Copyright

by

Rudrajit Datta

2011

The Dissertation Committee for Rudrajit Datta certifies that this is the approved version of the following dissertation:

Adaptable and Enhanced Error Correction Codes for Efficient Error and Defect Tolerance in Memories

Committee:

Nur A. Touba, Supervisor

David Z. Pan

Tony Ambler

Andreas Gerstlauer

Abhijit Jas

Adaptable and Enhanced Error Correction Codes for Efficient Error and Defect Tolerance in Memories

by

Rudrajit Datta, B.Tech.; M.S.E.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

December 2011

Dedicated to my family

Acknowledgements

I begin by thanking my advisor Dr. Nur Touba. Over the course of my PhD Prof. Touba has been a pillar of support, from starting me off in the right direction to helping me out with valuable suggestions whenever I needed them. It was always a joy to discuss new ideas with him, not least because he could so easily spot the loopholes that I, in my excitement at having discovered something new, would miss. He also had the innate ability to motivate without ever pressurizing and by firmly emphasizing quality over quantity, he laid down a benchmark which I intend to uphold in the future. The years I worked with him have been some of the best times of my life as a student and I can only thank him for this enriching experience.

I would also like to thank my other committee members Dr. David Pan, Dr. Tony Ambler, Dr. Andreas Gerstlauer and Dr. Abhijit Jas, for kindly agreeing to be part of my defense committee and for making vital suggestions without which this dissertation would never have been complete.

Joon-Sung Yang has been the most wonderful colleague, always there to entertain questions, not just pertaining to research but other interesting things and helping make graduate life a more wholesome experience. I tremendously enjoyed the long discussions we have had, which often resulted in productive findings. His frank comments and keen observations have only improved the quality of my work.

Shreepriya Das is a great flatmate. Extremely hardworking and dedicated – qualities I wish to emulate, he is a very entertaining man and totally hardcore.

Sushovan De is the person I turned to for help every so often and in the end came away with better coding practices and an improved sense of humor.

Varada Bal has been a tower of strength, always there for me through everything and a calm, stabilizing presence even in the most tumultuous of times, of which there were a few.

This dissertation would never have been possible without the support of my parents, Abhijit and Sunipa Datta. The desire to succeed while never compromising on honesty and integrity that they inculcated in me at a very early age has served me well throughout my life. They always emphasized on perfection, something I hope I have picked up, and their insistence that I learn to write and speak proper English has only made my life easier. Without their unfaltering support and strong conviction in my abilities, I am certain I would not have made it to where I am today. For all of this and for being the greatest source of inspiration, I want to thank them from the bottom of my heart.

I would like to thank Chris Wilkerson and Shih-Lien Lu of Intel Corporation for introducing us to the problem of low-voltage caches and for many helpful technical discussions and the National Science Foundation for helping support my research through generous grants.

I want to thank my high school teachers, specifically Mallar Roy, Partha Pratim Roy and Satyajit Banerjee, for raising the bar when it came to achieving success and for setting the example of good, sound teaching. I want to thank all my professors at the Indian Institute of Technology, Kharagpur and at the University of Texas at Austin for more than just the courses that they taught me. I would be doing a great disservice if I do not mention my undergraduate advisor, Prof Amit Patra, and the support he provided helping me get into the graduate school of my dreams. All my friends here at Austin, Nikhil Cheruku Reddy and Shayak Banerjee (until he left for New York) more so than others, without whom life would not have been anywhere as fun during the last five

years, I thank you. Melissa Campos and Debi Prather at the Computer Engineering Research Center deserve a special mention for taking care of so many things so seamlessly and Melanie Gulick for having an answer to all the questions I ever posed her and for always being so patient.

Adaptable and Enhanced Error Correction Codes for Efficient Error and Defect Tolerance in Memories

Publication No._____

Rudrajit Datta, Ph.D.

The University of Texas at Austin, 2011

Supervisor: Nur A. Touba

Ongoing technology improvements and feature size reduction have led to an increase in manufacturing-induced parameter variations. These variations affect various memory cell circuits, making them unreliable at low voltages. Memories are very dense structures that are especially susceptible to defects, and more so at lower voltages. Transient errors due to radiation, power supply noise, etc., can also cause bit-flips in a memory. To protect the data integrity of the memory, an error correcting code (ECC) is generally employed. Present ECC, however, is either single error correcting or corrects multiple errors at the cost of high redundancy or longer correction time.

This research addresses the problem of memory reliability under adverse conditions. The goal is to achieve a desired reliability at reduced redundancy while also keeping in check the correction time. Several methods are proposed here including one that makes use of leftover spare columns/rows in memory arrays [Datta 09] and another one that uses memory characterization tests to customize ECC on a chip by chip basis [Datta 10]. The former demonstrates how reusing spare columns leftover from the memory repair process can help increase code reliability while keeping hardware overhead to a minimum. In the latter case customizing ECCs on a chip by chip basis shows considerable reduction in check bit overhead, at the same time providing a desired level of protection for low voltage operations. The customization is done with help from a defect map generated at manufacturing time, which helps identify potentially vulnerable cells at low voltage.

An ECC based solution for tackling the wear out problem of phase change memories (PCM) has also been presented here. To handle the problem of gradual wear out and hence increasing defect rates in PCM systems an adaptive error correction scheme is proposed [Datta 11a]. The adaptive scheme, implemented alongside the operating system seeks to increase PCM lifetime by manifold times. Finally the work on memory ECC is extended by proposing a fast burst error correcting code with minimal overhead for handling scenarios where multi-bit failures are common [Datta 11b]. The twofold goal of this work – design a low-cost code capable of handling multi bit errors affecting adjacent cells, and fast multi bit error correction – is achieved by modifying conventional Orthogonal Latin Square codes into burst error codes.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Memories are very dense structures that are especially susceptible to defects. Transient errors due to radiation, power supply noise, etc., can cause bit-flips in a memory. To protect the data integrity of the memory during runtime operations, error correcting codes (ECC) of various class and strength is generally employed

A soft error occurs when a radiation event causes enough of a charge disturbance to reverse or flip the data state of a memory cell, register, latch, or flip-flop. The error is "soft" because the circuit/device itself is not permanently damaged by the radiation—if new data are written to the bit, the device will store it correctly. Recently, research has shown that commercial static random access memories (SRAMs) are now so small and sufficiently sensitive that single event upsets (SEUs) may be induced from the electronic stopping of a proton [Rodebell 07], [Heidel 08], [Sierawski 09], [Lawrence 09]. This sensitivity appears near the 65 nm technology node as the critical charge to upset a cell is on the order of 1 fC; merely 6,000 electrons are required to cause a change in data state. The lower critical charge required to cause a bit-flip has more pronounced effects on space applications compared to terrestrial ones [Sierawski 11].

Also low voltage operation can lead to greater number of failures, arising due to more pronounced effect of process variations. Voltage scaling, which is one of the most effective ways to reduce power consumption can lead to unreliable operations at lower voltages. Voltage scaling is limited by a minimum value referred to as V_{ccmin} beyond which circuits may not function reliably [Taur 98]. Voltage scaling beyond V_{ccmin} gives rise to reliability issues, most notably for the memory sub-systems. In order for V_{ccmin} to be reduced to enable ultra-low power modes in microprocessors and other circuits, some means for handling high memory bit failure rates is required.

While soft errors in cache or main memory are primarily corrected using error correction codes, some of the newer generations of memory have fundamentally different fault models. In this dissertation we have looked at Phase Change Memory (PCM) and how the reliability of a PCM based system can be improved. The primary fault model of PCM is based on wear out caused by memory writes. Permanent faults present a very different kind of challenge compared to soft errors.

1.1 ERROR CORRECTING CODES

. The most common error correcting code that is used is single-error-correcting, double-error-detecting (SEC-DED) codes [Hamming 50], [Hsiao 70]. These codes can correct single bit errors in any word of the memory and can detect double bit errors, have moderate redundancy in terms of check bits and are relatively easy to decode. Decoding and correction are done via syndrome method which takes single cycle. A special class of SEC-DED codes known as Hsiao codes [Hsiao 70] was proposed to improve the speed, cost, and reliability of the decoding logic. However some situations demand more stringent reliability requirements, thus necessitating error correction stronger than normal SEC-DED.

Stronger error correcting codes includes single byte-error-correcting, double-byte-error-detecting (SBC-DBD) codes [Berlekamp 68], [Reed 60], [Wolf 69], [Bossen 70] [Chen 96]. These codes perform at a higher order Galois field and consequently the encoding and decoding are more complex. Moreover, they require more check bits thereby increasing the size of the memory. There are also the double-error-correcting triple-error-detecting (DEC-TED) codes, which come at the cost of much larger overhead in terms of both the check bits and more complex hardware to implement the error correction and detection [Lin 83], [Berlekamp 68], [Lala 78]. The general drawbacks

with these methods are latency and speed. Most of these codes require several cycles to correct the first error unlike the SEC-DED codes. Moreover, the encoding and decoding are much more complex and require several table lookups for multiplication in higher order fields. However in spite of their low check bits overhead and single cycle decoding, SEC-DED codes are not able to provide requisite reliability under certain conditions.

As an alternative for fast, multiple errors correcting code, we have looked at Orthogonal Latin Square (OLS) codes in this dissertation. OLS codes have the ability to correct multiple errors in a single cycle which make them very attractive in systems with high error rates. In our work we have tried to make OLS codes better suited for implementation by optimizing their redundancy through different techniques.

1.2 Adaptable Error Correction Codes

The most crucial aspect of increasing memory reliability is to do so without increasing check bit redundancy too substantially so as to effect normal operation. One general approach that has been proposed in [Wilkerson 08], [Chisti 09] is to trade off cache capacity to store extra check bit for reliable low voltage operation. The goal is to have a stronger but more redundant code to mitigate extra error that may occur in low voltage mode. But this can prove to be over compensating at times.

One limitation of normal SEC-DED codes is that they can only detect, but not correct, double-bit errors whereas studies have shown that 1-5% of single event upsets (SEUs) can cause multiple-bit errors (MBUs) [Satoh 00], [Makihara 00], [Kawakami 04]. Also SEC-DED tends to miscorrect triple-bit errors with an alarmingly high probability of around 60% or more.

Some of these problems have been addressed by investigating different techniques which increases memory reliability on a need basis. However in order to adapt strength of

ECC based on circumstances, additional information is often required. This information can be obtained through post-silicon characterization tests or by involving the operation system.

1.3 ORGANIZATION

Over the course of the next few chapters, we propose four different ideas, all focusing on the central issue of making efficient usage of error correction codes. The applications range from caches to main memory to PCM based systems.

In chapter 2 we show how unused spare columns from the memory array can be used to improve reliability of the existing ECC. Spare columns are often included in memories for the purpose of allowing for repair in the presence of defective cells or bit lines. But in many cases, the repair process will not use all spare columns. We propose an extremely low cost method to exploit these unused spare columns to improve the reliability of the memory by enhancing its existing error correcting code [Datta 09]. Memories are generally protected with single-error correcting, double-error-detecting (SEC-DED) codes using the minimum number of check bits. In the proposed method, unused spare columns are exploited to store additional check bits which can be used to reduce the mis-correction probability for triple errors in SEC-DED codes or non-adjacent double errors in single adjacent error correcting codes (SEC-DAEC) codes. Our proposed has mimimal hardware overhead and the worst case performance is limited by the efficiency of the existing code.

Chapter 3 extends memory error correction to handle multiple soft errors. We propose a scheme by which an Orthogonal Latin Square code can be modified to correct burst-errors of specific length [Datta 11b]. The method discussed in this work models it as a graph coloring problem where the goal is to resolve conflicts in the existing OLS

code in order for it to correct burst-errors. Conflicts are resolved by reordering and/or reorganizing existing parity relations by inclusion of extra check bits. The graph coloring approach tries to minimize the number of additional check bits required. The final OLS code after reordering and/or reorganizing would be capable of correcting burst-errors of specific length in addition to its original error correction capabilities.

Chapter 4 introduces a new application for efficient error correcting codes – low voltage caches. In this chapter we talk about the idea of implementing a general multi-bit error correcting code based on Orthogonal Latin Square codes in on-chip hardware, but then selectively, on a chip-by-chip basis, using only a subset of the code's check bits (subset of the rows in its H-matrix) depending on the defect map for a particular chip [Datta 10]. The defect map is obtained from a memory characterization test which identifies which cells are defective or marginal. The idea proposed here is that if a general *t*-bit error correcting code is implemented in hardware and requires $c_{full} = n\text{-}k$ check bits for k information bits, then once the defect map is known, the defective cells become erasures w.r.t. the ECC. This fact can be used to select only a subset of the $n\text{-}k$ rows in the H-matrix which are sufficient to provide the desired error detection/correction capability in the presences of the known erasures. By selectively reducing the number of rows in the H-matrix, the number of check bits that are actually stored and used, c_{used}, can be restricted and the corresponding unused ECC hardware disabled. This reduces the check bit storage requirements and hence frees up more of the cache for storing data and improving performance. This strategy is applied to the problem of providing reliable cache operation in ultra-low voltage modes, and results indicate that with the proposed post manufacturing ECC customization, a fraction of the number of check bits are required compared to using a full OLS code for handling a particular defect rate.

In chapter 5 we investigate the application of ECCs to the reliability problem posed by phase change memories, one of the possible candidates for future generation of memory. We explore the concept of an adaptive multi-bit error correcting code for phase change memories that provides a manifold increase in the lifetime of phase change memories thereby making them a more viable alternative for DRAM main memory [Datta 11a]. A novel aspect of the proposed approach is that the error correction code (ECC) is adapted over time as the number of failed cells in the phase change memory accumulates. The operating system (OS) monitors the number of errors corrected on a memory line, and when the number of errors on a line begins to exceed the strength of the ECC present, the ECC strength is adaptively increased. As this happens, the performance of the memory system gracefully degrades because more storage is taken up by check bits rather than data bits thereby reducing the effective size of a cache line since less data can be brought to the cache on each read operation to the PCM main memory. Experimental results show that the lifetime of a phase change memory can be significantly extended while keeping the fraction of data to check bits as high as possible at each stage in the lifetime of the phase change memory.

Finally chapter 7 summarizes the contribution of the dissertation. We conclude by providing some direction for future work.

Chapter 2: Exploiting Unused Spare Columns to Improve Memory ECC

2.1. INTRODUCTION

Memories are very dense structures that are especially susceptible to defects. In most cases, memories take up a very large percentage of a chip's area. In order to improve yield, spare rows and columns are often included in a memory to allow for repairing the memory [Kim 98], [Zorian 03]. Defective cells, bit lines, or word lines, can be repaired by replacing the defective rows or columns with the spares. While the worst-case most defective memories on the tail end of the statistical curve may use all of the spare resources, most memories will have unused spare resources after the repair process. This chapter describes a methodology to exploit these unused resources, when available, to improve the reliability of the memory by enhancing its existing error coding.

Transient errors due to radiation, power supply noise, etc., can cause bit-flips in a memory. To protect the data integrity of the memory, an error correcting code (ECC) is generally employed. The most common error correcting code that is used is single-error-correcting, double-error detecting (SEC-DED) codes [Hamming 50], [Hsiao 70]. These codes can correct single bit errors in any word of the memory and can detect double bit errors. These codes require storing additional check bits in the memory. For a memory with 32 bit data words, 7 check bits are required. So the memory would need 39 columns for each logical word plus any additional spare columns that are included for repair. In some cases, check bits are used along with spare rows and columns to get combined fault-tolerance. In [Stapper 92], interleaved words with redundant word lines and bit lines are used in addition to the check bits on each word.

Some previous work has been done to enhance the reliability of memories without increasing the size of the memory. One limitation of SEC-DED codes is that if a triple-bit error occurs, it may not be detected, but rather it may be mis-corrected as if it were a single bit error [Hsiao 70]. The probability of mis-correction for triple bit errors for conventional SEC-DED codes for 32 bit data words is around 60% or more. A code with *r* check bits can protect up to *2r-1-r* data bits. Most memories will not have exactly that number of data bits, and hence *shortened codes* must be used. For shortened codes, there is a degree of freedom in selecting the set of columns in the parity-check matrix (*H*-matrix). In [Richter 08], a search procedure was described for selecting the columns in an *H*-matrix for a shortened code that minimizes the mis-correction probability for triple bit errors. For example, they found an SEC-DED code for 32 bit data words which has a triple error mis-correction probability of 47%. This increases the reliability of the memory at no additional cost other than a few extra XOR gates in the checker.

Another limitation of SEC-DED codes is that they can only detect, but not correct, double-bit errors. Studies have shown that 1-5% of single event upsets (SEUs) can cause multiple-bit errors (MBUs) [Satoh 00], [Makihara00], [Kawakami 04]. Most MBUs will affect nearby cells. In [Dutta 07], it was shown that by carefully selecting and ordering the columns in the *H*-matrix for an SEC-DED code, it is possible to correct all adjacent double-bit errors in addition to correcting all single bit errors thereby creating an SEC-DAEC code. Since the most likely double-bit errors will be adjacent, this is very useful. The limitation of SEC-DAEC codes is that they may not detect all non-adjacent double-bit errors. The SEC-DAEC code reported in [Dutta 07] for 32-bit data words has a 51% double-bit mis-correction probability. [Richter 08], a better code was found which has a 37% double-error mis-correction probability.

In this work, we propose an extremely low-cost scheme that exploits unused spare columns to store additional check bits which can be used to significantly reduce the mis-correction probability for triple-errors in SEC-DED codes or non-adjacent double-errors in SECDAEC codes. The proposed scheme does not require that any additional columns be added to the memory itself, but rather it simply exploits unused spare columns left over after memory repair. Implementing the proposed scheme requires only adding a few additional XOR gates to the check bit generation logic and providing a simple mechanism to disregard the additional check bits corresponding to the spare columns that become unavailable due to their being used for repair.

The sections of this chapter are organized as follows. Sec. 2.2 provides an overview of linear block codes and properties of SECDED and SEC-DAEC codes. Sec. 2.3 describes the proposed scheme for using extra check bits to reduce mis-correction. Sec. 2.4 describes the hardware implementation for the proposed scheme. Experimental results are shown in Sec. 2.5, and Sec. 2.6 is a conclusion.

2.2 LINEAR BLOCK CODES

Conventional SEC-DED codes [Hamming 50], [Hsiao 70] are systematic linear block codes [Peterson 72], [Pradhan 96]. In a *(n,k)* linear block code, *k* data bits are encoded by *n*-bit codewords. The number of check bits is *r=(n-k)*. The ($r \times n$) parity-check matrix (*H*-matrix) completely defines the code. *C* is a codeword of the code if and only if:

$$H \cdot C^T = 0$$

where C^T is the transpose of the codeword *C*. Let each element in the error vector *E* be a 1 if the corresponding bit is in error and a 0 if the bit is error-free, then an erroneous message can be represented as $V_{error} = V \square E$. The syndrome, *S*, is defined as:

$$S = H \cdot V_{error} = H \cdot (V \square E) = H \cdot V \square\ H \cdot E = H \cdot E$$

If there is no error (i.e., $E=0$), then the syndrome is all zero (i.e., $S=0$). If the syndrome is non-zero, then an error is detected. Let h_i represent the i-th column of the -matrix. If the i-th bit has a single error, then the error vector, E, is all zero with only the i-th bit being a one. The syndrome, which is equal to the product of H and E, will be equal to hi. For an SEC Hamming code, each column vector in the H-matrix is non-zero and distinct [Hamming 50]. This ensures that the syndrome for any single bit error will result in a unique syndrome. By decoding the syndrome, it is possible to determine which bit the error is in and flip the value of that bit to correct the error.

For a double-bit error, the syndrome is equal to the XOR of two columns of the H-matrix. If the XOR of any two columns is equal to the syndrome for any single bit error (i.e., equal to any column in the H-matrix), then the double-bit error syndrome would alias with the single-bit error syndrome. The correction logic would mis-correct the double-bit error thereby missing the error. To avoid this, it was shown in [Hsiao 70], that if every column of the H-matrix has an odd number of 1's and is distinct, then the code will be SEC-DED. The reason is that the XOR of any two columns with an odd number of 1's will produce a syndrome with an even number of 1's and hence is guaranteed to be different from any single column. This means that the syndromes for double-bit errors will always be different from the syndromes for single-bit errors, so the code will always detect double-bit errors and not miscorrect them. Hsiao codes are also called *odd-weight column codes*. Note that many double bit errors have the same syndrome, so it is generally not possible to correct double-bit errors since their syndromes cannot be distinguished.

For triple-bit errors, the syndrome is formed from three columns being XORed together. If the syndrome matches one of columns of the H-matrix, then it will be mis-

corrected as a single-bit error. The number of possible triple-bit errors is C_3^n , and the fraction of those that match columns of the H-matrix is the mis-correction probability for triple-errors. For most conventional SEC-DED codes, it is in excess of 50%.

In [Dutta 07], the *H*-matrix is constructed using odd-weight columns where the columns are carefully ordered so that adjacent columns when XORed together give a syndrome that is not equal to the syndrome for any single-bit error or the syndrome for any other adjacent double-bit error. The number of possible adjacent double-bit errors is equal to *n-1* and the number of single bit errors is *n*, so the combined set of *2n-1* syndromes must all be distinct from each other. This permits correction of both single-bit errors and adjacent double bit errors (i.e., SEC-DAEC). However, non-adjacent double-bit errors may match one of the *(n-1)* syndromes of the adjacent double-bit errors and hence may result in mis-correction.

2.3 PROPOSED SCHEME

The proposed scheme in [Datta 09] involves exploiting unused spare columns in the memory to store additional check bits. These additional check bits add extra rows to the *H*-matrix and increases the dimension of the syndrome. This makes it easier to distinguish syndromes thereby reducing the chance of mis-correction as well as reducing the chance of a multi-bit error's syndrome aliasing with the error-free all-zero syndrome and not being detected at all.

Note that if all the spare columns are used for repair, then for some chips, it may not be possible to store any additional check bits. Thus, the *H*-matrix that is selected should be such that if no additional check bits are available, it still retains the SEC-DED property. The easiest way to ensure this is to start with an SEC-DED code, and then incrementally add the extra rows to it.

The rows are added one at a time in a greedy fashion so that if only one spare is available after repair, then the maximum benefit for that one row is achieved. Consider the example in Fig. 2.1 which is a (7,3) SEC-DED Hsiao code. It has C_3^7 = 35 possible 3-bit errors, and 28 of those will result in mis-correction. In Fig. 2.2, an extra check bit is added to the *H*-matrix from Fig. 2.1. This results in an additional row (the bottom-most one) and an additional column (the right-most one). The last 5 columns in Fig. 2.2 correspond to check bits and hence form an identity matrix. The left three columns correspond to message bits. The bottom-most bit in the first three columns may be set to any value so as to minimize the mis-correction probability. In Fig. 2.2, the bottom-most bit in the second column is set to 1 and the others to 0. Now only 12 of the 35 possible triple-bit errors will result in mis-correction.

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 2.1: Example of (7, 3) Hsiao Code

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 2.2: Adding an extra row to the example in Fig. 2.1

Starting from an SEC-DED code, the proposed scheme adds rows one at a time. The columns corresponding to check bits form an identity matrix, so the degree of freedom is in selecting the 1's and 0's in the row for the columns corresponding to message bits. There are few different strategies that can be used. If the number of message bits is less than say 30, it is possible to do an exhaustive search. Each possible

combination of 1's and 0's for the row can be tried and the mis-correction probability computed. The one that minimizes the mis-correction probability is then selected. As the number of message bits gets larger, however, then an exhaustive search is no longer possible.

For larger codes, an alternative to an exhaustive search would be to do a random search and simply keep the best code found. The number of triple-errors is equal to Cn 3 which is manageable for n up to hundreds. It is feasible to enumerate all the triple-errors and compute the exact mis-correction probability for each candidate row. From our experiments, this gave quite good solutions. When comparing the results for an exhaustive search with those of a random search, there was not a significant difference in the results as can be seen in the experimental data in Sec. 1.5.

The procedure is the same for SEC-DAEC codes. In this case, the goal is to minimize the number of nonadjacent double-bit errors that mis-correct. This is even faster to evaluate since there are fewer possibilities.

When searching the codes, other criteria can be optimized as well such as total number of XOR gates or logic depth of the syndrome generator.

Each row is added one at a time up to the maximum number of spare columns available in the memory. In the best-case, if no spare columns are used for repair, then all the extra rows will be active for error detection and correction. In the worst-case, when all spare columns are used for repair, then none of the extra rows will be active, and hence only the original SEC-DED code that was used as the starting point will remain.

2.4 IMPLEMENTING PROPOSED SCHEME

The proposed scheme in [Datta 09] can be implemented with very little modification to a normal memory that uses spare columns and is protected with an SEC-

DED code. Figure 2.3 shows an example of the scheme assuming a single spare column. The additional logic that is added to support the scheme is the following:

1. An extra XOR tree in the check bit generator and syndrome generator to support one additional check bit.
2. An extra 2-input AND gate to disable the extra syndrome bit when determining error detection if the spare is used for repair.
3. An extra 2-input OR gate in the correction logic for each data bit to disregard the extra syndrome bit if the spare is used for repair. This is shown in Fig. 2.4.

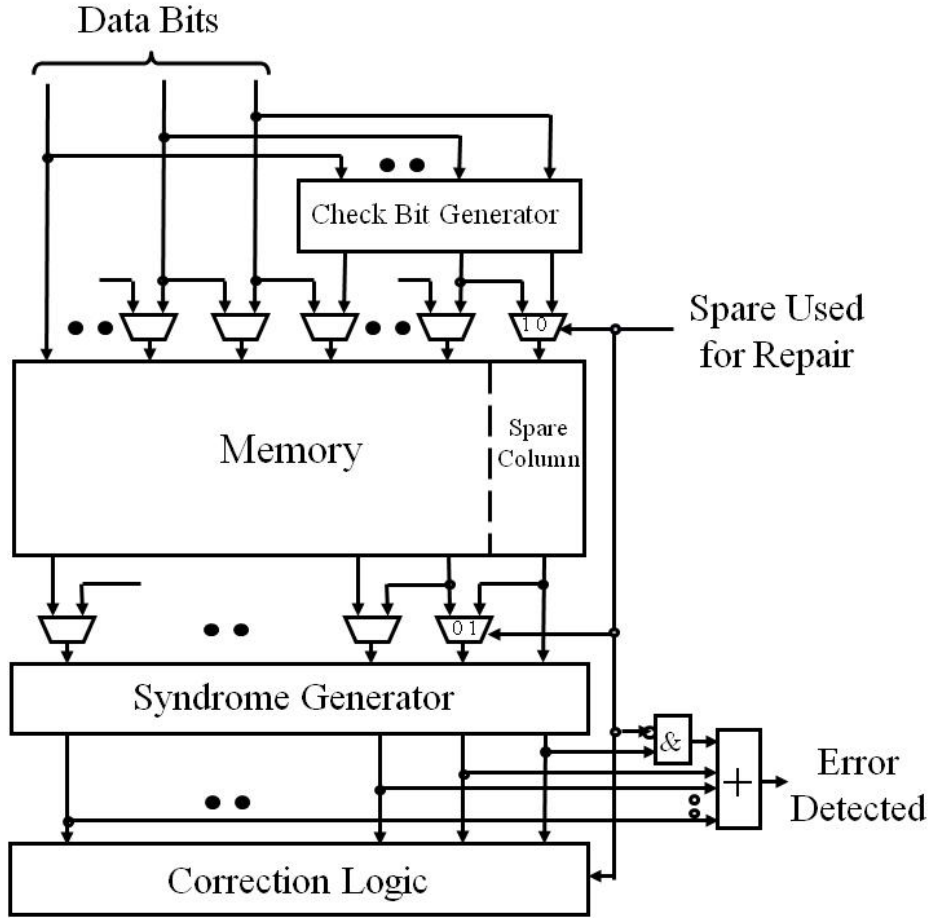

Figure 2.3: Block Diagram of Proposed Scheme for One Spare Column

Other than what is listed above, the rest of the circuitry is already present in a conventional memory with a spare column and SEC-DED ECC. If the spare is used for repair, then the MUXes at the input and output of the memory will shift the bits so that the defective column is bypassed. The control signal for the MUX on the far right will be a '1' if the spare is used for repair or if the spare column itself has a defect. If this control signal is a '0', then the spare is available for storing the extra check bit.

So if the spare is not used for repair, then the extra check bit generated by the check bit generator is stored in the spare column, otherwise, it is simply ignored. At the output of the memory, the extra syndrome bit that is generated is ignored if the spare is used for repair in which case error detection and correction are performed just as if that extra syndrome bit didn't exist. However, if the spare is not used for repair, then the extra syndrome bit is used to help increase the chance of detecting a multi-bit error as well as reduce the probability of mis-correction.

If multiple spare columns are used, then there are multiple control signals indicating whether each spare is used for repair or available for storing check bits. The extra control logic that was added to use one spare column would simply be replicated for each additional spare column.

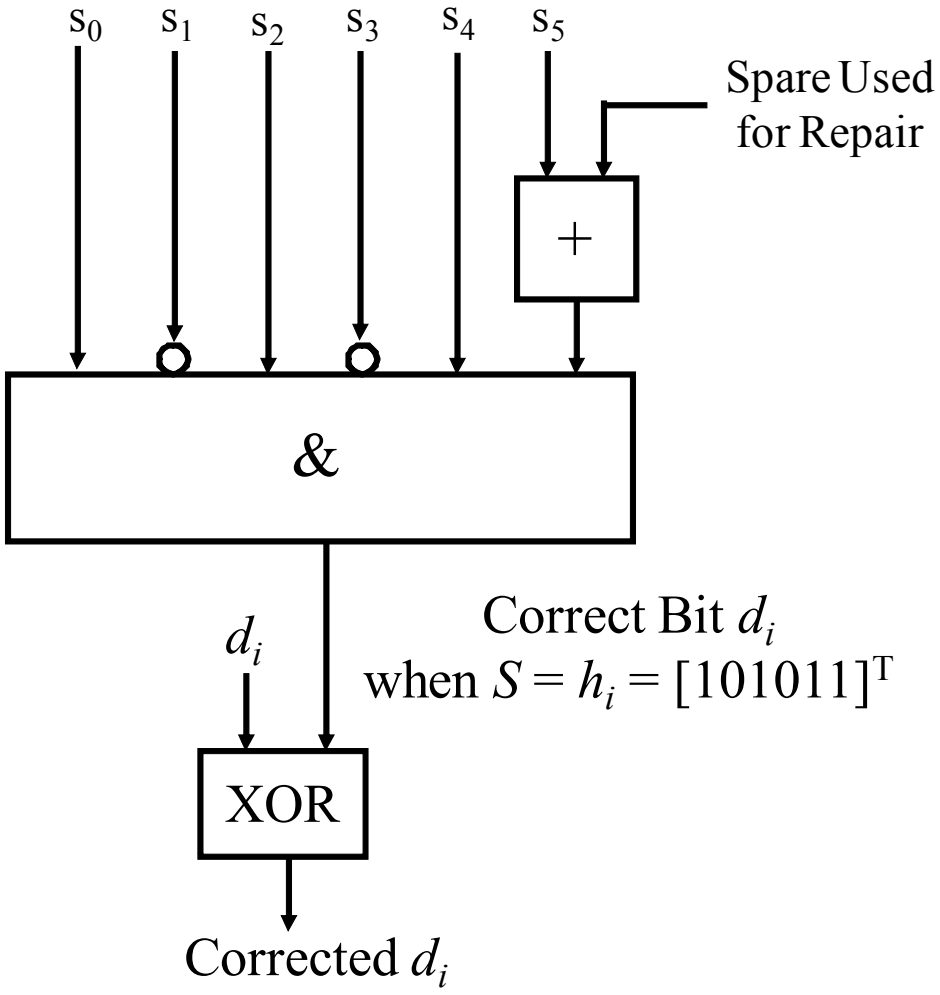

Figure 2.4: Example of Bit-Slice of Correction Logic for Proposed Scheme

2.5 EXPERIMENTAL RESULTS

Experiments were performed for common data word sizes to quantify the benefits of the proposed scheme. Table 2.1 shows the results for minimizing the triple-error mis-correction probability for SEC-DED codes. We started with a *H*-matrix that was original SEC-DED and then added one row to it. The extra row was selected at random from the 2^n possible options, n being the total number of check bits including the extra bits. For a regular SEC-DED code on 16 data bits, $n = 6$ + number of extra check bits.

For smaller codes we would add each possible extra row to the original *H*-matrix, finally retaining the one which had least number of triple error mis-corrections. For codes that were too large to simulate all possible row combinations, we would run a fixed number of iterations – calculating for each added row the total number of triple error mis-corrections – finally retaining the one that had the least number of mis-corrections.

Results are compared with the best codes from [Hsiao 70] and [Richter 08]. For each code, the number of 2-input XORs that is required is shown along with the raw number of triple-bit errors that are mis-corrected and the probability of mis-correction. For the proposed method, results are shown for the cases where one, two, and three spare columns are available after repair. As can be seen, the mis-correction probability is reduced dramatically at the cost of only a small number of additional XOR gates. For 2 and 3 spare columns, the code can detect nearly all triple-errors. Althought the prospect of having three unused or more unused spare columns leftover after the memory repair process is unlikely, our results show full potential of the proposed scheme's effectiveness if more spare columns are to be available.

Table 2.1: Comparison of Triple-Error Mis-correction Probability for SEC-DED codes

Data Bits	[Hsiao 70]		[Richter 08]		[Datta 09]					
					1 Spare Column		2 Spare Columns		3 Spare Columns	
	XORs	Mis-corrected	XORs	Mis-corrected	XORs	Mis-corrected	XORs	Mis-corrected	XORs	Mis-corrected
16	48	1,000 (64.9%)	-	-	58	448 (25.3%)	70	176 (8.7%)	76	52 (2.3%)
32	96	5,452 (59.66%)	115	4, 284 (46.88%)	118	2,548 (25.8%)	129	1,200 (11.3%)	138	588 (5.1%)
64	181	33,568 (56.28%)	250	26,616 (44.63%)	265	16,176 (26.0%)	308	9,084 (14.1%)	351	7,392 (11.0%)

Table 2.2. shows the results for SEC-DAEC codes. Here the goal is to minimize the number of non-adjacent double-bit errors that are mis-corrected. The simulations were again done in a similar manner. A code that is originally SEC-DAEC-DED was chosen

and an extra row was added to the *H*-matrix. In a similar fashion as described above, all possible combinations were tried out for smaller codes and a fixed number of iterations were run for the bigger ones – all the while the goal being the added row should minimize the total number of non-adjacent double error mis-corrections. Again, as can be seen, the mis-correction probability drops significantly with each added check bit. Also, the improvement in the number of mis-corrections when compared to similar codes in [Dutta 07] and [Richter 08] is substantial, with our scheme reducing mis-corrections to negligible proportions for more than one spare column.

Table 2.2: Comparison of Non-Adjacent Double-Error Mis-correction Probability for SEC-DAEC codes

Data Bits	[Dutta 07]		[Richter 08]		[Datta 09]					
					1 Spare Column		2 Spare Columns		3 Spare Columns	
	XORs	Mis-corrected	XORs	Mis-corrected	XORs	Mis-corrected	XORs	Mis-corrected	XORs	Mis-corrected
16	48	118 (56.2%)	-	-	55	68 (29.4%)	62	33 (13.0%)	67	24 (8.7%)
32	96	379 (53.4%)	115	274 (39.0%)	117	203 (27.4%)	130	108 (13.8%)	140	72 (8.8%)
64	224	1316 (53.0%)	250	864 (34.8%)	263	688 (26.9%)	306	469 (17.8%)	353	395 (14.6%)

As had been mentioned earlier, in the case of bigger size codes that did not allow exhaustive simulation – for choosing the extra row – we ran a fixed number of iterations and picked the best row based on number of mis-corrections. Table 2.3 shows the comparison between random and exhaustive searches for different size of codes. The very little difference in the two approaches justifies the random search for bigger code sizes.

Table 2.3: Comparison of Random and Exhaustive Searches for Obtaining Optimal Result

Data Bits	Triple-Error Mis-correction Probability Adding 1 Spare Row				Non-Adjacent Double-Error Mis-correction Probability Adding 1 Spare Row			
	Random Search		Exhaustive Search		Random Search		Exhaustive Search	
	XORs	Mis-corrected	XORs	Mis-corrected	XORs	Mis-corrected	XORs	Mis-corrected
16	56	453 (25.5%)	58	448 (25.3%)	56	72 (31.2%)	55	68 (29.4%)
18	63	352 (13.5%)	63	352 (13.5%)	65	51 (17.0%)	65	51 (17.0%)
20	76	496 (15.1%)	76	496 (15.1%)	73	65 (17.2%)	73	65 (17.2%)

2.6 CONCLUSIONS

In this chapter, a scheme for exploiting unused spare columns after repair is described for improving memory reliability. It is shown that very little additional hardware beyond what is already present for a memory with spare columns and SEC-DED ECC is required to use this scheme. The experimental results show that the mis-correction probability can be significantly reduced.

Note that if a memory has both spare rows and spare columns, then the spare rows could be used first thereby increasing the number of spare columns that remain for providing additional check bits.

Chapter 3: Generating Burst-error Correcting Codes from Orthogonal Latin Square Codes – a Graph Theoretic Approach

3.1 INTRODUCTION

Single event upsets (SEU) and soft errors generated by ionizing particles or neutron interactions with semiconductor devices have been identified as a critical and possibly dominant failure mechanism in modern CMOS circuits**.** Error detection and correction schemes in memories and microprocessor caches are common and drastically reduce the externally observable error rate.

A key consideration for these protection schemes is the treatment of multiple bit errors that can be generated when adjacent bits fail as a result of a single strike. Studies have shown that 1-5% of single event upsets (SEUs) can cause multiple-bit errors (MBUs) [Satoh 00], [Makihara 00], [Kawakami 04]. Most MBUs will affect nearby cells.

These events can prevent the detection of an error in parity protected circuits, or make an error uncorrectable in spite of the use of Error Correction Codes (ECC). Bit interleaving is commonly used to minimize the error rate contribution of multi-bit errors. It refers to a memory layout architecture in which physically adjacent bits belong to different logic words. The result is that from an error detection and correction standpoint, two adjacent failing bits appear as two single bit errors rather than **as** a double bit error in the logic word. Bit interleaving rules are often defined as the minimum physical distance separating two bits belonging to the same logic word. The quantification of their effectiveness requires a detailed understanding of the multi-bit failure probabilities and operating parameter sensitivities which are generally not available in the open literature [Maiz 03]. However bit interleaving would be limited by the width of the memory bus.

Moreover with continuous voltage scaling multiple-bit errors pose an important challenge, especially for memory sub-systems. A dramatic increase in MCU rates relative to SBU is projected for geosynchronous orbits, where direct ionization by heavy-ions dominates [Seifert 08].

In the non-volatile memory space, multilevel cell (MLC) NAND flash memories, which are widely used in mobile and wireless systems, have inferior data retention times compared to single bit cell (SLC) NAND flash. Although multi-leveling cell (MLC) improves memory density and performance of memory storage systems in general, they would also be prone to a greater number of errors caused due to shift of threshold voltage during cell programming operations [Micheloni 06] [Chen 08]. Such systems would require stronger ECC than traditional single error correction codes. Therefore, the data reliability has become an important issue in most communication and storage systems for high speed operation and mass data process.

In this work, a novel method is proposed for designing a multiple error correcting code specifically targeted towards correcting burst errors. We show how, given an Orthogonal Latin Square code, it can be converted into a code capable of correcting burst errors of a specific length in addition to its original capacity with minimal overhead [Datta 11b].

The rest of the chapter is organized as follows, section 3.2 talks about other knows methods of multi-bit error detection., section 3.3 explains OLS codes, section 3.4 explains our proposed methodology followed by the results in section 3.5. Finally section 3.6 concludes along with experimental results.

3.2 RELATED WORK

For high defect rates, memory repair schemes based on spare rows and columns are not effective. Much higher levels of redundancy are required that can tolerate multi-bit errors. The two-dimensional ECC proposed by [Kim 07] tolerates multiple bit errors due to non-persistent faults, but is slow and complicated to decode.

Conventional SEC-DED codes can only detect, but not correct, double-bit errors. In [Dutta 07], it was shown that by carefully selecting and ordering the columns in the H-matrix for an SEC-DED code, it is possible to correct all adjacent double-bit errors in addition to correcting all single bit errors thereby creating an SEC-DAEC code. Since the most likely double-bit errors will be adjacent, this is very useful. The limitation of SEC-DAEC codes is that they may not detect all non-adjacent double-bit errors. While scaling up conventional parity check code for correcting multi-bit errors requires less check bits, the additional number of syndromes that needs to be stored for correction purposes makes parity check matrices an unattractive solution for multi-bit errors.

In some cases, check bits are used along with spare rows and columns to get combined fault-tolerance. In [Stapper 92], interleaved words with redundant word lines and bit lines are used in addition to the check bits on each word. [Su 05] proposes an approach where the hard errors are mitigated by mapping to redundant elements and ECC is used for the soft errors. Such approaches will not be able to provide requisite fault tolerance under high bit error rates when there are not enough redundant elements to map all the hard errors.

[Micheloni 06] proposed a scheme that uses Bose- Chaudhuri-Hocquenghem (BCH) codes to correct multiple errors in NAND flash. However, NAND flash memory systems process with a large size of data such as a page or a block unit. Hence, BCH codes may not be appropriate for a NAND flash controller [Chen 08]. [Kim 10] proposed

a product code using a Reed-Solomon code scheme for NAND flash memories, capable of correcting multiple bit errors. Although Reed-Solomon codes are good for burst errors, the decoding time would be enormous (> 500 clock cycles) [Kim 10].

The application of OLS codes for handling the high defect rates in low power caches as described in [Christi 09] provides a more attractive solution. While OLS codes require more redundancy than conventional ECC, the one-step majority encoding and decoding process is very fast and can be scaled up for handling large numbers of errors as opposed to BCH codes, which while providing the desired level of reliability requires multi-cycles for decoding [Lin 83].

Since most multi-bit errors are likely to result in adjacent bit failures, a burst error code seems like an optimal solution. In this chapter we show how a regular OLS code can be converted to correct burst errors of specific lengths. This way we can combine the single-step decodable facet of OLS codes along with its high error correction capability. The capability of OLS codes to correct multiple errors in a single cycle is synergistic with a high performance memory system, in particular MLC NAND. This way even in the presence of multiple errors, a likely scenario in MLC NAND systems [Micheloni 06], the error detection and correction step would not be a bottleneck in the way of improved memory performance. Our proposed solution preserves this property of OLS codes while enabling it to correct burst errors with minimal overhead.

3.3 ORTHOGONAL LATIN SQUARE CODES

A Latin square [Hsiao 70] of order (size) m is an m x m square array of the digits 0, 1, . . . , m - ***1,*** with each row and column a permutation of the digits 0,1, … , m - 1. Two Latin squares are orthogonal if, when one Latin square is superimposed on the other, every ordered pair of elements appears only once.

In general, a *t*-error correcting majority decodable code works on the principle that *2t + 1* copies of each information bit are generated from *2t + 1* independent sources. One copy is the bit itself received from memory or any transmitting device. The other *2t* copies are generated from *2t* parity relations involving the bit. By choosing a set of *h* Latin squares that are pair-wise orthogonal, one can construct a parity check matrix such that the number of 1's in each column is $2t = h + 2$. The orthogonality condition ensures that for any bit ***d,*** there exists a set of *2t* parity check equations orthogonal on d_i**,** and thus makes the code self-orthogonal and one-step majority decodable. One-step majority decoding is the fastest parallel decoding method. The *t*-error correcting codes generated by OLS codes [Hsiao 70] have m^2 data bits and *2tm* check bits per word.

Let the m^2 data bits be denoted by a vector:

$$D = [d_0, d_1, ...d_{m^2-1}] \quad \text{..... (1)}$$

Then the *2tm* check-bit equations for *t*-error correcting are obtained from the following parity check matrix H:

$$H = \left[\begin{array}{c|c} \begin{matrix} M_1 \\ M_2 \\ M_3 \\ : \\ M_{2t} \end{matrix} & I_{2tm} \end{array}\right] \quad \text{..... (2)}$$

$\boldsymbol{I}_{2tm}$ is an identity matrix of order *2tm* and $\boldsymbol{M_1}, \ldots, \boldsymbol{M_{2t}}$ are submatrices of size *m* x m^2. These submatrices $\boldsymbol{M_1}....\boldsymbol{M_{2t}}$ have the form

$$M_1 = \begin{bmatrix} 11...1 & ... & \\ : & 11...1 & : \\ : & ... & 11...1 \end{bmatrix}_{mxm^2} \quad \text{..... (3)}$$

$$M_2 = [I_m \quad I_m \quad ... \quad ... \quad ... \quad I_m]_{mxm^2} \quad \text{... (4)}$$

The matrices $\boldsymbol{M_1}$....$\boldsymbol{M_{2t}}$ are derived from the existing set of orthogonal Latin squares $L_1, L_2, \ldots, L_{2t-2}$ of size m x m. Denote the set of Latin squares as,

$$L_1 = \left[l_{ij}^1\right]_{mxm}$$
$$L_2 = \left[l_{ij}^2\right]_{mxm}$$
$$\vdots$$
$$\vdots$$
$$L_{2t-2} = \left[l_{ij}^{2t-2}\right]_{mxm}$$

where $l_{ij} \in \{1, 2, \ldots, m\}$

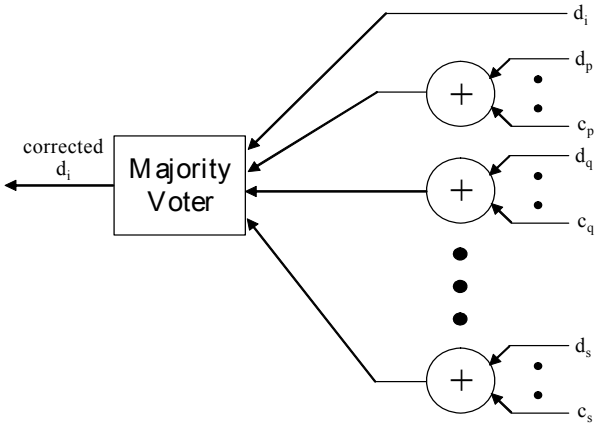

Figure 3.1: Decoding Data Bit d_i with Majority Voter

Once the *H*-matrix is constructed, the decoding for each data bit is done using a majority voter as illustrated in Fig. 3.1. When decoding data bit d_i, the set of bits in each of the *2t H*-matrix rows that d_i is present in are XORed together and serve as an input to a majority voter along with d_i itself giving a total of *2t+1* inputs. Since the set of inputs to the XOR gates are orthogonal, the OLS code will provide the correct output as long as the number of errors is less than half the number of inputs to each voter, i.e., *t* or less. Note that OLS coding does not need to generate a syndrome, but can "correct" errors directly from majority voting.

As an example, a single error correcting OLS code for 16 data bits is shown below. For 16 data bits, $m = 4$. Also, since this is a single error correcting code, $t = 1$. Therefore the total number of check bits will be $2*t*m = 8$. The H-matrix for the resulting (24, 16) code is shown below.

$$H = \begin{bmatrix}
d_0 & d_1 & d_2 & d_3 & d_4 & d_5 & d_6 & d_7 & d_8 & d_9 & d_{10} & d_{11} & d_{12} & d_{13} & d_{14} & d_{15} & c_0 & c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 \\
1 & 1 & 1 & 1 & & & & & & & & & & & & & 1 & & & & & & & \\
 & & & & 1 & 1 & 1 & 1 & & & & & & & & & & 1 & & & & & & \\
 & & & & & & & & 1 & 1 & 1 & 1 & & & & & & & 1 & & & & & \\
 & & & & & & & & & & & & 1 & 1 & 1 & 1 & & & & 1 & & & & \\
1 & & & & 1 & & & & 1 & & & & 1 & & & & & & & & 1 & & & \\
 & 1 & & & & 1 & & & & 1 & & & & 1 & & & & & & & & 1 & & \\
 & & 1 & & & & 1 & & & & 1 & & & & 1 & & & & & & & & 1 & \\
 & & & 1 & & & & 1 & & & & 1 & & & & 1 & & & & & & & & 1
\end{bmatrix}$$

As can be seen from the above matrix, each data bit, from d_0 to d_{15} can be reconstructed from two independent parity equations. For example, bit d_0 can be regenerated by XORing, d_1, d_2, d_3, c_0 and also d_4, d_8, d_{12}, c_4. These two independent equations along with bit d_0 itself would be the three inputs to the voter. Using such a scheme bit d_0 can be correctly decoded in the presence of one error. This shows the above H-matrix can tolerate a single error anywhere in the code.

3.4 PROPOSED SCHEME

The proposed scheme in [Datta 11b] is based on the fact that for decoding OLS codes, as long as each bit has requisite number of inputs feeding into the majority voter, multiple-bit errors can be corrected using a much smaller code. Say, for example bits d_x, d_{x+1}, d_{x+2} are in error. If each row of the OLS matrix contains at most only one of d_x, d_{x+1}, d_{x+2} and each of these bits occur in at least $2t$ different, orthogonal rows of the OLS matrix, then the OLS code is capable of handling any t error pattern and also a burst error pattern on the bits d_x, d_{x+1}, d_{x+2}. Moreover since by definition all rows of an OLS matrix

are mutually orthogonal, each particular conflict can occur only once in the code. This helps limit the number of conflicts that needs to be resolved. To summarize, in this example, if the parity relations are chosen carefully then $2t + k$ bits should be sufficient to detect all t-bit error patterns and any burst of length at most three. So, for a OLS code to correct all burst error patterns of length b, no parity relation should be comprised of bits adjacent by a distance b or less.

As part of our scheme we try to resolve all "conflicts" in a given OLS code and convert it into one capable of correcting all burst errors of length b. "Conflicts" here are defined as any row in the original OLS matrix where any combination of b adjacent bits appear together. The OLS matrix cannot correct all burst errors as long as even one such row is present. The problem has been modeled as a graph coloring problem. All the bits which cause conflicts are modeled as graph nodes. Once the set of nodes have been determined, the next step is to formulate the constraints that would dictate which bits can be grouped together. So, here we have a problem where a set of nodes needs to be grouped into as few sets as possible while adhering to some specific rules. If, now, we think of the set of nodes as a set of vertices in a graph whose edges are the specific rules binding their grouping, the problem then almost perfectly lends itself as a graph coloring problem.

Graph coloring refers to a problem, where we seek to assign a color to each node of an undirected graph G so that if (u, v) is an edge, then u and v are assigned different colors; and the goal is to do this while using a small set of colors. More formally, a k-coloring of G is a function $f: V \rightarrow \{1, 2, \ldots, k\}$ so that for every edge (u, v), $f(u) \neq f(v)$. For the burst-error problem under consideration, the number of colors required to color the graph is equal to the number of extra check bits required. There is however, one key difference between the traditional graph coloring problem and our problem at hand.

While in a conventional graph coloring problem, one would try to color all nodes of the graph using a minimum set of colors, all we need for our purposes is to select any one node from each conflicting pair and move them to a different line. The justification behind this is as follows. Each conflicting pair represents two bits, separated by a distance less than or equal to b. The structure of an OLS matrix is such that for all $b < m$, conflicting nodes will always be in pairs. Now if we choose any one node from each conflicting pair and move them to different lines, following pre-determined rules, we can convert the OLS code into a burst error correcting code. Hence once we single out the entire set of conflicting nodes (the set V of graph vertices) and determine the constraints binding their mutual positioning in a line of the OLS matrix, we trim the set of nodes by picking one from each conflicting pair. Then we have our final graph which we proceed to color in a manner described below with the final goal of using up as few colors as possible.

Since we expect the set of conflicting nodes to be fairly constrained as far as their mutual positioning in a single line goes, especially for smaller data sizes, we choose to represent the graph using an adjacency matrix. Identifying conflicting pair of nodes requires a single pass through each bit of the OLS matrix once. Since an OLS matrix capable of $2t$ errors has $2tm$ check bits and m^2 data bits, identifying all conflicting pair of nodes requires $O(m^3)$ time. Once all the conflicting pair of nodes has been recognized, the edges are determined following the rules listed below,

i) if any two nodes u, v of the graph appear in a row of the OLS matrix which produced neither u nor v, then (u, v) is an edge of G.

ii) if any two nodes u, v of the graph are separated by a distance less than or equal to b, then (u, v) is an edge of G.

This takes $O(n^2)$, n being the number of nodes in the graph [Cormen 01].

Once the graph G has been created, there are two separate problems which need to be solved in order. First we need to trim the graph, choosing one node from each conflicting pair. While choosing any one from the conflicting pair would preserve the functionality of our goal, choosing the node with fewer edges incident upon it helps us in the next step, where we do the coloring. Since two nodes sharing an edge between them needs to be colored differently, fewer edges would allow us to achieve our coloring goals using fewer colors which translate to fewer extra check bits. The argument rings true intuitively as well, since fewer edges mean less constraints and subsequently more freedom in coloring the graph.

Once the set of nodes, V, and the set of edges, E, has been trimmed, we are left with the final step in our problem i.e. to color G. It has been shown that k-coloring, for $k > 2$ is a NP complete problem [Kleinberg 06]. In this work we try solving the graph coloring problem using an underlying Breadth First Search (BFS) structure. In order to adapt the traditional BFS algorithm for the purposes of graph coloring we needed to use some additional data structures. Each node maintains an array listing what are the forbidden colors for that node. Thereafter the BFS algorithm is applied as follows,

i) start with a source node, s
ii) add s to the processing queue Q
iii) while $Q \neq \square$,
    a) $u \leftarrow$ *dequeue (Q)*
    b) make a single pass through the array listing forbidden colors, selecting the first color, say c_u not listed as forbidden for u
    c) for each node $v \in Adj\ (u)$
        1. set c_u as a forbidden color
        2. *enqueue (v)*
    d) set $u.visited \leftarrow 1$
iv) go through set of nodes, if $u.visited \neq 1$, run steps i)-iii) on it (after traversing the queue Q, if any node has $u.visited \neq 1$, it would mean that node is disconnected from the rest of the graph, hence a separate modified BFS algorithm needs to be run on that node to ensure that all nodes of the graph are covered)

In addition to traversing each node and each edge of the graph as part of our coloring algorithm, one has to go through an array for each node to determine what should be the correct color for that node. The cost for that traversal is of $O\ (k)$, for an array of size k. Hence, using our chosen adjacency-matrix representation of the graph, the complexity of the algorithm described above is bounded by $O\ (n^2k)$.

The number of colors used, k, signifies the number of extra check bits required to convert the original OLS code into a code capable of correcting all burst errors of length b or less. For our experiments, the array in each node keeping track of forbidden colors would be initialized to size k. If at any node, all k colors are designated as forbidden, that would mean the graph cannot be colored using k colors. Subsequent calls to the coloring function would be made with increasing values of k unless a valid solution was obtained. The next section lists some of the experimental results we obtained as proof of concept for our proposed scheme.

3.5 RESULTS

Table 3.1 shows experimental results where OLS codes that were originally double error correcting were converted to double error correcting and 3-bit burst error correcting code. Experiments were performed for different sizes of m. Although a double error correcting OLS code would include the sub-matrices M_1, M_2, M_3 and M_4, the structure of M_1 is such that there would be too many conflicts due to bit adjacency. Hence without any loss of generalization, a double error correcting OLS code was formed out of sub-matrices M_2, M_3, M_4 and M_5. This ensured a minimal number of conflicts and subsequently a better solution.

As shown in Table 3.1, the overhead of extra check-bits diminish with increasing size of code. This can be explained by virtue of the fact that for a larger *m* there is more freedom in the placement of bits giving rise to fewer conflicts.

Table 3.1: Check-bit overhead for 3-bit burst-error protection and Double Error Correcting OLS code

m	Original check-bits	Data Bits	Extra added check-bits	Percentage Overhead
4	16	16	4	25%
8	32	64	4	12.50%
16	64	256	3	4.69%

In Table 3.2, we show how attempting to build a code capable of handling longer burst errors affects check bit overhead. As expected we see that a larger number of check bits is necessary for stronger codes. We see from Fig. 3.1 that the increase in the number of extra check-bits follows a piecewise linear relationship with the length of burst-errors to be corrected.

Table 3.2: Check-bit overhead for DEC OLS code with *m*=16 while burst-error length is varied

Burst Error Length	Extra added check-bits	Percentage Overhead
3	3	4.69%
5	3	4.69%
7	4	6.25%
9	5	7.81%
11	5	7.81%

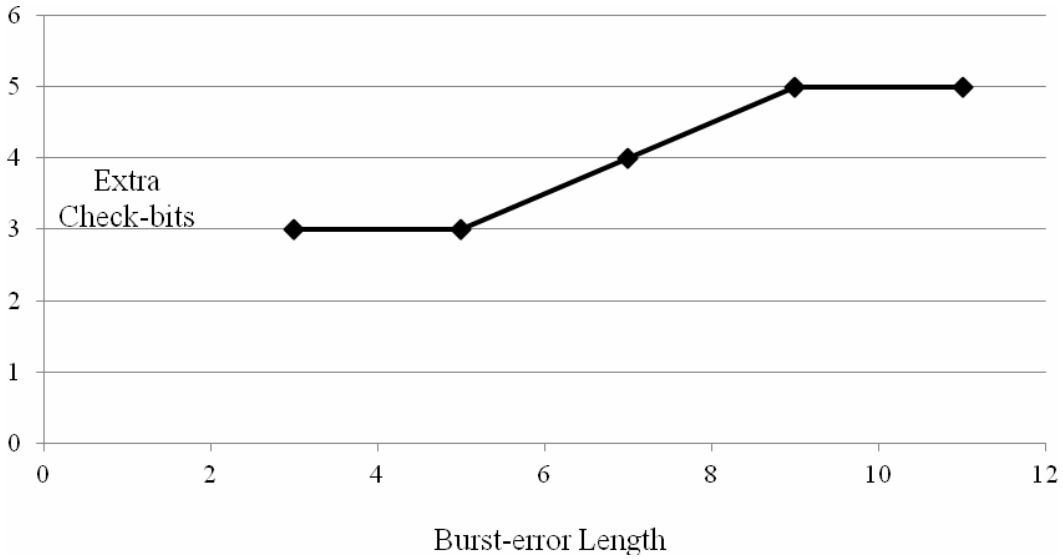

Figure 3.2: Burst-error length vs check-bit overhead

3.6 CONCLUDING REMARKS

In this chapter we have presented a scheme for generating one-step decodable burst-error correction codes. Our experimental results show that the check-bit overhead is a decreasing function for increasing code size, which makes this an attractive solution to counter the worsening problem of multiple-bit errors in memory systems.

Chapter 4: Post-Manufacturing ECC Customization Based on Orthogonal Latin Square Codes and Its Application to Ultra-Low Power Caches

4.1 INTRODUCTION

Voltage scaling, which is one of the most effective ways to reduce power consumption, is limited by a minimum value referred to as V_{ccmin} beyond which circuits may not function reliably [Taur 98]. Voltage scaling beyond V_{ccmin} gives rise to reliability issues, most notably for the memory sub-systems. In order for V_{ccmin} to be reduced to enable ultra-low power modes in microprocessors and other circuits, some means for handling high memory bit failure rates is required.

One general approach that has been proposed in [Wilkerson 08] is to trade off cache capacity for reliable low voltage operation. The idea is that in high-voltage operation, failure rate is low, so the entire cache is available. However, in low voltage operation, many memory cells become unreliable, so the cache size is sacrificed to increase reliability. Two approaches were proposed for this in [Wilkerson 08] which are based on performing a low voltage characterization test of the memory and identifying the failing cells. One approach (called word-disable) uses two physical lines to configure one logical line where only non-failing words are used. The other approach (called bit-fix) uses 25% of the cache to store a defect map which is used to bypass failing cells. Another approach that has been proposed in [Chisti 09] is based on using part of the cache to store the check bits for an Orthogonal Latin Squares (OLS) code [Hsiao 70] when operating in low voltage mode. An OLS code is a multi-bit error correcting code which is one-step majority decodable which allows faster encoding and decoding than traditional ECC at the cost of more check bits. The OLS code can be

used to encode each cache line and correct errors that may arise due to failing cells as well as due to transient errors. The amount of usable cache size depends on the number of check bits required for the OLS code which in turn depends one how many errors the OLS code can correct.

The idea proposed in this work is that after manufacturing a chip, a memory characterization test [Chang 07] can be performed to generate a defect map that identifies which cells fail or are vulnerable in low voltage operation. Once this information is known, the marginal cells effectively become erasures (i.e., errors with known locations), and it is possible to select a subset of the rows in the H-matrix for an OLS code which are sufficient to provide the desired level of error detection/correction capability in the presence of the defective cells for that particular chip [Datta 10]. This reduces the number of check bits that need to be stored in the cache thereby freeing up more of the memory for storing data and improving performance. Note that conventional approaches that use spare rows and columns for repairing memories can only repair a small number of defects and hence are not effective for high defect rates [Kim 98]. Because OLS codes use majority decoding for error correction, it is very easy to disable portions of the code by simply masking bits at the input to the voters and adjusting the threshold of the voter. An OLS code can be selectively reduced by storing one configuration bit for each row in the *H*-matrix which indicates whether or not that row should be included when encoding and decoding. By selectively reducing the number of rows in the *H*-matrix that are used, the number of check bits that need to be stored for each word is reduced thereby reducing the amount of redundancy. The idea of post-manufacturing customization of the ECC can be applied to the problem of providing reliable cache operation in ultra-low power modes of operation. A *t*-error correcting OLS code is selected for a particular cache design based on the expected defect rate and

implemented in full with on-chip hardware. It requires $c_{full}=n-k$ check bits for k-information bits. t must be selected large enough to handle the worst-case number of defects in any line of the cache. If some cache line has more than t defects, then the cache is unusable and the chip must be discarded. Based on the desired yield, t is selected appropriately. After a chip is manufactured, a memory characterization test is used to obtain a defect map. It may turn out that for a particular chip, no cache line has more than $(t-3)$ defects present. Thus, the t-error detecting code is overkill. Even if some line has t-errors present, it still may not be necessary to use the entire H-matrix of the OLS code. In section 4.3, a procedure is described for selecting a minimal number of rows in the H-matrix so that e additional transient errors can be corrected in addition to the permanent erasures identified in the defect map. In this way, the number of check bits that are actually used for a particular chip, c_{used}, is smaller than for using the full code, c_{full}, however, the code is still able to provide the desired level of protection. The configuration bits on the chip are set to indicate which rows of the H-matrix to use while all others are disabled. Compared to [Christi 09], the proposed method use up less of the cache for storing the check bits in low power mode, which means less caches misses and resultant memory accesses.

Even though the hardware for the full code, c_{full} is implemented on the chip with the proposed method, the functional design can be done assuming an a priori upper bound on c_{used}, the number of check bits that will actually be used post-manufacture, based on statistical analysis for the defect rate that is to be tolerated. This allows the desired level of fault tolerance to be achieved with fewer redundant memory cells storing check bits. In effect, the proposed method is exploiting the degree of freedom in selecting which portion of a larger code to use to get more leverage from a certain number of check bits in comparison to a conventional approach which uses the same code for every chip. It

provides a beneficial tradeoff where a small amount of extra ECC circuitry (hardware redundancy) is added in order to reduce the number of check bits (information redundancy) that needs to be stored in the cache. This increases the performance of the cache while still achieving the desired level of reliability.

4.2 Related Work

Most prior work in memory ECC has focused on low failure rates present at normal operating voltages, and has not focused on the problem of persistent failures in caches operating at ultra low voltage where defect rates are very high.

For high defect rates, memory repair schemes based on spare rows and columns are not effective. Much higher levels of redundancy are required that can tolerate multi-bit errors in each cache line. In addition to the techniques in [Wilkerson 08] mentioned earlier, other prior work includes the two-dimensional ECC proposed by [Kim 07] which tolerates multiple bit errors due to non-persistent faults, but is slow and complicated to decode. Similarly the approach in [Kim 98] can tolerate as many faults as can be repaired by spare columns, which would be insufficient in the present context with high bit-error rate. In some cases, check bits are used along with spare rows and columns to get combined fault-tolerance. In [Stapper 92], interleaved words with redundant word lines and bit lines are used in addition to the check bits on each word. [Su 05] proposes an approach where the hard errors are mitigated by mapping to redundant elements and ECC is used for the soft errors. Such approaches will not be able to provide requisite fault tolerance under high bit error rates when there are not enough redundant elements to map all the hard errors.

The application of OLS codes for handling the high defect rates in low power caches as described in [Christi 09] provides a more attractive solution. While OLS

codes require more redundancy than conventional ECC, the one-step majority encoding and decoding process is very fast and can be scaled up for handling large numbers of errors as opposed to BCH codes, which while providing the desired level of reliability requires multi-cycles for decoding [Lin 83]. The post-manufacturing customization approach proposed in this chapter can be used to reduce the number of check bits and hence the amount of redundancy required in the memory while still providing the desired level of reliability. Note that the proposed approach does not reduce the hardware requirements for the OLS ECC as the whole code needs to be implemented on-chip since the location of the defects is not known until post-manufacturing test is performed.

4.3 PROPOSED SCHEME

The proposed scheme in [Datta 10] leverages memory tests [Chang 07] that can be performed at manufacture time or out in the field when the system is booted up. These tests identify which are the vulnerable bits in the cache. The defect map can then be used to select a subset of the rows from the original *t*-error correcting OLS matrix. The suspect bits will be referred to as erasures (i.e., errors with known locations). For any cache line, given the defect map, the goal is to be able to correct all erasures along with one or more random errors that may occur on any other bit in that cache line.

For ease of explanation, two terms will be defined with respect to each information bit d_i – a "good row" and a "bad row". A "good row" for information bit d_i is a row of the OLS H-matrix that does not have a '1' in any bit position where there is an erasure in any line in the set of considered cache lines *C* except for erasures in bit *i* itself. Such a row can be used to unambiguously decode information bit d_i in the presence of the considered erasures. A "bad row" for an information bit d_i is one row of the OLS H-matrix which has a '1' in one or more bit positions where one or more erasures exist in

the set of considered cache lines *C*. A small example is shown in Fig. 2 where the set of considered cache lines contains two cache lines where 'E' denotes a vulnerable cell which is treated as an erasure. In this example, *H-row1* shown in Fig. 2 would be a good row for any information bit because it does not intersect with any erasure bit. *H-row2* intersects with the erasure in d_1, so it would only be a good row for d_1, but would be considered a bad row for all other information bits. *H-row3* intersects with erasures in both d_3 and d_5, so it would not be considered a bad row for all information bits.

Each information bit is generated by a majority voter whose inputs correspond to rows in the *H*-matrix plus the information bit itself. In a *t*-error correcting OLS code, each information bit is generated by a *2t+1* input majority voter. By construction of the code, at most *t* inputs to the majority voter can be bad rows at any given time, so the voter will always produce a correct output as long as the number of errors in any cache line is *t* or less. In the proposed approach, the goal is to tolerate *e* transient errors on top of the known erasures identified in the defect map. This can generally be accomplished with much fewer than *2t+1* inputs to the voter, and hence some inputs can be disabled (i.e., masked off). For information bit d_i, the set of good and bad rows can be identified for the considered erasures. Inputs to the voter for d_i can be disabled provided the following relationship is maintained:

$$\text{"good rows"} - \text{"bad rows"} \geq 2(e+1) \quad \text{(Condition 1)}$$

This ensures that if *e* transient errors occur which could cause *e* good rows to become bad rows, the voter will still produce a correct output even if d_i itself has an erasure. For example, if e=1, then the voter needs a minimum of 5 inputs with 4 of them coming from good rows and one input coming from d_i itself. In the worst case where d_i has an erasure and one row has a transient error, the 3 remaining good rows would out vote the two erroneous inputs. Alternatively, a 7-input voter could be used

with inputs coming from 5 good rows, one bad row, and d_i itself. Any size voter can be used provided the number of good rows is larger than the number of bad rows by a sufficient amount as per *Condition 1*.

	d_0	d_1	d_2	d_3	d_4	d_5	d_6	d_7
line1	-	E	-	-	-	E	-	-
line2	-	-	-	E	-	-	-	-
H-row1	1	0	0	0	1	0	1	1
H-row2	0	1	1	0	1	0	0	1
H-row3	1	0	0	1	0	1	1	0

Figure 4.1: Example where H-row1 good for all d_i, H-row2 good for only d_1, and H-row3 bad for all d_i

	d_0	d_1	d_2	d_3	d_4	d_5	d_6	d_7
row1	G	-	-	-	G	-	G	G
row2	-	G	B	-	B	-	-	B
row3	B	-	-	B	-	B	B	-

Figure 4.2: Covering matrix for example in Fig. 4.1

The problem of selecting a minimal set of rows so that the inputs to the voter for every information bit satisfies condition 1 for a given set of erasures can be formulated as a covering problem. A covering matrix can be formed where each column corresponds to an information bit, and each row corresponds to a row in the *H*-matrix. Each entry in

the matrix is a ‘G’ if the row is a good row for the corresponding information bit, a ‘B’ if it is a bad row, and a ‘-’ if the H-matrix row does not intersect with the information bit. The covering matrix for the small example in Fig. 4.1 is shown in Fig. 4.2. There are 8 information bits and three H-matrix rows. Note that this is not a complete OLS code, but only a small fragment to illustrate how the covering matrix is formed.

Once the covering matrix is formed, then a sufficient number of rows need to be selected to satisfy condition 1 for all information bits. As long as the following condition is satisfied for a *t*-error correcting OLS code, there will always exist a solution to the covering problem, i.e., if nothing else, the solution where all rows in the OLS H-matrix are included will work.

$$\textit{(Max erasures in any line)} + e \leq \quad t \quad \text{(Condition 2)}$$

While the covering problem is NP-complete, good heuristic algorithms can be employed. For example, a greedy procedure that first selects rows with the maximum number of G’s weighted by the difference between the G’s and B’s for each column could be used. Rows are iteratively selected until a valid solution is found that satisfies Condition 1. Other heuristic covering algorithms can be used as well [Vazirani 04]. Another strategy would be to start with all rows selected and iteratively remove rows as long as condition 1 is satisfied.

Note that the covering problem is solved w.r.t. a set of considered erasures. Ideally, the set of considered erasures (and hence the covering matrix) should be recomputed for each cache line and the covering problem for every cache line should be simultaneously solved. However, the computation complexity for this is infeasible. One solution to simplifying the problem would be to consider all erasures in the cache as occurring in the same line which would allow forming a single covering matrix to solve. However, this would badly over constrain the problem. The proposed procedure is to first

consider only the erasures in the worst-case cache line (i.e., the cache line that has the most erasures). Solve the covering problem for that, and then check if it is a valid solution for all other cache lines. If not, then the erasures for one of the cache lines that it is not a solution for are added to the set of considered erasures, and the procedure is repeated. This is done iteratively until a solution that works for all cache lines is found. Typically the worst-case cache lines are the limiting factor, so solving for them typically solves for all cache lines.

4.4 IMPLEMENTATION

One way to implement the proposed approach in [Datta 10] is as follows. The maximum number of erasures that needs to be tolerated in any line of the cache to achieve a desired yield is statistically computed based on the expected memory cell defect probability, the word size of the cache, and the number of lines in the cache. The OLS code is then selected so that it satisfies Condition 2 where the number of bits that the OLS code can correct is equal to the maximum number of erasures in any line plus the number of transient errors that are to be tolerated. Let c_{full} be the number of check bits for the selected OLS code. The maximum number of check bits that the proposed method will require to achieve the desired yield, c_{used}, can then be determined through Monte Carlo simulation. The memory is then designed so that it has c_{used} redundant columns for each line to store the check bits. For the application to low power caches (as described in [Christi 09]), the cache would be reconfigured in low power mode so that it could store c_{used} check bits for each line.

The full OLS code is implemented on the chip for generating all c_{full} check bits when writing to the cache, and for performing the decoding with all c_{full} check bits when reading from the cache. Configuration circuitry is added so that the full OLS code can

be reduced down to c_{used} check bits based on the configuration bits that are set for each chip after performing a memory characterization test as illustrated in Fig. 4. There is one configuration bit for each of the c_{full} check bits for the code. The configuration bit is a '1' if that check bit is to be used, and is a '0' if that check bit is to be disabled. The configuration bits can either be stored with fuses if they are to be one-time programmed at manufacture time, or they can be stored in flip-flops if they are to be programmed by software each time the chip is powered up after performing a memory BIST. Note that the switch networks in Fig. 4 can be simplified by placing constraints on the ways that c_{full} can be mapped to c_{used}.

The decoding process for each data bit is based on majority voting between the H-matrix rows associated with the data bit. The decoding process can be configured as shown in Fig. 5. Since some H-matrix rows may not be included in the reduced code, so inputs to the voter associated with those rows are masked off with AND gates. The majority voter is replaced with a threshold voter that gives a '1' output if the number of inputs that are equal to '1' is greater than or equal to the threshold T. The value of T is configured so that it is equal to $\lceil$(number of non-masked inputs)/2$\rceil$. For example, if there are 7 inputs to the voter, but 2 of them are not part of the reduced code, then they are masked off, and the threshold of the voter is set to 3. As long as no more than 2 out of the 5 non-masked voter inputs are correct, the output of the voter will be correct, so two errors can be tolerated. The control lines in Fig. 5 are generated by control logic that decodes the configuration bits (as shown in Fig. 4).

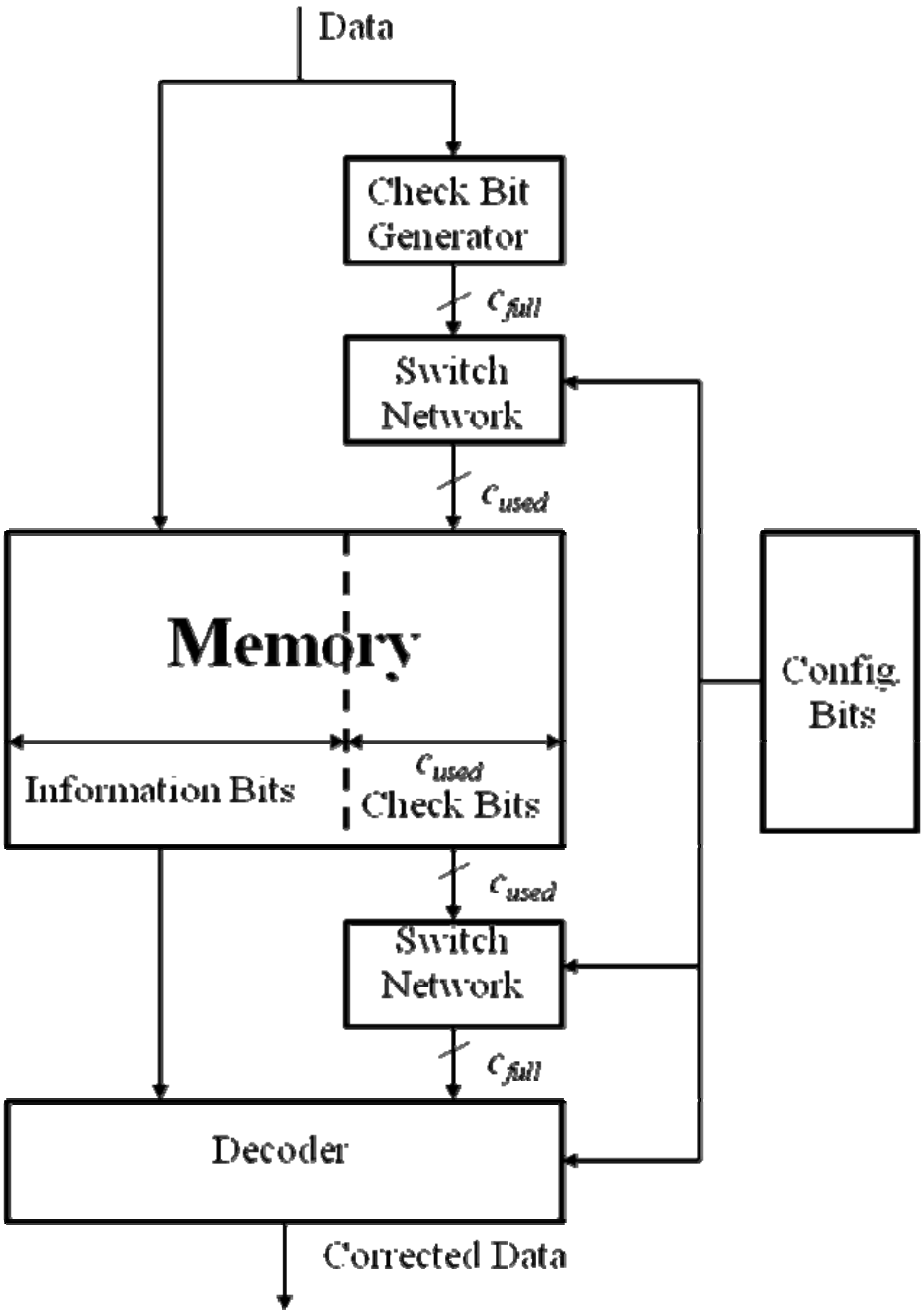

Figure 4.3: Block Diagram of Scheme

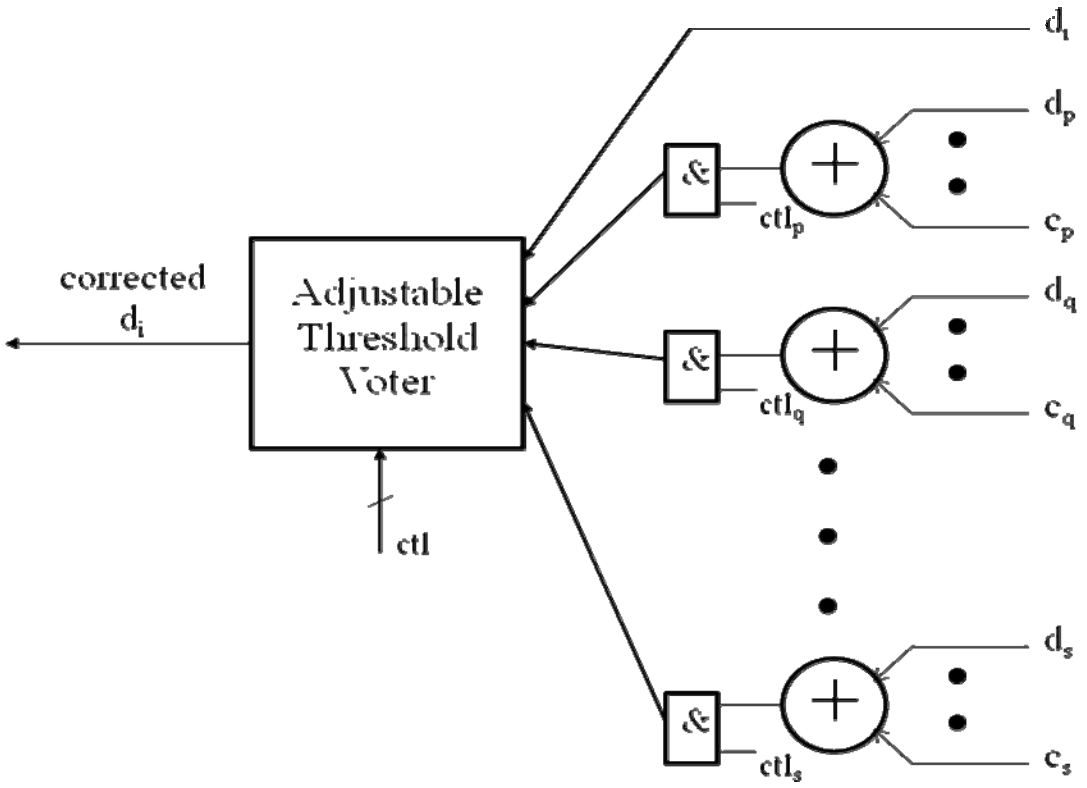

Figure 4.4: Decoding Logic for Data Bit

4.5 EXPERIMENTAL RESULTS

Table 4.1 shows the results for a constant cache size of 64KB where simulations were performed for 1000 caches. For each of the caches, defect maps were generated with random injection of errors but at a specific bit-error rate. The existing OLS code for each cache was then customized based on its corresponding defect map. The first column shows the word size, the second column shows the bit error rate (probability that a bit is defective), and the next two columns show the average and maximum number of check bits required for tolerating all defects among 1000 caches using a conventional OLS code. The last two columns show the same data using the proposed method where the

OLS code is customized for each cache for withstanding the effect of one transient error and all erasures.

As can be seen from the results in Table 4.1, significant reduction in the number of check bits can be achieved. The reduction becomes larger for higher bit-error probabilities and larger word sizes. The relatively less improvement for lower bit-error probabilities is due to the small number of erasures that are present. Also, smaller word sizes have less error correction capability. An OLS code for 64 data bits can correct up to 4 errors, an OLS code for 256 data bits can correct up to 8 errors, and an OLS code for 484 data bits can correct up to 11 errors.

Table 4.1: Results for Different Bit-Error Rates and Word Sizes across Constant Cache Size of 64KB

Word Size (Bits)	Bit-error Rate	Check bits for conventional OLS		Check bits for customized OLS	
		Avg	Max	Avg	Max
64	10^{-3}	63	64	55	64
	10^{-4}	41	64	35	56
	10^{-5}	32	32	32	32
256	10^{-3}	177	224	138	157
	10^{-4}	98	128	84	107
	10^{-5}	66	102	64	68
484	10^{-3}	295	396	198	230
	10^{-4}	143	176	117	139
	10^{-5}	92	132	89	115

Table 4.2 shows the results for different size caches with a word size of 256 bits, for tolerating one transient error on top of all the erasures present in the defect map. As can be seen, with the proposed method, the maximum number of check bits that are required is reduced by 30% to 42% which allows more of the cache to be used for storing data.

Fig. 4.5 shows the distribution of check bits required per cache for the proposed scheme compared to conventional OLS. If a threshold on the number of check bits is set at some point, it can be seen that the yield for proposed method would be much higher.

Table 4.2: Results for Different Cache Sizes with Word Size of 256 Bits and Bit-Error Rate of 10^{-3}

Cache Size (Bytes)	Check bits for conventional OLS		Check bits for customized OLS		Percentage reduction in Max. Check Bits
	Avg	Max	Avg	Max	
16 KB	155	224	117	145	35.27
32 KB	166	256	125	148	42.19
64 KB	175	256	134	156	39.06
128 KB	208	256	163	177	30.81

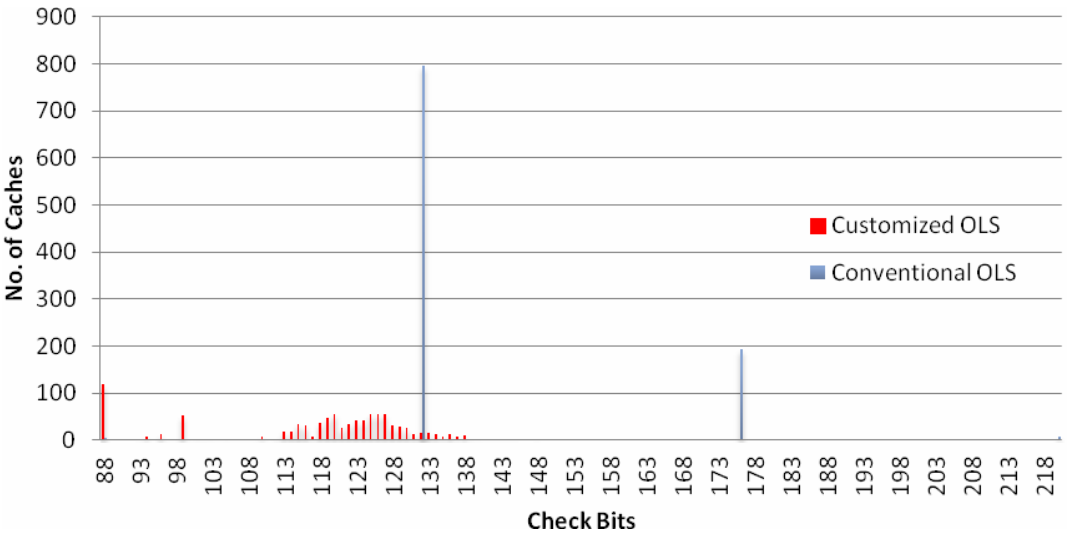

Figure 4.5: Distribution of 484-bit Word, 64 KB Cache at 10^{-3} Bit-Error Rate

4.6 CONCLUSIONS

The check bit overhead for tolerating large numbers of marginal cells in ultra-low power caches is quite large. The proposed method can be used to significantly reduce

the check bit overhead while still providing the same level of reliability especially when bit-error rates are higher.

The idea of post-manufacture ECC customization can be applied to other problems to provide efficient fault tolerance when high defect rates are present.

Chapter 5: Designing a Fast and Adaptive Error Correction Scheme for Increasing the Lifetime of Phase Change Memories

5.1 INTRODUCTION

Memory technology scaling drives increasing density, increasing capacity, and falling price-capability ratios. Storage mechanisms in prevalent memory technologies require inherently un-scalable charge placement and control. This in turn has put memory scaling, a first-order technology objective, in jeopardy. Dynamic Random Access Memory (DRAM) has been used as the main memory in computer systems for decades due to its high-density, high-performance and low-cost. However, DRAM technologies, facing both scalability and power issues, will be difficult to scale down beyond 50nm [Zhang 09] due to various limitations associated with device leakages and retention time.

Resistive memories, which arrange atoms within a cell and then measure the resistive drop through the atomic arrangement, are promising as a potentially more scalable replacement for DRAM and Flash. These technologies include spin-torquetransfer magnetoresistive memory (STT-MRAM), ferroelectric memory (FRAM), memristors, and phase-change memories (PCM). Of these emerging technologies, PCM has received the most research attention in the architecture literature, as it is closest to commercialization [Numonyx 07], [Samsung 06].

Phase change memory (PCM) provides a non-volatile storage mechanism amenable to process scaling. Phase change memories function by alternating between low resistance crystalline and high resistance amorphous states. The thermally induced phase transition is brought about by injecting current into the storage material during writes. The state of the cell is then detected during reads with the high resistance state being interpreted as a zero and the low resistance state as one. PCM, relying on analog current

and thermal effects, does not require control over discrete electrons. As technologies scale and heating contact areas shrink, programming current will also scale linearly. PCM scaling mechanism has been demonstrated in a 20nm device prototype and is projected to scale to 9nm [Lee 09]. As a scalable DRAM alternative, PCM could provide a clear roadmap for increasing main memory density and capacity.

However, one major challenge that needs to be addressed for PCM is its limited write endurance. PCM writes induce thermal expansion and contraction within the storage element, degrading injection contacts and limiting endurance to hundreds of millions of writes per cell at current processes. In current devices, a PCM cell typically supports around 10^7 writes [Ferreira 10]. Thus, PCM will wear-out quickly if used as a main memory.

This is a significant limitation and a prime reason why PCM is not yet a ready substitute for DRAM main memory. Current PCM prototypes are not designed to mitigate PCM endurance. One major challenge in designing ECC for PCM based systems is that the number of cell failures is a monotonically increasing function of memory writes. In order to mitigate the time-dependant nature of failures, this work proposes a novel adaptive error correction technique that increases PCM endurance several times. The core idea here is to start with a nominal ECC, depending on experimentally determined error rates for PCM, and then adaptively boost the ECC strength to keep up with increasing failure rates of PCM [Datta 11a]. The dynamic control over the ECC is achieved by involving the underlying operating system (OS). The OS monitors the maximum number of errors corrected per PCM line and compares this against the strength of the ECC currently in place. When number of errors corrected on a memory line read approaches the capacity of the existing error code, the ECC strength is increased. This can be done on the next reboot or done by writing main memory to

disk and reconfiguring the ECC when it is paged back in. The increase in ECC strength is achieved by breaking up a memory line into segments and then implementing separate ECC for each segment. Taken together, the combined effect of the segmented ECCs can correct up to tens of bits per memory line.

Note that as the ECC of the memory is increased, the performance of the memory system gracefully degrades because more storage is taken up by check bits rather than data bits. However, this strategy is much better than using a worst-case ECC which would give worst-case performance throughout the lifetime of the system. The degradation comes in the form of reducing the effective size of a cache line since less data can be brought to the cache on each read operation to the PCM main memory. Note that if the cache itself is implemented with more reliable SRAM, then the number of check bits stored in the cache does not need to be increased. So the total cache capacity remains the same. If the cache line size is reduced, then the cache can store more lines. Strategies for designing the cache to accommodate graceful degradation of the line size is discussed as well as the performance impact which is highly dependent on the number and locality of memory references for an application.

5.2 Related Work

Various approaches have been adopted to counter the limited write endurance of phase change memories. [Zhang 09] presents a hybrid PRAM/DRAM memory architecture that uses an OS level paging scheme to improve PRAM write performance and lifetime. They use a 7-error correcting BCH code for the ECC. BCH codes provide the desired level of reliability but require increasing number of cycles for correcting multi-bit errors [Lin 83]. [Xu 10] proposes a novel sensing mechanism for multi-level

PCM structures to address the reliability issue using either BCH or LDPC codes, for both of which the decoding time scale with number of errors being detected.

In [Schechter 10], the authors propose to correct permanent errors in PCM systems by encoding the locations of failed cells into a table and assigning cells to replace them. They describe a new data structure which includes information about failed cells and a spare cell to substitute the failed cell. While this approach is attractive because of its low check-bit overhead, it suffers from the same deficiencies of other fixed error correction techniques.

[Ferreira 10] shows how PCM writes can be minimized thereby increasing their lifetime. Note that this methodology could be used on top of the methodology proposed in this chapter. [Lee 09] uses buffer reorganization and partial write techniques to mitigate high energy PCM writes but improves PCM lifetime to only about 5.6 years.

Traditional schemes like using spare rows and columns as well as bit interleaving, as shown in [Stapper 92], are likely to prove insufficient because of the prohibitively high error rate in PCM systems.

While the PCM reliability issue has primarily been addressed from an architecture standpoint, solutions using novel ECC have yet to be fully explored. PCM differs from standard DRAM in a fundamental way in that the number of failures for a PCM cell is a function of time or more accurately a function of the number of writes/cell. The following section presents a detailed overview of the proposed scheme.

5.3 Overview of Proposed Approach

Given that bit failure rates for phase change memories increase with continuous usage, the proposed approach of adaptively increasing the strength of the ECC to keep up

with the increasing failure rate is a good strategy. Note that this strategy requires involving the operating system (OS).

ECC decoding and correction is performed by the memory controller after the data and check bits have been read from the memory chip and are in the processor core. The memory controller can be programmed as to what granularity of ECC is to be implemented. In the proposed method in [Datta 11a], every time the memory controller reads a memory line, the number of errors corrected is compared against a threshold which depends on the existing strength of the ECC in place. When the number of errors corrected for a particular read begins to approach the given threshold for the implemented ECC, the OS switches to a stronger ECC. This is accomplished by writing all of the main memory to disk and paging the memory back into physical memory with the stronger ECC. Note that this process of reconfiguring the ECC occurs very infrequently (on the order of months or years). Another way to implement it would be to record the need for an ECC reconfiguration and then perform it on the next reboot.

For the memory controller to be able to switch to an ECC of greater strength, the hardware has to be in place from before. Depending on up to how many levels the ECC will be stepped up, the full hardware to perform the necessary encoding-decoding will have to be implemented in the memory controller from the beginning. The memory controller will then choose which of the existing encoding-decoding schemes to employ using information received from the OS.

Figure 5.1 shows a possible implementation of the scheme proposed in [Datta 11a] that can switch between two modes of ECC. The OS signals when the memory controller needs to switch from the basic ECC scheme to the advanced scheme. In the general case where several different levels of ECC hardware are implemented, the OS will signal the memory controller when to step up the strength of ECC.

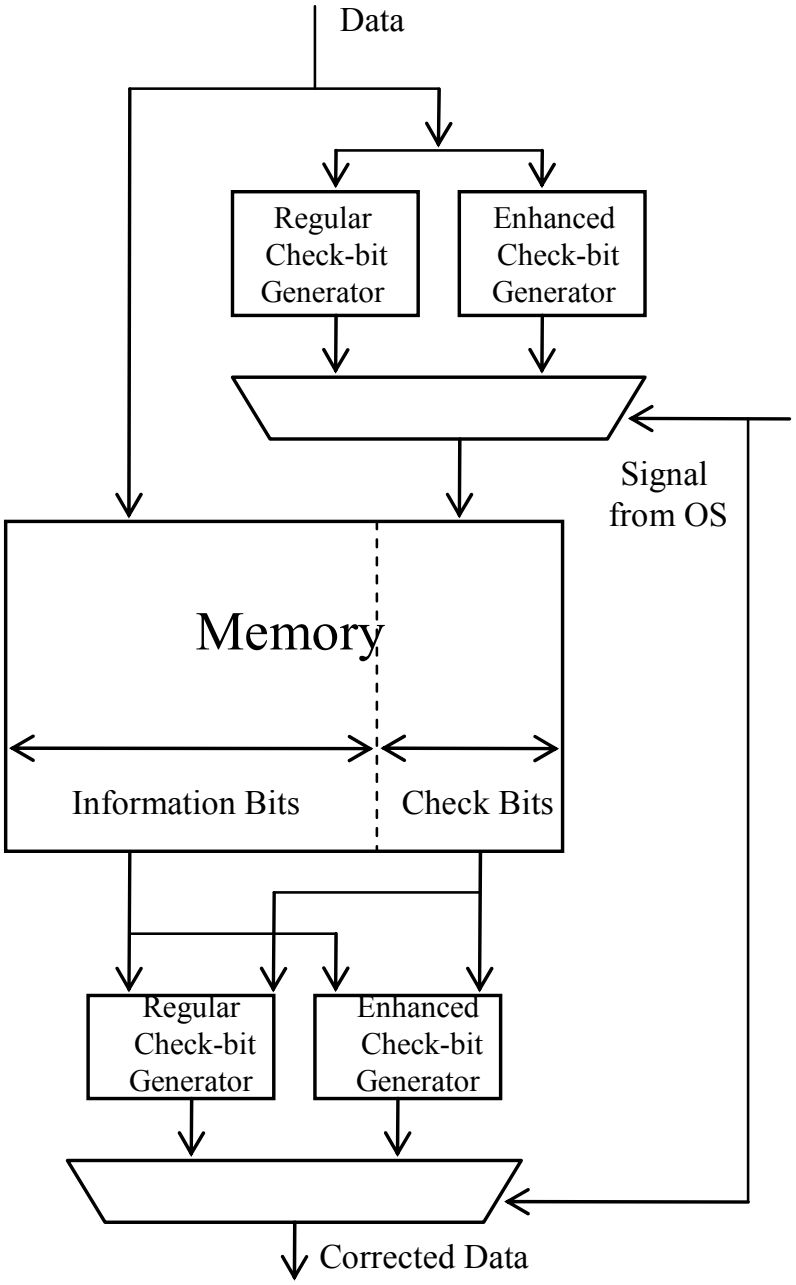

Figure 5.1: Adaptive ECC implementation

Strengthening the ECC requires increasing the redundancy of the code, which means more check bits and fewer data bits can be stored in each line of the memory. When a cache miss occurs, fewer data words can be brought into the cache in each main

memory access. For example, if a line in the memory initially contained 1024 data bits (i.e., 16 64-bit words), but then if the redundancy of the ECC is increased by 25%, then each line would only have 768 data bits (i.e., 12 64-bit words).

Note that if the cache is implemented with more reliable SRAM rather than PCM technology, then it is not necessary to store as many check bits in the cache as is needed in a PCM main memory, nor is it necessary to scale up the number of check bits in the cache over time. So as the number of check bits stored in the phase change main memory is increased over time, the total storage capacity of the cache is not affected. What is affected is the bandwidth coming into the cache. However, the reduction in system performance would be very application dependent and would depend on the locality of the data and number of required memory accesses. Experimental results are shown in Sec. 5.6 exploring the impact on performance.

There are a number of different options for how the cache is implemented to accommodate a gracefully degrading line size coming from the main memory.

One would be to have multiple valid bits for each original full size line in the cache. As the effective bandwidth is reduced when the ECC is strengthened in the phase change main memory, then each read from the main memory will only partially fill a line in the cache which would be indicated with the appropriate subset of valid bits for the line. If for example, the number of data bits was reduced by 50%, then each memory access would fill half of the cache line. Another way to think about it is that a cache with 1024 lines where each line originally stored 16 words would be effectively transformed to a cache with 2048 lines where each line stores only 8 words. The cache capacity doesn't change, only the effective line size changes.

Another way to implement the cache would be to adjust the associativity when the bandwidth from the PCM main memory is reduced. For example, a two-way set

associative cache could be converted to a four-way set associative cache when the line size is reduced in half.

In either of these cases, the impact of reducing the line size will depend on the locality and frequency of memory references in the application. The reduction in line size is partially offset by the increase in either the associativity or the number of lines.

This gives an overview of the approach which is general and could be used for any type of ECC. Next, one way for implementing the ECC with this scheme will be proposed which utilizes OLS codes. The advantage of using OLS codes over traditional multi-bit ECC such as BCH, LDPC codes, is that correction can be performed in a single clock cycle and the amount of time required is independent of the number of errors.

5.4 ORTHOGONAL LATIN SQUARE CODES

OLS codes, as the name suggest, are based on Latin squares. A Latin square [Hsiao 70] of order (size) m is an m x m square array of the digits 0, 1, . . . , m - ***1,*** with each row and column a permutation of the digits 0,1, … , m - 1. Two Latin squares are orthogonal if, when one Latin square is superimposed on the other, every ordered pair of elements appears only once.

As explained in [Datta 10], a t-error correcting majority decodable code works on the principle that $2t + 1$ copies of each information bit are generated from $2t + 1$ independent sources. One copy is the bit itself received from memory or any transmitting device. The other $2t$ copies are generated from $2t$ parity relations involving the bit. By choosing a set of h Latin squares that are pair-wise orthogonal, one can construct a parity check matrix such that the number of 1's in each column is $2t = h + 2$. The orthogonality condition ensures that for any bit d, there exists a set of $2t$ parity check equations orthogonal on d_i, and thus makes the code self-orthogonal and one-step majority

decodable. One-step majority decoding is the fastest parallel decoding method. The *t*-error correcting codes generated by OLS codes [Hsiao 70] have m^2 data bits and $2tm$ check bits per word.

5.5 ADAPTIVE ERROR CORRECTION CODE

The principle behind the proposed method is that as the number of permanent errors keeps increasing over time, the ECC needs to be increased in strength. The trade off comes in the form of using up more of the memory for storing the ECC check bits.

If for k data bits in a memory line, a t-error correcting code is used, then this is sufficient for all errors $\leq t$. To mitigate more than t errors, more check bits are required. They cannot be added without reducing the number of data bits per line because for off-chip main memory, the n-bit bus width transferring data and check bits from the memory to the processor is fixed such that

$$n = data\ bits + check\ bits$$

A straightforward approach for strengthening the ECC for an OLS code would be to simply directly increase t for the whole line. However, rather than doing that, it is more efficient to divide the line into fragments and increase the number of errors corrected in each fragment as will be shown in this section.

If an n-bit line consists of k data bits, it can be broken up into fragments each of size k_i, and r_i bits of ECC are separately implemented for each data fragment k_i such that

$$\sum_i \left(k_i + r_i\right) = n$$

Consider the case where all k bits have a t-error correction OLS code implemented on it. Then the total number of bits, data plus check bits, would be

$$k + 2t\sqrt{k} \quad \text{……..(5)}$$

Now if the line is broken up into fragments, each of size $k_i = k/f$ and a $t/\sqrt{f}$ - error correcting OLS code is implemented for each fragment. The total number of bits still remains

$$\left\{k/f + 2\left(t/\sqrt{f}\right)\sqrt{k/f}\right\} * f = k + 2t\sqrt{k} \quad \ldots(6)$$

Although the total number of bits is the same in both the cases, (5) and (6), the error correction capacity is different for both. In case (5), the line can withstand all error patterns affecting up to *t*-bits. In case (6), each fragment can handle up to $t/\sqrt{f}$ errors. But overall the line can handle all error patterns affecting up to $t/\sqrt{f}$ bits in each fragment and *some* error patterns affecting up to $t\sqrt{f}$ bits on the entire line. As is shown later in section 5.6, for randomly occurring errors, the property to correct some error patterns of size greater than the individual capacity of the ECC in each fragment is significant and as simulations show, a fair number of error patterns can be tolerated using this property.

So the overall idea is the following. An initial code over across all *n* bits is selected to protect the PCM memory based on characterization tests. Then during the course of operation as the number of failed cells accumulates over time, the strength of the ECC is increased by implementing ECC on increasingly smaller fragments.

Consider a numerical example to illustrate the scheme. Consider a memory line with 256 data bits. Initially a 3-error correcting OLS code is employed. Thus the total number of bits in the line is,

$$256 + 2 * 3 * \sqrt{256} = 352$$

In an enhanced ECC mode, 25% of the memory line is used to store extra check bits. Hence the total number of data bits per line now becomes 192. The rest

$$352 - 192 = 160$$

bits are used for storing ECC. But instead of implementing ECC over the entire 192 data bits at a time, the line is broken up into fragments of size 64, 64, 16, 16, 16 and 16 bits. Next a 3-error correcting OLS code is implemented on each of the 64-bit fragments and a 2-error correcting OLS code on each of the 16-bit fragments, bringing the total number of bits to

$$(64 + 2 * 3 * 8) * 2 + (16 + 2 * 2 * 4) * 4 = 352$$

But now instead of being able to correct only 3-error patterns all 2-error patterns, 99.97% of all 3-error patterns, 99.73% of all 4-error patterns and so on, up to a small fraction of 14-bit errors can be corrected. Thus the approach of breaking up a line into fragments and using separate ECC for each fragment is more efficient in terms of error correcting capacity than implementing a single ECC on the whole line.

The selection of fragment sizes and their respective ECC bits is a combinatorial problem. In the cases where there is more than one possible way to break up a memory line into identical division of data and check bits, the combination which can correct maximum number of errors is chosen.

5.6 Experimental Results

The bit error rate for a memory is defined as the number of failed bits divided by the total size of the memory (or in other words, the probability that each bit has failed). The bit error rate for PCM memories starts very small and grows over time as more cells fail. Figure 5.2 compares the proposed adaptive error correction scheme with the approach in [Zhang 09] where a 7-error BCH code is used and can tolerate bit error rates of up to 0.145%. The adaptive scheme discussed in this chapter was implemented with a line size of 1024 and starts with an initial correction capability of 3-errors. As the OS detects that the bit error rate is exceeding the strength of the ECC, 25% of the data bits

are converted to check bits by dividing the memory into fragments and adding check bits for each fragment. This process is repeated as the error rate continues to increase. As can be seen in Fig. 5.2, the proposed scheme can tolerate bit-error rates from 0.08% to a significantly higher 1.2%. The simulation for Fig. 5.2 was done on a memory of size 128MB with random injection of errors.

The error correction technique described in [Schechter 10] needs a mention here being similar to that of [Zhang 09]. In [Schechter 10] the Erroc Correction Pointer (ECP) data structure would hold the location of the failed cell and a spare cell to substitute for the failed one. Several such data structures would be maintained per line. ECP requires 1 full bit, *n* replacement bits, and *n* pointers large enough to address the original data bits. Thus the fractional space overhead S (ECP_n) for a row width *d* is

$$S(ECP_n) = \frac{1 + n + n..\log_2 d}{d}$$

Although botht these approaches, [Zhang 09] and [Schechter 10], using similar number of check bits, are very efficient when it comes to handling a fixed number of defects, neither of them would be capable of mitigating the growing number of defects as is common in phase change memories. This is where the advantage of our adaptive scheme, as shown in Fig. 5.2, becomes most prominent against other fixed error correction schemes

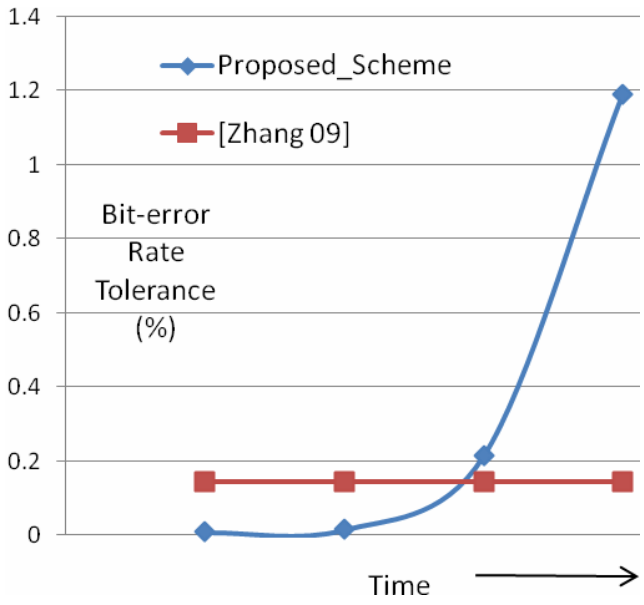

Figure 5.2: Adaptive fault tolerance

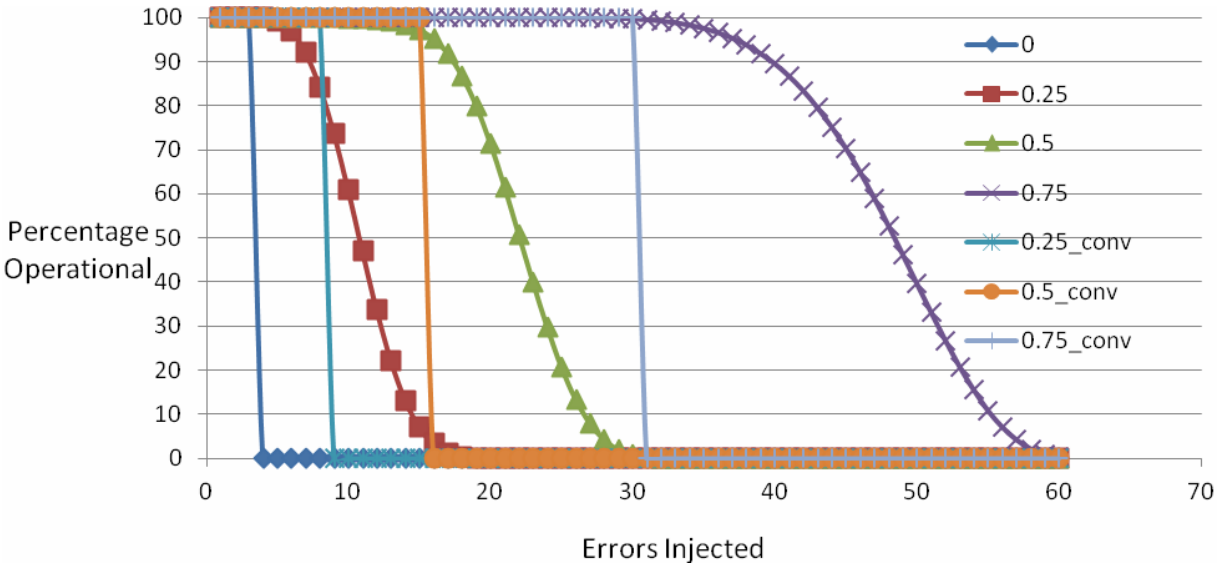

Figure 5.3: Percentage of operational cache lines versus number of errors injected (out of 100,000 experiments)

Table 5.1 shows how the error tolerance varies with size of the memory as the number of check bits is increased. For this experiment, random errors were injected into memories of various sizes. As more of the memory is used up to store extra check bits, the error tolerance grows alongside, reaching as high as 10^{-2} in the extreme where 75% of the memory has been used up to store extra check bits. For most applications, acceptable error tolerance is around 10^{-6} which shows that our proposed approach is capable of handling prohibitively high number of errors if necessary.

Table 5.1: Error Tolerance (no. of errors / no. of bits * 100) for varying line sizes

Memory Size	Fraction of Memory Used for Storing Extra Check-bits			
	0.0	0.25	0.5	0.75
128MB	0.008	0.015	0.213	1.190
256MB	0.006	0.042	0.205	1.117
1GB	0.005	0.026	0.154	0.989
4GB	0.003	0.020	0.125	0.916

The monotonic decrease in tolerance with increasing size can be explained by the fact that as the number of lines increase, the difference

$$n * E[\text{one line}] \sim E[n \text{ lines}]$$

is likely to increase, where n is the number of lines, $E[\text{one line}]$ is expected error tolerance of a single line and $E[n \text{ lines}]$ is the error tolerance of n lines. This mirrors a likely scenario where errors will accumulate faster on some lines than others unless the data read/write pattern is absolutely random, which is not the usual case due to data locality.

Another aspect of evaluating the proposed scheme is to study the distribution of error tolerance in each line for different ECC configurations. Figure 5.3 shows results for a 1024 bit line in which the initial stating point is a 3-bit error correcting ECC and then

25% of the data bits were converted to check bits, and then 50%, and finally 75%. Errors were injected at random, and Fig. 5.3 shows the percentage of lines that have not failed across 100,000 experiments. The *x*-axis corresponds to the number of errors injected in the line, and the *y*-axis corresponds to the percentage of lines that were able to tolerate that many errors for different configurations of the ECC. Results are shown for both the ECC scheme described in Sec. 5.5 which breaks up the line into fragments and implements ECC separately for each fragment, and the conventional case where the ECC was implemented across all the data bits at once. As can be seen from the results, the fragmented ECC scheme can easily tolerate more errors than the conventional method. The former is able to tolerate 20% more errors per line at 90% probability, than the conventional method, when half the line is used for storing check bits. When $3/4^{th}$ of the line is used for storing check bits, the fragmented scheme can tolerate 33% more errors at 90% probability.

Figures 5.4 and 5.5 show the effect of reducing memory line size to accommodate extra check bits. One way to implement this, as was described in Sec. 5.3, is to adjust the associativity and line size of the cache. In both Figs. 5.4 and 5.5, the *y*-axis plots cycles per instruction (CPI) for a set of SPEC2006 [Spec 06] benchmarks across different cache configurations defined as follows:

cache_config1 – line size 512B, associativity 4
cache_config2 – line size 256B, associativity 8
cache_config3 – line size 128B, associativity 16

The CPI was calculated assuming single cycle for all non-memory instructions and five cycles [Wulf 95] for instructions that caused a cache miss and needed to access main memory. As can be seen from the figures, there is little degradation on performance

for reduced line sizes and increasing associativity. Moreover, the fact that these changes are expected at an interval of every few years lessens the performance impact.

The simulations were done using Pin [Pin 04], a dynamic instrumentation tool. Pin was used to obtain memory traces of the benchmarks for various cache configurations. These memory traces were then used as an input to the DineroIV [Dinero IV] cache simulator to generate cache miss rates.

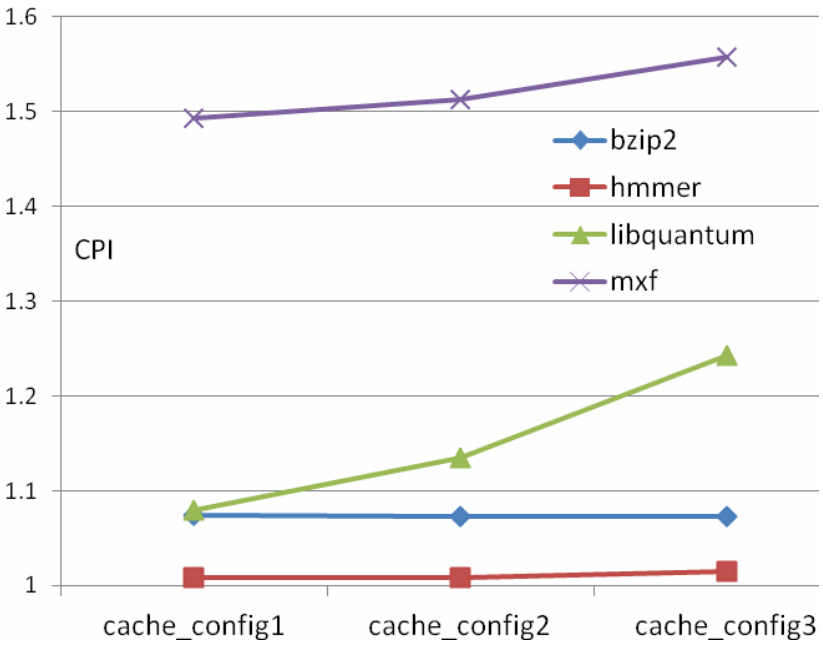

Figure 5.4: Variation of CPI for different cache configurations for four different SPEC2006 benchmarks for 64KB cache

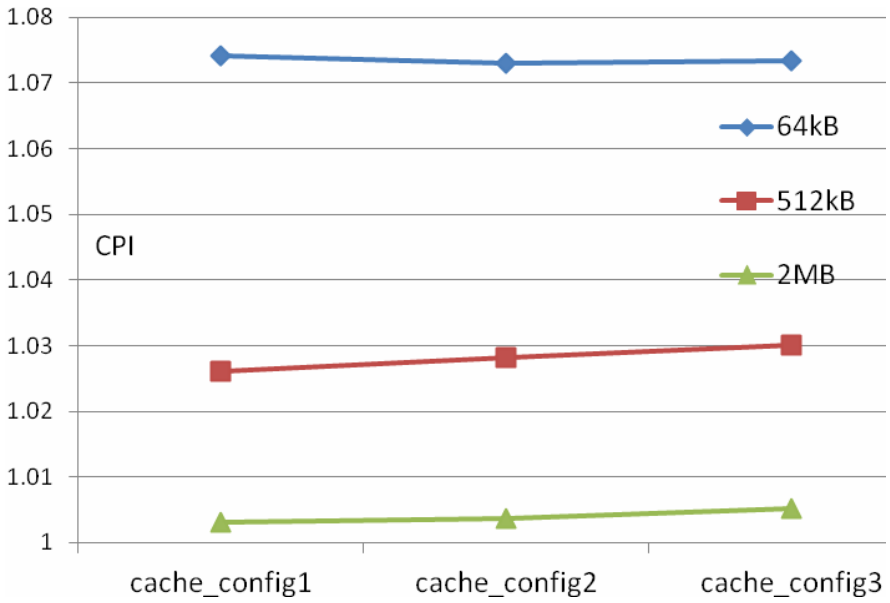

Figure 5.5: Variation of CPI for different cache configurations across different cache sizes for the SPEC2006 benchmark *bzip2*

5.7 CONCLUSIONS

This chapter described an adaptive error tolerance scheme that can extend to 8x more error tolerance than that of [Zhang 09] for similar initial redundancy. As the PCM memory degrades to the point where it exceeds the capability of the method in [Zhang 09] to continue operation, the proposed method can continue operation by adaptively increasing the ECC. The performance of the memory will gracefully degrade due to reducing the effective line size for each memory read to service a cache miss. However, the impact of this can be minimized by careful cache design since the total cache storage capacity as a whole is not impacted.

Chapter 6: Conclusions and Future Work

6.1 CONCLUSIONS

In this dissertation we have looked at various approaches to efficiently detect and correct online errors in memories. Memories are the densest structures in modern day microprocessors and also occupy the highest percentage of real estate on a chip. Thus it is absolutely essential that online fault tolerance in memories be handled with minimal overhead. That has been the common goal in every different approach we have discussed in this dissertation – how to get rid of that extra redundancy in the error correction code and make it more efficient.

We began in chapter 2 by presenting an extremely low-cost method that utilizes unused spare columns from the memory repair process to increase reliability of the existing code. We store additional check bits in the leftover spare columns and show considerable improvement in the mis-correction probabilities of SEC-DED (Single Error Correcting-Double Error Detecting) and SEC-DAEC (Single Error Correcting-Double Adjacent Error Correcting) code. This is achieved at the cost of very little extra hardware making this an attractive method. Chapter 3 looks at transforming multi-bit correcting OLS (Orthogonal Latin Square) code into burst error correcting codes. The motivation for this approach arises from the fact that most multi-bit errors are likely to affect adjacent bits. As before, our goal was to achieve the desired properties of burst error correction with minimal number of extra check bits. As our results show the check bit overhead actually goes down with increasing code size.

In chapter 4 we work towards making caches more resilient to errors at low voltages. Ultra low voltage caches, while very attractive from a power saving point of view, suffers from the increased failure rates that manifest at lower voltages, thereby

necessitating stronger, multi-bit correcting error codes with high redundancy. In our work we show that customizing the error code on a chip by chip basis leads to considerable savings in terms of check bit overhead. The customization is done with the help of a defect map – a list of cells vulnerable at lower voltages, generated at manufacture time by memory characterization tests. Our choice of ECC was OLS codes, because of its ability to correct multiple errors in a single cycle and more importantly for the modular nature of the code which allows reduction in check bits with relative ease.

Finally in chapter 5 we propose an adaptive error correction scheme for phase change memories, a new generation of memory elements. Phase change memories suffer from limited write endurance, thus needing stronger ECC with high redundancy. We discuss a novel method where by involving the operating system, the strength of the ECC is increased in steps in conjunction with the failure rate. This leads to a higher lifetime expectancy as well as graceful use of memory capacity to store extra check bits.

6.2 FUTURE WORK

As we move deeper into technological nodes, memories are likely to grow denser and more complex. This means that seamless operation of systems would require that online error correction/detection of memories is handled with minimum overhead and latency. This leaves ample room for improvement in terms of error correction codes.

In relation to the work presented in chapter 2, one possible extension could be to look at ways of making use of partially defective columns. Instead of discarding them completely during memory repair, if extra check bits can be stored in the good cells to improve system reliability.

For our work reducing check bit overhead by customization of OLS code, future work could look at other possible codes like BCH or LDPC, which are inherently less redundant than OLS codes.

For phase change memories, unless an alternate work around is found for the limited write endurance (possibly at the device level), error correction will have to handled at the system level. More approaches involving the operating system could look to explore the unique fault model of PCM systems, where error rate is always monotonically increasing. Ideas targeted towards multi-bit PCM systems where the fault model is usually asymmetric between different threshold levels should also be investigated.

Bibliography

[Berlekamp 68] Berlekamp, E. R., *Algebraic Coding Theory*, McGraw-Hill, New York, 1968.

[Bossen 70] Bossen, D. C., ”b-Adjacent Error Correction”, *IBM Journal of Research and Development,* Vol. 14, pp. 402-408, Jul. 1970.

[Chang 07] Chang, J., Ming Huang; Shoemaker, J., Benoit, J., Szu-Liang Chen, Wei Chen, Siufu Chiu, Ganesan, R., Leong, G., Lukka, V., Rusu, S., Srivastava, D., “The 65-nm 16-MB Shared On-Diie L3Cache for the Dual-Core Intel® Xeon Processor 7100 Series,,” *IEEE Journal of Soliid-state Circuits,* Vol.. 42, no. 4,,pp. 846-852, April,, 2007.

[Chen 96] Chen, C. L., ”Symbol Error Correcting Codes for Memory Applications”, *Proc. of Fault Tolerant Computing Systems,* pp. 200-207, 1996.

[Chen 08] B. Chen and X. Zhang, “Error Correction for Multi-Level NAND Flash Memory Using Reed-Solomon Codes,” *IEEE Workshop on Singal Processing Systems (SiPS)*, pp. 94-99, Oct. 2008.

[Chisti 09] Chishti Zeshan, Alameldeen Alaa R., Wilkerson Chris, Wu Wei and Lu S.-L., “Improving Cache Lifetime Reliability at Ultra-low Voltages”, *Proc. Of International Symposium on Microarchitecture*, Dec. 2009.

[Cormen 01] Cormen, T., Leiserson, C., Rivest, R., Stein, C., “Introduction to Algorithms”, *MIT Press*, 2001.

[Datta 09] R. Datta, N. A. Touba, "Exploiting Unused Spare Columns to Improve Memory ECC", *Proc. of IEEE VLSI Test Symposium*, pp. 47-52, May 2009.

[Datta 10] R. Datta, N. Touba, "Post-Manufacturing ECC Customization Based on Orthogonal Latin Square Codes and Its Application to Ultra-Low Power Caches", *Proc. of International Test Conference*, Paper 7.2, 2010.

[Datta 11a] R. Datta, N. A. Touba, "Designing a Fast and Adaptive Error Correction Scheme for Increasing the Lifetime of Phase Change Memories", *Proc. of IEEE VLSI Test Symposium 2011*, pp. 134-139, May 2011.

[Datta 11b] R. Datta, N. A. Touba, "Generating Burst-error Correcting Codes from Orthogonal Latin Square Codes – a Graph Theoretic Approach", *Proc. of IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 367-373, October 2011.

[Dinero IV] J. Edler, M. D. Hill, “DineroIV – Trace Driven Uniprocessor Cache simulator for memory references,” http://www.cs.wisc.edu/~markhill/DineroIV.

[Dutta 07] Dutta, A., and N.A. Touba, "Multiple-Bit Upset Tolerant Memory Using a Selective Cycle Avoidance Based SEC-DED-DEAC Code," *Prof. of VLSI Test Symopsium*, pp. 349-354, 2007.

[Ferreira 10] A.P. Ferreira, M. Zhou, S. Bock, B. Childers, R. Melhem, D. Mosse, "Increasing PCM Main Memory Lifetime", *Design Automation and Test in Europe*, pp. 914-919, 2010.

[Hamming 50] Hamming, R.W., "Error Correcting and Error Detecting Codes", *Bell Sys. Tech. Journal*, Vol. 29, pp. 147-160, Apr. 1950.

[Heidel 08] D. Heidel, P. Marshall, K. LaBel, J. Schwank, K. Rodbell, M. Hakey, M. Berg, P. Dodd, M. Friendlich, A. Phan, C. Seidleck, M. Shaneyfelt, and M. Xapsos, "Low energy proton single-event-upset test results on 65 nm SOI SRAM," *IEEE Trans. Nucl. Sci.*, vol. 55, no. 6, pp. 3394–3400, Dec. 2008

[Hsiao 70] Hsiao, M. Y., "A Class of Optimal Minimum Odd-weight-column SEC-DED codes", *IBM Journal of Research and Development*, Vol. 14, pp. 395-401, 1970.

[Kawakami 04] Kawakami, Y., et al., "Investigation of Soft Error Rate Including Multi-Bit Upsets in Advanced SRAM Using Neutron Irradiation Test and 3D Mixed-mode Device Simulation", *Proc. of IEEE Int'l Electronic Device Meeting*, pp. 945-948, Dec. 2004.

[Kim 98] Kim, I., Y. Zorian, G. Komoriya, H. Pham, F.P. Higgins, and J.L. Lewandowski, "Built In Self Repair for Embedded High Density SRAM," *Proc. of International Test Conference*, pp. 1112-1119, 1998.

[Kim 07] Jangwoo Kim, Hardavellas, N., Ken Mai, Falsafi, B., Hoe, J.C, "Multi-bit Error Tolerant Caches Using Two-Dimensional Error Coding", *Proc. of International Symposium on Micro-architectures*, December 2007.

[Kim 10] Kim, C., Rhee, S., Kim, J., Jee, Y., "Product Reed-Solomon Codes for Implementing NAND Flash Controller on FPGA Chip", *Second International Conference on Computer Engineering and Application*, pp 281-285, 2010

[Kleinberg 06] Kleinberg, J., Tardos, E., "Algortihm Design", *Pearson/Addison-Wesley*, pp/ 485-490, 2006.

[Lala 78] Lala, P. K.," An Adaptive Double Error Correction Scheme for Semiconductor Memory Systems," *Digital Processes,* Vol. 4, pp. 237-243, 1978.

[Lawrence 09] R. Lawrence, J. Ross, N. Haddad, R. Reed, and D. Albrecht, "Soft error sensitivities in 90 nm bulk CMOS SRAMs," in *Proc. of IEEE Radiation Effects Data Workshop*, pp. 123–126, Jul. 2009.

[Lee 09] B. C. Lee, E. Ipek, O. Mutlu, D. Burger, "Architecting Phase Change Memory as a Scalable DRAM Alternative", *Proc. of International Symposium of Computer Architecture*, pp. 2-13, 2009.

[Lin 83] Lin, S. Costello, D., .*Error Control Coding: Fundamentals and Applications*. Prentice Hall, 1983.

[Maiz 03] Maiz, J., Hareland, S., Zhang, K., Armstrong, P., "Characterization of multi-bit soft error events in advanced SRAMs", *Proc. of IEEE International Electron Devices Meeting*, pp. 21.4.1-21.4.4.

[Makihara 00] Makihara, A., et al., "Analysis of Single-Ion Multiple-Bit Upset in High-Density DRAMS", *IEEE Trans. on Nuclear Science*, Vol. 47, No. 6, Dec. 2000.

[Micheloni 06] R. Micheloni, R. Ravasio, A. Marelli, E. Alice, V. Altieri, A. Bovino, L. Crippa1, E. Di Martino, L. 'Onofrio, A. Gambardella, E. Grillea, G. Guerra, D. Kim, C. Missiroli, I. Motta, A. Prisco, G. Ragone, M. Romano, M. Sangalli, P. Sauro, M. Scotti, S. Won, "A 4Gb 2b/cell NAND Flash Memory with Embedded 5b BCH ECC for 36MB/s System Read Throughput", *IEEE Inter. Solid-State Circuits Conf*, pp. 497-506, February 2006.

[Numonyx 07] Numonyx. The basics of phase change memory technology. In *Numonyx White Paper*, 2007.

[Peterson 72] Peterson, W.W., and E.J. Weldon, Error Correcting Codes, MIT Press, Cambridge, MA, 1972.

[Pin 04] V.J.Reddi, A. Settle, D.A.Connors, R.S.Cohen, "PIN: A Binary Instrumentation Tool for Computer Architecture Research and Education", *Proc. of Workshop on Computer Architecture Education*, June 2004.

[Pradhan 96] Pradhan, D.K., *Fault-Tolerant Computer System Design*, Prentice Hall, Upper Saddle River, NJ, 1996.

[Reed 60] Reed, I. S., and G. Solomon, "Polynomial Codes Over Certain Fields", *J. Soc. Ind. Appl. Mat.,* Vol. 8, pp. 300-304, Jun. 1960.

[Richter 08] Richter, M., K. Oberlaender, and M. Goessel, "New Linear SEC-DED Codes with Reduced Triple Error Miscorrection Probability", *Proc. of International On-Line Testing Symposium*, pp. 37-42, 2008.

[Rodbell 07] K. Rodbell, D. Heidel, H. Tang, M. Gordon, P. Oldiges, and C. Murray, "Low-energy proton-induced single-event-upsets in 65 nm node, siliconon-insulator, latches and memory cells," *IEEE Trans. Nucl. Sci.*, vol. 54, no. 6, pp. 2474–2479, Dec. 2007.

[Samsung 06] Samsung. Samsung introduces the next generation of nonvolatile memory - pram. In *Samsung News Release*, Sept. 2006.

[Satoh 00] Satoh, S., Y. Tosaka, S.A. Wender, "Geometric Effect of Multiple-bit Soft Errors Induced by Cosmic-ray Neutrons on DRAMs", *Proc. of IEEE International Electronic Device Meeting*, pp. 310-312, June 2000.

[Schechter 10] Stuart Schechter, Gabriel H. Loh, Karin Strauss, Doug Burger, "Use ECP, not ECC, for Hard Failures in Resistive Memories", *Proc. of 37th International Symposium on Computer Architecture*, pp.141-152, June 2010.

[Seifert 08] Seifert, N., Gill, B., Foley, K., Relangi, P., "Multi-cell upset probabilities of 45nm high-k + metal gate SRAM devices in terrestrial and space environments", *IEEE International Reliability Physics Symposium*, pp. 181-186, 2008.

[Sierawski 09] B. Sierawski, J. Pellish, R. Reed, R. Schrimpf, K. Warren, R. Weller, M. Mendenhall, J. Black, A. Tipton, M. Xapsos, R. Baumann, X. Deng, M. Campola, M. Friendlich, H. Kim, A. Phan, and C. Seidleck, "Impact of low-energy proton induced upsets on test methods and rate predictions," *IEEE Trans. Nucl. Sci.*, vol. 6, no. 6, pp. 3085–3092, Dec. 2009.

[Sierawski 11] Sierawski, B.D., Reed, R.A., Mendenhall, M.H., Weller, R.A., Schrimpf, R.D., Wen, S., Wong, R., Tam, N., Baumann, R.C., "Effects of scaling on muon-induced soft errors", *IEEE Reliability Physics Symposium*, April 2011

[Spec 06] Standard Performance Evaluation Corporation. SPEC CPU2006 Benchmarks. http://www.spec.org/cpu2006/.

[Stapper 92] Stapper, Charles H., Hsing-san Lee, "Synergistic Fault-Tolerance for Memory Chips", *Proc of IEEE Transactions on Computers*, Vol. 41, No. 9, pp 1078-1087, Sep. 1992.

[Su 05] Su, Ching-Lung, Yeh, Yi-Ting, Wu, Cheng-Wen, "An Integrated ECC and Redundancy Repair Scheme for Memory Reliability Enhancement," *Proc. of Defect and Fault Tolerance in VLSI Systems*, pp. 81-89, 2005.

[Taur 98] Taur, Y., Ning, T.H., *Fundamentals of Modern VLSI Decices*, Cambridge University Press, 1998

[Vazirani 04] Vazirani, V.V., *Approximation Algorithms*, Springer, 2004.

[Wilkerson 08] Wilkerson, C. Hongliang Gao, Alameldeen, A.R., Chishti, Z., Khellah, M., Shih-Lien Lu, "Trading of Cache Capacity for Reliability to Enable Low Voltage Operation", *Proc. of International Symposium on Computer Architecture*, pp.203-214, June 2008.

[Wolf 69] Wolf, J. K.,"Adding Two Information Symbols to Certain Non-Binary BCH Codes and Some Applications", *Bell Systems Technical Journal*, Vol. 48, pp. 2405-2424, 1969.

[Wulf 95] W.A. Wulf, S.A. McKee, "Hitting the memory wall: implications of the obvious", *ACM SIGARCH Computer Architecture News*, Vol. 23 Issue 1, Mar.1995.

[Xu 10] W. Xu, T. Zhang, "Using Time-Aware Memory Sensing to Address Resistance Drift Issue in Multi-Level Phase Change Memory", *Proc. of International Symposium of Quality Electronic Design*, pp. 356-361, 2010.

[Zhang 09] W. Zhang, T. Li, “Exploring Phase Change Memory and 3D Die-Stacking for Power/Thermal Friendly, Fast and Durable Memory Architectures”, *Int. Conference on Parallel Architectures and Compiler Techniques*, pp. 101-112, 2009.

[Zorian 03] Zorian, Y., and S. Skoukourian, “Embedded-Memory Test and Repair: Infrastructure IP for SOC Yield,” *IEEE Design & Test of Computers*, Vol. 20, Issue 3, pp. 58-66, May 2003.

Vita

Rudrajit Datta was born in Kolkata, India on April 16, 1984. He received his Bachelor of Technology degree in Electrical Engineering from the Indian Institute of Technology, Kharagpur, India in 2007. He joined the University of Texas at Austin in 2007 and has been working in the Computer Aided Testing (CAT) Lab under the supervision of Prof. Nur Touba since, receiving his MS in Electrical and Computer Engineering with a thesis in Memory Reliability in 2009. He is currently pursuing his PhD degree researching on efficient error correction codes for improving online fault and defect tolerance of semiconductor memories.

Permanent Address: 301 Prince Anwar Shah Road,
Flat # 6C,
Kolkata – 700 045,
West Bengal, India.

This dissertation was typed by Rudrajit Datta.