as the pull-up network is normally operative in the low voltage mode

to hold the '1' state. So, in the place of a 6T SRAM cell, they proposed a 10T SRAM cell. The main advantages of this design are better read and write stability in the low voltage mode even at 160 mV. It becomes more tolerant to process variations. The proposed ST bit cell operates at 175 mV lower supply voltage than the 6T cell in same (10^{-3} FIT) read and write failure rate. The ST bit cell based cache causes 60% longer access time than 6T to keep the read operation stable. It requires about 2X area compared to 6T SRAM. However, the size and latency of the cache that uses Schmitt Trigger will significantly increase compared to cache that uses 6T SRAM.

2.3 Coding Techniques

Coding techniques detect and correct errors of SRAM cells by using error detecting and correcting codes (ECC). Yoon, et al. [9], proposed a two-tiered error correction and detection scheme, which stores the redundant information in low cost off chip DRAM instead of storing in SRAM. In this scheme, interleaved Tier-1 (T1EC) error detection code (a parity-based error detection code) is stored in a fixed place of last level cache. High-level Tier-2 (T2EC) ECC code (SECDED or Hamming ECC) is also stored in the memory system as addressable data instead of storing in every cache line. Thus, the error correction T2EC code can be cached in the Last Level cache (LLC) and eventually stored in low cost off chip DRAM. When a cache line is read from the LLC (writing data in L1 cache or write back in L2), the T1EC is used to detect errors. If an error is detected, correction is performed by T2EC, stored in on-chip register T2EC_base. The T2EC array and redundant information are stored in DRAM to correct data. At the same time of writing to the LLC, the T1EC is computed and stored in the T1EC portion of the cache

line. T2EC is only computed for dirty lines that are written back into the LLC from a previous-level L1 cache. This newly generated T2EC is mapped to a cacheable DRAM address, and if this T2EC address is already in the cache, it is updated with the newly computed T2EC. If the T2EC address is a cache miss, a line is allocated in the LLC and populated with the relevant T2EC information.

The main advantage of this technique is the use of DRAM to store ECC (T2EC) instead of storing it in costly SRAM. In this technique T1EC is stored in SRAM but T2EC is stored in DRAM as EDC takes only 2.4% storage overhead. The redundant data to fix errors is stored in DRAM, which reduces area overhead of SRAM. The DRAM access latency is high compared to SRAM. The DRAM access latency is 60-120ns, whereas the SRAM access latency is 5-24ns.

Yoon, et al. [10], proposed another two-tiered error protection scheme based on ECC FIFO. The difference between these two schemes is the Memory-Mapped ECC (MME) stores T2EC code as cacheable data in LLC, while ECC FIFO does not store any T2EC information in SRAM. Instead, when T1EC detects any errors, the T2EC redundant information is read and decoded from the T2EC FIFO allocated in DRAM. The T2EC FIFO is then searched starting from the newest entry until a matching tag is found and the redundant information can be retrieved for that detected error. When a data is written in the LLC from DRAM, a T1EC is encoded and written along with the data into the LLC. A T2EC is encoded only when a dirty line is written into the LLC from the write back L1 cache. The encoded T2EC is combined with the tag, which is a pointer to the corresponding physical data line in the LLC. A tag is composed of the set number and

the way number of the cache line so that the T2EC can later be associated with a detected error. When a data line is read from LLC and the T1EC detects an error, T2EC FIFO is searched to find the tag for that line to find the redundant ECC data. It is possible to use better error detection and correction schemes (such as SECDED) in T1EC in both ECC FIFO and MME schemes, but it requires more area overhead. Overall performance degradation of using ECC FIFO is similar to MME. MME increases DRAM traffic less compared to ECC FIFO.

Chishti, et al. [3], proposed multi-bit segmented ECC (MS-ECC) to address persistent and non-persistent failure, and to improve cache lifetime reliability at low voltages. MS-ECC corrects bits using majority voting by implementing an Orthogonal Latin Square Code (OLSC) [11]. In the low voltage mode, the cache is divided into data ways and ECC ways at different granularities depending on the required reliability level. Each data way and ECC way is again sub-divided into multiple segments and stores the ECC for each segment in the corresponding ECC way. For example, the paper uses 64-bit segments, so each 64B cache line contains eight segments. There are separate ECC decoders and encoders for each of the eight segments that decode and encode segments in parallel by using information from both the data and ECC ways. On a read hit, both the data line and the corresponding ECC line are fetched and decoded in segments. The decoded segments are then concatenated to obtain the entire 512-bit line. On a write hit, the ECC of the data line is obtained from the ECC encoder. Then the new data is written to the data line and the new ECC is written to the corresponding ECC line. As both the data way and the ECC way need to be accessed simultaneously, this mechanism requires

doubling the bus width. The disadvantage of the MS-ECC is the cost of OLSC stored in SRAM. In the worst case, OLSC may occupy half of the cache area to correct 4 bits per segment in a 64B line.

Sadler, et al. [12], proposed Punctured ECC Recovery Cache (PERC) that fetches the error correction bits from PERC to the L1D cache after errors are detected. This scheme tolerates more errors than ECC (L1)/ ECC (L2) with slightly better performance and lower power. In this scheme, error codes add r check bits to each k -bit piece of data to create n -bit ($n=r+k$) code words that contain information redundancy. The r check bits are divided into punctured error detection bits r_d (EDC_p) and punctured error correction bits r_c (ECC_p). For the L1D cache, the r_c bits are stored in PERC. For the L2 cache, the error detection and correction codes are stored as non-punctured code ECC_{np} in the L2 cache. The number of EDC_p and ECC_p bits determines number of possible errors that could be detected and corrected. The punctured cache has a similar number of frames and set associativity as the L1D cache. Each data word in the L1D cache has corresponding ECC bits in PERC. The main advantage of this scheme is that error correction is possible without storing ECC bits in the expensive L1D cache. The access latency in PERC is only 1% higher compared to an unprotected cache. Compared to other ECC schemes (SEC, DEC, TEC, QEC), PERC gives higher bit error correction at the same latency. This scheme does not require extra bandwidth between the L2 cache and the L1D cache. But this scheme has an overhead of one bit per word in the L1D cache. It also requires the PERC structure to store ECC for the L1D cache.

Kim, et al. [13], proposed an area-efficient non-uniform error protection scheme for the L2 cache, which applies ECC in dirty lines and parity bits in clean lines. To reduce the number of dirty lines in the cache, they clean the dirty cache lines by periodically writing them back to main memory. Cache line cleaning is performed by the cleaning logic that includes a cycle counter and a latch storing the next cache set number. The cleaning logic checks cache lines belonging to the cache set number stored in the latch after a predefined number of cycles. The L2 cache has a parity-bits array for each cache way and one ECC array for all cache ways. As the write back in memory may increase bus memory traffic, they propose to determine the best dead time of the cache line to write in memory. This technique reduces area overhead by 59% compared to a normal ECC technique, but it increases memory latency by 1% compared to unprotected cache.

3. SPARE BLOCK CACHE ARCHITECTURE

In this chapter, we explain our proposed spare block cache architecture, cache failure probability model, and critical path delay. We design a cache controller that skips the faulty words, and shifts the data to the adjacent fault-free word position. We have added spare words to store the overflow of words. The critical path of our design has been estimated at low voltage mode with 16 spare blocks.



Fig 3.1: A 4-way set associative cache

We have proposed a cache architecture that can operate reliably at low voltage. At low voltage many SRAM cells fail to operate reliably. We have designed a cache that disables the faulty words and shifts the data to the adjacent fault-free word positions to repair the failing bits. We have designed cache access controller circuits. The cache access controller is designed using shifters. The shifters shift the word line data to the adjacent fault-free word that was configured to store at the faulty word location. The control circuit is used to connect the data word line with the next fault-free SRAM words. In Figure 3.1, we show a single set in a 4-way set associate cache. Each cache line has 4 words. Therefore, in a cache set there are 16 words. In our design, we have used spare blocks to back up a cache set. Assume four spare words have been added to back up this

16-word cache set. In Figure 3.2, the nominal cache words are w_0-w_{15} , and the spare words are sw_0-sw_3 . In our proposed technique, we have used “disable bit” to tag the words as faulty or fault-free. If the disable bit is ‘1’, then the corresponding word is faulty, similarly if disable bit is ‘0’, the word is fault-free. Figure 3.2 show the additional 20 disable bits that have been used to tag the 20 words of this cache set.

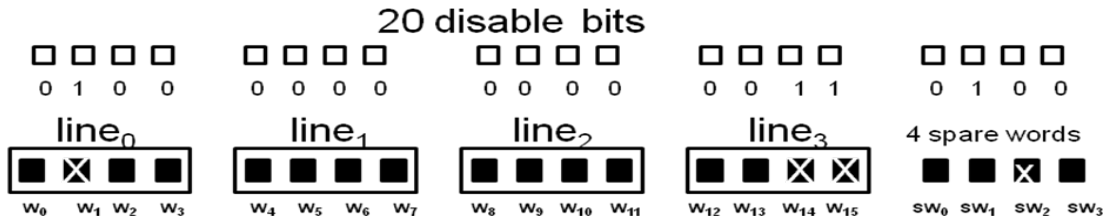


Fig 3.2: Faulty words and disable bits

We assume there are four faulty words detected in the cache set. Figure 3.2 shows w_1 , w_{14} , w_{15} , and spare word sw_2 as faulty. The faulty words are marked as ‘X’. In our approach, the data word lines are not connected directly to the SRAM word. Switches are used between SRAM words and data word lines to switch the data word line data to a fault-free SRAM word. Figure 3.3 illustrates word shifting. The design data word line₀ is connected with word w_0 , while data word line₁ is not connected with word w_1 , as word w_1 is faulty. pMOS switches connect the data word line₁ with the next fault-free word w_2 and data word line₂ stores data in word w_3 . The rest of the data word lines store data by shifting one word until another faulty word has been detected. Consequently, data word line₁₂ has been stored in word w_{13} . In Figure 3.3, we show that next two faulty are w_{14} and w_{15} . Data word line₁₃ cannot be stored data at word w_{14} or w_{15} , since those are faulty. Therefore, data word line₁₃ stores data in the first spare word sw_0 , and the data word

line14 connects with sw_1 . The next data word line15 cannot store at the next spare word sw_2 , as this spare is also faulty, so it connects with spare word sw_3 . The faulty data word line is skipped, and data is stored in fault-free words.

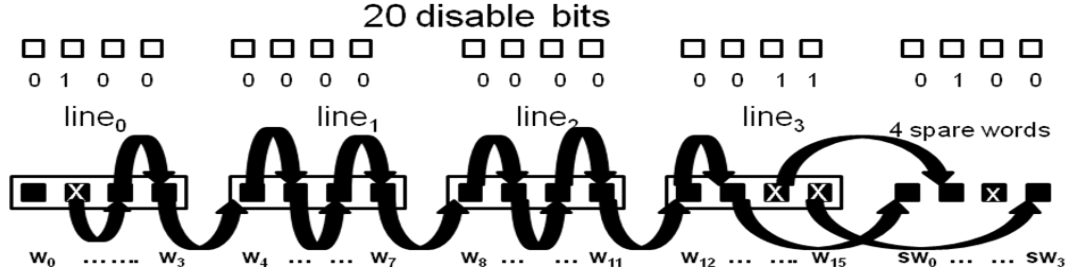


Fig 3.3: word shifting

For the purposes of this example, we assume that every word has four SRAM bits (b_{00} - b_{03}). Therefore, there are four bit lines (bl_{0_0} - bl_{0_3}) in a data word line (data word line₀). In Figure 3.4, we observe that a bit in word w_1 is faulty (any of the bits b_{10} to b_{13}). As we are disabling at a word granularity, we tag the whole word w_1 as faulty. Therefore, the data word line₁ connects to the word w_2 instead of word w_1 , as w_2 is the next fault-free word. Similarly, data word line₂ stores data in word w_3 instead of word w_2 , as w_2 has already been used by bits from data word line w_1 . Therefore, the bit line bl_{1_0} connects with the SRAM bit b_{20} instead of connecting with SRAM b_{10} . Similarly, all other bit lines (bl_{1_1} to bl_{1_3}) of data word line w_1 store data using bits of word w_2 . This switching of connections is done by the pMOS gates.

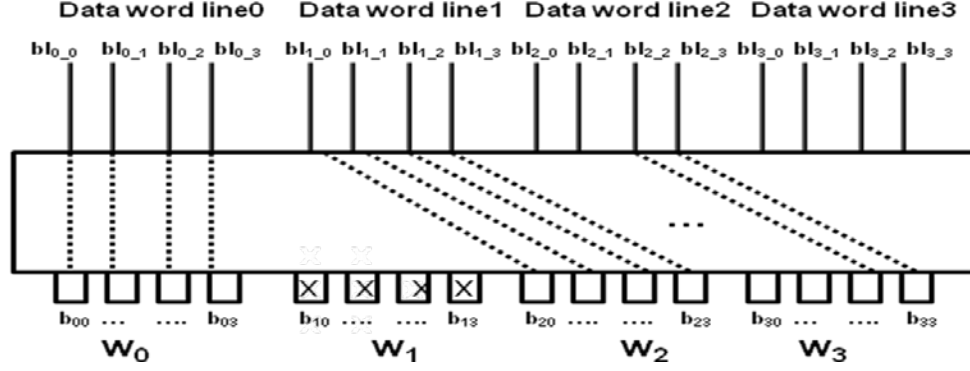


Fig 3.4: Four consecutive words with faulty word w_1

In our approach, we focus on designing a reliable cache that operates at low voltage without sacrificing cache capacity significantly. The shifting mechanism employs a physical organization utilizing the sub-block disable technique [9]. However, we allow faulty cache blocks to reside in the next fault-free position. Once a faulty block is disabled, all blocks are displaced by at least one position. The cache is able to operate at low voltages by disabling and shifting words simultaneously into fault-free positions.

Figure 3.5 presents an 8-way set associative cache with 16 spare words. In a traditional cache, every cache line has 16 words and a set of eight lines has 128 words. We add a spare block of 16 words to back up the 128 words. In our proposed architecture, a cache set has a total of 144 words. We use 144 extra bits to tag the words as faulty or fault-free.

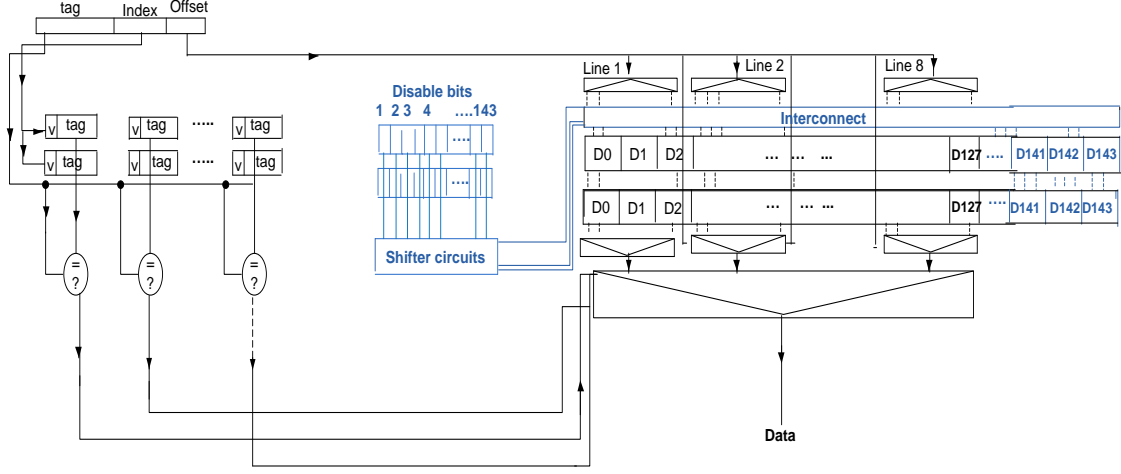


Fig 3.5: Proposed cache architecture with spare blocks

In the following sections, we describe the cache failure probability model and the architecture of our proposed spare block cache architecture. We have added spare words to back up a cache set to keep the cache set failure probability acceptable at low voltage. In the next section, we present the fault model of our proposed cache design.

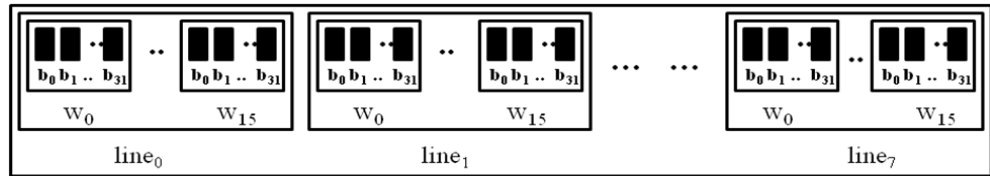


Fig 3.6: Cache set

3.1. Cache Failure Probability Model

We developed a model to estimate the SRAM failure probability of our spare block architecture. We use the bit failure probability measured by Kulkarni, et al. [6]. We denote the failure probability of an SRAM cell as $P_{fail}(b)$. The fault-free probability of a

cell is denoted by $\overline{P_{fail}(b)}$, where $\overline{P_{fail}(b)} = 1 - P_{fail}(b)$. We divide the cache line at a word granularity. There are 32 bits in a word, and bit failures are independent of each other. The fault-free probability of an SRAM word is, $\overline{P_{fail}(w)}$, where $\overline{P_{fail}(w)} = (1 - P_{fail}(b))^{32}$. In our design, we have implemented 32KB and 2MB 8-way set associative caches, in which each cache set has eight lines and each line has 16 words. Thus, a cache set has 128 words. The fault probability of a cache line and cache set can be expressed as, $P_{fail}(l)$ and $P_{fail}(s)$ respectively. Therefore, fault-free probability of a cache set is, $\overline{P_{fail}(s)} = (\overline{P_{fail}(w)})^{128}$. If at least one bit cell fails to operate reliably, the cache set is treated as faulty. But, at low voltage some bit cells fail to operate reliably. Therefore, we tag a word as faulty if at least one bit cell in that word fails to operate reliably at low voltage mode. In our proposed cache architecture, we used spare words to back up a cache set. In our proposed fault probability model, we denote the cache set failure probability as $S_{fail}(s)$. In our design, if one spare word is used, the cache set is fault-tolerant to one cache word failure. Here, after a faulty word has been found, all the words shift by one word position. The spare word stores the last word of the cache set. Thus using a number of spare words actually increases the fault coverage of the cache set. We have denoted the number of words in a cache set and spare words in a cache set as, N_w and N_{sw} respectively. Hence, with the spare block cache architecture, the cache set fault-free probability is, $\overline{S_{fail}(s)}$, where,

$$\overline{S_{fail}(s)} = \sum_{n=0}^{n=N_{sw}} {}^{N_{sw}+N_w}C_n P_{fail}^n(w) \overline{P_{fail}(w)}^{N_{sw}+N_w-n} \dots\dots\dots (3.1)$$

We calculated both of the probabilities of availability of fault-free sets in the cache without any disable mechanism and with proposed spare block mechanism. We tried to estimate the acceptable failure probability of a cache set. Then we have estimated how many spare words need to be used to back up the cache set to achieve that cache set failure probability at low voltage. We have seen that if we use 16 spare words, we can operate the cache reliably at 485mV with acceptable cache set failure probability than a conventional cache operating at nominal voltage.

In the equation 3.1, the fault-free probability of a cache set without any spare block is $\overline{P_{fail}(w)}^{N_w}$. However, when the spare block is added, the spare words back up the faulty words. Therefore, the number of acceptable faulty words can be less than or equal to the number of spare words. Thus using of number of spare words actually increases the fault-free probability a cache set. In our design, we use 16 spare words to back up a cache set.

In the nominal condition when all of the words are fault-free, the cache set fault-free probability for the spare block architecture is higher than the nominal cache set fault-free probability without spare words (i.e. $\overline{S_{fail}(s)} > \overline{P_{fail}(s)}$). The fault-free probability of our cache word is, $\overline{S_{fail}(w)} = \overline{S_{fail}(s)}^{1/128}$, as there are 128 words in a cache set. Similarly, as there are 32 bits in a word, the bit fault-free probability of our proposed architecture can be described as, $\overline{S_{fail}(b)} = \overline{S_{fail}(w)}^{1/32}$. Hence, the bit failure probability is, $S_{fail}(b) = 1 - \overline{S_{fail}(b)}$. The bit failure probability of our proposed cache architecture is

lower than that of the nominal cache architecture without spare blocks (i.e. $S_{fail}(b) < P_{fail}(b)$) due to use of spare words.

3.2. Cache Controller Architecture

We have designed a cache controller to back up a cache set at low voltage. In our proposed technique, we detect errors in cache set using memory tests, and disabled at a word granularity. We add spare cache blocks to a conventional set associative cache. We designed serial shifter circuits to control the switching between SRAM words and data word lines. The pMOS gates are used as switches between SRAM words and data word lines.

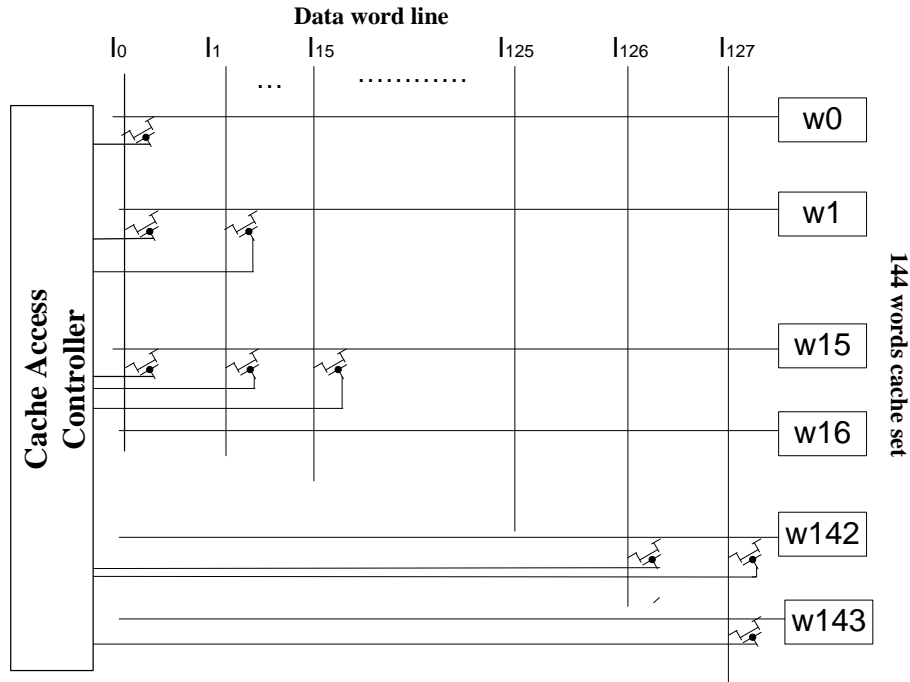


Fig 3.7: Cache controller and pMOS switches between words and data word line

In our architecture, data cannot be accessed directly by I/O lines to SRAM cells. In this design, the pMOS gates control the access to the SRAM. We have tagged the

words of a cache set by disable bits to indicate whether the word is faulty or fault-free. These disable bits are also stored in separate SRAM cells. The disable bits are used as control bits of the shifters. The output of the shifter is used as the controller of the pMOS gates that control 16 consecutive words. Between these 16 words, a maximum of one pMOS gate gets logic low '0' from the shifter output. If one of these sixteen bits is logic '0', then the data can be stored in the corresponding word position controlled by the pMOS gate. Thus a word line can store in any of 16 consecutive word positions. The architecture of the conventional cache and the proposed spare cache is presented below.

Figure 1.1 presents a 6T SRAM cell which is used in cache memory. In conventional design, every storage cell is connected with two column lines and one row line (I/O lines). Information is written into the cell and read out of the cell through the column lines and controlled by the row lines. The memory management unit (MMU) controls cache accesses. MMU includes a small amount of memory that holds a table matching the virtual addresses to physical addresses. We have presented a nominal cache set in figure 1.2. The memory is separated as following: index, tag, and offset bits. The table in which the address is stored is called the Translation Look-aside Buffer (TLB). All requests for data are sent to the MMU, which determines whether the data is in the cache. A read/write controller controls read and write operations in cache. The address lines are usually tied to the memory system address bus, while the I/O data lines are tied to the data bus. The I/O lines are bidirectional. For write operations, the I/O lines carry the data to be written into the memory cells. For read operations, the lines carry the output of the memory cells. When the address is present in TLB, the index bits enable the

matched row. In read operations, tag bits of the line address are compared with the tag bits of the corresponding set in the cache. If any tag matches with the tag of a line, the corresponding data is selected by the multiplexer and data is sent to the data bus. Offset bits select the portion of data needed by the processor from the line by using a multiplexer. In write operations, the cache controller picks one of the set lines to store data using the cache replacement policy.

In the conventional 8-way set associative L1 cache, there are eight lines in a set. Each line stores 16 words of data. Any of these 16 words can be selected by the offset bits. The match of tag bits selects the data line of a set. In our proposed design, we have tested all the words of the set and tagged by disable bits. Thus, in a set of eight lines of 128 words, there are 128 disable bits. As described earlier at low voltage mode, as many cells start to fail, a spare cache block of 16 words is added. In our proposed design we also verified the 16 spare words and tagged them by disable bits. Thus 144 bits have been used as disable bits for a cache set. We have used these disable bits as control bits of the shifter.

3.2.1 Serial Design for Cache Controller

We designed a shifter circuit to skip the faulty word position and move the data to next fault-free word position. At first, we design the shifter as a barrel shifter which controls all the pMOS switches between data word lines and SRAM words. The shifter also generates input for next shifter, which is the next word access controller of the set.

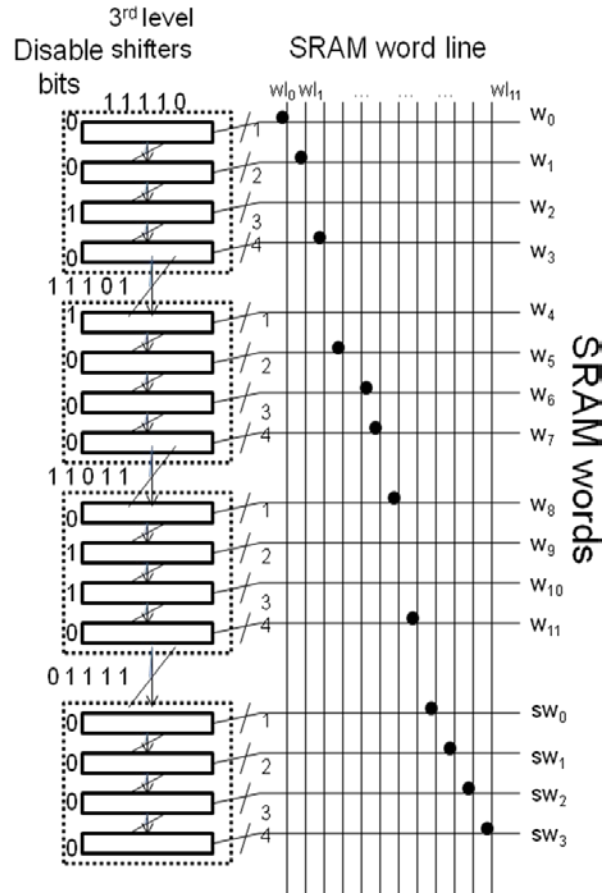


Fig 3.8: Serial Access for Cache Controller

In Figure 3.8, we present a serial cache access controller of twelve words in a cache set. We added four spare words to back up the cache set at low voltage. Therefore, there are sixteen disable bits to mark the sixteen words as faulty or fault-free. The first shifter generates pMOS gate inputs to switch between first data word line wl_0 and SRAM word w_0 . It also generates the output for second the shifter input. Each of the shifter inputs depends on the previous shifter's output. Outputs from all the shifters are required to generate input of the sixteenth shifter. The timing overhead becomes crucial when

there are many words in a cache set. In our conventional caches, there are 128 words in a cache set, and we add 16 spare words to back up the cache set. Storing the data in the 144th word requires passing all the previous 144 shifters.

Only using this shifter would significantly increase cache access latency. We design two other shifters (i.e., 1st and 2nd level shifters) that generate this type of shifters input in parallel. From here forward we are going to use “3rd level shifter” to describe the shifter we described in this section. We divide the disable bits into groups, and the first 3rd level shifter of each group gets its input from the 1st or 2nd level shifters instead of the previous 3rd level shifter.

3.2.2 Controller Implementation

In our proposed cache architecture, we designed the cache controller with three types of shifter circuits. Those are defined as the 1st, 2nd, and 3rd level shifters. The output of the 3rd level shifter is used as the controller of pMOS gates to use as a switch between data word lines and SRAM words. We divide the 144 disable bits into 18 groups, where each group includes eight consecutive disable bits. The disable bits of a group are used as inputs to the 1st level shifter. Moreover, every disable bit is used as the control bit of a 3rd level shifter. Inputs of the first shifter of each group are initialized by the output of the prior 1st level shifter circuit.

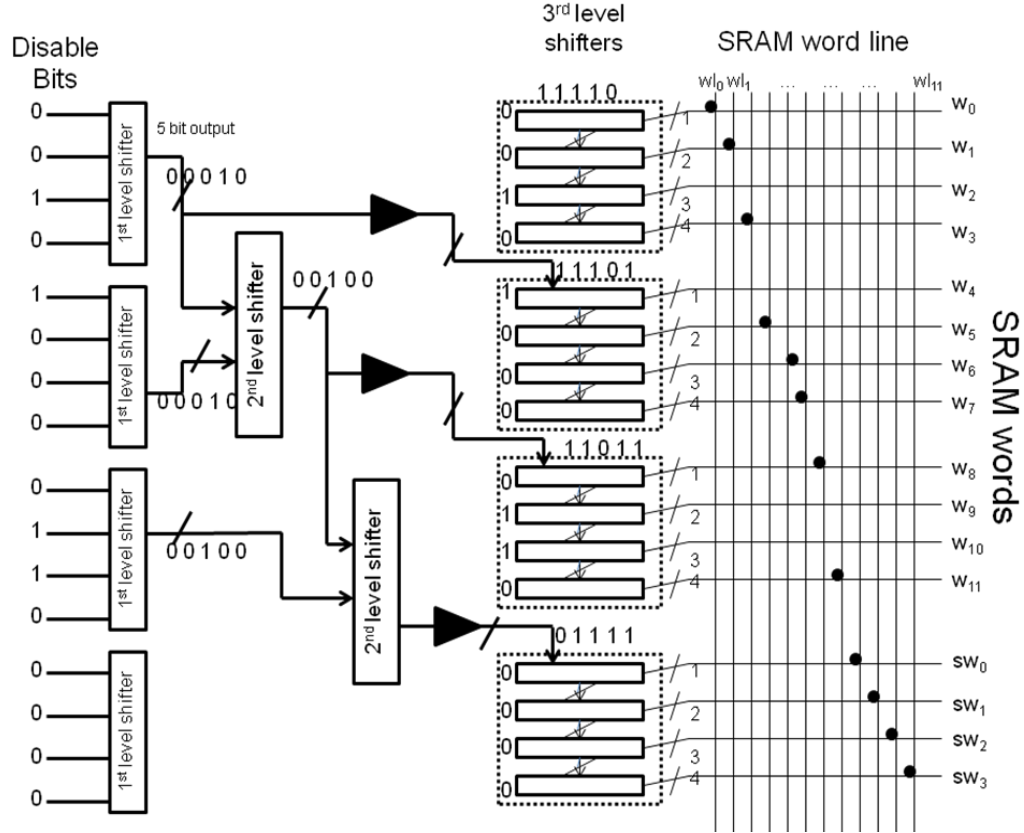


Fig 3.9: Cache access controller with three types of shifters

The design and operation of the cache access controller are demonstrated in Figure 3.9 with a small cache. The cache has 12 SRAM words and 12 data word lines in a cache set. In our proposed spare block cache architecture, we add 4 spare words to back up the cache set. The cache words are marked as faulty or fault-free with the disable bits. We assume that the 3rd, 5th, 10th, and 11th words are faulty and set the disable bit to ‘1’. We use the 3rd level shifters to control the data word line access to the SRAM words. The 3rd level shifter is a barrel shifter, whose input is from the previous 3rd level shifter and the control bits are the disable bits. Thus, the sixteenth 3rd level shifter inputs depend on

the fifteenth 3rd level shifter output, the fifteenth 3rd level shifter input depends on the fourteenth 3rd level shifter output, and so on. As the third level shifters control the connection between data word lines and SRAM words, the inner SRAM words access require a lot of time. That significantly increases cycle time. Therefore, we design two other types of shifters that parallelize cache word access.

We divide the 16-word cache set into 4 groups. There are four 3rd level shifters and their corresponding 4 disable bits in each group. In the first group, the disable bit sequence is 0010. Here, the third disable bit is '1', which demonstrates the 3rd word is faulty. The 1st level shifter is designed such a way that it adds the number of '1's in a disable bit group, which is eventually the number of faulty words found in a group. The output of the 1st level shifter is left-shifted according to the number of faulty words in a group. Therefore, the output of the first group 1st level shifter is 00010, and the output of third 1st level shifter is 00100 since the 1st and 3rd groups have one and two faulty words, respectively. The 2nd level shifter adds the number of faulty words found in a group and left-shifts the position of '1'. As an example, the first 2nd level shifter adds the number of faulty words of the 1st and 2nd group. The number of output bits of the 1st and 2nd level shifters depends on the number of spare words used in a cache set. Here 1st and 2nd level shifters output are five bits, as we have four spare words. Each word can be stored in any of five possible positions. The first 3rd level shifter of each group gets shifter inputs from 1st or 2nd level shifter outputs and the remaining three 3rd level shifters get their inputs from the previous 3rd level shifter. Next, we discuss shifters and pMOS gates of our proposed spare block cache architecture.

The 1st level shifter

We have designed the 1st level shifter circuit as an eight input parallel left-shifter using domino gates. The shifter takes eight consecutive disable bits of a group as inputs and generates 16 bits as output. We used seventeen 1st level shifters for 144 disable bits. The last eight disable bits have been used only as the control bits of the corresponding 3rd level shifters. The working mechanism of the 1st level shifter is as an adder, but it is designed as a shifter. The output of the 1st level shifter is used as an input to the 2nd level shifter or directly to the 3rd level shifter. When the output of the 1st level shifter is used as the input to the 3rd level shifter, it is inverted. The logical equation of every bit of the 1st level shifter output is:

$$P_0 = (f_0 + f_1 + f_2 + \dots + f_7)', \quad \dots \quad (3.2)$$

$$P_1 = f_0 (f_1 + f_2 + \dots + f_7)' + f_1 (f_0 + f_2 + \dots + f_7)' + \dots + f_7 (f_1 + f_2 + \dots + f_6)', \quad \dots \quad (3.3)$$

.....

$$P_8 = f_0 f_1 f_2 \dots f_7, \quad \dots \quad (3.4)$$

and P_9 to P_{15} is equal to *zero*, where f is the disable bit and P is the output bit of 1st level shifter.

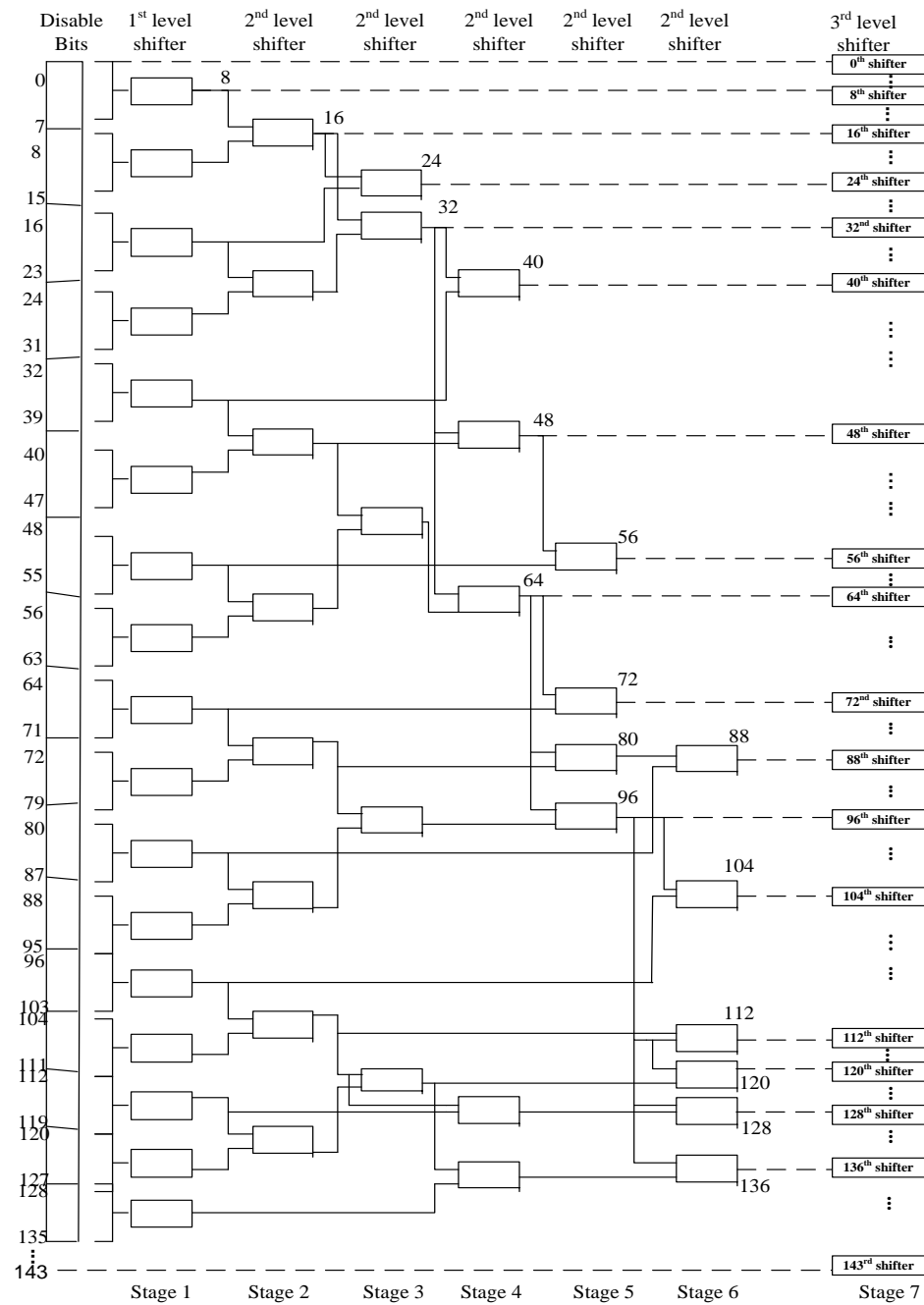


Fig 3.10 Shifter design in cache controller with 64B lines and 16 spare words

As an example, if all the words of a group are fault-free, the sequence of eight disable bits is ‘00000000’. Hence, the output of the 1st level shifter is sixteen bits long, ‘0000000000000001’. Number of 1’s in the disable bits string of a group determines how many faulty words were found in the group and how much the output would shift left. As an example, if the disable bit string is ‘10111001’, there are five faulty words and the output is left-shifted by five positions as ‘000000000100000’. In the proposed design, the output of 1st level shifter is inverted while being used as an input to 3rd level shifter. However, the output of the 1st level shifter is un-inverted while being used as an input to the 2nd level shifter.

The 2nd level shifter

In our proposed technique, we have designed the 2nd level shifter using domino gates. As mentioned earlier, the adders are designed as a shifter circuit but they generate output as an adder. The input of the 2nd level shifter is either from two 1st level shifter outputs, or both 1st and 2nd level shifter outputs, or two 2nd level shifter outputs. The two inputs of the 2nd level shifter are strings of sixteen bits, where any one of the sixteen bits could be one. The position of ‘1’ from least significant bit determines the number of faulty words found in the previous level (if input is ‘0000000000010000’, there are 4 faulty words found). 2nd level shifters add the number of the faulty words found from the previous stages. The output bits of the 2nd level shifters can be described by:

$$R_N = \sum_{n=0}^{n=N-1} P_{In} \cdot P_{2.(N-n)}, \quad \dots \quad (3.5)$$

where, R_N is the output bit number, P_{In} and $P_{2(N-n)}$ are the two input bit numbers. The output of the 2nd level shifter is used as an input to either of the 3rd level shifters, or to the 2nd level shifter. The output of the 2nd level shifter needs to be inverted when it is used as the input to the 3rd level shifter.

Let the output of two 1st level shifters be ‘0000000000001000’ and ‘0000000001000000’. There are three and six faulty words in these two groups, respectively. Hence, the outputs of the two 1st level shifters get added by the 2nd level shifter. Thus, the output of the 2nd level shifter is ‘0000001000000000’, where the ‘1’ is left-shifted by nine positions. If this output is used as the 3rd level shifter input, then it is inverted to ‘1111110111111111’. The un-inverted output is used as an input to the 2nd level shifter.

The 3rd level shifter

In our proposed technique, the 3rd level shifter works as a barrel shifter, in which the disable bit is used as the control bit. We have used this shifter as the control circuit to connect between the bit line and the SRAM cell. We divided the disable bits into eighteen groups. The first 3rd level shifter of each group gets its inputs from the 1st level or 2nd level shifters via an inverter. Otherwise, the rest of the 3rd level shifters receive inputs from the previous 3rd level shifter outputs. The shifter generates two types of outputs: one type is used as input to the next 3rd level shifter, and the other is used as control bits for the gate inputs of pMOS to access the SRAM cells.

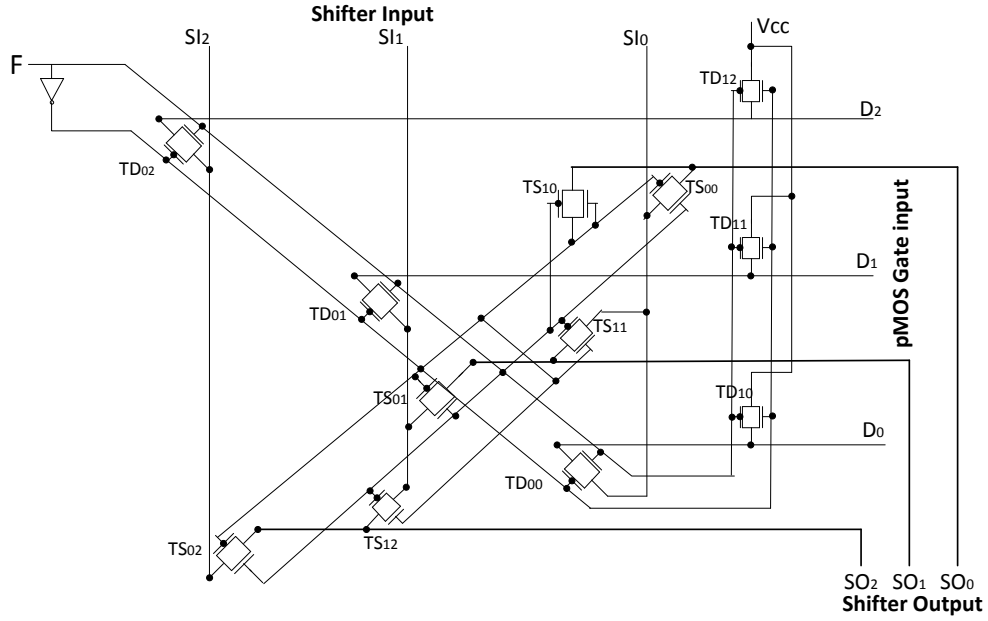


Fig 3.11: Three bit 3rd level shifter circuits

The first 3rd level shifter outputs of the first group can be denoted as D^0 , and SO^0 , where D^0 is the pMOS gate control input and SO^0 is the shifter output. The logical equation of the pMOS gate control input (D^0) for the first 3rd level shifter of the group can be expressed as,

$$D^0 = \overline{F_0} SI^0 + F_0 \cdot (1111111111111111), \quad \dots \quad (3.6)$$

where, F_0 = first disable bit, SI^0 = the first 3rd level shifter input of the first group. Table 3.1 presents the truth table of the 3rd level shifter circuit.

Table 3.1: Truth table for 3rd level Shifter Circuit

Disable bit	Shifter Input(SI)	pMOS gate control input (D)	Shifter Output(SO)
0	Q	Q	Q
1	Q	All 1	1 left shifted Q

In the conventional design, a direct connection is possible between SRAM words and data word lines, as a word can be stored in only one position. However, in our design, a word can be stored in any of 16 consecutive word positions. In Figure 3.11, the 3rd level shifter output ' D ' is used as the pMOS gate control bit to access the SRAM cell. The 3rd level shifter input (SI) represents the number of faulty words found in all previous word positions. The disable bit (F) represents if this word is faulty (1) or fault-free (0). In Table 3.1, if the disable bit is '0', the shifter output (SO) is the same as the shifter input (SI); and if the disable bit is '1', the shifter output (SO) is left-shifted by one. Furthermore, if the disable bit (F) is '0', the pMOS gate control input (D) is same as the shifter input (SI), and if the disable bit (F) is '1', the pMOS gate control input (D) is all ones. In Figure 3.11, we show a three-bit 3rd level shifter circuit. If the disable bit (F) is '0', the transmission gates TD_{00} , TD_{01} , and TD_{02} are 'ON' and shifter inputs SI_0 , SI_1 , and SI_2 connect with pMOS gate control input D_0 , D_1 , and D_2 . Also when the disable bit (F) is '0', the shifter output (SO) is same as shifter input (SI). If the disable bit is '1', transmission gates TD_{10} , TD_{11} , TD_{12} , TS_{10} , TS_{11} , and TS_{12} are 'ON', pMOS gate control

input, D_0 , D_1 , and D_2 are set to '1', and the shifter input (SI) is left-shifted by one position in the shifter output (SO).

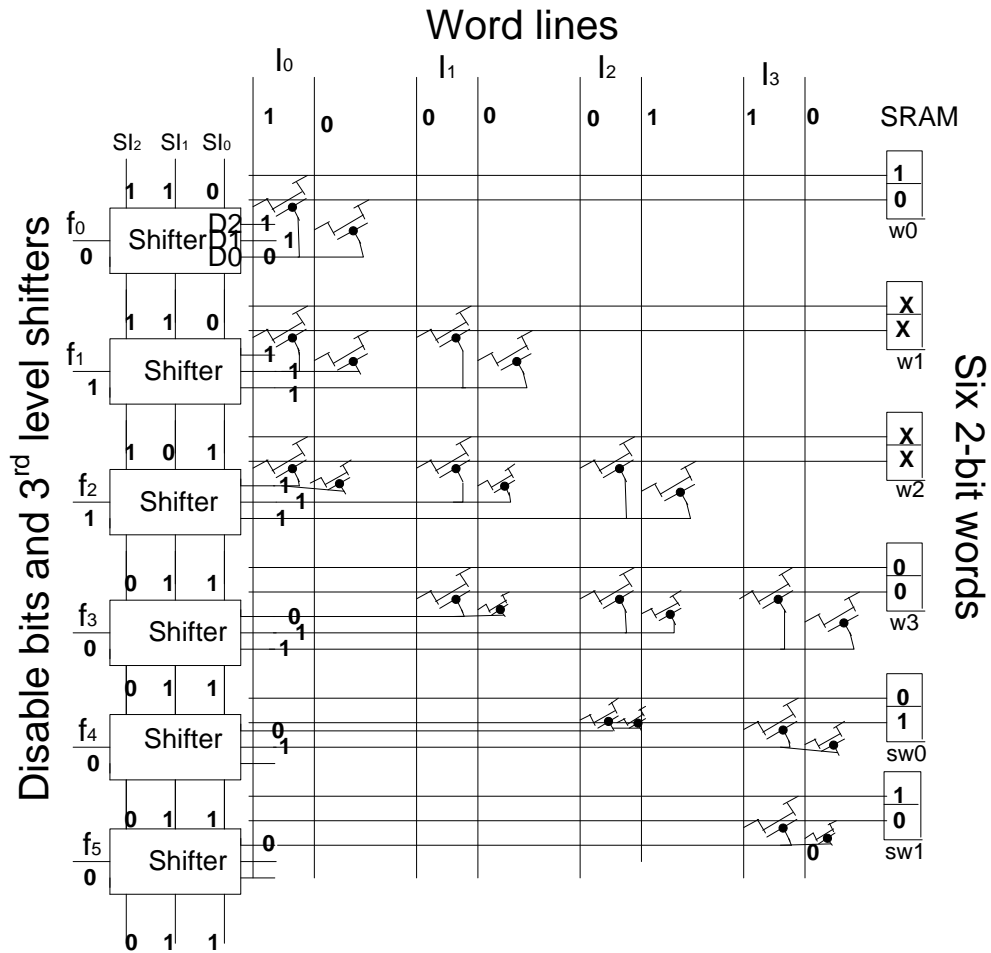


Fig 3.12: Small spare block cache architecture with 3rd level shifters

3rd level shifter and SRAM cell access example

Figure 3.12 presents a small spare cache architecture where the cache has four 2-bit words. Two 2-bit word spare blocks are used to back up the 4-word cache, so six disable bits are required to tag the cache words. For example, we need to store four words of two bits (10), (00), (01), (10) in the data word line sequence of I_0, I_1, I_2, I_3 in the cache

with the disable bit sequence of 0, 1, 1, 0, 0, 0. Initial input to the shifter is $SI_2 SI_1 SI_0 = 110$, where the LSB is $SI_0 = 0$. Since the first disable bit is '0', the first word pMOS control input (D) is equal to the shifter input 110. Therefore, the first word's pMOS gates get a '0' input and get turned ON, and data in first data word line I_0 is stored in the first cache word w_0 . The first shifter output (SO) stays same as shifter input (SI). The second disable bit is '1', so second shifter output (SO) is left-shifted to 101. The pMOS gate control bits (D) are set to all 1's, i.e., 111. Therefore, no data is stored in O_1 as the pMOS gates are turned off. The next word is also faulty, so a similar condition will arise, as the shifter output (SO) is left shifted (011) and the pMOS control bits (D) are all '1' (111). The fourth word is fault-free so the shifter output and pMOS gate control bits are the same as the input '011'. Thus a connection between second data word line and fourth word SRAMs is created. This allows us to have data word lines I_0, I_1, I_2, I_3 store data at the w_0, w_3, sw_0 , and sw_1 word positions.

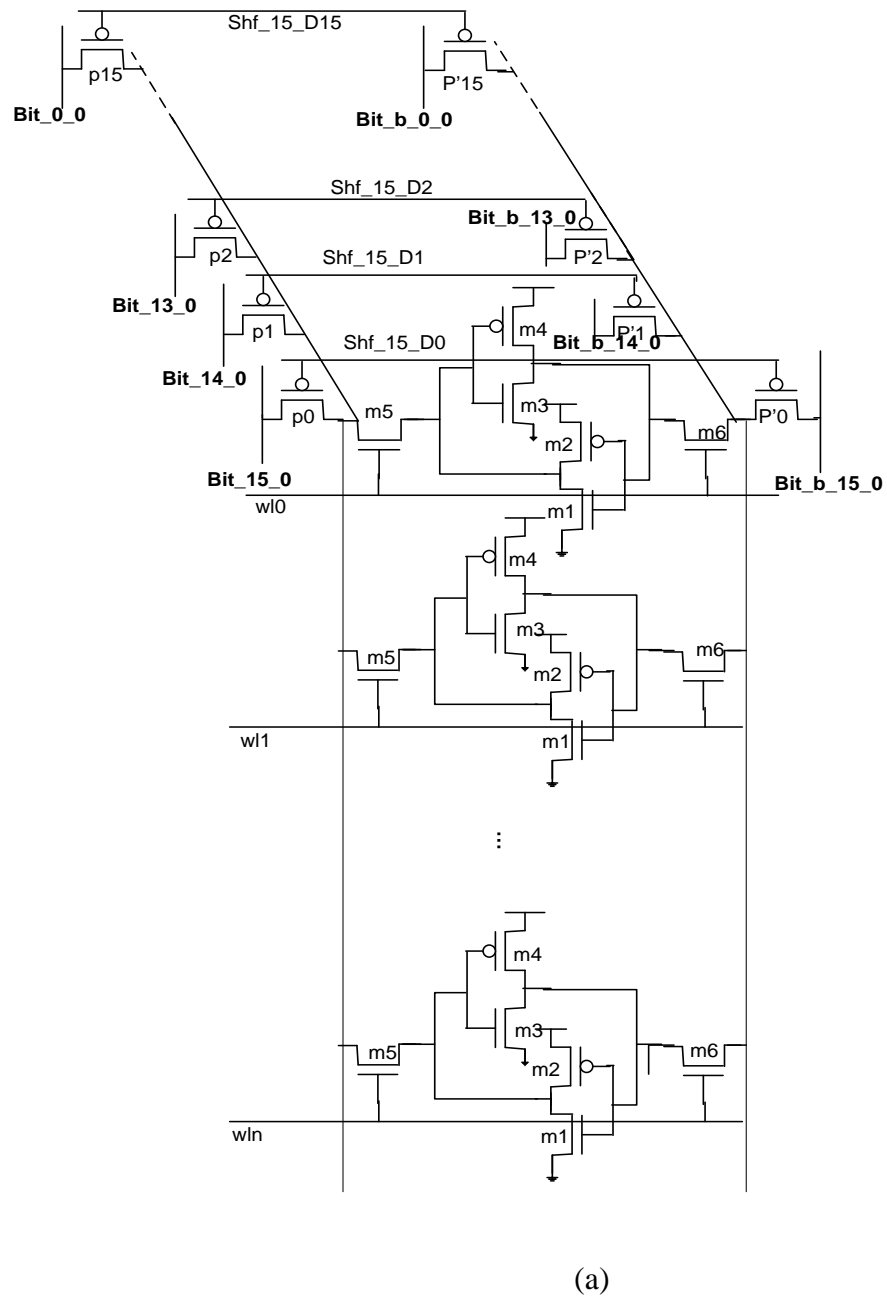


Fig 3.13: SRAM cell access control across multiple cache sets

3.2.3 SRAM cells with control gates

In Figure 3.13, we present SRAM access control in our proposed spare block cache. The fifteenth 3rd level shifter controls the access to the 15th word. All SRAM bits are controlled by the same 3rd level shifter. The pMOS gates are used as a switch only once in each column of SRAM bits. The 3rd level shifter controls access to every word in the cache set. Every bit line in a data word line can store any one of the 16 SRAMs in any of 16 consecutive words. The drains of 16 parallel pMOS gates ($p_0 - p_{15}$) are connected with access transistors (m_5) and the sources are connected with the bit lines Bit_{15_0} to Bit_{0_0} , respectively. Inverse bit-lines ($\overline{Bit_{15_0}}$ to $\overline{Bit_{0_0}}$) are similarly connected through parallel pMOS gates ($\overline{p_0}$ to $\overline{p_{15}}$). We used each of these 16 pMOS gates as switches between the bit lines and SRAM. The pMOS gates are controlled by the pMOS gate control input from the 3rd level shifter output (D). As an example, the source of pMOS p_1 is connected with the first bit line (Bit_{14_0}) of the 14th word. Similarly, pMOS $\overline{p_1}$ is connected with the first inverse bit line ($\overline{Bit_{14_0}}$) of 14th word. All SRAM cells of a column are connected to the same line. The access transistor m_5 determines which SRAM stores the data through the switching of word line wl_0 to wl_n .

Example of SRAM Cell Access Control Across Multiple Cache Sets

In the conventional cache design, index bits are used to select a cache set. However, in our design, index bits select the disable bits in addition to the cache set. SRAM cell access is controlled by the pMOS transistor. Each word is controlled by the corresponding 3rd level shifter. The data word line is connected to consecutive 16 SRAM

words. The pMOSs are connected with all the SRAM cells at the same position in different indexes. In conventional SRAM, the access transistor is the only route to access an SRAM cell. However, in our design, any of 16 bit lines have access to an SRAM cell through pMOS gates. Figure 3.13 demonstrates that each bit SRAM can be accessed by 16 bit lines through pMOS gates. As an example, the twentieth 3^{rd} level shifter input (SI) is ‘1111111111111011’. As two faulty words have been found and twentieth word is fault-free, the eighteenth data word line is connected to the twentieth word position. All the SRAM cells of the twentieth word connect to the 18^{th} word bit-lines at the corresponding positions through the pMOS gates. Therefore, pMOS gate control input is ‘1’ at pMOS gates from p_0 to p_{15} except p_2 . Gate p_2 gets logic ‘0’ as an input, and will be “ON”. So word 2 is skipped and each word starting from word 2 is shifted by one word position.

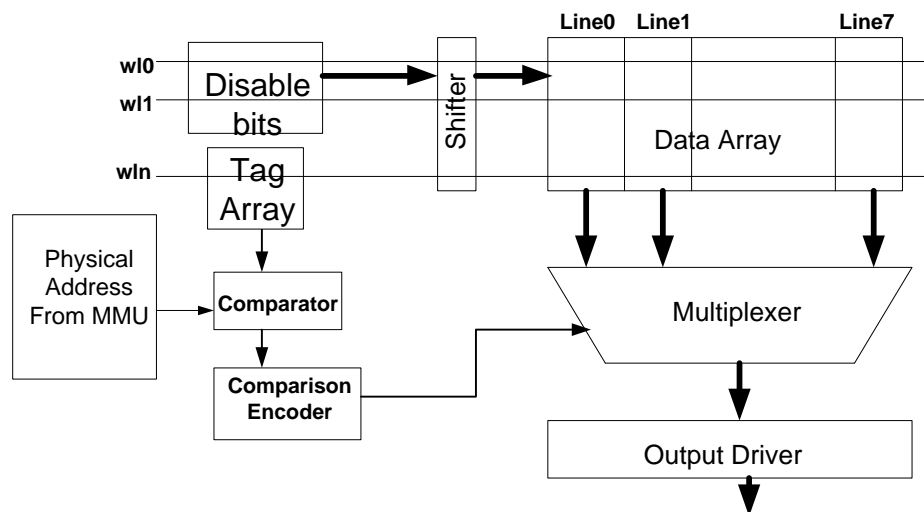


Fig 3.14: Critical path of Spare block cache

3.2.4 Critical Path for Cache Access

The longest path or slowest logic path in a circuit is called the critical path. The critical path of a cache is dependent on cache access during the read operation. The cache access is determined by index and tag bits of the requested physical address. When the data address is available, the index bits are used to activate the cache set that should contain the cache line. The cache is tagged by the physical address, and it requires translating the virtual address. In a conventional cache, tag access time is estimated to be longer than data access time due to comparator delay [17]. Therefore, the critical path in this model is during the tag access operation. In a set associative cache, when the address is available, the appropriate cache set is activated. This set is shared by all associative way-lines. In each way-line, the bit-lines will get the values of the cache block stored in that way. The sense amplifier detects transitions on the bit lines and produces logic values at the inputs to the output way multiplexor. The tag array is also divided into separate way banks, holding the tag information for the cache blocks in the corresponding data ways. All way banks in the tag array are probed in parallel to produce inputs to the n tag comparators. Tag comparators compare the stored tags to the tag bits of the address. The results of these tag comparisons are used to generate the select lines for the output way multiplexor on the data side. Once these select lines are available, the output way multiplexor will output the correct cache block onto the output data bus. The set-associative cache introduces significant critical path delay because of the time to select lines from the tag comparators to the output multiplexer, the time to switch the multiplexer, and to provide the result on the data bus (Figure 1.2).

In our proposed technique, the critical path changes from the conventional cache design. Since the shifters consume significant time to operate, shifters have a significant effect on critical path delay. We estimate the critical path delay by computing path effort (F) and parasitic delay (P) using the method of logical effort [21]. The equations can be described as:

$$\text{Path effort: } F = GBH \quad \dots\dots\dots (3.7)$$

$$\text{Stage effort: } f = F^{1/N} \quad \dots\dots\dots (3.8)$$

$$\text{Delay: } D = N F^{1/N} + P \quad \dots\dots\dots (3.9)$$

Here the path logical effort (G) is the products of the logical efforts of each stage along the path, and the path electrical effort (H) is the ratio between the output capacitance the path must drive and the input capacitance presented by the path. The path branching effort (B) is the product of the branching efforts between stages, and N is the number of stages across the path. In Figure 3.15 we show that there are six stages in the critical path of our shifters.

In our proposed design, we read the physical address from the translation lookaside buffer (TLB) and use it to access the cache. The index will select the cache set and disable bits. The 1st level shifter operation starts when the disable bit line is selected and shifters get disable bits for that cache set. However data and tag access start in parallel with shifter operation. In order to access SRAM, 3rd level shifters need to provide pMOS gate input. Hence, the bit line is unable to store or fetch bits from the SRAM cells before the pMOS is turned ON. The pMOS transistor is controlled by the output of 3rd

level shifter. We have designed 1st and 2nd level shifters with domino gates instead of static gates to minimize the number of transistors used.

The 1st level shifter is an 8-bit input, 16-bit output adder. We computed maximum effort and parasitic delay of all output bits to estimate the logical effort and delay of the shifters. Therefore, the logical effort and parasitic delay of 1st level shifter is: $G_I = 9/3$, and $P_I = 73/3$ (Figure 3.15).

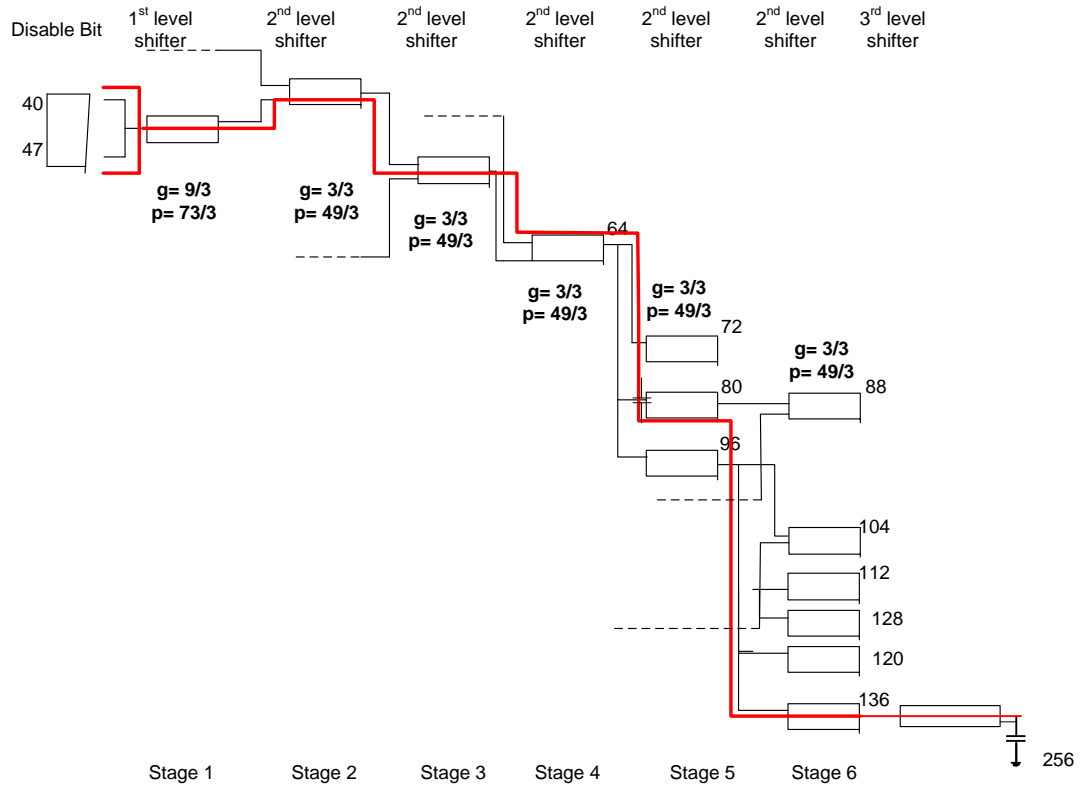


Fig 3.15: Critical path of the cache controller

The 2nd level shifter is also a 16-bit input, 16-bit output adder. The logical effort is equal to all output bits, $g_0 = g_1 = \dots = g_{15} = 3/3$, however, the parasitic is different

for all output bits. Hence the maximum parasitic is $p_{max} = 49/3$. Therefore, the logical effort and parasitic delay of the 2nd level shifter is $G_2 = 1$, and $P_2 = 49/3$.

We have designed the 3rd level shifter with transmission gates, where each of the shifters controls 16 pMOS gates. In our design, the 3rd level shifter gets its input from the previous 3rd level shifter of that group through only one transmission gate. The 2nd level shifter provides the input to the first 3rd level shifter of a group. Therefore, reaching the last 3rd level shifter of a group requires passing 8 transmission gates. Each of these transmission gates controls 32 pMOS transistors. Each pMOS controls an SRAM cell. Hence, we have assumed total load driven by the last 2nd level shifter is 128 pMOS transistors. If we assume that the width of pMOS is twice that of nMOS width (C) due to mobility, the total pMOS load will be 256C [21].

In Figure 3.15, we show that there are six stages in the critical path of the cache controller. The longest logical path starts from the fifth 1st level shifter, which takes 8 disable bits from 40 to 47 as inputs, and generates outputs that are used as inputs for the 2nd level shifter. After crossing five 3rd level shifters, the 136th 3rd level shifter gets inputs and generates pMOS gate inputs. In a conventional cache design, the critical path delay is 17.4 FO4 [17]. In our design the delay can be estimated by path effort, $G = 3$, branch effort, $B = 9$, electrical effort, $H = 256 / 9$. Therefore, path effort, $F = GBH = 256$

Thus, the Delay is: $D = N F^{1/N} + P = 121.12$

We can convert the delay to FO4 unit by dividing by 5 [17, 21]. The Delay is, therefore, $D = 121.12/5 = 24.22$ FO4

We use FO4 as the delay unit since FO4 is a process-independent delay unit. One FO4 is the delay of an inverter, driven by an inverter 4 times smaller than itself, or driving an inverter 4 times larger than itself. Our spare block cache architecture delay is 24.22 FO4, which means that our circuit requires 24.22 more time to drive compared to an inverter.

In our design, we observe that the main contribution in delay is from the parasitic components. The parasitic is much higher in static gates compared to domino gates. Therefore, we chose to design shifters using domino gates.

4. OVERHEAD AND FAULT COVERAGE

4.1 Hardware overhead

We designed a cache access controller for our proposed spare block cache architecture. The size of the access controller is independent of the cache size. We used three types of shifters and pMOS gates to design the cache access controller. We use disable bits to mark whether the word is faulty or fault-free. The disable bits are used as the input to the shifters. The shifters generate pMOS gate control signals to switch between bit lines and SRAM bits. We also used sixteen words to back up the faulty words in each cache set in low voltage mode. It takes 144 SRAM cells to store disable bits to mark the faulty words in each cache set. This is equivalent to adding 16.0156% extra SRAM words in each cache.

Table 4.1: Transistors and Overhead of proposed technique in L1 and L2 cache

Source	Transistors	Number of times used	total(32KB)	Total(2MB)
1st level shifter	2474	17	42058	42058
2nd level shifter	304	28	8512	8512
3rd level shifter	130	144	18720	18720
pMOS at interconnect	32	4740	151680	151680
Valid bits	864	64(32KB), 4096(2MB)	55296	3538944
Back up words	3072	64(32KB), 4096(2MB)	196608	12582912
Total transistors in overhead			472874	16342826
Conventional Cache set	24576	64(32KB), 4096(2MB)	1572864	100663296
Overhead(%)			30.06	16.24

Table 4.1 describes the estimated overhead at the transistor level for our proposed cache design. We used domino gates to design the shifters. Each 1st level, 2nd level, and

3rd level shifter required 2474, 304, and 130 transistors to design, respectively. The total overhead is the overhead of all the transistors we added to our design. The percentage overhead is the ratio between the number of extra transistors in our design and the number of transistors in a conventional SRAM cache. We note that in Table 4.1, the overhead is about 30% in a 32KB cache and 16.24% in a 2MB cache. The overhead percentage decreases significantly as the cache size increases. This shows that our proposed architecture is scalable.

We also implemented our technique in a 6MB L3 cache. It is important to note that the high level cache access is different in our design. In a low level cache, we store the data of faulty words to the next fault-free word of the same or the next cache line. But in a high level cache, we store the faulty words directly to the spare block words. Thus, the selected cache line, spare block words, and the disable bits are activated only when the tag matches. We have used a sequence of OR gates, comparators, and MUXs to generate the disable bit sequence for the 3rd level shifter inputs to the spare blocks for the selected cache line. In a high level cache design, the main drawback of this technique is the increase in the cache access latency. We have computed the transistor overhead increases by 16.87% in a 6MB L3 cache with our design.

4.2 Cache Failure Probability Model

We implemented the cache failure probability model of our design in GCC. We used the high precision C library, GMP, to evaluate the fault coverage at low voltage. We use the high precision library GMP to measure the failure probability accurately with a 32-bit processor. The bit failure probability is computed by using the failure probabilities

for different voltages between 782mV and 405mV [1]. We simulated the fault model for 8 and 16 spare blocks per cache set, and evaluated the results for a 32KB L1-data and 2MB L2-unified cache.

Table 4.2: Cache set failure probability for the spare block cache architecture

		Base Cache Set	With 8 spare word per set	With 16 spare word per set
V_{cc} (mV)	Bit Failure, $P_{fail}(b)$	Failure, $P_{fail}(s)$	$S_{fail}(s)_{8spare}$	$S_{fail}(s)_{16spare}$
405	1×10^{-2}	1	1	1
485	1×10^{-3}	9.83×10^{-01}	0.029	3.411×10^{-06}
55	1×10^{-4}	3.36×10^{-01}	8.062×10^{-10}	1.33×10^{-21}
605	1×10^{-5}	4.01×10^{-02}	1.134×10^{-18}	1.925×10^{-38}
655	1×10^{-6}	4.09×10^{-03}	1.173×10^{-27}	1.398×10^{-53}
702	1×10^{-7}	4.10×10^{-04}	1.177×10^{-36}	1.3831×10^{-65}
743	1×10^{-8}	4.10×10^{-05}	1.178×10^{-45}	1.383×10^{-77}
782	1×10^{-9}	4.10×10^{-06}	1.178×10^{-54}	1.384×10^{-89}

Table 4.2 presents the cache set failure probability with eight and 16 spare words. Our simulation results show that the failure probability of the cache sets without using any spare block is 4.10×10^{-06} at 782mV, 9.83×10^{-01} at 485mV, and 1 at 405mV. The bit failure probability at 782mV is 1×10^{-9} . Below this voltage, bit failure probability increases. With this bit failure the cache set failure probability is found 4.1×10^{-06} . Therefore we can assume that the tolerable failure probability of the cache set is less than or equal to 4.10×10^{-06} . Our design directly affects the cache set failure probability as we use spare blocks to back up the cache set. We also note that the cache set failure probability increases as the voltage decreases due to increase of bit failure probability.

Also note from Table 4.2 that eight spare words per cache set result in a failure rate of 1.18×10^{-54} at 782mV, and 8.06×10^{-10} at 550mV. We conclude that if we use eight spare words in each cache set, we can operate the cache at around 550mV with an acceptable cache set failure probability. If we use 16 spare words, we can operate at 485mV with set failure probability of 3.41×10^{-06} , which is slightly better failure rate than a conventional cache design at 782mV.

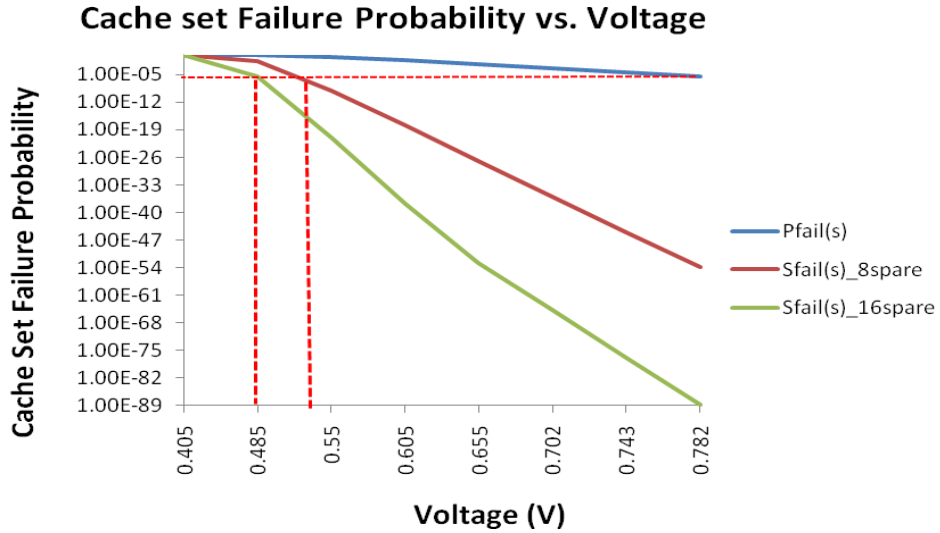


Fig 4.1: Voltage vs. Cache Set Failure Probability

Figure 4.1 shows the voltage vs. cache set failure probability. The X-axis of the graph is the failure probability of a cache set, and the Y-axis represents the operating voltage. We note that the spare block design results in a cache set which has higher fault tolerant behavior at a voltage below 500mV.

We evaluate the cache failure probability of 32KB and 2MB 8-way set associative caches. Each cache set has eight lines in each set. In each line there are 16 words, so each

set has 128 words. Thus, a 32KB cache has 64 sets and a 2MB cache has 4096 sets. We have evaluated the failure probability of these two types of caches with 8 and 16 spare blocks. The simulation results show that both of these caches can reliably operate reliably around 550mV and 485mV when 8 and 16 spare blocks are used, respectively. We plotted three graphs of the cache failure probability at different configurations: the nominal cache without any spare blocks, with eight spare blocks, and sixteen spare blocks. The cache failure probability is calculated from bit failure probability at different voltages [1]. A horizontal dotted line is drawn from the point where the nominal cache can operate reliably. We used that line as the margin of acceptable cache failure probability. So, the intersection point of the dotted line and a plotted graph represents acceptable voltage for the given configuration. The cache failure rate is acceptable at 485mV when 16 spare words are used and at 530mV when 8 spare words are used.

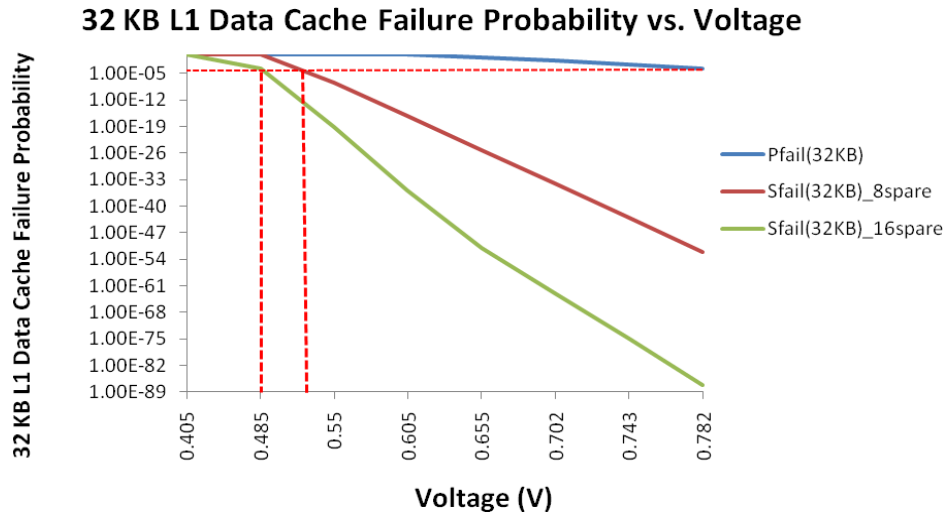


Fig 4.2: 32 KB cache failure probabilities at different voltages

Figures 4.2 and 4.3 show the 32KB and 2MB cache failure probability at different voltages. It indicates the baseline 32KB cache failure probability is 2.62×10^{-4} , and the baseline 2MB cache failure probability is 1.66×10^{-2} at 782mV. The horizontal dotted line denotes the acceptable limits for reliable cache operation. The notation 32KB_6T denotes a 32KB L1 data cache with conventional design. 32KB_8Spare and 32KB_16Spare denotes a 32KB L1 data cache with 8 spare words and 16 spare words, respectively. The horizontal dotted line intersects the 32KB_8Spare graph at 530mV, and the 32KB_16Spare graph at 485mV. This line shows that our 32KB L1 designs can operate reliably at 530mV and 485mV.

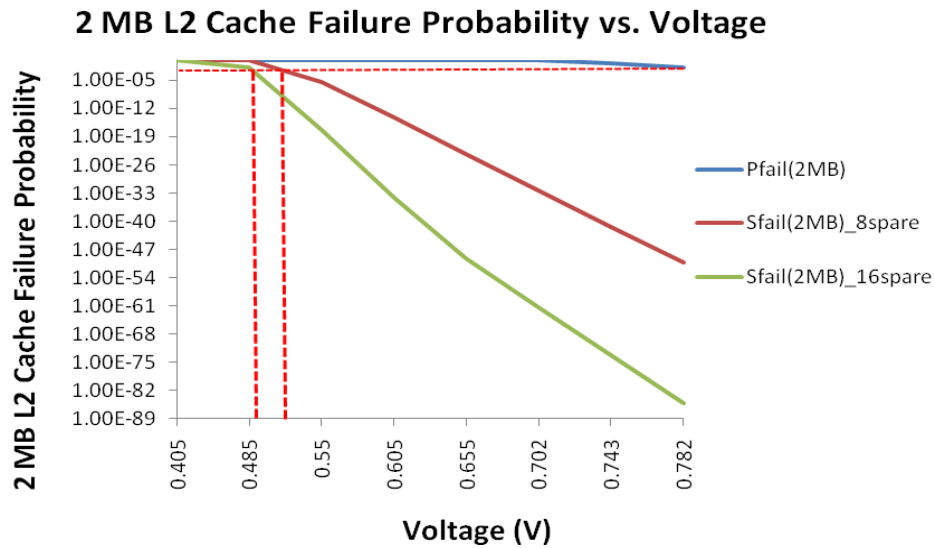


Fig 4.3: 2 MB cache failure probabilities at different voltages

Figure 4.3 describes the 2MB cache failure probability at different voltages. The notation 2MB_6T denotes a 2MB L2 unified cache with conventional design. 2MB_8Spare and 2MB_16Spare respectively denote a 2MB L2 cache with 8 spare words

and 16 spare words. The horizontal dotted line intersects the 2MB_8Spare graph at 530mV and 2MB_16Spare graph at 485mV. This line shows that our 2MB designs can operate reliably at 530mV and 485mV.

5. PERFORMANCE AND POWER EVALUATION

We evaluated our spare block cache architecture using six benchmarks and compared the results with a nominal 6T SRAM cache and with the Bit-Fix mechanism. In this section, we present the experimental results of our proposed cache design. Section 5.1 presents our simulation methodology. The performance impact of our design and the tradeoffs involved in choosing cache parameters at each cache level are described in Section 5.2. Section 5.3 compares our design with the 6T SRAM nominal cache and the Bit-Fix mechanism in terms of area overhead, power consumption, and energy efficiency.

5.1 Simulation Methodology

We used the cycle-accurate SimpleScalar simulator [22]. We simulate an out-of-order processor core with two-level cache memory hierarchy and a 20-stage pipeline. The fetch, decode, and issue width of the processor is eight instructions per cycle, and memory access width is sixteen bytes. The size of the Register Update Unit (RUU) is 256 instructions and issue width is eight instructions per cycle. In order to quantify the relative performance loss as a result of higher cache latency and smaller cache size, we also simulated a defect-free reliable baseline cache. The (unrealistic) baseline cache has no latency overhead or capacity loss at the nominal voltage. In order to simplify our discussion, we fix the low voltage at 485mV instead of using a different voltage for each mechanism. In our experiments, we simulated six different benchmarks. We use instructions per cycle (IPC) as the performance metric. We normalize the IPC of each

category to the baseline to show relative performance. We evaluated our cache in such a way that our cache size plus overhead is equal to the size of the conventional cache. Thus the effective size of our cache is less than the effective size of the conventional cache. Chapter 3.2.4 presents the critical path delay estimation by which we calculate the increase in cache access time in our design. The cache access time of our design is 1.3 times greater than the conventional cache. We evaluate our design in L1 and L2 caches separately and in a combined fashion (in both L1 and L2 cache). The L1 instruction cache is not affected in all our experiments. When we evaluate our design in only the L1 or L2 cache, we use the conventional design of the other cache (L2 or L1).

Table 5.1: Baseline processor configuration

Voltage Dependent Parameter		
	High Vol.	Low Vol.
Memory Access latency		
L1 Data	3 cycle	4 cycle
L2 Data	20 cycle	26 cycle
Cache ways		
L1 Data cache	8 ways	5 ways
L2 Unified cache	8 ways	7 ways
Voltage	782mV	485mV
Voltage independent Parameter		
Fetch/schedule/retire width	8/8/8 Byte/word	
Branch Prediction Type	Comb (bimodal & 2-level)	
Memory Access latency	400 cycles	
Memory Bus Width	16 bytes	
Number of Integer ALU	6	
Number of Floating Point ALU	6	
Pipeline Length	20	
Cache line size	64 bytes	
L1 Data Cache	32KB, 8 ways, 3 cycles	
L2 Unified cache	2MB, 8 ways, 20 cycles	
Number of instruction skipped	100 million	
Number of instruction executed	50 million & 100 million	

5.2 Performance

In this section, we compare the performance of our cache with a conventional cache at nominal voltage. Table 5.1 shows the baseline processor configuration. We normalized our data to baseline processor performance. As a base configuration, we use an 8-way set-associative 32KB L1 data cache and a 2MB unified L2 cache. The hit latency for the nominal L1 data and L2 unified cache is 3 cycles and 20 cycles, respectively. The critical path delay of our design is 1.3 times the delay of the nominal cache. The L1 data cache hit latency becomes 4 cycles, and the L2 unified cache hit latency becomes 26 cycles in our design. We have evaluated our design with six benchmarks: *gcc*, *jpeg*, *li*, *compress*, *vortex*, and *m88ksim* from the *SPEC95* benchmark suite. The area overhead is 30% and 16.24% respectively in the 32KB and 2MB cache using our spare block cache architecture. Therefore, we evaluate a 5-way L1 data cache and a 7-way L2 cache instead of 8-way for our design. This way reduction decreases the effective size of the 32KB L1 data cache to 22.4KB and the 2MB cache to 1.675MB. Figure 5.1, 5.2, 5.3, 5.4, and 5.5 describe performance of L1_spare, L2_spare, and L1L2_spare as we have implemented our design with L1 data cache only, L2 unified cache only, and both L1 and L2 cache, respectively.

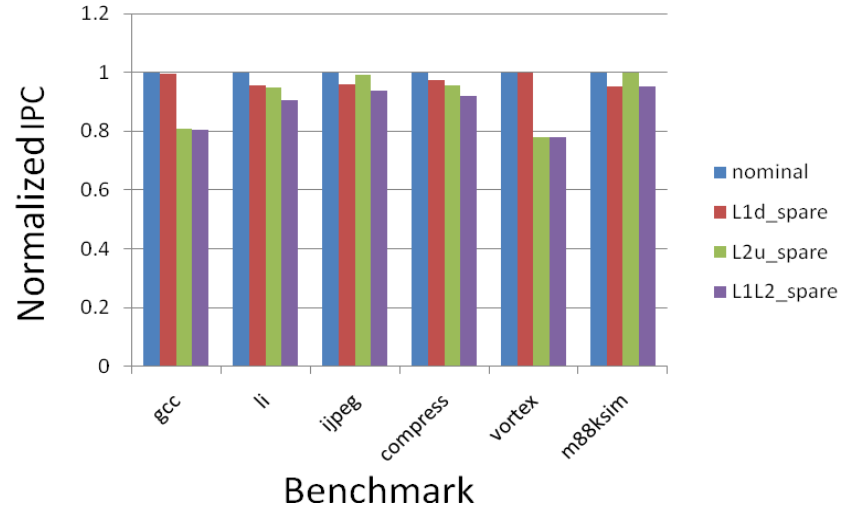


Fig 5.1: Normalized IPC comparison in different benchmarks

Figure 5.1 compares the instruction per cycle (IPC) of our cache with a conventional cache operating at nominal voltage. The figure shows that the maximum IPC decreases when the spare cache has been implemented in both caches. To quantify the performance of our cache we have normalized the IPC relative to the IPC of defect-free conventional cache. It shows that if our technique is implemented in the L1 data cache, the IPC decreases by 3%. However, the IPC decreases 10% when our design has been implemented in the L2 unified cache only. If we implement our design in both the L1 data and L2 unified cache, the IPC reduction is about 12%.

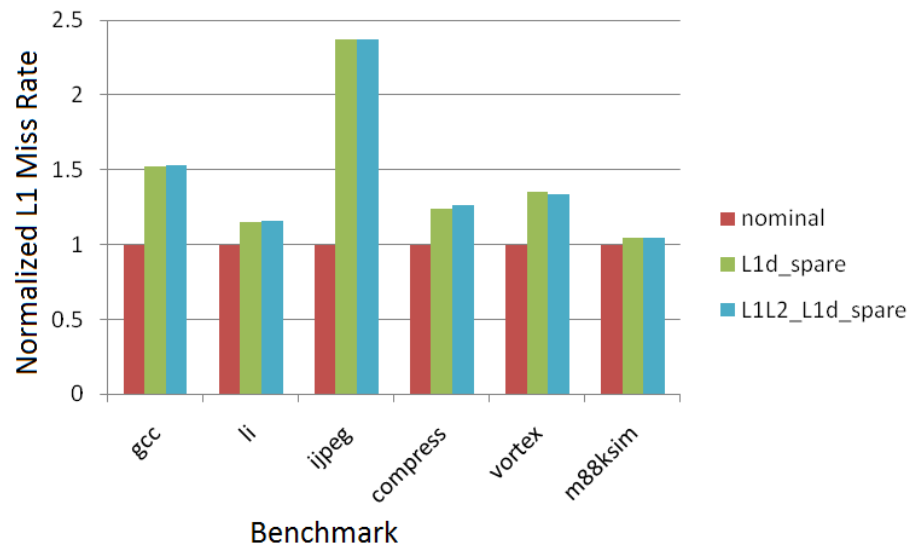


Fig 5.2: Normalized L1 data cache miss rate in different benchmarks

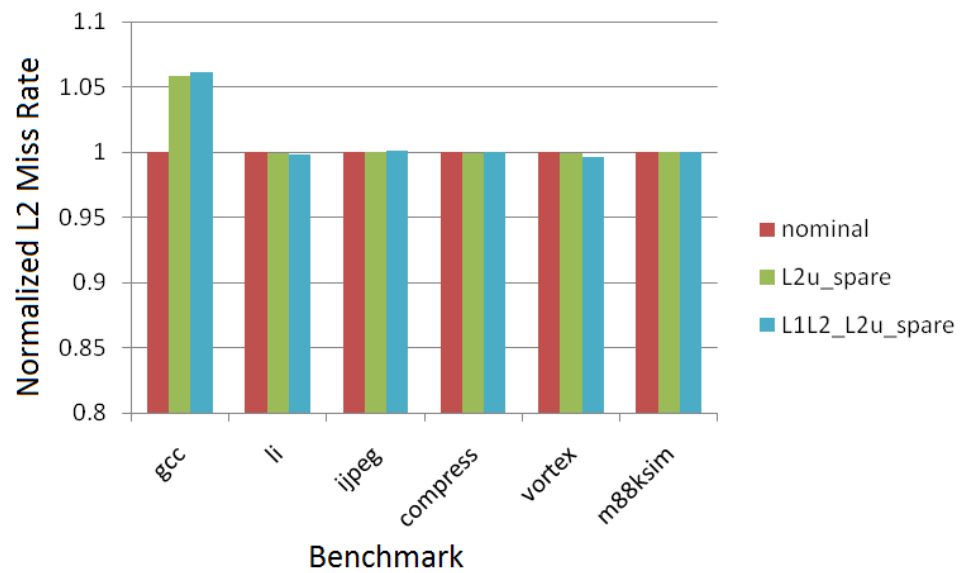


Fig 5.3: Normalized L2 unified cache miss rate in different benchmarks

Figure 5.2 and 5.3 show the normalized cache miss rate for the L1 data, and the L2 unified cache. The L2 cache accesses increase about 2X in the *jpeg* benchmark. However, the L2 cache miss rate does not increase much when the L2 cache size is reduced.

Figure 5.4 shows that the L1 data cache access increases by 5% in our design. This increase is mainly because the L1 data cache has been reduced by 30% in our design. The effective cache size reduces to 70% of the nominal cache, and the cache becomes 5-way set-associative cache instead of 8-way. The cache miss rate increases more in the L1 data cache. Similarly, the L2 unified cache becomes a 7-way cache in our design, but the cache miss rate is almost the same as the nominal cache.

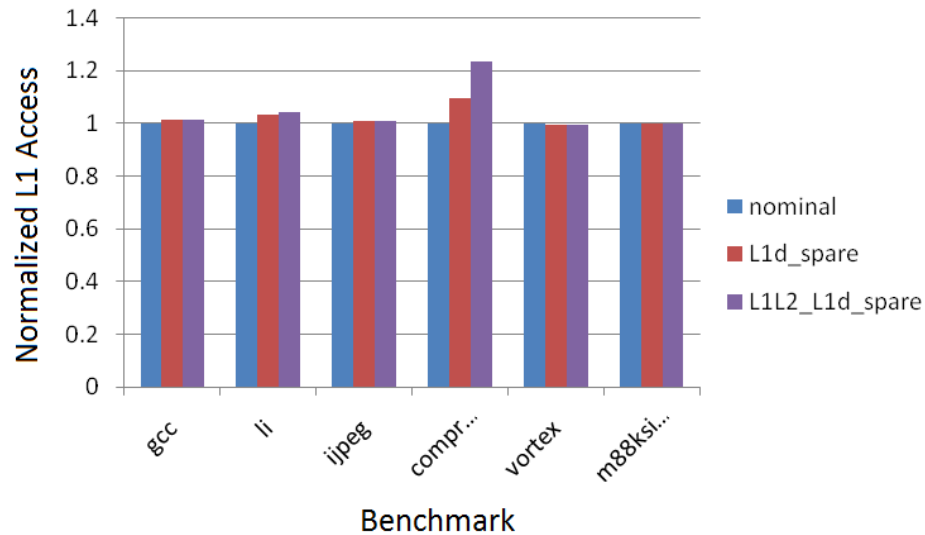


Fig 5.4: Normalized L1 data cache access in different benchmark

In Figure 5.4, the L1 cache accesses are almost equal to the conventional cache accesses in all benchmarks except *compress*. Most instructions of this benchmark are in a

small loop with a very few branches. Therefore, when the L1 cache size decreases, it needs to access cache more to write the data more often. We can see in Figure 5.1 that the L1 cache miss rate increases significantly in the *compress* benchmark. In Figure 5.4, we show that the L1 cache accesses increase when we implement our design in both of the L1, and L2 caches (L1L2_L1d_spare) compared to when our design implemented only in L1 data cache. In Figure 5.5 we show the normalized L2 cache accesses. We observe that the L2 cache accesses increase when L1 cache failure rate increases.

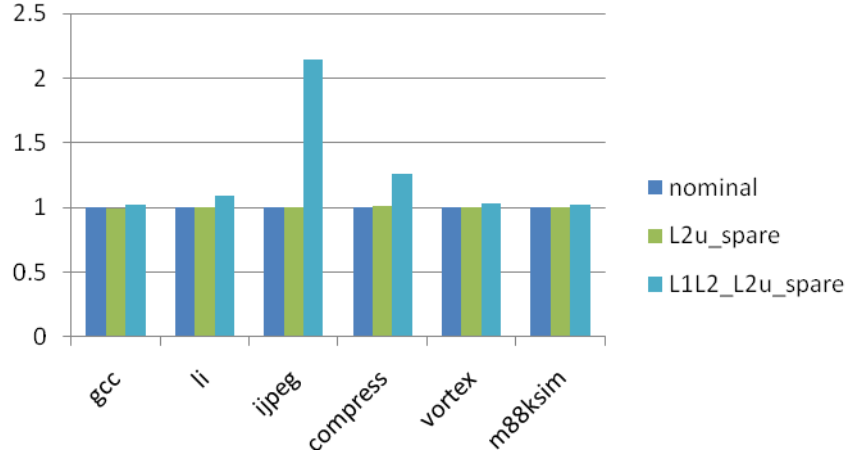


Fig 5.5: Normalized L2 unified cache access in different benchmark

Table 5.2: Conventional cache, Bit-Fix mechanism, and our proposed architecture in L1 and L2 cache

Scheme	Vccmin (mV)	Norm. cache area	Norm. power	IPC	Norm. EPI
Conventional	782	1	1	1	1
L1d_spare	485(L1), 782(L2)	1.005	0.988	0.972	0.987
L2u_spare	485(L2), 782(L1)	1.158	0.25	0.91	0.275
L1L2_spare	485(L1), 485(L2)	1.162	0.239	0.881	0.262
L1_bitfix	500(L1), 782(L2)	1.005	0.989	0.989	0.993
L2_bitfix	782(L1), 500(L2)	1.325	0.273	0.991	0.276
L1L2_bitfix	500(L1), 500(L2)	1.33	0.261	0.981	0.265

Table 5.2 presents the detailed comparison between our spare block cache at low voltage and a conventional cache at nominal voltage. We also implemented the Bit-Fix mechanism [1] and evaluated its performance with SimpleScalar. We normalize the area, power, IPC, and EPI results of each technique to the corresponding results of the conventional 6T cell-based cache. The lowest operating voltage (V_{ccmin}) of our cache is 485mV. However, the V_{ccmin} is 782mV in the conventional cache. In order to estimate the power consumption, we assume that the dynamic power scales quadratically with supply voltage (i.e. power \propto voltage²) and linearly with frequency [1, 3]. We also assume that static power scales with the cube of supply voltage [1]. We have also assumed that the frequency is linearly proportional to voltage (frequency \propto voltage). Therefore, we have estimated the dynamic power as the cube of supply voltage. As an example, in the

scheme L1d_spare, the L1 data cache is operating at 485mV, and the L2 cache is operating at 782mV. The dynamic power of the L1 data cache is proportional to 0.485^3 , and the L2 cache is proportional to 0.782^3 . As the size of the L2 cache is 64 times larger than the L1 data cache, the total dynamic power for the L2 cache is 64 times that of L1 cache. Therefore, we calculated total power as a summation of power for both caches operating at their own voltage. We normalize the cache power with the nominal L1 and L2 cache power operating at 782mV. We assume the leakage is an exponential function of operating voltage (V_{cc}) [1, 3]. We calculated the IPC as the geometric mean of all the benchmark's IPC. We also assume that energy scales with the cube of supply voltage (energy \propto voltage³) operating at a fixed amount of time. We use the number of cycles to execute 50 million instructions to estimate time. We have found the operating frequency of the processor at different voltages measured by Wilkerson, et al. [1], and Chishti, et al. [3]. Thus, the energy is measured for the entire execution time. Energy per Instruction (EPI) is the average energy required to execute each instruction. Table 5.2 shows compared to the processor with the 6T cell-based cache, our design reduces the power consumption by 1% when it is implemented in the L1 data cache only, by 75% when implemented in the L2 unified cache only, and by 76% when implemented in both caches. Table 5.2 shows that the increase in cache area is lower in our design compared to the Bit-Fix mechanism. However, the cache access latency increases 1.3 times in our design compared to the conventional cache and Bit-Fix mechanism. Therefore, the total cycle requirement to execute 50million instruction is higher in our design, and the cache access latency of our design reduces the IPC.

6. CONCLUSION

Reducing the supply voltage is the most effective way to reduce power consumption in a microprocessor. However, reducing voltage is limited by large memory structures, such as caches, where many cells can fail at low voltage operation. As a result, voltage scaling is limited by a minimum voltage (V_{ccmin}), below which some components of the processor may not operate reliably. We have proposed a cache architecture that can operate reliably at low voltages. Our evaluation shows that, at 485mV, our designed cache operates with an equivalent bit cell failure probability to a conventional cache operating at 782mV.

We designed a novel spare block cache architecture that is power-efficient and requires less overhead compared to prior techniques. At low voltage, we skip the faulty word and move the data to an adjacent fault-free word position. We used spare words to store the overflow of words. The proposed cache can tolerate 16 faulty words in a cache set. The designed cache controller controls the cache word access. We have designed shifters that are used in the cache access controller to move the data to an adjacent fault-free word position. The pMOS gates are used as a switch between the bit lines and SRAM bits. Our developed cache access controller is independent of cache size. Our optimized controller increases cache access latency by 30% compared to conventional cache access latency.

We proposed a robust design to tolerate failures at word granularity. Our designed 3rd level shifter generates the pMOS gate input for all of the cache words. We have designed the 1st and 2nd level shifter to speed up generating the pMOS gate input. We used domino gates in designing shifters to decrease latency and overhead. Due to the added spare word blocks, the cache set failure probability improves at low voltage. We proposed a fault model to estimate the cache failure probability of our cache at low voltage. Our fault model shows that our proposed cache can operate at cache failure probability of 2.18×10^{-04} for 32KB L1 data cache and 1.39×10^{-02} for 2MB L2 unified cache at 485mV. This failure probability is better than a conventional cache operating at 782mV. We implemented our cache in the SimpleScalar simulator. We have compared our design to a baseline using 8-way set associative 32KB L1 cache and 2MB L2 cache, and to the Bit-Fix mechanism.

Our design increases the overhead relative to the baseline L1 and L2 caches by about 30.06% and 16.24%, respectively. Our estimation of critical path delay indicates that the critical delay of our proposed cache is 24.22FO4, which is 1.3 times the critical delay of the conventional cache (17.4FO4). Therefore, we estimate the cache access time is 1.3X larger than the conventional cache access time. The power consumption is about 76% less than the conventional cache. The IPC decreases by 3%-12% for our cache versus a conventional cache.

The efficient execution of multithreaded programs on future multi-cores requires fast cache operation at ultra low voltage. The current work can be extended to caches in

multi threaded processors. In multi-threaded processor the cache access latency may have a more significant effect.

REFERENCES

- [1] C. Wilkerson, H. Gao, A. Alameldeen, Z. Chishti, M. Khellah and S. Lu, "Trading off Cache Capacity for Reliability to Enable Low Voltage Operation," *International Symposium on Computer Architecture*, pp.203-214, June 2008.
- [2] D. Roberts, N. Kim, and T. Mudge, "On-chip cache device scaling limits and effective fault repair techniques in future nanoscale technology," *10th Euromicro Conference on Digital System Design Architectures (DSD), Methods and Tools*, pp. 570–578, August 2007.
- [3] Z. Chishti, A. Alameldeen, C. Wilkerson, W. Wu and S. Lu, "Improving Cache Lifetime Reliability at Ultra-low Voltages," *International Symposium on Micro-architecture (MICRO)*, pp. 89-99, 2009.
- [4] L. Li, V. Degalahal, N. Vijaykrishnan, M. Kandemir, M. Irwin, "Soft Error and Energy Consumption Interactions: A Data Cache Perspective," *IEEE Transactions on Dependable and Secure Computing*, vol. 3, no. 4, pp. 353-364, October 2006.
- [5] A. Ansari, S. Gupta, S. Feng, and S. Mahlke, "Zereh Cache: Armoring Cache Architectures in High Defect Density Technologies," *International Symposium on Micro-architecture (MICRO)*, pp. 100-110, December 2009.
- [6] J. Kulkarni, K. Kim, and K. Roy, "A 160 mV Robust Schmitt Trigger Based Subthreshold SRAM," *IEEE Journal of Solid State Circuits*, vol. 42, no. 10, pp. 2303-2313, October 2007.
- [7] B. Calhoun, and A. Chandrakasan, "A 256-kb 65-nm Sub-threshold SRAM Design for Ultra-Low-Voltage Operation," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 3, pp. 680-688, March 2007.
- [8] A. Alameldeen, Z. Chishti, C. Wilkersonson, and W. Wu, "Adaptive Cache Design to Enable Reliable Low-Voltage Operation", *IEEE Trans. On Computers*, vol. 60, no. 1, pp. 50-63, 2011.
- [9] J. Abella, J. Carretero, P. Chaparro, X. Vera, A. González, "Low V_{ccmin} Fault-Tolerant Cache with Highly Predictable Performance," *International Symposium on Micro-architecture (MICRO)*, pp. 111-121, 2009.

- [10] D. Yoon, and M. Erez, "Flexible Cache Error Protection using an ECC FIFO," *Conference on High Performance Networking and Computing*, no. 49, pp. 1-12, November 2009.
- [11] H. Y. Hsiao, "Orthogonal Latin Square Codes," *IBM Journal of Research and Development*, vol. 14, no. 4, pp. 390-394, July 1970.
- [12] N. Sadler and D. Sorin "Choosing an Error Protection Scheme for a Microprocessor's L1 Data Cache," *In Proc. the Int'l Conf. Computer Design (ICCD)*, pp. 449-505, October 2006.
- [13] S. Kim, "Area-Efficient Error Protection for Caches," *In Proc. of the Conference on Design Automation and Test in Europe (DATE)*, pp. 1-6, March 2006.
- [14] D. Nassimi and S. Sahni, "A self routing benes network," *International Symposium on Computer Architecture*, pp. 190 – 195, 1980.
- [15] D. Yoon, M. Erez, "Memory Mapped ECC: Low-Cost Error Protection for Last Level Caches," *International Symposium on Computer Architecture*, pp. 116-127, 2009.
- [16] X. Vera, J. Abella, A. Gonz'alez, And R. Ronen, "Reducing Soft Error Vulnerability of Data Caches," *IEEE Transactions on Dependable and Secure Computing*, vol. 3, no. 4, pp. 353-364, October 2006.
- [17] V. Degalahal, L. Li, V. Narayanan, M. Kandemir, and M.J. Irwin, "Soft Errors Issues in Low-Power Caches," *IEEE Transactions on VLSI Systems*, vol. 13, no. 10, pp. 1157 – 1166, October 2005.
- [18] P. Jung-Wook, P. Gi-Ho, P. Sung-Bae, and K. Shin-Dug, "Power-aware deterministic block allocation for low-power way selective cache structure", *In Proc. of the IEEE International Conference on Computer Design (ICCD)*, pp. 42-47, 2004.
- [19] A. Badulescu, and A. Veidenbaum, "Power Efficient Instruction Cache for Wide-issue Processors", *In Proc. of the Innovative Architecture for Future Generation High-Performance Processors and Systems (IWIA)*, pp. 1-12, 2001.
- [20] S. Unsal, I. Koren, C. Mani Krishna, and C. Moritz, "The Minimax Cache: An Energy-Efficient Framework for Media Processors", *International Symposium on High-Performance Computer Architecture - HPCA*, pp. 131-140, 2002.
- [21] N. Weste and D. Harris, "CMOS VLSI Design: A Circuits and Systems Perspective", 3rd edition, Addison-Wesley, 2005.
- [22] D. Burger, and T. Austin, "The SimpleScalar tool set, version 2.0", *ACM SIGARCH Computer Architecture News*, Volume 25 Issue 3, pp. 13-25, June 1997.