Design and Analysis of an Adjacent Multi-bit Error Correcting Code for Nanoscale SRAMs

by

Adam Neale

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical & Computer Engineering

Waterloo, Ontario, Canada, 2014

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Increasing static random access memory (SRAM) bitcell density is a major driving force for semiconductor technology scaling. The industry standard 2x reduction in SRAM bitcell area per technology node has lead to a proliferation in memory intensive applications as greater memory system capacity can be realized per unit area. Coupled with this increasing capacity is an increasing SRAM system-level soft error rate (SER). Soft errors, caused by galactic radiation and radioactive chip packaging material corrupt a bitcell's data-state and are a potential cause of catastrophic system failures. Further, reductions in device geometries, design rules, and sensitive node capacitances increase the probability of multiple adjacent bitcells being upset per particle strike to over 30% of the total SER below the 45 nm process node. Traditionally, these upsets have been addressed using a simple error correction code (ECC) combined with word interleaving. With continued scaling however, errors beyond this setup begin to emerge. Although more powerful ECCs exist, they come at an increased overhead in terms of area and latency. Additionally, interleaving adds complexity to the system and may not always be feasible for the given architecture.

In this thesis, a new class of ECC targeted toward adjacent multi-bit upsets (MBU) is proposed and analyzed. These codes present a tradeoff between the currently popular single error correcting-double error detecting (SEC-DED) ECCs used in SRAMs (that are unable to correct MBUs), and the more robust multi-bit ECC schemes used for MBU reliability. The proposed codes are evaluated and compared against other ECCs using a custom test suite and multi-bit error channel model developed in Matlab as well as Verilog hardware description language (HDL) implementations synthesized using Synopsys Design Compiler and a commercial 65 nm bulk CMOS standard cell library. Simulation results show that for the same check-bit overhead as a conventional 64 data-bit SEC-DED code, the proposed scheme provides a corrected-SER approximately equal to the Bose-Chaudhuri-Hocquenghem (BCH) double error correcting (DEC) code, and a 4.38x improvement over the SEC-DED code in the same error channel. While, for 3 additional check-bits (still 3 less than the BCH DEC code), a triple adjacent error correcting version of the proposed code provides a 2.35x improvement in corrected-SER over the BCH DEC code for 90.9% less ECC circuit area and 17.4% less error correction delay.

For further verification, a 0.4-1.0 V 75 kb single-cycle SRAM macro protected with a programmable, up-to-3-adjacent-bit-correcting version of the proposed ECC has been fabricated in a commercial 28 nm bulk CMOS process. The SRAM macro has undergone neutron irradiation testing at the TRIUMF Neutron Irradiation Facility in Vancouver, Canada. Measurements results show a 189x improvement in SER over an unprotected memory with no ECC enabled and a 5x improvement over a traditional single-error-correction (SEC) code at 0.5 V using 1-way interleaving for the same number of check-bits. This is comparable with the 4.38x improvement observed in simulation. Measurement results confirm an

average active energy of 0.015 fJ/bit at 0.4 V, and average 80 mV reduction in VDD_{MIN} across eight packaged chips by enabling the ECC. Both the SRAM macro and ECC circuit were designed for dynamic voltage and frequency scaling for both nominal and low voltage applications using a full-custom circuit design flow.

Acknowledgements

I would like to take this opportunity to express my gratitude and thanks to my supervisor **Professor Manoj Sachdev** at the University of Waterloo. I could not have asked for a better supervisor to work with. Without his guidance, knowledge, kindness, and patience, this work would not have been possible. I would also like to thank my examining committee **Dr. Bill Bishop**, **Professor Vincent Gaudet**, **Professor Edlyn Teske-Wilson**, and **Professor Ali Sheikholeslami**. I realize how valuable everyone's time is, and how much of it is required to be a part of a Ph.D. examining committee. Thank you for agreeing to be a part of mine, and thank you for all of your valuable comments and suggestions.

It has been a great pleasure to be a part of the CMOS Design and Reliability (CDR) Group for the last six years throughout this degree and my M.A.Sc. degree. The dedication and talent within this group and its simple but powerful motto of "Aim High" has always inspired me to strive for my best in everything I do. My sincere appreciation goes out to all of the past and current members of this group especially Dr. David Rennie, Dr. Jaspal Singh Shah, and Pierce Chuang.

I would also like to thank the varsity track and field team for supporting me throughout my studies. Having running as an outlet has allowed me to stay physically active, and remain (quasi-)sane over the last four years. Thank you Shane, Kate, Jacob, Oly, Lawrence, Trevor, Justin, Chantel, Kofi, Jenny, Nathan, Mo, Tommy, and everyone else.

Lastly, I would like to acknowledge the funding that I received throughout the duration of my degree from the Natural Sciences and Engineering Research Council (NSERC), Waterloo Institute for Nanotechnology (WIN), and the University of Waterloo. Thank you for believing in my ability to do research.

Contents

List of Tables

List of Figures

List of Abbreviations

$GF(2)$ Binary Galois Field

Q_{col} Collected Charge

Q_{crit} Critical Charge

$\mathbf{I}_r$ $r \times r$ Identity Matrix

a_n Neutron Fluence Acceleration Factor

a_t Irradiation Time Acceleration Factor

d_{min} Minimum Hamming Distance

G-matrix ECC Generator Matrix

H-matrix ECC Parity Check Matrix

6T 6 Transistors

BCH Bose Chaudhuri Hocquenghem

BL/BLB SRAM Complementary Bitline and Bitline Bar Pair

CMOS Complementary Metal Oxide Semiconductor

DAEC Double Adjacent Error Correction

DEC Double Error Correcting

DEC-TED Double Error Correcting Triple Error Detecting

DED Double Error Detecting

DICE Dual Interlocking Cell Element

DRAM Dynamic Random Access Memory

DVFS Dynamic Voltage and Frequency Scaling

ECC Error Correcting Code, or Error Correction Code

EG-LDPC Euclidean Geometry Low Density Parity Check

FF Fast n-type Transistor Devices, Fast p-type Transistor Devices

FIT Failures in Time, 1 FIT = one failure in 10^9 device-hours

HDL Hardware Description Language

HK+MG High κ Dielectric plus Metal Gate

ICB Increased Check-Bit codes

IIMS Increased Identity Matrix Size codes

JEDEC Joint Electron Device Engineering Council

LET Linear Energy Transfer

MBU Multi-Bit Upset

MCU Multi-Cell Upset

NIF Neutron Irradiation Facility

PCB Printed Circuit Board

PGA Pin Grid Array

RAM Random Access Memory

RS Reed-Solomon

S*b*EC-D*b*ED Single Byte Error Correcting Double Byte Error Detecting

SBU Single Bit Upset

SEC Single Error Correcting

SEC-DED Single Error Correcting Double Error Detecting

SEC-DED-xAEC-yAED Single Error Correcting Double Error Detecting x-bits Adjacent Error Correcting y-bits Adjacent Error Detecting

SEC-DED-DAEC Single Error Correcting Double Error Detecting Double Adjacent Error Correcting

SET Single Event Transient

SEU Single Event Upset

SOI Silicon on Insulator

SPICE Simulation Program with Integrated Circuit Emphasis

SRAM Static Random Access Memory

SS Slow n-type Transistor Devices, Slow p-type Transistor Devices

TAEC Triple Adjacent Error Correction

TRIUMF Tri-University Meson Facility

TT Typical n-type Transistor Devices, Typical p-type Transistor Devices

UE Uncorrectable Error

VDD$_{MIN}$ Minimum Operating Supply Voltage

VLSI Very Large Scale Integration

WL SRAM Wordline

xAEC A variable degree of Adjacent Error Correction

yAED A variable degree of Adjacent Error Detection

zBED A variable degree of Burst Error Detection

Chapter 1

Introduction

This chapter provides an introduction to the soft error problem in nanoscale integrated circuits with an emphasis on SRAMs. The sources of soft errors and their underlying mechanisms are first described, and the increasing SRAM soft error rate in light of technology scaling is used as the main motivating factor driving this research.

1.1 Problem Statement

Driven by consumer demand for increased functionality and reduced power consumption, transistor device dimensions and chip operating voltages for the complementary metal oxide semiconductor (CMOS) technology used in computer electronics continues to scale. The increased demand for chip functionality has caused a proliferation in memory intensive applications. New applications requiring large capacity memory modules can occupy a considerable portion of a chip's total area. To fulfill the demand for increasing on-chip storage capacity, while mitigating against chip real estate penalties, embedded memories are being designed using state-of-the-art technologies with minimum features sizes currently on the order of 14-16 nm [1, 2, 3]. These gains however do not come without an associated cost. Technology scaling, coupled with reduced operating voltage, can lead to an increase in a memory cell's sensitivity to external radiation effects [4]. This issue is particularly noticeable in static random access memory (SRAM), and Flash-based memories [5]. *As devices scale deep into the sub-45 nm regime, radiation induced soft errors become a serious reliability challenge facing embedded memories* [6].

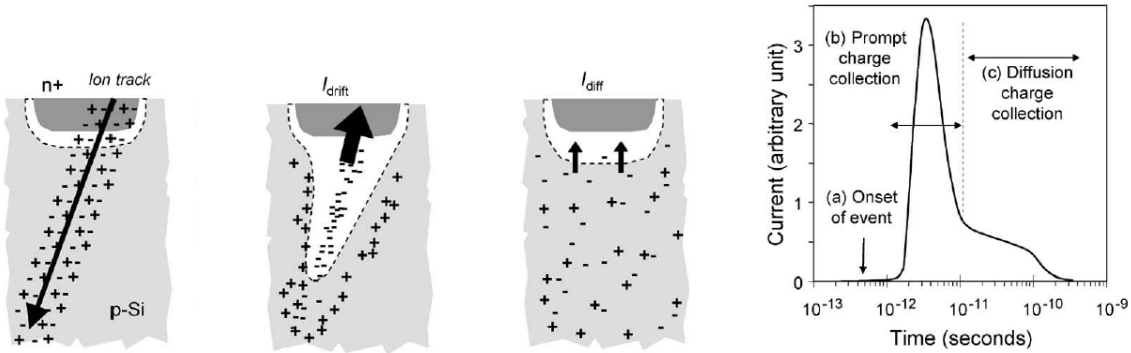
Soft errors arise as a result of energetic neutrons from galactic particles and from alpha particles emitted by chip packaging material [7]. Originally termed "antenna pickup", seemingly random upsets in digital electronics were reported as early as the 1940's, but it was not until the 1970's that radiation was considered to be a potential source of these

upsets [7]. In 1978, Intel reported that radioactive uranium contamination was causing "soft fails" in some of their dynamic random access memory (DRAM) and microprocessor chips [8], then in 1979, Ziegler and Lanford predicted that cosmic rays could cause a major electronic reliability problem at both terrestrial sites and aircraft altitudes [9]. The high energy particles create free electron-hole pairs as they pass through semiconductor devices which can lead to large current/voltage transients if they come within close proximity of a transistor's depletion region. These single event transients (SET) appear at circuit nodes and can unintentionally corrupt data stored in memory cells and other storage elements. When a SET leads to the data corruption of one of these storage elements it is defined as a single event upset (SEU). An SEU corrupting a single-bit of data is classified as a single bit upset (SBU), while an SEU corrupting multiple data cells is defined as a multi-cell upset (MCU). A MCU that corrupts multiple bits within the same logical memory word is defined as a multi-bit upset (MBU). Since each of these upsets corrupt the data state but do not necessarily permanently damage the device itself, they are referred to as soft errors.

Until recently, large node capacitances and high noise margins in storage devices have prevented soft errors at ground level from becoming a major concern. As a byproduct of aggressive technology scaling and increased bitcell packing density however, the nature of soft errors is changing, and designers are being forced to deal with these issues during earlier stages of the design cycle. Although many solutions have been developed for high reliability in extreme environments such as military and aerospace applications, they are not necessarily being adopted for commercial applications at ground level due to the significant area, performance, and power consumption penalties they incur where these metrics are at a higher premium. Additionally, *as bit-cell packing densities increase, many adjacent cells are now being corrupted by a single SEU as the external radiation source remains constant. SEUs that would have once caused only a single SBU are now causing MBU and MCUs. This leads to the failure of traditional error mitigation strategies, and severely limits the prospect of continued device scaling* [10].

1.2 Soft Error Mechanisms

When an energetic cosmic neutron enters a silicon substrate, it can either pass directly through the substrate, or cause a disturbance by striking an atom, thereby generating an ion [11]. Alpha particles are already ionized and therefore can cause a disturbance directly on contact with the substrate. The magnitude of these disturbances depends on the linear energy transfer (LET) of the ion or particle, and is measured in megaelectron volt square centimeter per milligram (MeV-cm^2/mg). The LET is dependent on the mass and energy of the particle, and the material through which it is traveling. The reverse-biased p-n junctions are the most charge-sensitive areas of a circuit [4]. This is due to the



(a) (b) (c) (d)

Figure 1.1: Charge generation and collection phases in a reverse-biased junction and the resultant current pulse caused by the passage of a high-energy ion [4].

high electric field within their depletion region allowing for efficient charge collection [7]. Charge collection generally occurs within a few microns of the junction, and the collected charge (Q_{col}) for these events range from 1 to several 100 fC depending on the type of ion, its trajectory, and energy over the path through or near the junction.

At the onset of an ionizing radiation event, as shown in Figure 1.1(a), a cylindrical track of electron-hole pairs is created in the wake of the incident ion's path. When one of these ion tracks is in the vicinity of a reverse biased junction, carriers are quickly collected by the junction's electric field creating a large current/voltage transient at the particular circuit node Figure 1.1(b). This high intensity charge collection process happens briefly over the period of no more than a nanosecond, and is then followed by more a subdued phase of diffusion dominated charge collection lasting over several hundred nanoseconds Figure 1.1(c). This phase continues until all of the excess carriers have been collected, recombined, or diffused away from the junction area. The resulting current transient associated with this phenomena is shown in Figure 1.1(d).

The amount of charge collected at a junction is inversely proportional to the distance from which the radiation event occurs. The farther away the strike is from the junction, the lower the probability that the event will cause an upset. The overall charge collection process is further complicated however, as circuit nodes do not exist in isolation, but rather as part of a more complex circuit structure with many nodes in close proximity to one another. The magnitude of Q_{col} depends on a complex combination of factors including the size of the device, biasing of the surrounding circuit nodes, substrate structure, device doping, the type of ion, its energy, its trajectory, the initial position of the event within the device, and the data-state of the device itself.

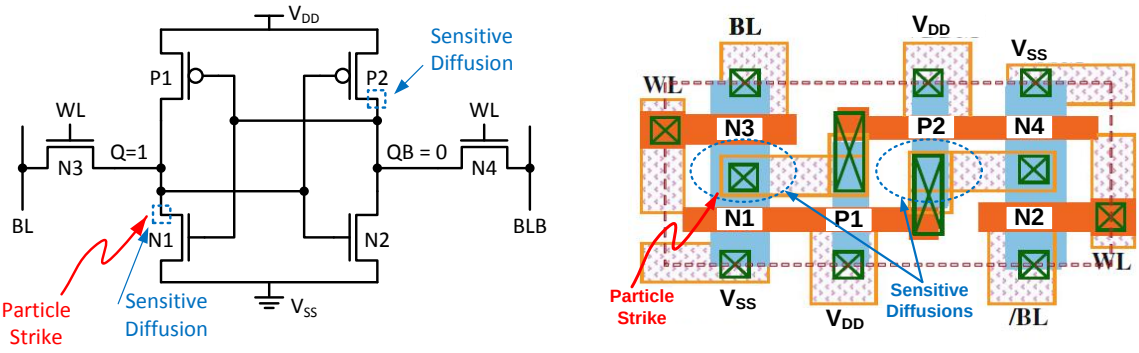
For an upset to occur, the Q_{col} must exceed a certain critical value. This minimum critical charge threshold (Q_{crit}) is defined as the minimum amount of charge necessary to trigger a change in the data state of the storage device. A SET will only cause an upset in the event that $Q_{col} > Q_{crit}$, otherwise the storage element will be able to return to its original state, and no upset will occur. The Q_{crit} has several dependencies of its own including, node capacitance, operating voltage, and strength of any feedback components, and is used as a figure of merit to assess the soft error susceptibility of a particular storage element.

The soft error rate (SER) is used to measure the frequency of occurrence of soft errors. It is measured in terms of failures in time (FIT), where one FIT is the equivalent to one failure in 10^9 device-hours. SER is typically normalized either per-bit to describe the cell-level soft error susceptibility or per-device as a measure of the system-level susceptibility. To put the SER into perspective, the aggregate failure rate for traditional "hard" failure mechanisms (such as gate oxide breakdown, metal electronmigration, latch-up, etc...) in advanced technologies is typically in the range of 50-200 FIT/device. Whereas the unabated SER for memories can easily exceed 50 000 FIT/device [4]. As an example, if we consider a 16 MB SRAM (as in the Intel Xeon L3 cache [12]) and assume a bit-level SER of 100-1000 FIT/Mb [5], this provides between 19 200-192 000 FIT/device.

While Q_{crit} remains essentially unchanged for scaled DRAMs, the critical charge in SRAM and Flash memories scale with the technology, and as memory bit-cell densities increase, system-level soft error rates continue to grow [4, 5]. Due to the different data storage mechanisms and shear contrast in capacity between Flash and SRAM memories, mitigation design approaches exploiting these differences can lead to more targeted, memory-type-specific solutions. Since on-chip embedded SRAM constitutes more that 50% of the die area for state-of-the-art microprocessors and systems-on-chip (with this value expected to increase in the future), this work is in the context of SRAM soft error mitigation. Although many of the concepts could be transferred to other memory types, SRAM scaling limitations are at the forefront computer memory research, and are of immediate concern.

1.3 Soft Errors in SRAM

The conventional six-transistor (6T) SRAM bit-cell, shown in Figure 1.2, stores one bit of data and its complement in a cross coupled inverter pair. Since the inverters continuously drive one another, the cell is able to retain its data provided the power supply remains on. For the sake of explanation, we can assume the cell is holding the data '1' and '0' at the nodes Q and QB respectively. Under these conditions, transistors N1 and P2 are off, and the drain diffusion junctions of these devices are reverse biased and sensitive to upset. When an energetic particle strikes one of these sensitive cell diffusions, the charge



(a) Schematic

(b) Layout [13]

Figure 1.2: 6-Transistor SRAM Cell. Assuming the cell is holding the data ($Q = 1, QB = 0$), the drain diffusions for devices N1 and P2 are sensitive to particle induced upsets.

collected at the junction results in a transient current through the struck transistor [14]. As this current flows through the device, the restoring transistor (P1 for N1 and N2 for P2) sources current in an attempt to balance the particle-induced SET. As current flows through the restoring transistor, a voltage transient occurs at the struck drain's storage node. If this transient voltage exceeds the switching threshold of the inverter that is being driven by the node, then the inverter can switch, causing the forward feedback action of the inverter pair to unintentionally latch the transient data. Assuming the cell data is reversed (i.e., $Q = 0$, and $QB = 1$), then the upset sensitive diffusions are the drains of transistors N2 and P1 respectively.

Bit-cell area, critical charge, and operating voltage are shown as a function of technology node in Figure 1.3. Transistor device scaling allows for a roughly two-fold reduction in SRAM bit-cell area per technology generation [15]. These shrinking device dimensions lead to a reduction in the Q_{crit} necessary to corrupt a bit-cell. Coupled with reductions in SRAM operating voltage, technology scaling should lead to an increase in the bit-level SER; however, the reduction in device depletion region cross sectional area leads to a decrease in cell charge collection efficiency. For the 180 nm and 130 nm nodes these effects essentially cancel one another out and lead to a saturation in the bit-level SER shown in Figure 1.4. As the supply voltage saturates below 100 nm technologies however, the bit-level SER begins to decrease. Despite this decrease, the system-level SER increases with technology generation. This is shown by the seven-fold increase in system-level SER between the 130 nm and 22 nm nodes brought on by the increase in memory system capacity per unit area made possible by the smaller cell size.

The reduction is SRAM bitcell cross sectional area also reduces the distance between adjacent bitcells. This, coupled with a lower Q_{crit}, allows for not only lower energy particles

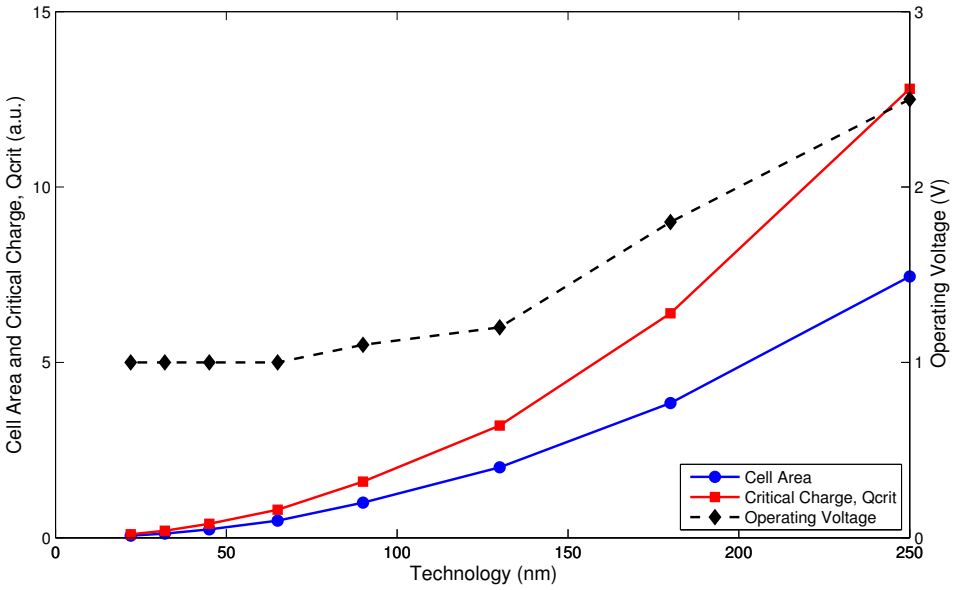


Figure 1.3: SRAM parameters - normalized cell area, normalized Q_{crit}, and operating voltage as a function of technology node [4], [10].

to cause SEUs but for more cells to be upset per particle strike [17]. Also shown in Figure 1.4 is the MCU percentage of the total (system) SER as a function of technology generation from the 250 nm to 22 nm node [6]. From the figure, we can see that MCUs comprise over 30% of the SER below the 45 nm node.

Although the MCU percentage of the total SER is increasing, not all MCUs are of the same size and shape. If an MCU occurs down a single bitline column, each of the cells will reside in a separate memory word and can be recovered individually. It is the upsets that corrupt multiple bits within the same memory word (i.e., MBUs) that require a higher degree of circuit complexity to correct. A distribution of the percentage of different width upsets for 45 nm, 32 nm, and 22 nm technologies is shown in Table 1.1 [6]. From the table it is clear that the number and size of these upsets increase with scaling. Further, Figure 1.5 shows a measured same-word MBU distribution for a 65 nm and 90 nm SRAM [16]. The figure shows that the frequency of larger sized MBUs decays exponentially with the number of bits upset per SEU. This means that although, the proportion of MBUs is on the rise, the majority of MBUs will still be comprised of only a small number of bits.

Despite the rarity of larger MBUs, there is still a non-zero probability of their occurrence. In Figure 1.6 the maximum adjacent upset size per particle strike is shown as a function of technology scaling. The data presented with squares has been collected from a

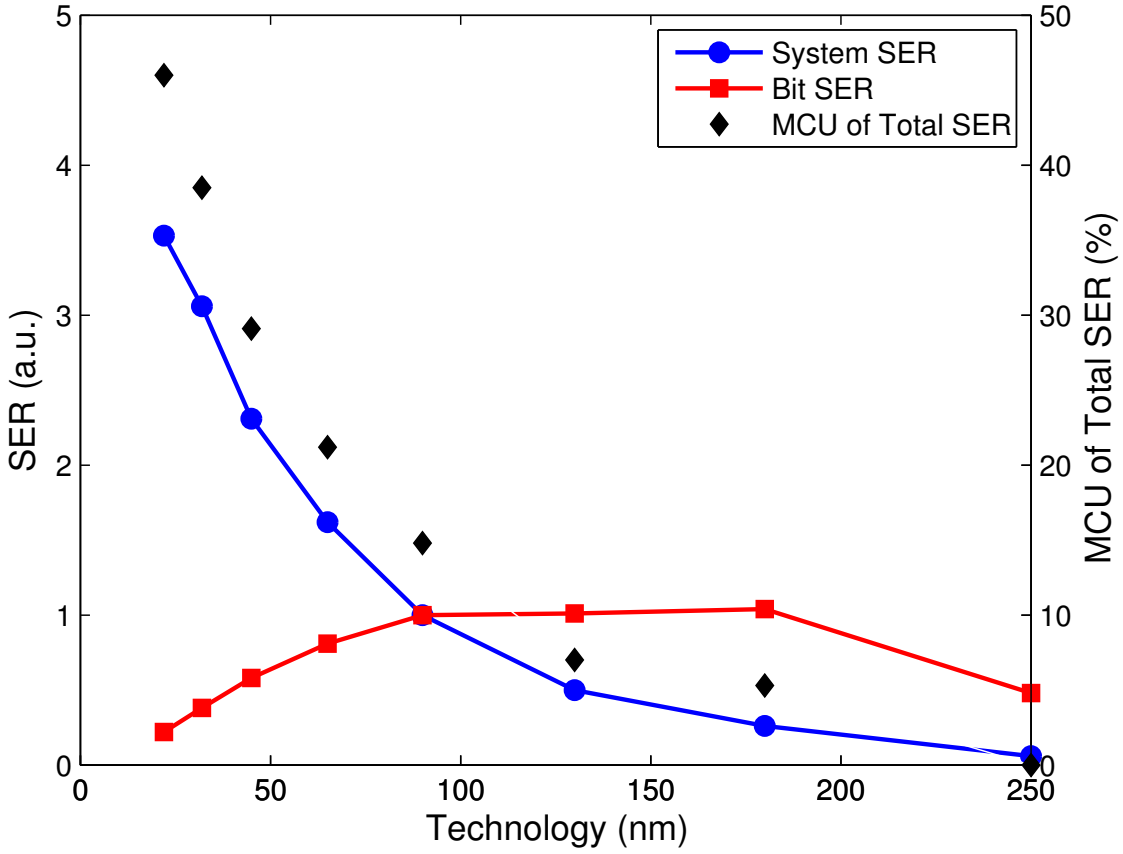


Figure 1.4: System-SER, bit-SER and percentage MCU of the total SER as a function of technology node [6].

Table 1.1: MCU bit width and percentage of the total SER for 45 nm, 32 nm, and 22 nm technologies [6]

Technology (nm)	% of Total SER	Bit width				
		2	3	4-8	>8	Max (bits)
45	2.2	1.9	0.2	0.1	<0.1	6
32	3.1	2.6	0.2	0.3	<0.1	16
22	3.6	3.0	0.2	0.3	0.1	18

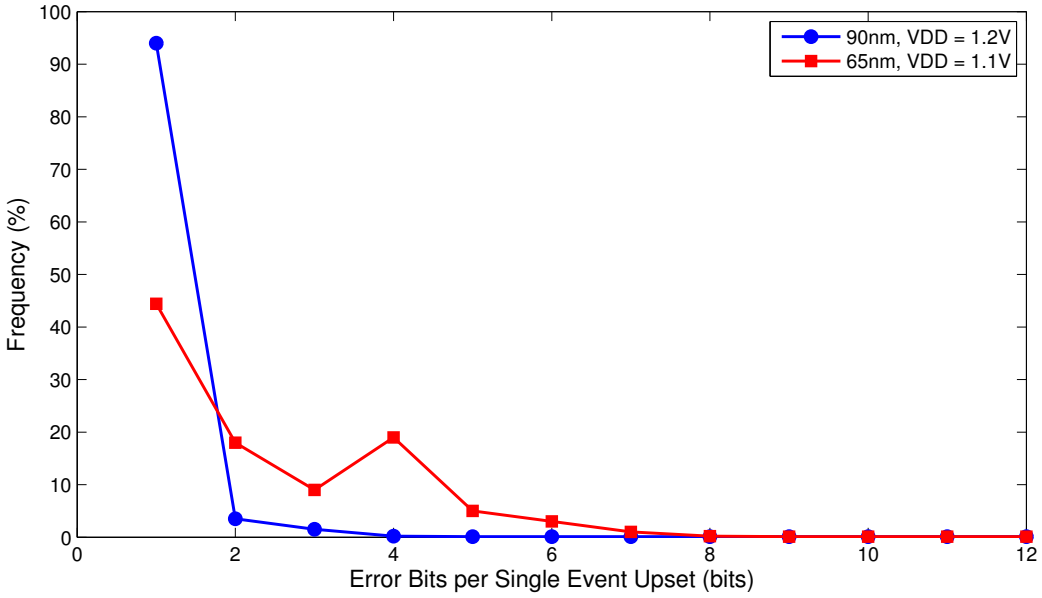


Figure 1.5: Frequency distribution of the number of soft errors generated per SEU due to neutron irradiation of 65 nm and 90 nm SRAMs [16].

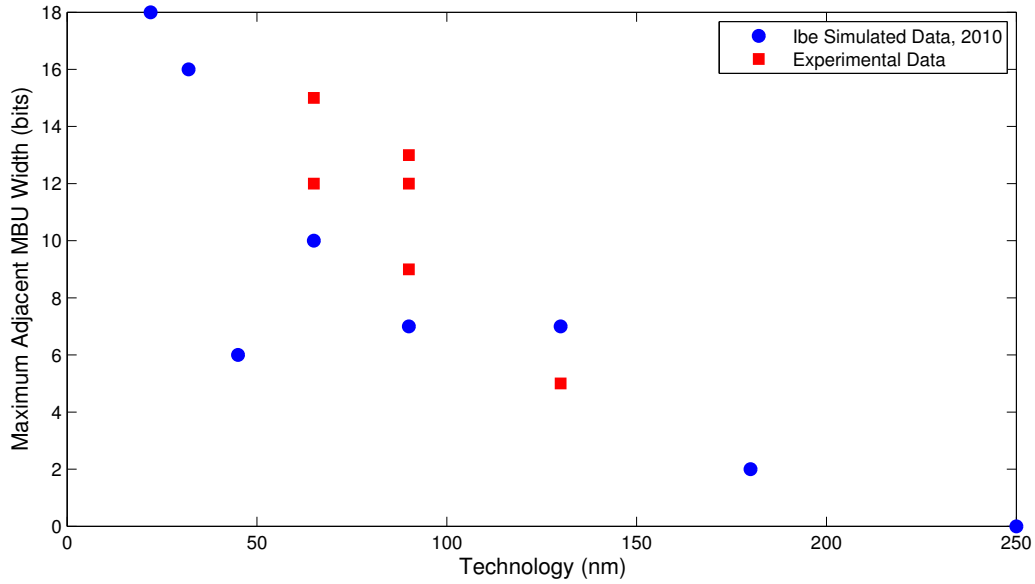


Figure 1.6: Maximum adjacent MBU width as a function of technology. Derived from [18, 19, 16, 20, 6].

series of radiation experimental results ranging from 130 nm to 65 nm, and shows between 12-15 adjacent-bit upsets at the 65 nm node [18, 19, 16, 20]. The data presented with circles uses a predictive model ranging from 250 nm to 22 nm and predicts 10 adjacent-bit upsets at the 65 nm node, and 18 adjacent-bit upsets at the 22 nm. The actual number of adjacent bit upsets depends heavily on the design and layout of the SRAM bitcell, but this data provides an indication of what to expect for a given technology. Finally, it also provides an indication that *for scaling to continue, SRAM SEU mitigation strategies must move beyond their current SBU focus, and increase their robustness to adjacent MBUs.*

1.4 SRAM Soft Error Mitigation Strategies

Recent developments in raw material purification during the semiconductor fabrication process has lead to a significant reduction in alpha particle induced upsets caused by packaging material [21]. Reducing cosmic neutron induced upsets unfortunately proves to be a greater challenge. Concrete has been shown to reduce the cosmic radiation flux at a rate of approximately 1.4x per foot of concrete thickness [4]. Thus, by operating a system in a basement surrounded by many feet of concrete, the SER due to cosmic neutrons can be reduced. While this may be a viable option for mainframes, it is not very practical for personal computers or other portable electronic devices. Therefore, rather than reducing cosmic radiation flux, cosmic neutron induced soft errors require on-chip mitigation strategies to minimize the SER. The soft error mitigation strategies for SRAMs can be classified into three separate categories, including: process-, circuit-, and architectural-level techniques.

1.4.1 Process

The primary method for soft error mitigation at the process level is to reduce the amount of charge collection at sensitive nodes. This can be achieved by increasing the doping of the p-well [22], using triple [23] or quadruple-well [24] structures, or utilizing a silicon-on-insulator (SOI) substrate. SRAM SERs have been reported with a 5x reduction relative to bulk CMOS when fabricated in a partially-depleted SOI technology [25]. Although process-level techniques significantly improve the soft error performance of SRAMs, the techniques do require modification of the standard CMOS fabrication process. Therefore these techniques are not readily available to companies that do not have control over the fabrication process, such as fab-less design groups. Additionally, process mitigation techniques require additional processing costs, which may further detract from their practicality.

1.4.2 Circuit

Circuit and architecture-level techniques provide simpler solutions to reduce the SER compared to process-level techniques. At the circuit level, the SRAM bitcell can be soft error hardened by either slowing down the cell's response to current/voltage transients or by increasing the cell's critical charge at its sensitive nodes. The cell's response to the transient can be reduced by adding either a resistor in the cell's feedback path [26] or a coupling capacitor between the cell's storage nodes [27]. The cell's critical charge can be increased either through cell transistor sizing, or by adding redundant storage nodes to the cell. The dual interlocking cell element (DICE) [28] and Quatro [29] cell topologies each use four data storage nodes rather than the traditional two for the 6T SRAM cell to store the cell's data. These techniques generally come at the expense of increased cell area. The DICE cell for instance incurs an approximately 80% area overhead relative to the conventional 6T cell [30].

1.4.3 Architecture

While circuit-level mitigation techniques are able to improve SRAM SER, they incur a significant area overhead. Since these techniques require over-sized or additional cell components, a hardened memory array can be significantly large compared to an un-hardened array, thus reducing the benefit of scaling. This area penalty can be reduced by using architectural-level solutions. There are four factors that make architecture-level mitigation techniques more attractive than circuit-level techniques. First, the definition of what an error is lies at the architecture level; an error may not even cause an issue if new data is written to the cell before the corrupted data is read. Second, the error may result from a physical weakness of the cell (e.g., process variability) in addition to a particle strike. In this case, circuit hardening may not help. Third, architecture-level solutions can incur less area overhead than circuit-level techniques. For example, a single error correcting double error detecting (SEC-DED) error correcting code (ECC) has an overhead of 8 check-bits for every 64 data-bits (i.e., 13%) [31], whereas radiation-hardened cells can have an area overhead on the order of 30-100% depending on the aggressiveness of the technique [32]. Finally, ECC can also be used to correct hard errors or parametric faults, which is another limiting factor to SRAM yield.

Error Correcting Codes

ECCs provide channel encoding in the form of parity check-bits to protect user data as it is stored in the memory array. The check-bits can then be recalculated and compared against the previously written bits whenever the word is read from memory. Any upsets

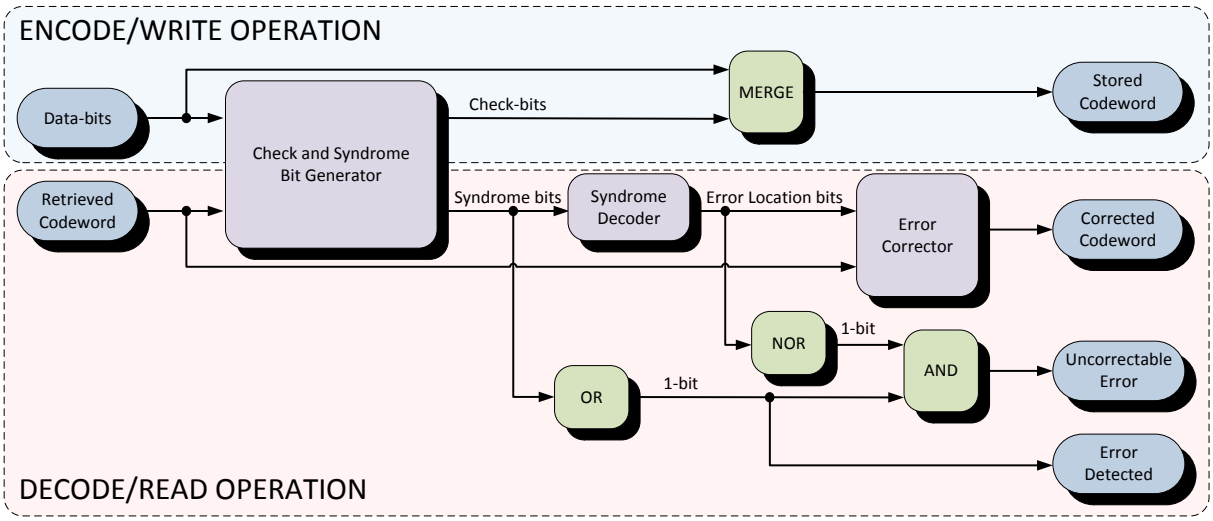


Figure 1.7: ECC Block Diagram

will result in discrepancies between the two sets of check-bits, also known as the syndrome. The syndrome can be used to correct or detect upsets depending on the code's complexity. This process is shown in Figure 1.7. Following the generation of a particular syndrome pattern, it can be decoded into an error location. This error location vector can then by applied against the originally received codeword to correct any errors. In the event that an error can be corrected, the correction process is completed transparent to the outside of the memory. Additional combinational logic is used to determine whether an error can be corrected or detected. If used in a multi-tiered cache memory system, detected errors can send a request for data to be re-written from a higher level in the memory hierarchy. The types of errors that can be corrected or detected depend on the encoding of the check-bits, and the sophistication of these errors types have a direct impact on the overhead of the scheme.

Error correcting codes used in early computer memory systems were designed using the class of SEC-DED codes created by Hamming in 1950 [33]. A SEC-DED code is capable of correcting one single-bit error and detecting two single-bit errors in a codeword. The double-error-detecting capabilities serves to guard against data loss. In 1970, Hsiao improved upon the efficiency of these codes by introducing odd column weight codes to the SEC-DED family [31]. For the same coding efficiency as Hamming's algorithm, odd-weight-column codes provide improvements over the Hamming codes in terms of speed, cost, and reliability of the decoder logic. As a result, odd-weight-column codes have been widely implemented by the computer industry worldwide [34], and are still in use today [35]. Hamming codes are discussed in more detail in Section 2.8.2, and 2.8.3, while Hsiao codes

are discussed in Section 2.8.4.

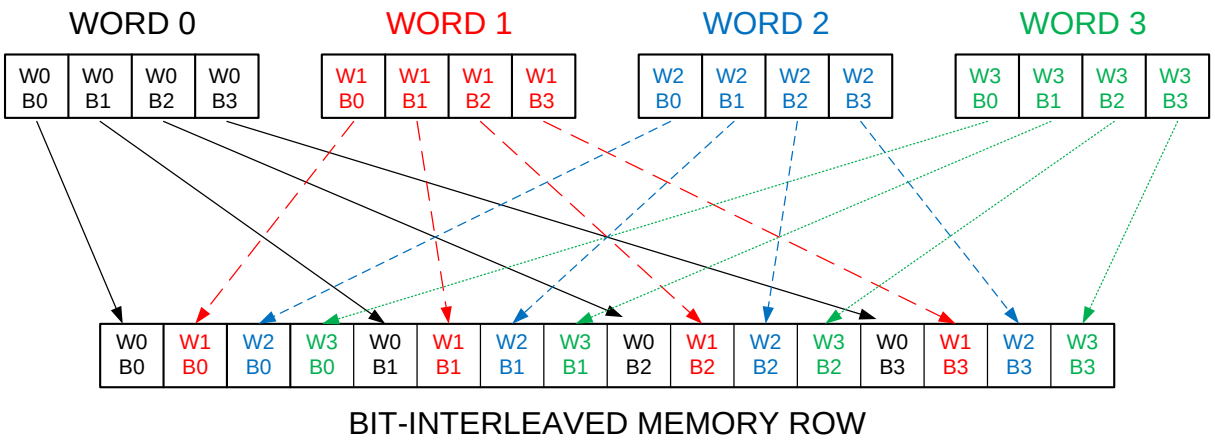
Although Hamming and Hsiao SEC-DED codes are capable of correcting one error and detecting all possible double errors, it is becoming apparent that they will not be able to provide adequate protection against MBUs in modern aggressively scaled SRAMs [35]. To address this concern, codes utilizing a higher degree of check-bits and/or encoder-decoder logic are required. An effective approach to mitigate MBUs is through multi-bit error correcting codes. The Bose Chaudhuri Hocquenghem (BCH) codes [36], Reed-Solomon (RS) codes [37], Golay codes [38], Euclidean Geometry Low Density Parity Check (EG-LDPC) codes [39], and Two-dimensional error codes [40] have all been designed to deal with multiple bit errors in memories. Due to their high decoding complexity and redundancy requirement however, the implementation of these codes is not popular for high-speed RAM applications [41]. Further, high-speed embedded SRAMs require hard, fixed cycle time, decoding strategies and cannot rely on probabilistic soft-decoders using iterative, multi-cycle, solution methods such a Turbo [42] and LDPC [43] codes. The issue is that, while more powerful ECCs exist, they come at too high a penalty compared to the SEC-DED codes for everyday applications [35]. For example, a double error correcting-triple error detecting (DEC-TED) BCH codes require twice the number of check-bits and an approximately 60% increase in delay as compared to a SEC-DED code. Therefore, *extending ECC cache protection capabilities for use in next generation microprocessors is still an open issue.*

To address this issue, Dutta, in 2007, presented a single error correcting-double error detecting-double adjacent error correcting (SEC-DED-DAEC) coding algorithm [44]. For the same check-bit overhead, and only slightly increased decoder size, these codes offer all of the protection of a SEC-DED code with the added benefit of 2-bit adjacent error correction to help mitigate against adjacent-MBUs. While these codes offer an alternative to SEC-DED and more powerful codes, there is still a great deal of design space between the two mitigation options to be explored. SEC-DED, BCH, RS, and SEC-DED-DAEC codes are discussed in more detail, complete with design examples, in Chapter 2, while the proposed class of ECC codes is presented in Chapter 3.

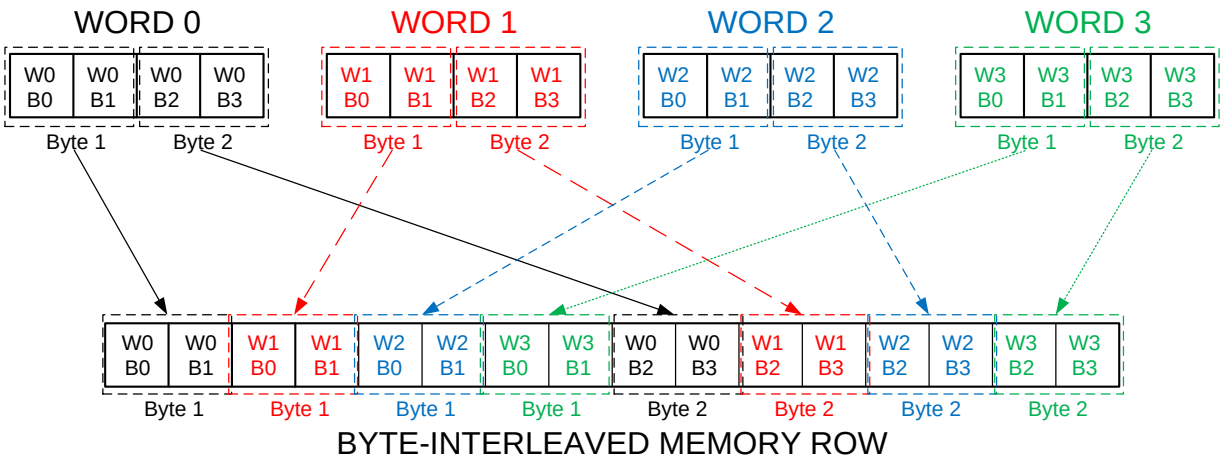
Interleaving

Interleaving is a method of extending ECCs to protect against adjacent-MBUs [45]. Rather than storing a word's bits in adjacent memory cells, interleaving physically separates the logical bits of multiple words throughout a memory row. This concept is illustrated in Figure 1.8(a).

In the bit-interleaving example in Figure 1.8(a), the data from four 4-bit words have been interleaved to form a 16-bit row in the memory. Notice that the first bit in the row is bit-0 from word 0, the second bit is bit-0 from word 1, the third bit is bit-0 from word



(a) Bit Interleaving



(b) Byte Interleaving

Figure 1.8: Types of Interleaving

2, and the fourth bit is bit-0 from word 3. Next, the fifth bit in the memory row is bit-1 from word 0. This data pattern is repeated throughout the memory row. For the byte interleaving example in Figure 1.8(b), the same four 4-bit words are each comprised of two 2-bit bytes. The four word's bytes are then interleaved throughout the memory row. The byte organized Reed Solomon codes lend themselves more readily to byte-wise interleaving. This will be discussed in more detail in Section 2.8.6 and Section 4.1.2.

By storing data in this interleaved manner, adjacent-MBUs can be distributed across multiple logical words to form multiple upsets that can be corrected individually. Interleaved data can be selected using a series of multiplexers, and as such the number of interleaved word is referred to as the number of 'ways' that the data can be selected as input. The 4-way interleaving scheme shown in Figure 1.8(a) is capable of dividing a 4-bit adjacent MBU into four separate SBUs across four separate words. Each of these SBUs can then be corrected by a simple single-bit correcting ECC. Interleaving can be added to an ECC scheme to extend the adjacent-error reliability of the system. Situations do arise however in which the degree of interleaving is limited or infeasible such as in register files or fixed aspect ratio memories [46]. In these situations, MBU error correction and detection becomes the sole responsibility of the implemented ECC.

Memory Scrubbing

Memory scrubbing is a method for preventing errors from accumulating beyond the error handling capabilities of an ECC [47]. Rather than performing the error correction or detection procedure when a word is being read from memory, scrubbing circuits continuously cycle through the memory array preemptively addressing any errors upon their detection. This prevents separate SEUs that could be handled by the ECC from accumulating over time to become uncorrectable upsets by the time they are eventually read from memory. Choosing the scrubbing cycle time interval presents a trade-off between power consumption and product reliability, as excessive scrubbing may needlessly exercise the memory when it could be sitting idle, and relies heavily on the system's end application. Scrubbing has been used in some implementations to remove the ECC logic from the SRAM read circuitry's critical path to improve the system's read cycle time; however, this benefit comes at the cost of system reliability and data integrity [48].

1.5 Goal of This Research

The main focus of this research is to explore the design space between SEC-DED and DEC-TED codes to develop a class of light-weight, soft error robust ECCs targeted toward on-chip embedded SRAMs in state-of-the-art technologies. These codes target the adjacent-MBUs

more prevalent in advanced technology nodes and provide a varying degree of adjacent error correction (xAEC) and adjacent error detection (yAED) while maintaining overheads less than those of the multi-bit ECCs. These gains are realized through the use of the check-bit difference between the SEC-DED and DEC-TED codes, and are verified both through simulation and test chip irradiation measurement results.

1.6 Outline

The thesis is organized as follows. Chapter 2 describes error correcting codes for SRAM. Chapter 3 presents the proposed class of adjacent-MBU robust ECCs complete with high-level model and synthesized circuit results in comparison to other ECCs. Chapter 4 provides SER reduction verification in the form of simulation results and radiation measurements for a test chip fabricated in a 28 nm bulk CMOS technology. Finally, Chapter 5 concludes the report and summarizes the contributions of this research.

Chapter 2

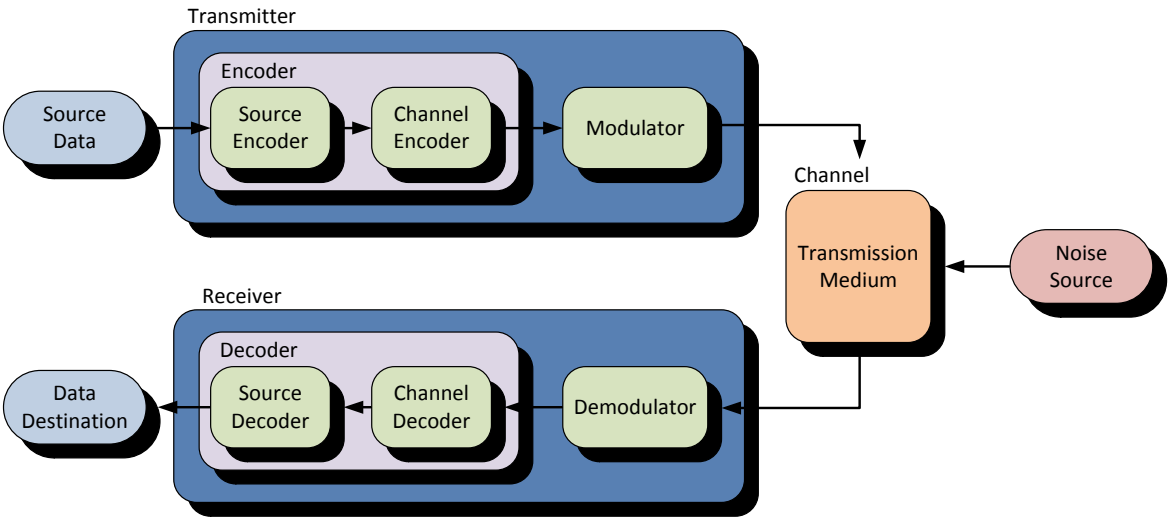
Error Correcting Codes in Computer Memories

This chapter discusses error correcting codes in the context of embedded SRAMs. A brief mathematical foundation for ECCs is provided, and the codes used for comparison in subsequent chapters are discussed.

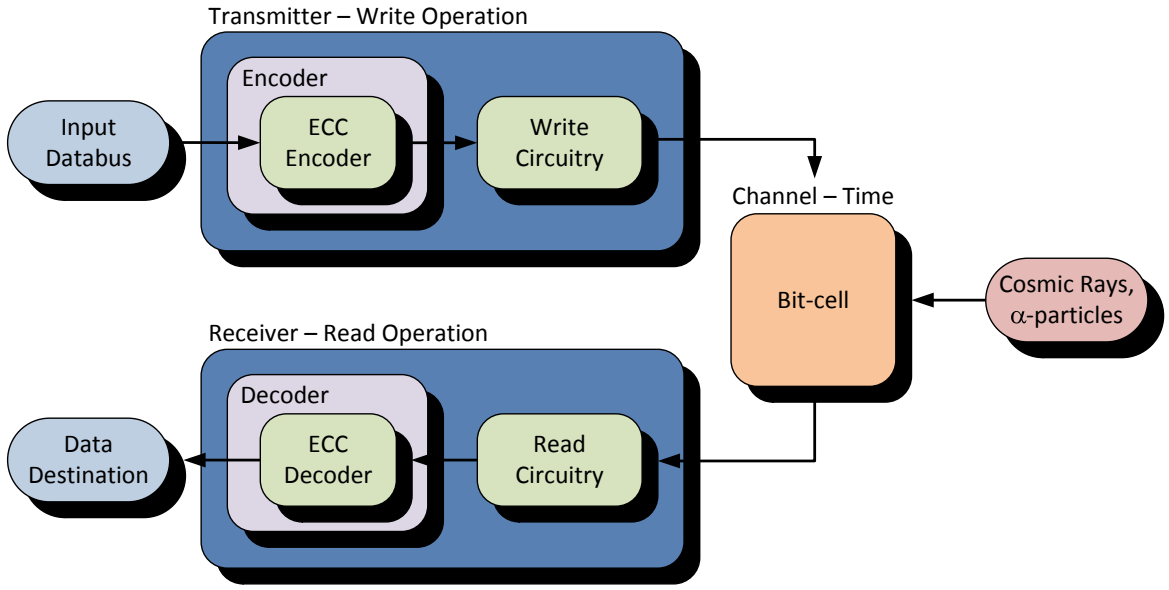
2.1 The Communication Channel

Error correcting codes are an effective technique for improving data integrity when transmitting information over a channel. The channel encoding procedure involves adding redundancy in the form of parity check-bits and encoding-decoding logic circuits to improve data robustness to noise during transmission. The channel encoding problem is one of designing an ECC with sufficient error handling capabilities while maintaining its area and delay penalties to within the system specifications. The most appropriate choice of ECC depends on the system application and the behaviour of the noise in the channel.

In traditional data communication systems, data is transmitted spatially over a channel. This is illustrated in Figure 2.1(a) [49]. Data originates from a source and is first prepared for channel transmission. Preparation can take place in the form of data-compression source encoding for efficient data transfer rates, ECC channel encoding for transmission reliability, and modulation for the actual propagation through the channel. Data is then transmitted spatially over the channel from point A to point B. At any point during transmission noise may corrupt a portion of the outgoing data. Once received by the receiver, data can then be demodulated and used by the receiving system. This process involves an inversion, or decoding, of the encoding processes.



(a) Communication Channel



(b) Memory System

Figure 2.1: Block diagrams for the communication channel and memory storage systems including ECC blocks. Adapted from [49, 50].

The communication channel model is directly applicable for memory systems, as is shown in Figure 2.1(b) [50]. Instead of data being transmitted spatially however, it is transmitted temporally (through time) as it resides in the memory cell. During a write operation, data originating from an external source (input databus) is channel encoded by the ECC and then prepared for storage by the memory's write circuitry. As the data resides in memory, it "travels" temporally while it is susceptible to external radiation induced noise (i.e., galactic radiation and alpha particles). Upon a read request, the data is demodulated by the memory's read circuitry, and then channel decoded by the ECC. Once error handling is complete, the data is then placed on the memory's output databus.

Since both of these domains undergo channel encoding, the concepts of error control coding theory can be applied to either discipline. System specifications dictate which class of code is most appropriate for the given application. For example, while data communication systems may require large amounts of information to be transmitted serially over long distances, very-large-scale-integration (VLSI) on-chip embedded memory systems store data in much smaller segments and transmission occurs in parallel to ensure high-speed data rates. With the end application in mind, ECCs can be designed to leverage these system-level properties for optimal performance.

This work focuses on ECCs for high-speed, high-density VLSI on-chip embedded memory applications. These systems typically have a fixed dataword length, and require single cycle read and write execution. In SRAMs in particular, the memory cell packing density combined with a low critical charge influences the size and type of upsets that occur in scaled technologies.

2.2 Recent Advances in Coding Theory

Recent advances in coding theory have focused on high noise channel environments for applications such as mobile communication systems and deep space satellite communication [51]. In these environments, the emphasis is on reliable information transfer rather than bandwidth or latency constraints. Turbo codes, first presented in 1993 [42], and Low Density Parity Check (LDPC) codes, presented and dismissed in 1963 [43] then rediscovered in 1996 [52], have been at the forefront of error correction research recently [53, 54, 55]. Both codes use an iterative, probabilistic decision making approach to progressively increase the system's confidence in the fidelity of the received codeword.

Turbo codes work by iteratively comparing and modifying the results of two separate decoder units until they converge on the same output solution. This process can upwards of 10 cycles to complete [56]. Turbo codes have been used extensively in the 3G and 4G mobile long term evolution (LTE) telephony standards, and IEEE 802.16 (WiMAX) wireless metropolitan network standard [57, 58]. Alternatively, LDPC codes use multiple,

smaller decoder units to provide soft probabilistic decisions on the most likely data being received by each decoder unit. The decoders with high confidence in their output data, pass this information to the lower confidence decoders to iteratively build up the overall confidence in the received codeword. LDPC codes are currently used in the digital video broadcasting standard (DVB-S.2) for satellite transmission of digital television, 10GBase-T Ethernet, and the Wi-Fi 802.11n and 802.11ac high throughput specifications of the Wi-Fi 802.11 standard [59, 60, 61].

While these codes provide a high level of error protection, the selection of an error correcting code for a particular application depends heavily on the bit error rate of the error channel in which it will be exposed to and the required performance specifications. For high-speed, high-density on-chip embedded memory applications, performance is a critical factor and a multi-cycle delay penalty can be excessively prohibitive. Further, for low error rates when error correction is only necessary for a small fraction of read or write operations, this penalty becomes unnecessarily crippling to system performance. This makes the iterative decision making Turbo codes and LDPC codes poor candidates for embedded, high-speed SRAM cache memory applications. To counteract the high performance penalty, codes that include the encoding and decoding operations within the single write and read cycle memory access are preferred [11, 19].

In the context of high-speed embedded SRAMs, selective bit placement codes, described in more detail Section 2.8.7, are emerging as a leading candidate for light-weight error correction due to their minimal impact in terms of performance and additional number of check-bits. A shortcoming of these codes however is their potential for miscorrection, an issue that is partially addressed with the proposed class of in Chapter 3.

2.3 SRAM Organization

Due to its high random access speeds and compatibility with the CMOS logic process, SRAM has had a long history of being at the top of the computer memory hierarchy [13]. The memory hierarchy refers to the hierarchical configuration of memory storage units in modern computer systems. By blending a series of fast, but less dense, low capacity memories with dense, but slower, higher capacity memory types, the system presents the illusion of having a single high-speed, high-capacity storage unit. The pyramid-like hierarchy, shown in Figure 2.2, is indexed using multiple tiers, or levels, to describe the functional purpose of each memory unit. The large-to-small capacity configuration reflects the increasing speed requirement and cost per bit (both in real estate and monetary terms) as the pyramid is traversed from the bottom level-5 (L5) remote secondary storage tier to the topmost (and closest to the central processing unit) level-0 (L0) register tier. Historically, SRAMs have occupied the L0-L2 range of the hierarchy, but due to recent scaling advance-

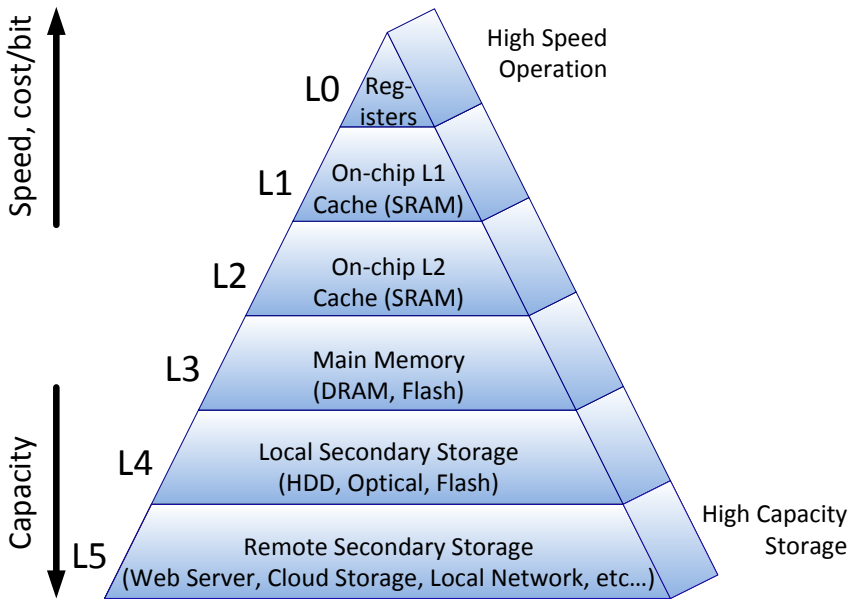


Figure 2.2: Computer memory hierarchial organization structure. SRAM is typically used as the main building block for system registers, on-chip cache, and various buffers.

ments, some system implementations have inserted an additional on-chip (embedded) L3 SRAM cache between the L2 and main memory hierarchy layers [62]. SRAM bitcells can be optimized at the transistor level for high performance using slightly larger device sizes (these would be used for L0 and L1 memories), or for density, using near minimum device features sizes, for L2 and L3 memories. In Section 4.2.3 and Section 4.5.3 three different vendor supplied SRAM cells are considered; these cells have been optimized for high-speed, high-performance (e.g., very high speed), and high-density.

A typical SRAM block configuration consists of an $N \times M$ array of memory cells, organized as N horizontal rows and M vertical columns [63]. This is shown in Figure 2.3. As shown in Figure 1.2, the 6T SRAM bit-cell has three external data signals, including: wordline (WL), and a pair of complementary bit-lines (BL and BLB). These cells are designed for data to be accessed in a row-wise manner. When a read or write operation is performed, the memory's address decoder enables an entire row's WL. This exposes the bit-cells' data to its respective column-wise BL/BLB pair. This allows an entire row of data to be read or written to within a single cycle, while accessing data in a column-wise fashion would require a full read cycle for each bit of data to be read. The organization of an SRAM array hence lends itself readily to a row-wise ECC scheme. Therefore, the check-bits for each dataword are typically contained within the same row as the dataword itself. This allows for codewords (data-bits and their respective check-bits) to be read or

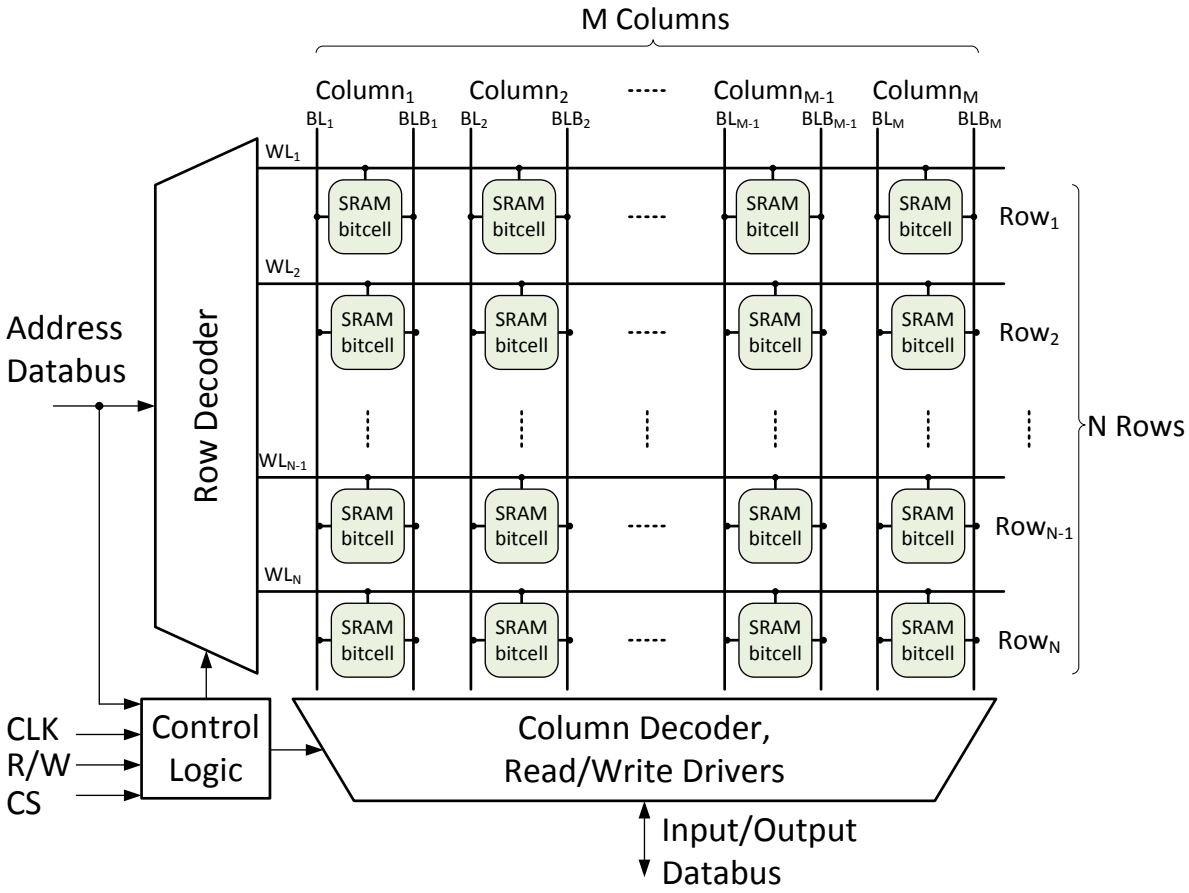
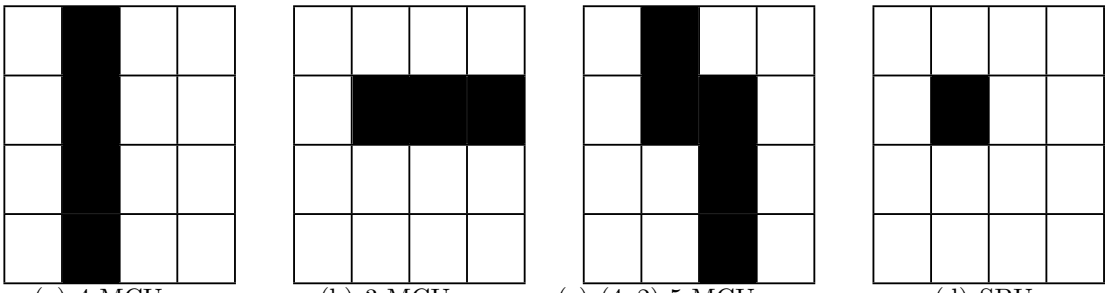


Figure 2.3: $N \times M$ SRAM Array with Peripheral Circuitry



(a) 4-MCU$_{BL}$ (b) 3-MCU$_{WL}$ (c) (4x2) 5-MCU$_{Cluster}$ (d) SBU

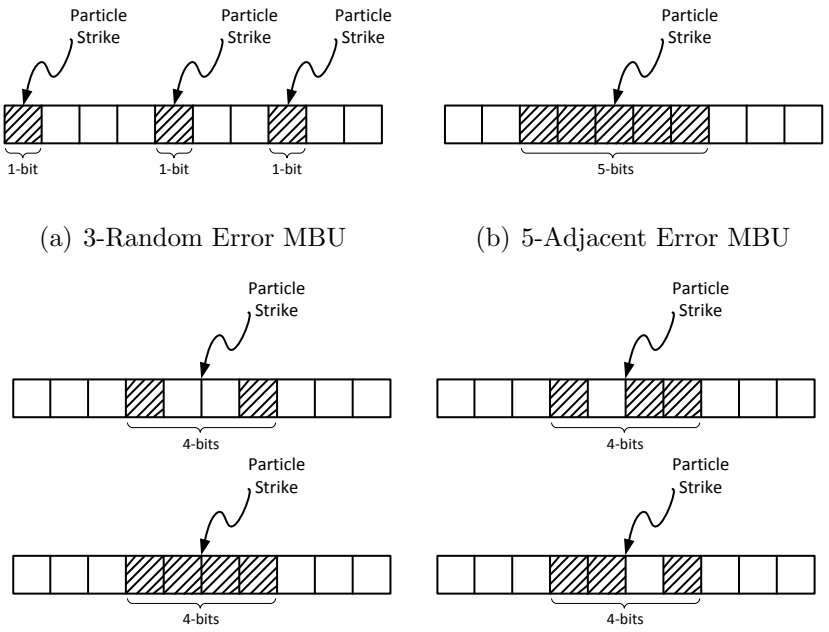
Figure 2.4: MCU Upset Classifications

written to within a single cycle, and it also mitigates against column-wise MCUs. Much like bit-interleaving discussed in Section 1.4.3, x-bit MCUs occurring vertically along a column will be distributed across multiple codewords. This results in x more manageable SBUs as opposed to a more disruptive x-bit MBU within the same codeword. *MBUs occurring horizontally throughout a codeword are those most disruptive to the memory system. These are the focus of multi-bit ECCs.*

2.4 Error Types

Error types are classified by the number of bits that they upset, and the matter in which they upset them. Single bit upsets, or SBUs, as the name implied are those in which only one bit within the memory word is corrupted by the error mechanism. Correction and detection of these errors are the simplest for ECCs to deal with. Multi-cell upsets, or MCUs, also as the name implies, involved the corruption of multiple bitcells. Multi-bit upsets, or MBUs, are a specific type of MCU in which multiple bits within the same memory word have been corrupted. These error types can be significantly more challenging to handle. The distinction between MCUs and MBUs depends on the organization of the memory words. MCUs can be classified into three different categories: a). MCUs running down a single bitline MCU$_{BL}$, b). MCUs running along a single wordline MCU$_{WL}$, and c). MCUs spread across multiple bitlines and wordlines MCU$_{Cluster}$. All MCUs are prefaced with the number of corrupted bits involved in the upset, e.g., 3-MCU$_{WL}$, while MCU$_{Cluster}$ includes the upset's size in terms of wordline width by bitline height, e.g., (4x2) 5-MCU$_{Cluster}$. Since memory words are traditionally organized row-wise, the MCU types most likely to cause MBUs beyond the error handling capabilities of the ECC are the MCU$_{WL}$ and MCU$_{Cluster}$. Each of these patterns are shown in Figure 2.4 along with a single bit upset pattern.

MBUs occur within the same logical memory word and can be further classified into three separate sub-categories. These include: random, adjacent, and burst errors. Each



(a) 3-Random Error MBU

(b) 5-Adjacent Error MBU

(c) Four separate examples of 4-Burst Error MBUs

Figure 2.5: MBU Upset Classifications

of these error types are illustrated in Figure 2.5, and a numerical prefix is added to each MBU type to indicate the extent of the error (i.e., a 2-random error indicates an upset of two randomly separated bits within the memory word).

A random MBU is the occurrence of multiple SBUs accumulating over time leading to the upset of multiple bits within a memory word. There is no correlation between these temporal errors occurring within the memory word. An example 3-random error is shown in Figure 2.5(a). Memory scrubbing can be combined with an ECC scheme to prevent the accumulation of multiple SBUs. If the scrubbing interval is less than the mean time between errors, then most of the temporal MBUs can be eliminated [32].

An adjacent MBU is a single upset that corrupts multiple side-by-side adjacent bits within a memory word. Figure 2.5(b) shows an example of a 5-adjacent error. In the example, all five adjacent bits have been corrupted. These errors make up a significant portion of the total SER as the memory cell size is reduced below that of the effective

cross-section of the imposing radiation source [20].

Burst MBUs are similar to the adjacent MBU; however, not all of the adjacent bits are necessarily upset. The size of a burst error is defined by the distance between the two most extreme upset bit positions within the memory word. The corruption status of the intermediary bits between these two extreme bits is irrelevant and may vary. For instance, the burst errors shown in Figure 2.5(c) are all examples of 4-burst errors. The error mechanism for these is similar to the adjacent MBU, but not all of the affected bits are corrupted. This may occur due to fabrication process variability increasing the critical charge of one of the affected bits relative to the others, or a cell having a data-dependent asymmetrical critical charge (i.e., if the memory cell has been designed to hold a '1' more effectively than a '0'). Adjacent and burst error are both examples of spatial MBUs. They each affect multiple bits separated by a physical distance and are caused by a single particle strike.

Finally, without temporal knowledge, random errors may appear as either burst or adjacent errors. The 3-random error in Figure 2.5(a) for instance could be considered an 8-burst error if no temporal information is included with the error pattern. Likewise, the 4-burst error in Figure 2.5(c) affecting all four bit positions can also be considered a 4-adjacent error. Errors can be categorized as multiple types depending on the context of the situation.

2.5 ECC Classifications

Error control coding theory has been studied for over half a century [33]. During this time designers have developed a multitude of error correcting and detecting codes and algorithms for mitigation against upsets and ensuring data reliability in both communication and memory systems. On-chip embedded VLSI memories rely predominately on algebraic bit-error control codes to ensure data integrity. Each coding algorithm provides a certain amount of error protection for an associated overhead in terms of delay and die area. Intuitively, an ECC offering n-bits of correcting capability will have a higher overhead than a code offering $(n-1)$-bits of correcting capability.

Algebraic coding theory applies to those codes that can be expressed algebraically [50]. Within this area, there are two major coding classifications: linear block codes, and convolution codes. Convolution codes process input data streams in a serial manner and work by associating a code sequence with a data sequence. Due to the serialized nature of the codes, they are often used in serial communication systems. In contrast, linear block codes process predefined lengths of data-bits in parallel to form useable codewords. Since memory data is already divided into predefined size blocks of datawords, memory systems lend themselves more readily to using linear block codes. Linearity is an important feature

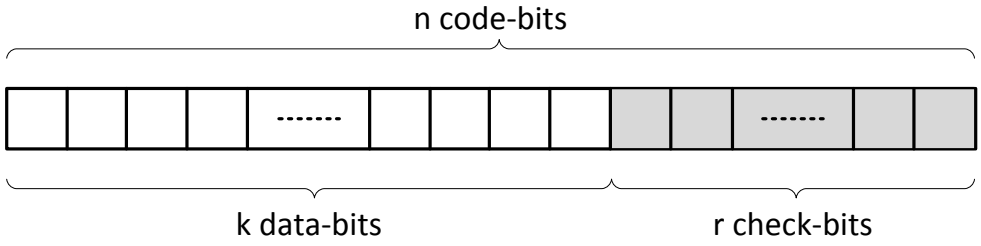


Figure 2.6: $(n,\ k)$ codeword structure. The n-bit codeword consists of k data-bits and r parity check-bits.

of these codes. It dictates that the sum of any two codewords forms another codeword. This property is used extensively in determining the validity of a received codeword. Error correcting linear block codes can be classified using three main parameters:

1. The error handling capability they provide,
2. The length of the dataword they are designed to protect, k, and
3. The overall length of the resulting codeword, n.

For example, a $k = 32$-bit dataword using 7 check-bits to provide SEC-DED protection has a codeword length of $n = 39$ bits, and is expressed as a (39, 32) SEC-DED code. The organization structure of these bits is illustrated in Figure 2.6. The dataword, of length k-bits, stores the actual information relevant to the user. This information is encoded to form an n-bit codeword, where $n = k + r$, and r represents the number of parity check-bits used to encode the data.

For a k-bit dataword, there are 2^k possible datawords. For a linear block code, this set of 2^k valid codewords forms a k-dimensional subspace $\mathbf{C}$ in the 2^n possible n-bit codeword vector space $\mathbf{V}$. Each of the codewords in $\mathbf{C}$ has a one-to-one correspondence with its respective dataword. Due to the linearity of these codewords, the modulo-2 sum of any two valid codewords must form another valid codeword within the subspace. In circuit design, this bit-wise modulo-2 addition can be performed using XOR gates [11].

2.6 Mathematical Foundation

The research in error correcting codes relies to a large extent on powerful structures in modern algebra. Many of the multi-bit error correcting codes are based on the structure of

rings and Galois fields. This adds a significant level of complexity to their implementation. Fortunately, single-random-bit and multiple-adjacent-bit codes can be constructed under binary $GF(2)$ Fields, requiring only two elements in the symbol library, i.e. '0', and '1'. This minimizes the required mathematical and implementation complexity. Since memory designers do not necessarily have a strong background in algebraic coding theory, an effort has been made to explain the following ECCs in their simplest terms using circuit terminology, while minimizing the mathematics to only what is necessary. The section begins with some discussion on the basic algebraic terms used in coding theory, and then considers the basic structure of the parity check matrix and syndrome decoder. The material presented in this section has been adapted from [11, 50, 64].

2.6.1 Algebraic Definitions for ECC

In Hamming's 1950 seminal work he defined the two important concepts of code weight and code distance, and by combining these concepts he defined a minimum distance requirement for each code [33]. Each of these concepts are presented in turn below.

Hamming Weight

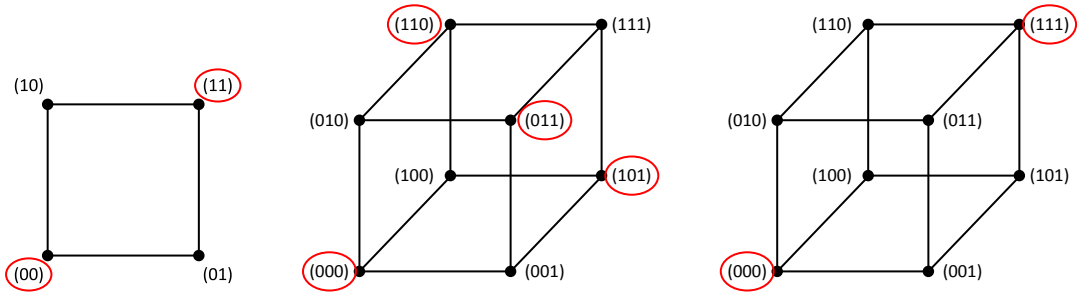
The *Hamming weight*, or simply weight, of a vector $\mathbf{u} = (u_0, u_1, \ldots, u_{n-1})$, denoted by $w(\mathbf{u})$, is the number of nonzero elements of $\mathbf{u}$. For instance, the vector $\mathbf{d} = (1101\ 1101)$, has a Hamming weight of 6, or $w(\mathbf{d}) = 6$.

Hamming Distance

The *Hamming distance* between two vectors $\mathbf{u}$ and $\mathbf{v}$, denoted by $d(\mathbf{u}, \mathbf{v})$, is the Hamming weight in $\mathbf{u} - \mathbf{v}$. The Hamming distance is equal to the number of positions by which the two vectors differ. That is,

$$d(\mathbf{u}, \mathbf{v}) = w(\mathbf{u} - \mathbf{v}) = w(\mathbf{v} - \mathbf{u}). \tag{2.1}$$

The Hamming distance and Hamming weight can be used to understand the error control capabilities of a particular code. For instance, when the codeword $\mathbf{v}$ is transmitted, and the received word $\mathbf{r}$ is received, the Hamming distance between $\mathbf{v}$ and $\mathbf{r}$, given by $d(\mathbf{v}, \mathbf{r})$, is equal to the number of errors introduced in the channel. This value is also expressed by the Hamming weight of the error vector $\mathbf{e} = \mathbf{v} - \mathbf{r}$, $w(\mathbf{e})$.



(a) (2,1) Code, $d_{min} = 2$ (b) (3,2) Code, $d_{min} = 2$ (c) (3,1) Code, $d_{min} = 3$

Figure 2.7: Minimum Hamming Distance Example

Minimum Hamming Distance

The *minimum Hamming distance,* d_{min}, of a linear block code is the minimum distance between all pairs of codewords. The metric is critical in determining the random error handling capabilities of a code. Minimum Hamming distance is illustrated in Figure 2.7.

If we let the four point plane in Figure 2.7(a) represent a (2, 1) code where $\mathbf{C} = \{00, 11\}$ represents the set of valid codewords, then we can see by inspection that the minimum distance between valid codewords is two, that is $d_{min} = 2$. Extending this concept to Figure 2.7(b), if we let the eight point cube represent a (3, 2) code in which $\mathbf{C} = \{000, 011, 110, 101\}$ represents the set of valid codewords, then this code also has $d_{min} = 2$. If instead we let the cube represent a (3, 1) code with $\mathbf{C} = \{000, 111\}$ representing the set of valid codewords, as shown in Figure 2.7(c), we can see that this code has a $d_{min} = 3$.

Based on the previous example, it can be rationalized that if the code has a minimum distance of at least d, then the code can detect any error pattern of weight $d - 1$ or less. Since valid codewords are each separated by a distance of d_{min}, a received codeword with up to $d - 1$ errors will result in an invalid codeword. The validity of a codeword can then be determined by the ECC.

In terms of error correction, a code with a minimum Hamming distance of

$$d_{min} \geq 2t + 1 \tag{2.2}$$

can correct all patterns of t or fewer errors. This is shown with the aid of an example in Figure 2.8 for a code with $d_{min} = 4$. For any valid codeword that has suffered an SBU, there is only one valid codeword within a distance of $d = 1$ away from the corrupted codeword. It is therefore safe to assume that this is the valid codeword. If two bits are upset, then

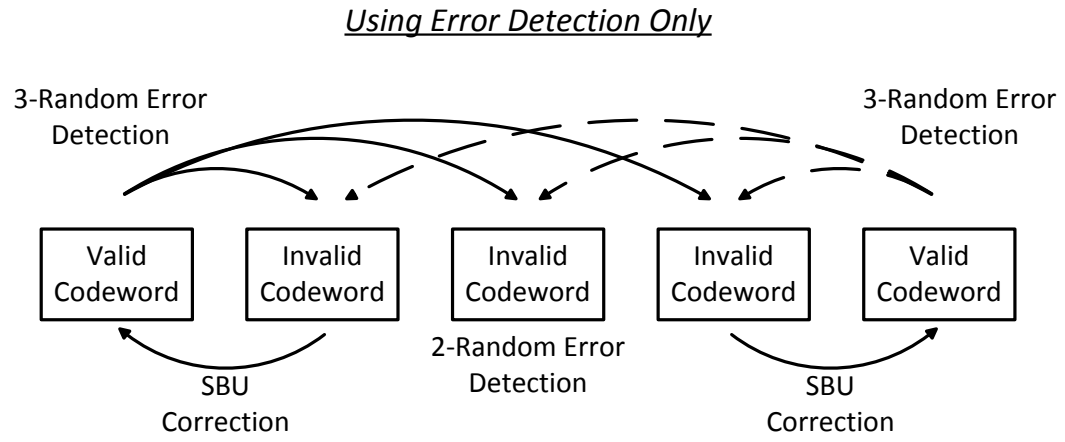


Figure 2.8: Provided that $d_{min} = 4$, the code is capable of either 1-bit error correction, 3-bit error detection, or 1-bit error correction with 2-bit error detection.

there are two potential candidate codewords to "correct" the word to. Since the code has no way to determine which is the correct codeword, no correction can be performed. If only error detection is being implemented, the ECC can detect up to three error bits since $d_{min} - 1 = 3$.

When performing a combination of both error correction and detection, for a minimum Hamming distance

$$d_{min} \geq t + d + 1 \tag{2.3}$$

the code can correct any combination of t-bit errors and detect up to d-bit errors where $d \geq t$. To visualize this, consider again Figure 2.8, for which $d_{min} = 4$. In this case, the ECC can correct single bit errors ($t = 1$) and detect double bit errors ($d = 2$). In the event of a triple bit error, this code could potentially miscorrect the error to form a different codeword than the original. Miscorrections are discussed in more detail in Section 2.7.3. By increasing the minimum Hamming distance for an ECC, more sophisticated error handling can be performed. The proposed set of codes rely on having a non-minimum Hamming distance between codewords differing in adjacent bit positions, while still meeting a given minimum Hamming distance requirement for codes differing in non-adjacent bit positions.

2.6.2 Generator and Parity Check Matrices

The characteristics of a linear block code can be completely described by either its generator matrix, **G**, or its parity check matrix, **H**. The **G**-matrix is responsible for encoding (generating) the check-bits during a write operation, while the **H**-matrix is responsible for calculating the syndrome-bits used for the decoding process. The syndrome, and syndrome decoding process, will be discussed in more detail in Section 2.6.3.

Traditionally, the generator matrix **G** is an $r \times k$ matrix used to generate the set of r parity check-bits $\mathbf{c} = (c_0 c_1 \dots c_{r-1})$ for a given k-bit dataword. Each of the r rows in **G** correspond to a check-bit calculation, while each of the k columns corresponds to a specific data-bit in the dataword. The matrix is organized as follows:

$$\mathbf{G} = \begin{array}{c} \\ c_0 \\ c_1 \\ \vdots \\ c_{r-1} \end{array} \begin{array}{c} \begin{array}{cccc} d_0 & d_1 & \cdots & d_{k-1} \end{array} \\ \begin{pmatrix} h_{0,0} & h_{0,1} & \cdots & h_{0,k-1} \\ h_{1,0} & h_{1,1} & \cdots & h_{1,k-1} \\ \vdots & \vdots & \ddots & \vdots \\ h_{r-1,0} & h_{r-1,1} & \cdots & h_{r-1,k-1} \end{pmatrix} \end{array}.$$

For a given dataword, $\mathbf{d} = (d_0 d_1 \dots d_{k-1})$, the r check-bits are calculated according to

$$\mathbf{c} = \mathbf{d} \cdot \mathbf{G}^T, \tag{2.4}$$

where $\mathbf{G}^T$ is the transpose of **G**. That is,

$$\begin{aligned} c_0 &= d_0 h_{0,0} \oplus d_1 h_{0,1} \oplus \cdots \oplus d_{k-1} h_{0,k-1} \\ c_1 &= d_0 h_{1,0} \oplus d_1 h_{1,1} \oplus \cdots \oplus d_{k-1} h_{1,k-1} \\ &\vdots \\ c_{r-1} &= d_0 h_{r-1,0} \oplus d_1 h_{r-1,1} \oplus \cdots \oplus d_{k-1} h_{r-1,k-1}. \end{aligned}$$

For all calculations, all $h_{i,j}$ belong to the set $\{0, 1\}$, and all calculations are performed modulo-2, using the XOR function. By appending the r check-bits to the k-bit input dataword, the n-bit codeword $\mathbf{v}$ is generated as $\mathbf{v} = [\mathbf{d}\ \mathbf{c}]$. This gives the codeword $\mathbf{v} = (d_0 d_1 \dots d_{k-1}\ \ c_0 c_1 \dots c_{r-1})$.

To decode the received codeword, the $r \times n$ parity check matrix, **H** is defined. In its simplest form, $\mathbf{H} = [\mathbf{G}\ \mathbf{I}_r]$ where **G** is the generator matrix and $\mathbf{I}_r$ is the $r \times r$ identity matrix. Since the parity check matrix contains the generator matrix, the **H**-matrix is sufficient to describe the ECC. The generalized **H**-matrix is given by

$$\mathbf{H} = [\,\mathbf{G} \quad \mathbf{I}_r\,] = \begin{bmatrix} \mathbf{h}_0 \\ \mathbf{h}_1 \\ \vdots \\ \mathbf{h}_{r-1} \end{bmatrix} = \overbrace{\begin{bmatrix} h_{0,0} & h_{0,1} & \cdots & h_{0,k-1} & 1 & 0 & \cdots & 0 \\ h_{1,0} & h_{1,1} & \cdots & h_{1,k-1} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ h_{r-1,0} & h_{r-1,1} & \cdots & h_{r-1,k-1} & 0 & 0 & \cdots & 1 \end{bmatrix}}^{n} \Big\updownarrow r$$

$$\underbrace{\qquad\qquad\qquad}_{k}\underbrace{\qquad\qquad}_{r}$$

The product of a codeword $\mathbf{v}$ and the transpose of the parity check matrix, $\mathbf{H}^T$ provides what is defined as the syndrome, $\mathbf{S}$. For a valid codeword, the syndrome is equal to the null vector, that is, $\mathbf{S} = \mathbf{0} = (00\cdots0)$. This occurs since

$$\mathbf{v} \cdot \mathbf{H}^T = \begin{bmatrix}\mathbf{d}\ \mathbf{c}\end{bmatrix} \cdot \begin{bmatrix}\mathbf{G}^T \\ \mathbf{I}_r\end{bmatrix} \tag{2.5a}$$

$$= \mathbf{d}\mathbf{G}^T \oplus \mathbf{c}\mathbf{I}_r \tag{2.5b}$$

$$= \mathbf{c} \oplus \mathbf{c} \tag{2.5c}$$

$$= \mathbf{0}. \tag{2.5d}$$

Equation 2.5a comes from direct substitution of definitions, that is $\mathbf{v} = [\mathbf{d}\ \mathbf{c}]$ and $\mathbf{H} = [\mathbf{G}\ \mathbf{I}_r]$. Equation 2.5b comes from matrix multiplication. Equation 2.5c is a substitution of Equation 2.4, while Equation 2.5d is a direct result of the XOR function. This occurs because $\mathbf{v}$ is a codeword in the subspace $\mathbf{C}$. By inspection of the parity check matrix $\mathbf{H}$, it can be seen that by concatenating the encoding generator matrix $\mathbf{G}$ and the identity matrix $\mathbf{I}_r$, the syndrome is the recalculation of each of the encoded check-bits compared with their respective stored value of the originally encoded check-bits. Provided that no upsets have occurred, the stored and recalculated check-bit values will be identical, and thus when XOR'ed together will produce the null vector, $\mathbf{0}$. The calculation of a non-zero syndrome value indicates the detection of a invalid codeword.

Parity check matrices are not necessarily limited to the $\mathbf{H} = [\mathbf{G}\ \mathbf{I}_r]$ format. $\mathbf{H}$-matrix of the form $[\mathbf{G}\ \mathbf{I}_r]$ can be modified by matrix row matrix operations and still maintain their random error correcting and detecting properties [11]. Other forms of $\mathbf{H}$-matrices can consist of the concatenation of a series of sub-matrices, each responsible for providing certain error correction and detection or performance based properties. Figure 2.9 shows a set of alternative $\mathbf{H}$-matrix organization examples. For each matrix, the subscript provides the size of the sub-matrix, while the superscript acts as the sub-matrix index. The Single Parity Bit code shown in Figure 2.9(a) and described in Section 2.8.1 consists of a single, $1 \times n$ all-1's matrix row, while the Extended Hamming SEC-DED code (Figure 2.9(b))

$$\mathbf{H}_{\text{Single-Parity-Bit}} = \begin{bmatrix}\mathbf{1}_{1\times n}\end{bmatrix}$$

(a) Single Parity Bit code

$$\mathbf{H}_{\text{Extended Hamming SEC-DED}} = \begin{bmatrix}\mathbf{G}_{(r-1)\times k} & \mathbf{I}_{(r-1)\times(r-1)} & \mathbf{0}_{(r-1)\times 1} \\ & \mathbf{1}_{1\times n} & \end{bmatrix}$$

(b) Extended Hamming SEC-DED code

$$\mathbf{H}_{\text{BCH DEC}} = \begin{bmatrix}\mathbf{G}_{r\times k} & \mathbf{I}_{r\times r}\end{bmatrix} = \begin{bmatrix}\mathbf{H}^1_{(r/2)\times k} & \mathbf{I}_{((r/2)\times(r/2)} & \mathbf{0}_{(r/2)\times(r/2)} \\ \mathbf{H}^2_{(r/2)\times k} & \mathbf{0}_{(r/2)\times(r/2)} & \mathbf{I}_{(r/2)\times(r/2)}\end{bmatrix}$$

(c) BCH DEC code

$$\mathbf{H}_{\text{Reed-Solomon}} = \begin{bmatrix}\mathbf{I}_{b\times b} & \mathbf{I}_{b\times b} & \cdots & \mathbf{I}_{b\times b} & \cdots & \mathbf{I}_{b\times b} \\ \mathbf{I}_{b\times b} & \mathbf{H}^1_{b\times b} & \cdots & \mathbf{H}^i_{b\times b} & \cdots & \mathbf{H}^{2^b-2}_{b\times b} \\ \mathbf{I}_{b\times b} & \mathbf{H}^2_{b\times b} & \cdots & \mathbf{H}^{2i}_{b\times b} & \cdots & \mathbf{H}^{2(2^b-2)}_{b\times b}\end{bmatrix}$$

(d) Reed-Solomon SbEC-DbED code

$$\mathbf{H}_{\text{Extended Reed-Solomon}} = \begin{bmatrix}\mathbf{I}_{b\times b} & \mathbf{I}_{b\times b} & \cdots & \mathbf{I}_{b\times b} & \cdots & \mathbf{I}_{b\times b} & \mathbf{I}_{b\times b} & \mathbf{0}_{b\times b} & \mathbf{0}_{b\times b} \\ \mathbf{I}_{b\times b} & \mathbf{H}^1_{b\times b} & \cdots & \mathbf{H}^i_{b\times b} & \cdots & \mathbf{H}^{2^b-2}_{b\times b} & \mathbf{0}_{b\times b} & \mathbf{I}_{b\times b} & \mathbf{0}_{b\times b} \\ \mathbf{I}_{b\times b} & \mathbf{H}^2_{b\times b} & \cdots & \mathbf{H}^{2i}_{b\times b} & \cdots & \mathbf{H}^{2(2^b-2)}_{b\times b} & \mathbf{0}_{b\times b} & \mathbf{0}_{b\times b} & \mathbf{I}_{b\times b}\end{bmatrix}$$

(e) Extended Reed-Solomon SbEC-DbED code

Figure 2.9: Example **H**-matrix Configurations Using Sub-matrices

described in Section 2.8.3 horizontally appends the Single Parity Bit **H**-matrix to a $[\mathbf{G}\ \mathbf{I}_r]$ formatted Hamming SEC code (along with an $r-1 \times 1$ all-0's row) to provide double random bit error detection. The BCH DEC code (Figure 2.9(c)) described in Section 2.8.5 can be expressed in either the $\mathbf{H} = [\mathbf{G}\ \mathbf{I}_r]$ format or divided into in sub-matrices determined by its construction process. For the Reed-Solomon codes (Figure 2.9(d) and Figure 2.9(e)), the horizontally concatenated $b \times b$ identity matrices align with the byte boundaries for the code, and help provide the within-byte error correction capabilities for the code. The repeated identity matrix structure is leveraged for the set of proposed codes to provided its adjacent error detection capabilities, and in conjunction with other sub-matrices, helps facilitate the adjacent error correction functionality. The repeated identity matrix structure is described in more detail in Section 3.2.1.

2.6.3 Syndrome

Once data has been written into memory, it will not necessarily remain constant. As the data resides in memory, it is continually being exposed to external radiation such as alpha particles and galactic rays. This leaves the memory cells susceptible to upsets. These upsets must be recognized and handled by the ECC decoder. To model this mathematically, we let $\mathbf{v} = [\mathbf{d}\ \mathbf{c}] = (d_0 d_1 \ldots d_{k-1} c_0 c_1 \ldots c_{r-1})$ be a codeword written into memory. Since this codeword is stored under noisy circumstances, we can let $\mathbf{r} = (d'_0 d'_1 \ldots d'_{k-1} c'_0 c'_1 \ldots c'_{r-1})$ be the received codeword during a read operation. The received code word $\mathbf{r}$ may or may not differ from the transmitted codeword $\mathbf{v}$. The addition of errors to the vector $\mathbf{v}$ can be expressed by the vector sum modulo-2 (i.e., XOR) as

$$\mathbf{r} = \mathbf{v} \oplus \mathbf{e} \tag{2.6}$$

where $\mathbf{e} = (e_0 e_1 \ldots e_{n-1})$ is an n-bit vector representative of the error pattern injected into the transmitted codeword. For $e_i = 1$ the i-th element of $\mathbf{r}$ is not equal to the corresponding i-th element of $\mathbf{v}$. This indicates an error at the i-th element in the received codeword $\mathbf{r}$. When $e_i = 0$, the i-th elements in $\mathbf{r}$ and $\mathbf{v}$ are the identical, and thus the i-th element of $\mathbf{r}$ does not contain an error.

Upon receiving $\mathbf{r}$, the decoder must determine the existence of errors within the received codeword. Depending upon the extent of the error, appropriate error correction or detection may occur. When $\mathbf{r}$ is received, the syndrome generator performs the following computation

$$\mathbf{S} = \mathbf{r} \cdot \mathbf{H}^T. \tag{2.7}$$

The output of the generator $\mathbf{S} = (S_0 S_1 \cdots S_{r-1})$ is defined as the syndrome of $\mathbf{r}$. If $\mathbf{S} = \mathbf{0}$, then $\mathbf{v}$ is considered to be a valid codeword, and no detectable errors exist within $\mathbf{r}$. If $\mathbf{S} \neq \mathbf{0}$, then an error has occurred and $\mathbf{r}$ is not a valid codeword. A critical feature of the syndrome calculation is its independence of the transmitted codeword $\mathbf{v}$. The syndrome calculation is solely dependent only on the error pattern $\mathbf{e}$. Since $\mathbf{r}$ is the vector sum of $\mathbf{v}$ and $\mathbf{e}$, we have the following computation,

$$\mathbf{S} = \mathbf{r} \cdot \mathbf{H}^T = (\mathbf{v} \oplus \mathbf{e}) \cdot \mathbf{H}^T = (\mathbf{v} \cdot \mathbf{H}^T) \oplus (\mathbf{e} \cdot \mathbf{H}^T).$$

Since $\mathbf{v} \cdot \mathbf{H}^T = \mathbf{0}$ from Equation 2.5, $\mathbf{S}$ depends only on the error vector $\mathbf{e}$, where

$$\mathbf{S} = \mathbf{e} \cdot \mathbf{H}^T. \tag{2.8}$$

Once the syndrome has been calculated, it must be processed to determine the originally transmitted codeword, $\mathbf{v}$. To correct a particular error pattern, the error must map to a unique error correcting syndrome. In the simplest case, if $\mathbf{S} = \mathbf{0}$, then the error vector $\mathbf{e} = \mathbf{0}$ and no error has occurred. In the event that $\mathbf{S} \neq \mathbf{0}$, then two possible cases arise:

1. The syndrome is identical to an error correcting syndrome, or
2. The syndrome does not match an error correcting syndrome.

Based on the syndrome calculation procedure, the generated syndrome value will always produce the modulo-2 linear combination (XOR combination) of the columns in the **H**-matrix corresponding to the upset bits in the received codeword. That is, provided that a single-bit-error occurred, the syndrome pattern will match the i-th column vector of **H** corresponding to the i-th bit in $\mathbf{r}$. Therefore, to facilitate for single-bit-error correction, all of the **H**-matrix column vectors must be unique, and non-zero. The correction of other error types depends on the uniqueness of their syndrome values. This will be discussed in more detail in Section 3.1.1. Once the error location has been identified, error correction can proceed by inverting the bit(s) in error and then forwarding the corrected data to the output data bus and optionally rewriting the corrected data back into memory.

In the event that $\mathbf{S} \neq \mathbf{0}$ and the syndrome does not match an error correcting syndrome, then an error has been detected, but its location cannot be determined. This situation occurs when the error type is beyond that of the ECC's correcting capability. This information can be forwarded external to the memory and handled at the system-level. How the system handles the detection of the error will depend on its application, but from the memory standpoint, the error has been recognized, and it is left to the user or system to determine how to proceed. For example, if an error is detected in a multi-tiered cache memory system, the dataword can be reloaded from a higher level in the memory hierarchy.

2.6.4 Encoding-Decoding Example

The following is an example of the encoding-decoding process for a (7,4) Hamming SEC code. The generation of this code is described later in Section 2.8.2. The **H**-matrix contains three rows (one for each check/syndrome-bit) and seven columns (one for each code-bit). The matrix is the concatenation of the encoding generator matrix **G** and the identity matrix, $\mathbf{I}_r = \mathbf{I}_3$.

$$\mathbf{H}_{SEC} = [\mathbf{G}\ \mathbf{I}_3] = \begin{array}{c} \\ S_0 \\ S_1 \\ S_2 \end{array} \begin{array}{c} \begin{array}{cccccccc} d_0 & d_1 & d_2 & d_3 & & c_0 & c_1 & c_2 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 0 & & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & & 0 & 0 & 1 \end{pmatrix} \end{array}$$

Consider the $k = 4$-bit dataword $\mathbf{d} = (1100)$. Multiplying **d** by the transpose of **G** provides the parity (or check) bits $\mathbf{c} = (010)$.

$$\mathbf{c} = \mathbf{d} \cdot \mathbf{G}^T = (1100) \cdot \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} = (010)$$

This can also be expressed as,

$$\begin{aligned} c_0 &= d_0 \oplus d_1 \oplus d_2 = 1 \oplus 1 \oplus 0 = 0 \\ c_1 &= d_0 \oplus d_2 \oplus d_3 = 1 \oplus 0 \oplus 0 = 1 \\ c_2 &= d_0 \oplus d_1 \oplus d_3 = 1 \oplus 1 \oplus 0 = 0 \end{aligned}$$

Appending **c** to **d** provides the codeword $\mathbf{v} = [\mathbf{d}\ \mathbf{c}] = (1100\ 010)$. This codeword is then stored in memory. Provided that no bits are upset before the codeword is read from memory, the syndrome produces the null vector, that is $\mathbf{S} = \mathbf{v} \cdot \mathbf{H}_{SEC}^T = (000) = \mathbf{0}$.

In the event that a bit error does occur, the syndrome will equal the column vector of $\mathbf{H}_{SEC}$ representative of the error. Consider the example where the third bit, d_2, in the codeword **v** is corrupted. Since $\mathbf{v} = [\mathbf{d}\ \mathbf{c}] = (d_0 d_1 \underline{\mathbf{d}_2} d_3\ c_0 c_1 c_2)$, the received codeword, **r** will be $(11\underline{1}0\ 010)$, notice the difference in the third bit. Calculating the syndrome produces $\mathbf{S} = (110)$.

$$\mathbf{S} = \mathbf{r} \cdot \mathbf{H}_{SEC}^{T} = (1110\ 010) \cdot \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = (110)$$

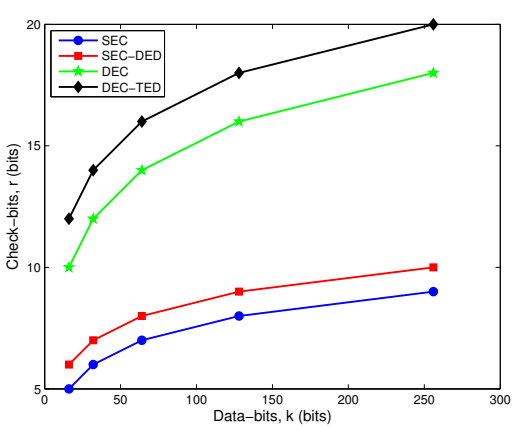
This syndrome pattern is equal to the third column vector in $\mathbf{H}_{SEC}$ (or third row vector of $\mathbf{H}_{SEC}^{T}$), indicating an error in the third position of the received codeword $\mathbf{r}$. Once the location of a bit error has been determined, it can be corrected by inverting the bit. The corrected data can then be forwarded to the output data bus, and optionally rewritten into the memory.

2.7 Performance Metrics

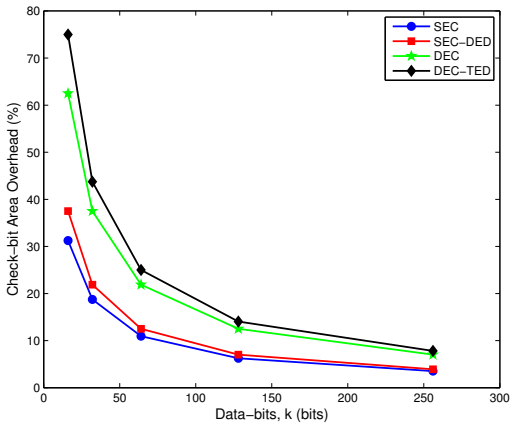
This section describes the high-level performance metrics used to evaluate and compare ECCs. These measures are based solely on the parity check matrix and allow for a high-level comparison during the pre-circuit design phase. Each of these metrics correspond to a post-circuit implementation metric including, area overhead, performance impact, and robustness.

2.7.1 Area Overhead

The area overhead associated with a particular ECC is measured in terms of the number of check-bits required for the code and the logic gate count of the encoder-decoder logic circuitry. Since check-bits must be added for each stored dataword, the number of check-bits required for a code will have a significant impact on the area overhead. For instance, a (39, 32) SEC-DED code requires seven check-bits for every 32 data-bits. This corresponds to a 21.9% increase in bit-cell array area. The number of required check-bits for a given ECC corresponds directly with the number of rows in its parity check matrix. Figure 2.10(a) shows the absolute number of check-bits required versus dataword length, while Figure 2.10(b) expresses the check-bit area overhead as a percentage of data-bits per codeword. These figures compare the associated check-bit overhead for SEC, SEC-DED, DEC BCH, and DEC-TED BCH codes. Notice that the incremental number of check-bits for a DEC code is always double that of a SEC code independent of dataword length. The same is true for DEC-TED relative to SEC-DED codes. The proposed codes explore the design space bounded by these curves.



(a) Check-bits vs. Data-bits



(b) Check-bit Area Overhead vs. Data-bits

Figure 2.10: Area overhead comparisons for SEC, SEC-DED, DEC BCH, and DEC-TED BCH codes of various data lengths.

The area required for the check- and syndrome-bit generation logic is proportional to its XOR logic gate count. Since each check- and syndrome-bit is generated using modulo-2 addition, XORing the indicated bits within each matrix row will generate the respective check- or syndrome-bit value. The total number of 2-input XOR gates required to generate these values is determined by the total weight (total number of 1's) in the parity check matrix. This is given by

$$\text{2-Input XOR Gates}(\mathbf{H}) = \text{Total Weight}(\mathbf{H}) - \text{Number of Rows}(\mathbf{H}). \tag{2.9}$$

Since an n-input XOR gate can be implemented using $n-1$ 2-input XOR gates, the total number of 2-input XOR gates required per syndrome-bit is equal to that bit's row weight minus one. The total number of 2-input XOR gates required for the check and syndrome calculations is equal to the total weight of the parity check matrix minus its number of rows (or check-bits). Finally, since the check-bit value is regenerated during the syndrome-bit calculation, the same hardware can be used for each set of calculations.

2.7.2 Performance Impact

Integrating an ECC circuit into a memory system will add an additional delay penalty to each read and write operation. Although some ECC schemes exist where the circuit is placed off of the memory's read and write critical paths, they do so at the expense of system reliability, compromising up-to-date data reliability for speed [48].

The maximum delay penalty associated with an ECC circuit is comprised of the delay through the syndrome-bit generation XOR logic tree, syndrome decoder, and error corrector during the ECC decoding phase. This value is always greater than the check-bit generation delay during the encoding phase since each check-bit is regenerated during the decoding phase. The syndrome-bit generation delay is determined by the maximum number of XOR gate logic levels in the syndrome-bit generation logic, and is given by

$$\text{Max Syndrome Logic Depth}(\mathbf{H}) = \lceil \log_2(\text{Maximum Row Weight}(\mathbf{H})) \rceil. \tag{2.10}$$

Since an n-input XOR gate can be implemented using $n-1$ 2-input XOR gates, the maximum number of XOR logic levels will be equal to the number of XOR gate delays in the longest syndrome-bit XOR tree. This figure of merit can be used to compare the relative speed penalties associated with various ECC implementations. Equation 2.10 assumes each of the syndrome-bit calculations are independent of one another. A modified version of this equation is presented in Section 3.2.5 to include calculation dependencies. This modification is required for the proposed codes.

2.7.3 Robustness

When an error occurs in a memory that is beyond the error handling capabilities of the ECC, various outcomes may occur during a subsequent read operation. Since the ECC has no means of determining that an error has occurred beyond its capability (otherwise, the error would at least be detectable) the codeword will still be processed by the decoder circuitry and a syndrome will still be generated. The ideal situation occurs when a non-zero syndrome not matching any of the error correcting syndromes is produced. This error will be detected as per any other non-error-pattern-matching syndrome.

More destructively, a syndrome can match one of the error correcting syndromes and the error "correction" process will proceed, or the zero syndrome will be produced, indicated that no error has occurred at all. Either way, undetected erroneous data will reach the output data bus. Errors that go undetected by the ECC are called miscorrection errors. As an example, if an error were to upset bits d_0, d_1, and d_2 in the (7, 4) SEC code in Section 2.6.4, this 3-adjacent error pattern would match that of the single-bit error for bit c_0. This miscorrection of bit c_0 would produce an additional error in the codeword leading to four corrupted bits rather than the intended corrected codeword. Miscorrection probabilities for each error type can be calculated directly from the parity check matrix.

For a given error type, the miscorrection probability is equal to the number of error patterns that are shared with error correcting syndromes divided by the total number of

error patterns producible for the given error type minus the number of error correcting syndromes. The miscorrection probability of an e-random error for a particular (n, k) code is given by

$$P(e\text{-Random Error}) = \frac{\text{Sharable Syndromes}}{\binom{n}{e} - e\text{-column correcting syndromes}} \tag{2.11}$$

where Sharable Syndromes is the number of linear combinations of e columns vectors in the **H**-matrix that falsely produce either an error correcting syndrome or the zero vector. This value is then divided by the total number of e-bit combinations for the given codeword size minus the number of e-column error correcting syndrome patterns. Miscorrection probabilities can be calculated for any error type including, random, adjacent, and burst errors.

2.8 Example Codes

In this section, a subset of the commonly used error correcting and detecting codes are introduced with examples. These include: single-parity-bit [11], Hamming SEC and SEC-DED [33], Hsiao SEC-DED [31], BCH [11], Reed-Solomon [65], and Dutta SEC-DED-DAEC [44] codes. Each of these codes rely on the mathematical foundation presented in Section 2.6. Any additional concepts required for a specific code are provided within the code's respective subsection.

2.8.1 Single-Parity-Bit Codes

The simplest form of error detection requires only one additional parity check-bit per dataword. The check-bit ensures that the total number of 1's in a stored codeword is always even. For this reason, single-parity-bit codes are also known as even parity code. The (k+1, k) even parity code is defined by the all 1's $1 \times n$ **H**-matrix.

$$\mathbf{H}_{\text{Single-Parity-Bit}} = (1\ 1\ \cdots\ 1)$$

As an example, assuming a eight-bit dataword $\mathbf{d} = (0111\ 0101)$, a parity bit, given by: $p = d_1 \oplus d_2 \oplus \cdots \oplus d_8$, produces a single parity check-bit of value '1'. This value is then appended to $\mathbf{d}$ resulting in the stored codeword $\mathbf{v} = (\mathbf{d}\ \underline{p}) = (0111\ 0101\ \underline{1})$. Notice that the stored codeword contains an even number of 1's. When the codeword is later read, the decoder can determine its validity by ensuring the even parity of the codeword. The reading of a non-even (i.e., odd) weight codeword indicates the detection an error.

Since this encoding scheme contains no information regarding the location of the bit-error, it can only be used to detect errors. Additionally, in the event that two bits are upset, the scheme would incorrectly return a valid codeword, thus showing the limitation of the $d_{min} = 2$ code. By using multiple single-parity-bit codes and judiciously selecting which bits in the dataword are used in each parity calculation, more complex codes with increased degree of error correction and detection can be created.

2.8.2 Hamming SEC Codes

An example (7, 4) Hamming SEC code was shown in Section 2.6.4 to illustrate the basic ECC encoding and decoding process. This code has a minimum Hamming distance $d_{min} = 3$, and is capable of correcting all single bit errors. Constructing a Hamming SEC **H**-matrix requires that all column vectors be unique and non-zero. To meet these requirements, $n \leq 2^r - 1$, where n is the codeword length, and r is the number of check-bits. The encoder and decoder circuit implementations for the (7, 4) Hamming SEC code example is shown in Figure 2.11.

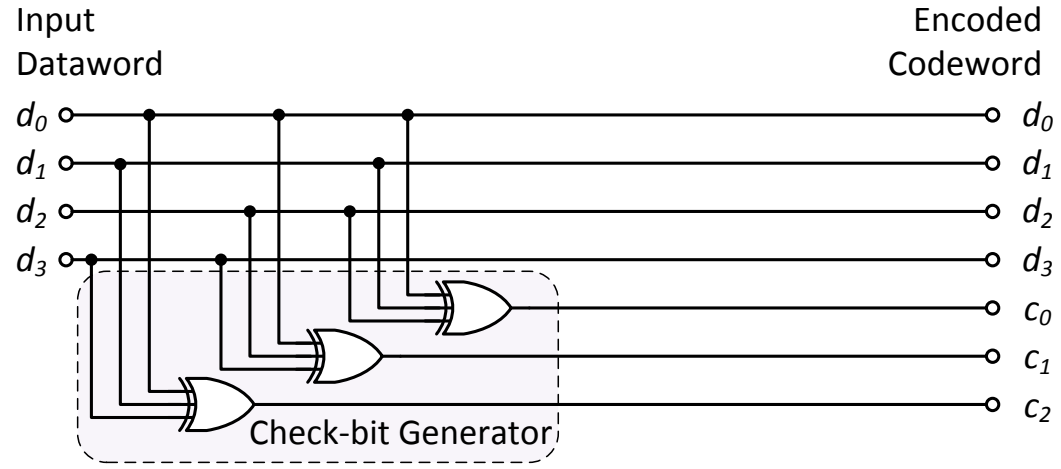
The encoder check-bits are computed from the input dataword by the following equations using XOR gates:

$$\begin{aligned} c_0 &= d_0 \oplus d_1 \oplus d_2 \\ c_1 &= d_0 \oplus d_2 \oplus d_3 \\ c_2 &= d_0 \oplus d_1 \oplus d_3. \end{aligned}$$

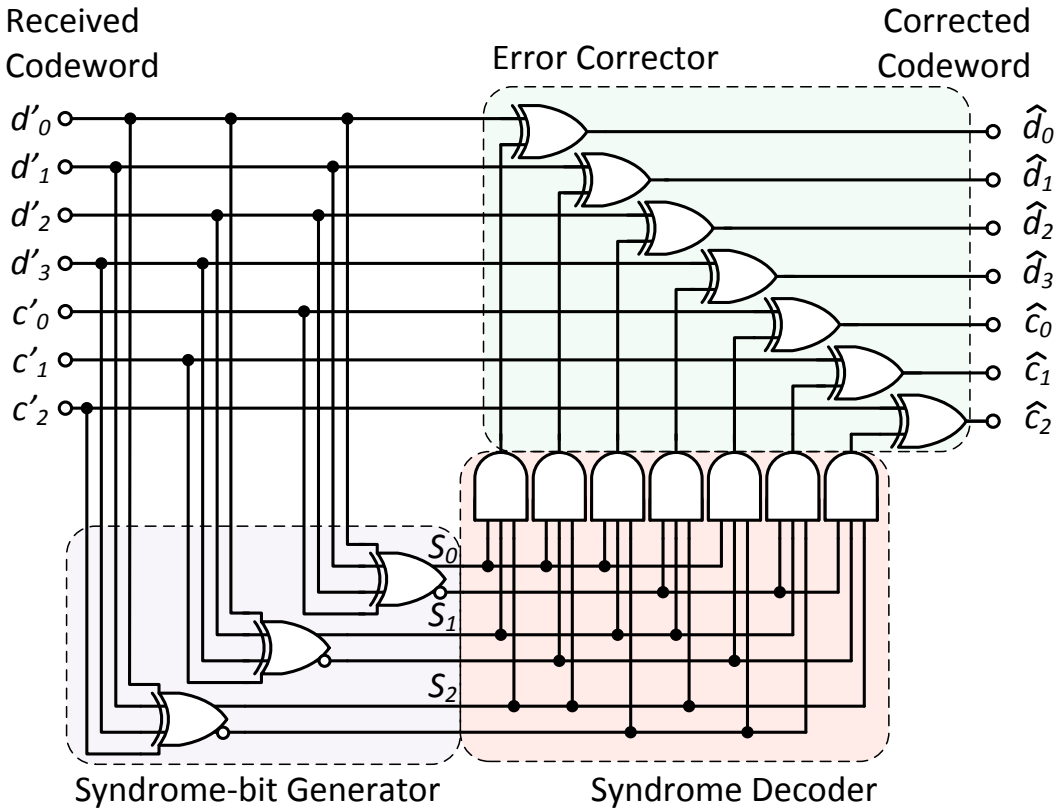
During decoding, the syndrome **S** is calculated for the received word $\mathbf{r} = (d'_0 d'_1 d'_2 d'_3\ c'_0 c'_1 c'_2)$ as,

$$\begin{aligned} S_0 &= d'_0 \oplus d'_1 \oplus d'_2 \oplus c'_0 \\ S_1 &= d'_0 \oplus d'_2 \oplus d'_3 \oplus c'_1 \\ S_2 &= d'_0 \oplus d'_1 \oplus d'_3 \oplus c'_2. \end{aligned}$$

The syndrome $\mathbf{S} = (S_0 S_1 S_2)$ is then decoded via a set of AND gates to indicate the location of any single-bit errors that may have occurred. Error correction occurs only if the binary syndrome pattern matches any of the binary column vectors in **H**. The erroneous bit corresponding to the matching column vector is then corrected via bit-wise inversion using an XOR gate.



(a) Encoder



(b) Decoder

Figure 2.11: Encoder and decoder of the Hamming (7,4) SEC code.

2.8.3 Hamming SEC-DED Codes

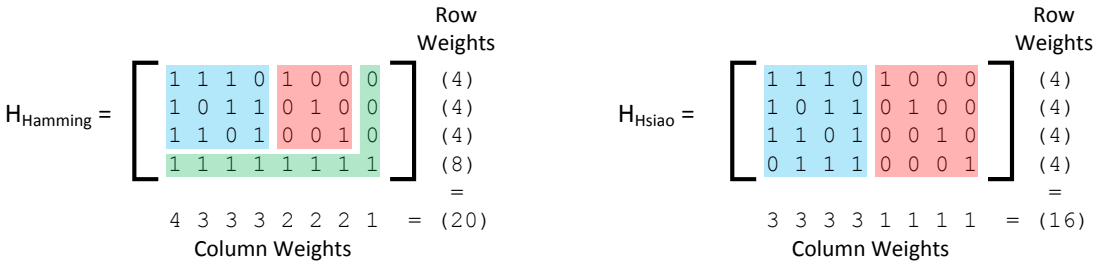
A Hamming SEC code can be extended to detect double-random-bit errors in addition to its single-bit error correcting property to form a Hamming SEC-DED code. The code adds an overall even parity check-bit to an existing Hamming SEC code. The $\mathbf{H}_{SEC}$ matrix is modified by adding an all 1's row vector and a weight-1 column vector with upper $r-1$ all 0's to an existing $(r-1) \times (n-1)$ binary $\mathbf{H}_{SEC}$ matrix. The $\mathbf{H}$-matrix is written as

$$\mathbf{H}_{\text{SEC-DED}} = \begin{bmatrix} \boxed{\mathbf{H}_{\text{SEC}}} & \begin{matrix} 0 \\ 0 \\ \vdots \\ 0 \end{matrix} \\ \begin{matrix} 1 & 1 & \cdots & 1 \end{matrix} & 1 \end{bmatrix} \begin{matrix} \updownarrow r-1 \\ \updownarrow 1 \end{matrix}$$
$$\longleftarrow n-1 \longrightarrow \leftarrow 1 \rightarrow$$

The overall parity-bit provides an even parity check for the entire codeword, including both the data and check-bits. This is used in conjunction with the standard SEC check-bits to provide the DED functionality when decoding the syndrome. If $\mathbf{S} = \mathbf{0}$, then the received codeword is error-free. If $\mathbf{S} \neq \mathbf{0}$ and the overall parity bit is '1', then an odd number of errors have occurred. If the syndrome pattern is identical to a column vector in the $\mathbf{H}$-matrix, then the corresponding bit-error is corrected. If the syndrome pattern is not identical to a column in $\mathbf{H}$, then three or more odd number of errors have occurred. These errors are detectable, but not correctable. Finally, in the event that $\mathbf{S} \neq \mathbf{0}$ and the overall parity bit is '0', then an even number of errors have occurred and are detectable. It can easily be shown that the maximum codeword length for an (n, k) binary SEC-DED code is $2^r - 1$.

2.8.4 Hsiao SEC-DED Codes

Hsiao SEC-DED codes, also known as odd-weight-column codes, provide the same level of coding efficiency as Hamming SEC-DED codes in terms of check-bit overhead but provide improvements in terms of speed, area, and reliability of the decoding logic. Improvements are realized through the implementation of the code's double error detection functionality. Rather than relying on an all 1's row in its $\mathbf{H}$-matrix, Hsiao codes leverage odd Hamming weighted columns to produce the DED functionality. Since the sum of any two odd weight vectors will result in an even weight vector, any double-bit errors will result in an even



(a) Hamming (b) Hsiao

Figure 2.12: (8,4) Hamming and Hsiao SEC-DED Matrices

weighted syndrome vector. This vector will be unique from all of the odd weighted column vectors, and hence allows the Hsiao code to provide a minimum Hamming distance of $d_{min} = 4$.

A comparison between the (8, 4) SEC-DED Hamming and Hsiao matrices is shown in Figure 2.12. The Hsiao SEC-DED matrix has a lower total weight (16) compared to the Hamming SEC-DED matrix (20). This results in a decrease in the total number of XOR gates (Hsiao = 12, Hamming = 16), and a lower number of logic levels in the XOR decoder tree (Hsiao = $\lceil \log_2(4) \rceil = 2$, Hamming = $\lceil \log_2(8) \rceil = 3$). These features produce smaller decoders and reduced delay penalties for the Hsiao SEC-DED implementations as compared to the Hamming SEC-DED implementations. For this reason, throughout the remainder of this work, the Hsiao codes are used for all SEC-DED performance evaluations.

2.8.5 BCH Codes

Bose, Ray-Chaudhuri, Hocquenghem (BCH) codes provide error correcting capabilities beyond those of the Hamming and Hsiao SEC-DED codes. They were invented independently by Hocquenghem in 1959 and by Bose and Ray-Chaudhuri in 1960 [11]. These polynomial-based codes provide error correction for multiple random bit errors at the expense of an increase in the number of check-bits and decoder complexity. BCH codes using higher order Galois fields can be used to correct many random errors; however, these codes require a substantial overhead in terms of area and delay and are not typically considered for use in high-speed embedded memories. Here we briefly consider binary BCH codes used for correcting two random bit errors. This scheme has, although in a very limited amount, been used in some embedded SRAM applications [19, 12]. For a detailed description of BCH codes, the reader is referred to any number of texts on coding theory [64, 50, 66].

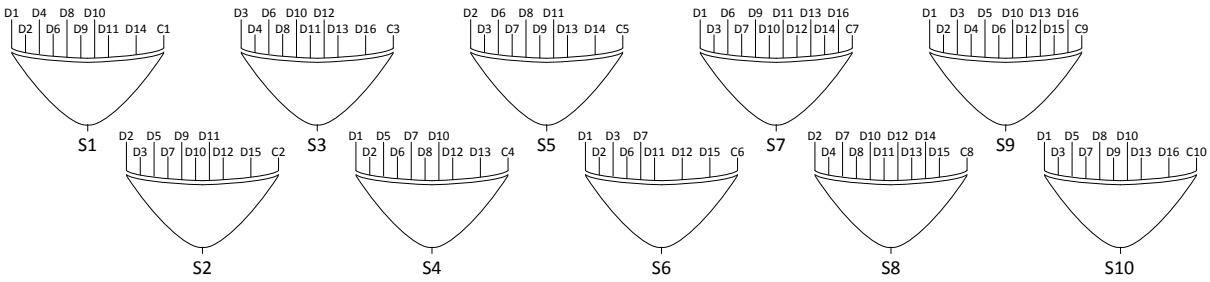


Figure 2.13: (26, 16) BCH DEC Syndrome Generation Logic

The double error correcting (DEC) BCH code uses a generator polynomial $\mathbf{g}(x)$ and its roots α^i to construct an **H**-matrix in which every 4 columns are linearly independent (i.e., every XOR sum of any two or fewer **H**-matrix columns is distinct, and can therefore produce a distinct error correcting syndrome pattern). **H**-matrices for the implemented (26, 16), (44, 32), and (78, 64) BCH DEC codes are provided in Appendix A.

BCH check- and syndrome-bits are generated in the same manner as the previously presented single error correcting codes using XOR gates. The syndrome-bit generation logic for the (26, 16) BCH DEC code is shown in Figure 2.13. Syndromes can be decoded either serially using a linear feedback shift register (this method consumes minimal area, but requires multiple clock cycles to execute), or in parallel (requiring a decoding circuit for each correctable syndrome pattern). This requires a higher area overhead but can be executed in a single cycle using combinational logic. All of the BCH DEC circuits in this work have been designed for single cycle execution. For the (78, 64) BCH DEC code there are 14 syndrome bits and $\binom{78}{2} = 3003$ double bit random bit error patterns given the 78-bit codeword size. As such, the parallelized syndrome decoder is effectively a 14-to-78 decoder with 78 correctable single bit syndrome patterns and 3003 correctable double bit patterns.

2.8.6 Reed-Solomon Codes

Reed-Solomon (RS) codes are a subclass of non-binary BCH codes in which error correction is performed on symbols, or bytes, as opposed to individual bits [65]. Each byte consists of a cluster of b-bits, and any sized burst error contained within a given byte can be corrected. For this work, we consider the single byte error correcting-double byte error detecting (SbEC-DbED) Reed Solomon codes. These are similar to SEC-DED codes, only instead of handling bit errors, they consider byte errors. Figure 2.14 shows the byte-wise organization for a (25, 16) code using a $b = 3$ byte size. Notice that the 25-bit codeword has been segmented into $\lceil n/b \rceil = 9$ bytes, eight equal to the $b = 3$ bytes size, and one of size 1-bit, (25 mod 3 = 1).

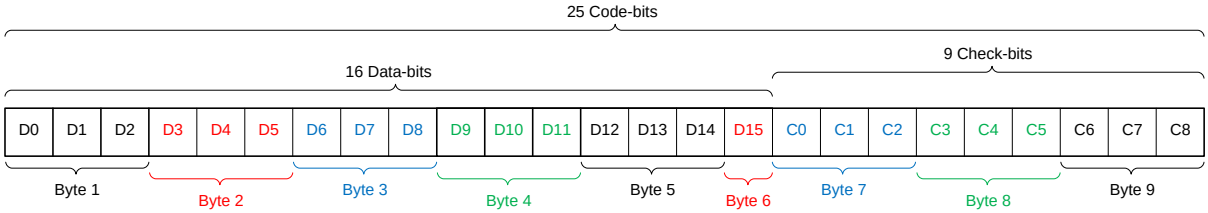


Figure 2.14: (25, 16) $b = 3$ Reed Solomon code byte organization

Table 2.1: Code Parameters for RS SbEC-DbED Codes

b(bits)	n_{max}(bits)	k_{max}(bits)
2	12	6
3	30	21
4	72	60
5	170	155

The RS code is able to correct any b-bit burst error pattern so long as it resides within a single byte. To provide this level of error correction the code requires 3 check-bytes ($3 \cdot b$ check-bits). If an upset crosses over the byte boundary however, it can be detected but not corrected. By interleaving the bytes of multiple codewords, as in Figure 1.8(b), adjacent bytes within a codeword can be separated by as many as b-bits multiplied by the degree of interleaving, I, minus one, that is $b \cdot (I - 1)$. A limiting factor for the RS codes is that its maximum codeword length is determined by the expression $N = b \cdot (2^b + 2)$ and, as such there is a maximum data-bit length k permitted per byte size, shown in Table 2.1. This requires RS codes to provide a certain amount of error protection based on their number of databits, and prohibits them from providing a lower check-bit overhead.

Similar to the BCH codes, RS codes can be decoded either serially over multiple clock cycles or using a parallelized combinational logic decoder circuit in a single clock cycle. In this work, parallelized decoders have been implemented, and the **H**-matrices for the implemented (25, 16) S3EC-D3ED, (44, 32) S4EC-D4ED, and (79, 64) S5EC-D5ED RS codes are listed in Appendix A.

2.8.7 Dutta SEC-DED-DAEC Codes

SEC-DED-DAEC Dutta codes [44] attempt to compromise between the low area overhead of SEC-DED codes and the higher error correcting capabilities of BCH and Reed-Solomon codes. They use the same number of check-bits as the Hsiao and Hamming SEC-DED codes, but add an additional constraint on the **H**-matrix column selection procedure. This

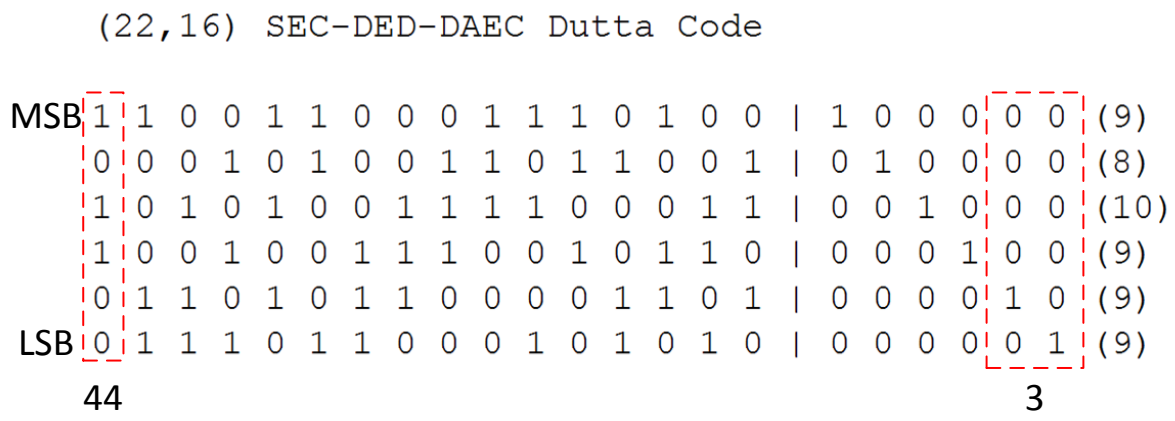


Figure 2.15: (22, 16) SEC-DED-DAEC Dutta code. Adapted from [44].

constraint provides the DAEC property. To ensure proper functional operation for all SEC-DED-DAEC codes, all column vectors within the parity check matrix are selected such that:

1. All columns are non-zero, (i.e., no column consists of the null vector, **0**),
2. All of the columns are distinct,
3. No linear dependencies exist involving 3 or fewer columns (i.e., the XOR of any two columns does not equal any of the individual columns), and
4. The linear combination of any two adjacent columns does not equal any of the individual columns or the linear combination of any other two adjacent columns.

The first three constraints are identical to those of the Hamming SEC-DED codes, while the fourth constraint, proposed by Dutta, requires that the linear combinations of any two adjacent columns vectors $h_i \oplus h_{i+1}$ be distinct from one another and from each individual column vector h_i. While Constraints 1 and 2 provide distinct syndrome patterns for all single-bit errors, Constraint 4 increases the set of error correcting syndromes to include all 2-adjacent bit-errors. Figure 2.15 shows a (22, 16) SEC-DED-DAEC Dutta code, and Table 2.2 provides the decimal equivalent for each of the code's columns and linear combination of 2-adjacent columns. For each column, the top row is considered to be the most significant bit, while the bottom row is the least significant bit. The values are listed in order starting with the left-most side of the **H**-matrix. Notice that each value is distinct.

Table 2.2: (22, 16) SEC-DED-DAEC Dutta Code Column Uniqueness Table

Column Values, h_i (22)	{ **44**, 35, 11, 21, 42, 49, 7, 14, 28, 56, 41, 52, 19, 38, 13, 26, 32, 16, 8, 4, 2, 1 }
2-Adjacent Column XOR Results, $h_i \oplus h_{i+1}$ (21)	{ 15, 40, 30, 63, 27, 54, 9, 18, 36, 17, 29, 39, 53, 43, 23, 58, 48, 24, 12, 6, **3** }

A key feature of Dutta's SEC-DED-DAEC code is that it uses the same number of check-bits as the conventional Hamming and Hsiao SEC-DED codes. A drawback to this code however is that to maintain the iso-check-bit overhead, some of the adjacent double-bit error correcting syndromes are shared with some of the non-adjacent double-bit error syndrome patterns. This leads to potential miscorrections for the 2-random-bit error type. The miscorrection probability is determined using Equation 2.11. For the (22, 16) Dutta code, we first consider the number of shareable syndrome patterns between the non-adjacent 2-random-bit error patterns and the correctable syndrome patterns. Through exhaustive search, 135 sharable syndrome patterns are counted. For the 22-bit codeword size, there are $\binom{22}{2} = 231$ possible 2-bit error patterns, of which $n-1 = 21$ of these have the bit errors in adjacent positions, (there are therefore $231 - 21 = 210$ non-adjacent two-bit syndrome patterns). Therefore, using Equation 2.11, the probability of a non-adjacent 2-random bit error being miscorrected as a correctable error pattern is equal to $135/210 = 64.3\%$. Although this is undesirable, the miscorrection penalty can be justified in the context of MBU soft errors since the likelihood of two adjacent-bit upsets is much higher than two non-adjacent bit upsets. It is still however desirable to minimize this quantity; and as such, this is considered with the proposed class of codes. **H**-matrices for the implemented (22, 16), (39, 32), and (72, 64) SEC-DED-DAEC codes are listed in Appendix A.

2.9 Summary

In this chapter, relevant background material regarding the SRAM organization structure, error type classifications, and ECC comparison metrics have been discussed. Further, all of the ECC schemes used for comparison in the subsequent chapters have been introduced.

Chapter 3

Proposed Class of Error Correction Codes

*In this chapter the basic structure of the proposed codes is introduced and a set of rules for constructing their parity check matrices is presented. The details of the code construction procedure is then discussed, and an **H**-matrix solution solver/evaluator designed in Matlab is described. A set of the proposed codes generated by the solver is then evaluated relative to existing codes. The chapter concludes with the presentation of a number of modified codes derived from the basic proposed code's structure.*

3.1 Code Structure

The proposed class of error correcting codes is designed to mitigate the soft error problem for on-chip embedded SRAMs in aggressively scaled technologies. Radiation strikes that would have appeared as SBUs in previous technology generations are now appearing as burst and adjacent MBUs in sub-100 nm technologies. Further, these upsets not only affect more bits, but become more frequent with continued scaling. As a baseline, the proposed codes offer the traditional SEC-DED functionality found in Hamming and Hsiao codes and the DAEC found in Dutta codes. They also offer scalable adjacent and burst error detection (yAED, zBED), the option for triple adjacent error correction (TAEC), and a reduction in miscorrection probabilities compared to other codes. Parity check matrices have been constructed for the proposed codes with up to 11-bits of adjacent error detection for memories with 16, 32, and 64 data-bits per codeword. In these codes, adjacent error correction and detection is emphasized since these type of MBUs form the primary error pattern limiting SEC-DED ECCs in scaled memory technologies. Additionally, they emphasize a large degree of error detection coverage since if instantiated in a multi-tiered

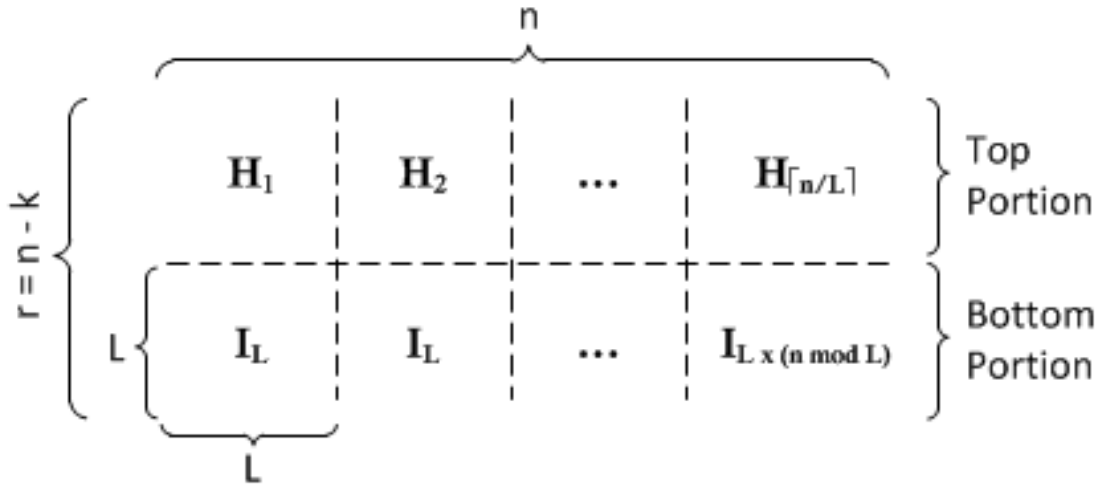


Figure 3.1: Generalized $(n,\ k)$ $\mathbf{I}_L$ **H**-matrix structure for the proposed class of ECCs

$$\mathbf{H} = \begin{pmatrix} 0000 & 1010 & 0000 & 0101 & 1010 & 111 \\ 0101 & 0101 & 0000 & 1010 & 0000 & 111 \\ 0101 & 1010 & 1010 & 0000 & 0101 & 000 \\ & & & & & \\ 1000 & 1000 & 1000 & 1000 & 1000 & 100 \\ 0100 & 0100 & 0100 & 0100 & 0100 & 010 \\ 0010 & 0010 & 0010 & 0010 & 0010 & 001 \\ 0001 & 0001 & 0001 & 0001 & 0001 & 000 \end{pmatrix}$$

Figure 3.2: Example (23, 16) $\mathbf{I}_4$ SEC-DED-DAEC-5AED Code

cache system, upon detection of an uncorrectable error, data can be reloaded from a higher level in the memory system hierarchy.

The generalized **H**-matrix structure for the proposed class of ECCs is shown in Figure 3.1 and follows the conventional $(n,\ k)$ naming nomenclature appended with an identity matrix size, $\mathbf{I}_L$. The **H**-matrix structure leverages the repeated identity matrix configuration seen in the Reed-Solomon matrix examples in Figure 2.9(d) and Figure 2.9(e), along with a series of H_i sub-matrices that have each been designed using a selective bit-placement strategy in the spirit of the Dutta codes presented in Section 2.8.7. The provided degree of error protection coverage includes the standard SEC-DED coverage along with x-bits of adjacent error correction, and y-bits of adjacent error detection (SEC-DED-xAEC-yAED). An example (23, 16) $\mathbf{I}_4$ SEC-DED-DAEC-5AED code is shown in Figure 3.2. It provides double adjacent error correction and 5-bits of adjacent error detection in addition to the basis SEC-DED protection. For brevity, a particular code may be referred to using only its

xAEC. The matrix is divided into two separate portions (top and bottom) each responsible for providing different behavior. The bottom portion consists of a series of horizontally concatenated $L \times L$ identity matrices, $\mathbf{I}_L$, truncated to fit the n-bit codeword size. This physically separates the code-bits used to calculate each of the syndrome-bits by a physical distance of L bitcells, and essentially mimics the bit-wise interleaving process within the **H**-matrix. The top portion contains $\lceil n/L \rceil$ sub-matrices, H_i. For each H_i sub-matrix the odd column vectors are identical to one another, and likewise for the even column vectors. For example, for a four column H_i sub-matrix (as in Figure 3.2), the first, and third columns are identical to one another, and the same is true for the second and fourth columns. The four column H_i sub-matrix is described by

$$H_i = (h_{\mathrm{o}_i},\ h_{\mathrm{e}_i},\ h_{\mathrm{o}_i},\ h_{\mathrm{e}_i}) \tag{3.1}$$

where h_{o_i} represents the odd column vectors for the i-th top portion sub-matrix and h_{e_i} represents the even column vectors. When vertically concatenated with its respective bottom portion $\mathbf{I}_L$ matrix, these H_i matrices each form a new $(r \times L)$ sub-matrix each comprised of L unique columns. This property eases the code construction procedure, and will be discussed in Sections 3.2.4.

3.1.1 Design Constraints

The proposed codes' **H**-matrix structure is defined by a set of constraints designed to elicit the desired error handling behaviour. The Constraints 1-4 are identical to those of the SEC-DED-DAEC Dutta codes presented in Section 2.8.7, while Constraints 5 and 6 provided the yAED and TAEC features respectively. Constraint 7 is a soft constraint designed to reduce the miscorrection probabilities for various error patterns. The constraints for the proposed codes' parity check matrices are as follows:

1. All columns must be non-zero.
2. All columns must be distinct.
3. The XOR result of any two columns must not equal any of the individual columns.
4. The XOR result of any two adjacent columns must be distinct, and different from any column in **H**.
5. The XOR result of any adjacent columns greater than two and less than or equal to y must produce a non-zero vector not equal to any the error correcting syndromes.

6. The XOR result of any *three* adjacent columns must be distinct and different from the single-bit and double-adjacent-bit error correcting syndromes. (*Required for triple adjacent error correction only*).

7. The amount of sharable syndromes between adjacent and non-adjacent errors should be minimized. (*Soft constraint*).

Constraint 1 ensures that all single-bit errors cause a non-zero syndrome and are thus detectable. Constraint 2 guarantees that all of the single-bit error syndromes are unique and thus correctable. Constraint 3 restricts the syndromes of all double-bit errors from matching those of the single-bit errors, meaning that no linear dependencies exist involving 3 or fewer columns. This allows for all double bit errors to be detectable. These three constraints provide the SEC-DED functionality. Constraint 4 ensures that the syndrome of all adjacent double bit errors are distinct, and combined with Constraints 2 and 3 ensure that these syndromes are different from the single-bit error correcting syndromes. This ensures that all single-bit and double-adjacent-bit error syndromes are distinct and thus correctable. This provides the SEC-DED-DAEC functionality.

Constraints 5 and 6 provide the functionality that separates this work from previous research. Constraint 5 allows for variable yAED functionality by ensuring that any error in y-adjacent bits produces a detectable (non-zero) syndrome that does not conflict with the error correcting syndrome patterns. The degree of error detectability is determined by the size of the identity matrix, $\mathbf{I}_L$, used during the construction of the $\mathbf{H}$-matrix. Constraint 6 provides the TAEC functionality. This constraint is not necessary for the DAEC codes, but is required for the TAEC codes. By expanding the set of distinct syndrome patterns to include those of all triple-adjacent-bit errors, these errors can be corrected in addition to the double-adjacent-bit and single-bit errors. Including this functionality attempts to increase the memory's reliability in the presence of adjacent-MBU soft errors.

Constraint 7 is added as a soft constraint in an effort to minimize the miscorrection probabilities of various error types. These miscorrections occur when non-zero syndrome patterns match those of the correctable syndrome patterns (i.e., single-bit, double-adjacent-bit, and triple-adjacent-bit error patterns), but the error is of a different error type. Traditionally, these errors are those beyond the error handling capabilities of the code; however, with the introduction of adjacent error correction, some adjacent and non-adjacent errors of the same bit-weight share syndrome patterns. Non-adjacent MBUs may therefore be misinterpreted as adjacent MBUs of the same weight, thus resulting in a miscorrection. This overlap in syndrome use comes as a result of the limitation of the code's minimum Hamming distance for its given number of check-bits. For the proposed codes, the miscorrection probabilities are less than those presented in [44] and can be further reduced by increasing the $\mathbf{I}_L$ matrix size. An effort is still made however to reduce the miscorrection probability through careful selection of the syndrome-bit calculation equations.

```
(24, 16) I-5 Code
0 4, 1 6, 4 2, 2 1, 5 0,

  C   C   C       C       C C                       C   C
  4   6   8       3       1 2                       5   7
  0 1 0 1 0 | 0 1 0 1 0 | 1 0 1 0 1 | 0 0 0 0 0 | 1 0 1 0 (9)
  0 0 0 0 0 | 0 1 0 1 0 | 0 1 0 1 0 | 1 0 1 0 1 | 0 0 0 0 (7)
  0 0 0 0 0 | 1 0 1 0 1 | 0 0 0 0 0 | 0 1 0 1 0 | 1 0 1 0 (7)
  ---------------------------------------------------------
  1 0 0 0 0 | 1 0 0 0 0 | 1 0 0 0 0 | 1 0 0 0 0 | 1 0 0 0 (5)
  0 1 0 0 0 | 0 1 0 0 0 | 0 1 0 0 0 | 0 1 0 0 0 | 0 1 0 0 (5)
  0 0 1 0 0 | 0 0 1 0 0 | 0 0 1 0 0 | 0 0 1 0 0 | 0 0 1 0 (5)
  0 0 0 1 0 | 0 0 0 1 0 | 0 0 0 1 0 | 0 0 0 1 0 | 0 0 0 1 (5)
  0 0 0 0 1 | 0 0 0 0 1 | 0 0 0 0 1 | 0 0 0 0 1 | 0 0 0 0 (4)
```

Figure 3.3: Example (24, 16) $\mathbf{I}_5$ SEC-DED-DAEC-7AED Parity Check Matrix

3.2 Code Design Procedure

The design of a particular code's **H**-matrix is essentially a search process to find a set of column vectors that satisfies all of the constraints mentioned in Section 3.1.1. The process of constructing **H**-matrices is NP-complete [44]. For a standard binary $r \times n$ matrix there are $2^{r \times n}$ potential matrices; each of which form a potential candidate **H**-matrix solution. Exhaustive search methods quickly become impractical as values of n and r become even moderately large. For example, a (22, 16) code has $2^{6 \times 22} = 2^{132}$ possible solutions to consider. Even for a Hsiao code that uses distinct, minimum, odd-weighted columns, thereby limiting the search space, (and assuming the check-bit columns use the 1-weight per column identity matrix) there are still $\binom{6}{3} = 20$ potential 3-weight column vectors for the 16 data-bit columns. This yields $\binom{20}{16} = 4845$ column vector combinations, and for each combination there are 16! column permutations. Therefore, the exhaustive search solution space still contains $4845 \times 16!$ candidate **H**-matrices ($\approx 2^{56.5}$), and this value only increases for larger memory word sizes. For this reason, an effort has been made to narrow the search space for the proposed SEC-DED-xAEC-yAED codes. As an example, a (24, 16) $\mathbf{I}_5$ SEC-DED-DAEC-7AED code is shown in Figure 3.3. This **H**-matrix provides double adjacent error correction and 7-bit adjacent error detection.

Each of the sub-matrices of the SEC-DED-DAEC-7AED **H**-matrix are clearly indicated. The bottom portion of the matrix consists of a repeating $\mathbf{I}_5$ matrix structure, and is used to invoke the 7AED behaviour. The top portion contains a set of $\lceil n/L \rceil - 1$ sub-matrices

of size $((r-L)\times L)$ and one $((r-L)\times(n \mod L))$ sub-matrix, where n is the codeword length, r is the number of check-bits, and L is the identity matrix size. For the (24, 16) $\mathbf{I}_5$ SEC-DED-DAEC-7AED **H**-matrix, $n = 24$, $k = 16$, $r = 8$, and $L = 5$ produces four (3×5) sub-matrices in the top portion each with a respective $\mathbf{I}_5$ matrix, and one (3×4) sub-matrix with an associated truncated identity matrix in the bottom portion. The column truncation is done to fit the 24-bit codeword length. The columns indicated with a 'C' are those columns indicating the check-bit positions for the code. The check-bit column selection procedure is described in Section 3.2.5. The check-bits are labeled 1 through 8. The bracketed number on the right-hand side of each matrix row is that row's row-weight. Finally, the set of decimal number pairs underneath the matrix title is a consolidated code for the particular **H**-matrix implementation. Each pair represents the decimal equivalent of the column vectors for each of the **H**-matrix's top portion H_i sub-matrices. Since the columns alternate between two values within each sub-matrix, only two elements are needed to represent each sub-matrix. For each column, the first row represents the most significant bit and the last row in the top portion represents the least significant bit. With this mapping, it can be seen that the first, third, and fifth column of the first top portion sub-matrix can be represented with a '0', while the second and fourth columns can be represented with a '4'. Continuing with this pattern provides the {(0,4), (1,6), (4,2), (2,1), (5,0)} consolidated code. This code, paired with an identity matrix size (L), codeword length (n), and list of check-bit columns is sufficient to completely describe a parity check matrix for the proposed code class.

3.2.1 Degree of Adjacent Error Detection

An identity matrix is used to provide the variable yAED behaviour for the syndrome. Larger AED coverage requires a larger identity matrix, and a parity bit is required for each row of the matrix. The parity check-bits for the identity matrix rows (rows 4-8) in the (24, 16) $\mathbf{I}_5$ example code are

$$\begin{aligned} c_4 &= b_1 + b_6 + b_{11} + b_{16} + b_{21} \\ c_5 &= b_2 + b_7 + b_{12} + b_{17} + b_{22} \\ c_6 &= b_3 + b_8 + b_{13} + b_{18} + b_{23} \\ c_7 &= b_4 + b_9 + b_{14} + b_{19} + b_{24} \\ c_8 &= b_5 + b_{10} + b_{15} + b_{20} \end{aligned}$$

where c_i represents the i-th check-bit and b_i represents the i-th codeword bit. The symbol b is used rather than d in this case to show that code-bits, including both check-bits and

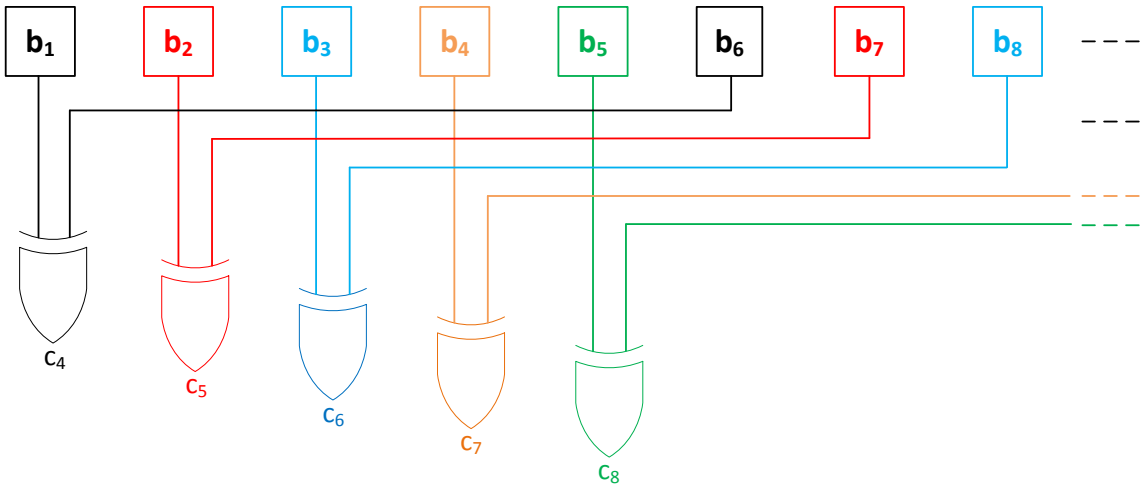


Figure 3.4: Identity matrix check-bit organization. Selecting check-bits in this manner allows for adjacent error detection.

data-bits, are used in the calculation rather than strictly just the data-bits, d. This can be generalized for an arbitrary codeword length of arbitrary identity matrix size as

$$c_{(r-L)+i} \quad = \quad b_i + b_{i+L} + \cdots + b_{i+L\cdot\lfloor\frac{n-i}{L}\rfloor} \;\; ; \;\; i = 1, 2, \ldots, L \tag{3.2}$$

where L is the identity matrix size. An example of the XOR gate implementation of this strategy is shown in Figure 3.4.

Using this implementation, the code-bits for each parity bit are separated by a physical distance of L-bits. Any upset affecting an odd number of bits in a syndrome-bit equation will generate a value of '1'. When a radiation strike upsets L-adjacent bits, only one bit in each of the L syndrome-bit equations is affected. This produces the all 1's syndrome for this set of bits, thus generating a non-zero, detectable syndrome. For all adjacent errors less than or equal to L, the error pattern will either be correctable, or produce a detectable non-zero syndrome. As the adjacent error size grows beyond L, the corruption of an even number of code-bits within a syndrome calculation will cancel one another out, thus producing syndrome-bit values of '0'. A $2L$ adjacent bit error will result in an all zero sub-syndrome for these check-bits, and thus error detection will rely solely on the syndrome bits of the top portion of the **H**-matrix.

For adjacent-bit errors, this scheme is able to detect up to

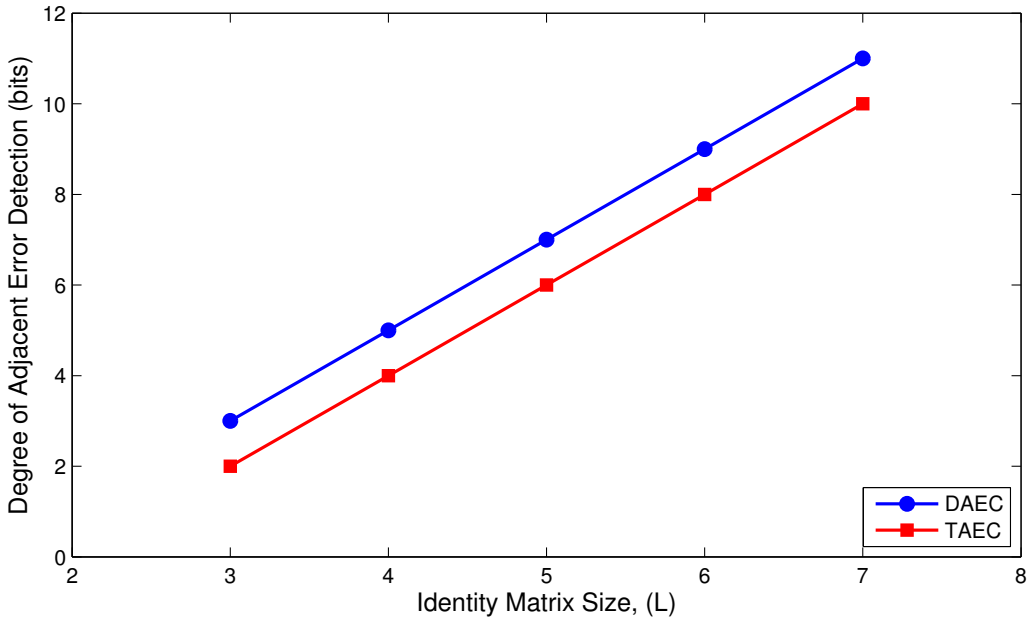


Figure 3.5: Degree of adjacent error detection versus identity matrix size, L

$$yAED = 2L - (xAEC + 1) \text{ bits} \tag{3.3}$$

where $xAEC$ is the degree of adjacent-correctable-bit errors. For larger adjacent-bit errors, the syndrome error patterns of the bottom portion of the **H**-matrix will match those of correctable error syndrome patterns, and may produce miscorrections depending on the error syndrome patterns produced by the top portion of the **H**-matrix. The degree of yAED is therefore dependent on the amount of xAEC provided and the size of the identity matrix, L. Figure 3.5 shows the degree of yAED for different sized identity matrices within the **H**-matrix for different amounts of xAEC.

For the DAEC codes, the $\mathbf{I}_3$ will yield up to three bits of AED. This value increases linearly with the size of the identity matrix, and implementations of the proposed code have been instantiated for up to $\mathbf{I}_7$. The DAEC $\mathbf{I}_7$ codes are capable of providing up to 11 bits of AED. As for the TAEC codes, these will always yield a degree of error detection one bit less than the DAEC codes for the same size identity matrix.

3.2.2 Degree of Burst Error Detection

In addition to providing adjacent error detection, AED, the proposed class of codes also provide a certain degree of burst error detection, BED. The degree of BED is always less than the degree of AED, and for the proposed codes is independent of the provided degree of AEC. For the proposed codes

$$zBED = L - 1 \tag{3.4}$$

where L is the size of the bottom portion identity matrix. BED is beneficial when not all of the cells affected by a soft error are corrupted. This may occur when the Q_{crit} of a storage cell is asymmetrical either due to data-value dependence or process variability. Since Hamming, Hsiao, and Dutta codes are limited to an error detection level of only two random bits, these codes are not capable of reliably detecting burst errors beyond the trivial case of two adjacent bits, while the proposed codes offer scalable protection proportional to L.

3.2.3 Required Number of Check-bits

The columns in the top portion of the **H**-matrix must be selected such that each complete column (concatenation of the top and bottom column portions) comply with the constraints detailed in Section 3.1.1. For this purpose, knowledge of the repeated identity matrix in the bottom portion of the **H**-matrix can be leveraged during the construction of the top portion.

First, the required number of check-bits, r, must satisfy the following relation

$$2^{r-L} \geq \left\lceil \frac{n}{L} \right\rceil \tag{3.5}$$

where r is the number of check-bits, n is the length of the codeword, and L is the size of the repeated identity matrix. This will ensure that there are a sufficient number of check-bits such that each complete column is unique.

Additionally, since the bottom portion of the **H**-matrix contains the repeated $\mathbf{I}_L$ matrix, the top portion of the matrix consists of $\lceil n/L \rceil$ sub-matrices each containing a maximum of L columns. Since the $\mathbf{I}_L$ is repeated, the i-th column of each top portion sub-matrix will have an identical column from the bottom portion $\mathbf{I}_L$ beneath it. To ensure the overall column uniqueness constraint, the top portion of the **H**-matrix can be divided into L bins (one for each column in the identity matrix) each containing at most $\lceil n/L \rceil$ unique columns. To express these $\lceil n/L \rceil$ unique binary columns, at least 2^{r-L} bits are required,

where $r - L$ is the number of rows, or check-bits, used in the top portion of the matrix. In the case of the (24, 16) $\mathbf{I}_5$ SEC-DED-DAEC-7AED code ($r = 8$, $L = 5$) there are $2^{r-L} = 2^{8-5} = 2^3 = 8$ different columns to choose from and $\lceil n/L \rceil = 5$ columns to be chosen per bin, or $\binom{8}{5} = 56$ different combinations.

3.2.4 Column Vector Selection Procedure for DAEC

In addition to the distinct column constraint (Constraint 2), the XOR result of all adjacent column pairs must be distinct from one another and from all individual columns (Constraints 3 and 4). The repeated $\mathbf{I}_L$ matrix will ensure the XOR uniqueness result of any two adjacent columns from any individual column. This is because the XOR result of any two adjacent bottom portion columns will have a weight of two, whereas the individual bottom portion column only have a weight of one. To ensure the XOR uniqueness result of all two adjacent column pairs requires a careful selection of the top portion columns.

For an n-bit codeword, rather than attempting to find an ordered set of n unique columns with $n-1$ unique adjacent column XOR pairs in one attempt, the problem can be subdivided using a two-phase solution approach. Within the first phase, columns satisfying the uniqueness constraints are selected for each individual H_i sub-matrix. These sub-matrices are then arranged in the second phase to satisfy the overall uniqueness constraints for the entire **H**-matrix. This procedure is described in Figure 3.6 using 3×4 H_i sub-matrices concatenated with $\mathbf{I}_4$ matrices as an example. For clarity, decimal equivalent values are used for the column values of each H_i matrix. For each of these column values, the top row represents the most significant bit, and the bottom row represents the least significant bit.

First, notice that the matrix shown in Figure 3.6(a) is a 3×4 H_i sub-matrix of the form described in Equation 3.1 concatenated with the $\mathbf{I}_4$ matrix, and this matrix conforms with all of the constraints listed in Section 3.1.1. For this matrix, only the two $r - L$ bit column vectors $h_{o_i} = 0$ and $h_{e_i} = 3$ need to be considered to meet the constraint requirements. The first phase of the solution method consists of finding a set of $\lceil n/L \rceil$ distinct $H_i = (h_{o_i}, h_{e_i})$ column vector pairs with a distinct set of $h_{o_i} \oplus h_{e_i}$ values. In the case of the $H_i = (0, 3)$ matrix, this gives a value of 3. In Figure 3.6(b) a second H_i sub-matrix (1, 2) is added. The (0, 3), (1, 2) matrices each have individually unique columns, but they produce the same $h_{o_i} \oplus h_{e_i}$ values. This causes the **H**-matrix to fail Constraint 4. By replacing the H_i pair (1, 2) with (4, 2), as is shown in Figure 3.6(c), these matrices produce distinct $h_{o_i} \oplus h_{e_i}$ values. Once the set of column vector pairs have been successfully selected, this marks the end the Phase 1 solution.

The second phase of the solution method consists of arranging the H_i matrices in a manner such that the XOR result of the two adjacent columns in each pair of adjacent H_i

$$\begin{bmatrix} H_i \\ \hline \mathbf{I}_4 \end{bmatrix} = \begin{bmatrix} 0\ 0\ 0\ 0 \\ 0\ 1\ 0\ 1 \\ 0\ 1\ 0\ 1 \\ \hline \\ \mathbf{I}_4 \\ \\ \end{bmatrix}$$

H_i	(0,3)
$h_{o_i} \oplus h_{e_i}$	3
$h_{e_i} \oplus h_{o_{i+1}}$	-

(a)

$$\begin{bmatrix} 0\ 0\ 0\ 0 \\ 0\ 1\ 0\ 1 \\ 0\ 1\ 0\ 1 \\ \hline \\ \mathbf{I}_4 \\ \\ \end{bmatrix}, \begin{bmatrix} 0\ 0\ 0\ 0 \\ 0\ 1\ 0\ 1 \\ 1\ 0\ 1\ 0 \\ \hline \\ \mathbf{I}_4 \\ \\ \end{bmatrix}$$

H_i	(0,3), (1,2)
$h_{o_i} \oplus h_{e_i}$	3, 3
$h_{e_i} \oplus h_{o_{i+1}}$	2

(b)

$$\begin{bmatrix} 0\ 0\ 0\ 0 \\ 0\ 1\ 0\ 1 \\ 0\ 1\ 0\ 1 \\ \hline \\ \mathbf{I}_4 \\ \\ \end{bmatrix}, \begin{bmatrix} 1\ 0\ 1\ 0 \\ 0\ 1\ 0\ 1 \\ 0\ 0\ 0\ 0 \\ \hline \\ \mathbf{I}_4 \\ \\ \end{bmatrix}$$

H_i	(0,3), (4,2)
$h_{o_i} \oplus h_{e_i}$	3, 6
$h_{e_i} \oplus h_{o_{i+1}}$	7

(c)

$$\begin{bmatrix} 0\ 0\ 0\ 0 \\ 0\ 1\ 0\ 1 \\ 0\ 1\ 0\ 1 \\ \hline \\ \mathbf{I}_4 \\ \\ \end{bmatrix}, \begin{bmatrix} 1\ 0\ 1\ 0 \\ 0\ 1\ 0\ 1 \\ 0\ 0\ 0\ 0 \\ \hline \\ \mathbf{I}_4 \\ \\ \end{bmatrix}, \begin{bmatrix} 1\ 0\ 1\ 0 \\ 0\ 0\ 0\ 0 \\ 1\ 1\ 1\ 1 \\ \hline \\ \mathbf{I}_4 \\ \\ \end{bmatrix}$$

H_i	(0,3), (4,2), (5,1)
$h_{o_i} \oplus h_{e_i}$	3, 6, 4
$h_{e_i} \oplus h_{o_{i+1}}$	7, 7

(d)

$$\begin{bmatrix} 1\ 0\ 1\ 0 \\ 0\ 0\ 0\ 0 \\ 1\ 1\ 1\ 1 \\ \hline \\ \mathbf{I}_4 \\ \\ \end{bmatrix}, \begin{bmatrix} 0\ 0\ 0\ 0 \\ 0\ 1\ 0\ 1 \\ 0\ 1\ 0\ 1 \\ \hline \\ \mathbf{I}_4 \\ \\ \end{bmatrix}, \begin{bmatrix} 1\ 0\ 1\ 0 \\ 0\ 1\ 0\ 1 \\ 0\ 0\ 0\ 0 \\ \hline \\ \mathbf{I}_4 \\ \\ \end{bmatrix}$$

H_i	(5,1), (0,3), (4,2)
$h_{o_i} \oplus h_{e_i}$	4, 3, 6
$h_{e_i} \oplus h_{o_{i+1}}$	1, 7

(e)

Figure 3.6: Column vector selection procedure.

sub-matrices are distinct. For the case of an even $\mathbf{I}_L$ matrix size, this requires that the set of $h_{e_i} \oplus h_{o_{i+1}}$ values form a distinct set. In Figure 3.6(d), the column vector pairs (0, 3), (4, 2), and (5, 1) are individually unique, and provide a valid Phase 1 solution; however, the adjacent columns for adjacent H_i sub-matrices fail to satisfy Constraint 4. This can be remedied by rearranging the H_i matrices as is shown in Figure 3.6(e), to produce a valid Phase 2 solution. A complete solution consists of an ordered set of $\lceil n/L \rceil$ H_i sub-matrices that satisfy all of the constraints listed in Section 3.1.1, and reduces the required number of columns to be considered down from n to a maximum of $2 \times \lceil n/L \rceil$. For the (24, 16) $\mathbf{I}_5$ code, this reduces the number of columns to be considered from 24 down to 10. This allows for the solution search space to be reduced from $2^{n \times r}$ candidate **H**-matrices down to

$$\begin{aligned} &\left(\binom{2^{r-L}}{\lceil n/L \rceil} \times \lceil n/L \rceil! \right)^2 \times \lceil n/L \rceil! \\ &= \left({}_{(2^{r-L})}P_{(\lceil n/L \rceil)} \right)^2 \times \lceil n/L \rceil! \end{aligned} \tag{3.6}$$

potential candidates. In the first phase of the solution, each column in the set of the column pairs can be arranged in $\binom{2^{r-L}}{\lceil n/L \rceil}$ combinations permuted $\lceil n/L \rceil!$ ways. Since there are two columns to be chosen per pair, this value is then squared. Once the Phase 1 column pairs have been chosen, there are $\lceil n/L \rceil!$ ways to arrange the pairs in Phase 2 of the solution process. For the (24, 16) $\mathbf{I}_5$ SEC-DED-DAEC-7AED **H**-matrix in Figure 3.3, this reduces the search space from $2^{24 \times 8} = 2^{192}$ candidate **H**-matrices down to $(\binom{8}{5} \times 5!)^2 \times 5! \approx 2^{32.3}$ candidate matrices. Additionally, by choosing only those vectors that will produce a minimum matrix weight, this preselects 4 of the 5 Phase 1 columns to be chosen (i.e., 000, 001, 010, and 100), and limits the 5th choice to only 3 values (011, 101, or 110) reducing the number of candidate columns to $\binom{3}{1}$, and the number of candidate solutions to $2^{23.9}$.

By examining the example (24, 16) $\mathbf{I}_5$ SEC-DED-DAEC-7AED **H**-matrix in Figure 3.3 it is clear that the odd columns (1, 3, and 5) of each of the top portion's sub-matrices are identical, and the even columns (2, and 4) of each sub-matrix within the top portion are identical as well. The matrix also satisfies all of the individual column and 2-adjacent XOR column uniqueness constraints. By considering each column of the parity check matrix in Figure 3.3 to be a binary value with its most significant bit in the first row, and least significant bit in the r-th row, these values are converted to their decimal equivalents and the uniqueness of this matrix is shown in Table 3.1.

The first two rows of the table contain data pertaining to the uniqueness of each complete column of the (24, 16) $\mathbf{I}_5$ code example in Figure 3.3. The first row shows the 24 distinct decimal equivalent values of each column in the matrix, while the second row shows the 23 distinct XOR results of each 2-adjacent column pair. Each of these data values are listed from left to right. Although these uniqueness results are necessary to

Table 3.1: (24, 16) $\mathbf{I}_5$ SEC-DED-DAEC-7AED Column Uniqueness Table

Unabridged Data	
Column Values, h_i (24)	{16, 136, 4, 130, 1, 48, 200, 36, 194, 33, 144, 72, 132, 66, 129, 80, 40, 68, 34, 65, 176, 8, 164, 2}
2-Adjacent Column XOR, $h_i \oplus h_{i+1}$ (23)	{152, 140, 134, 131, 49, 248, 236, 230, 227, 177, 216, 204, 198, 195, 209, 120, 108, 102, 99, 241, 184, 172, 166}
Consolidated Data	
Column Pairs, H_i (5)	{(0, 4), (1, 6), (4, 2), (2, 1), (5, 0)}
Phase 1 2-Adjacent Column Pair XOR, $h_{o_i} \oplus h_{e_i}$ (5)	{4, 7, 6, 3, 5}
Phase 2 2-Adjacent Column Block XOR, $h_{o_i} \oplus h_{o_{i+1}}$ (4)	{1, 5, 6, 7}

obtained the desired level of error handling, they can be expressed more succinctly in their consolidated forms as shown in the final three rows in Table 3.1. Each of these rows contain data pertaining to only the top portion of the **H**-matrix and leverage the knowledge of the repeated identity matrix in its bottom portion. The third row contains the set of column pairs used for each top portion sub-matrix in Figure 3.3, while the fourth and fifth rows contain the Phase 1 and Phase 2 column pair XOR results respectively. The Phase 1 2-adjacent column pairs XOR results are $0 \oplus 4 = 4$, $1 \oplus 6 = 7$, $4 \oplus 2 = 6$, etc... and the Phase 2 2-adjacent column block XOR results are $0 \oplus 1 = 1$, $1 \oplus 4 = 5$, $4 \oplus 2 = 6$, etc... Notice the uniqueness of every first element and every second element in the set of column pairs, as well as the uniqueness of the Phase 1 and Phase 2 XOR results. This uniqueness is sufficiently equivalent to the unabridged uniqueness result shown in the first two rows of the table, and is sufficient to provide the SEC-DED-DAEC-7AED coverage (Constraints 1-5) for the (24, 16) $\mathbf{I}_5$ code.

3.2.5 Check- and Syndrome-bit Generation XOR Logic Depth and Check-bit Selection

The use of the repeated identity matrix in the bottom portion of the proposed codes' **H**-matrices facilitates the AED capability of the code; however, it prevents the use of the $\mathbf{I}_r$ matrix (or the set of 1-weighted columns) for the check-bit columns. Since an all-zero column vector within the bottom portion of the **H**-matrix will disrupt the AED capabilities of the proposed codes, some check-bit columns will have a weight greater than one. This causes the generation of some of the check-bits to be dependent upon the generation of other check-bits. This has a negative impact on the maximum check-bit generation logic depth, adding an additional delay penalty to the encoder logic. Functionally, any set of

r columns may be chosen to represent the check-bit locations, but by selecting the check-bit columns such that no dependencies exist within the top portion rows, and limiting the number of dependencies in the bottom portion rows the impact on encoder delay can be minimized. The maximum encoder check-bit logic depth considering dependencies is calculated as

$$\text{Max Check-bit Logic Depth}_{\text{Dep.}}(\mathbf{H}) = \lceil \log_2(\text{MRW}(\mathbf{H}) - 1) \rceil + \left\lceil \log_2 \left(\left\lceil \frac{r-L}{L} \right\rceil + 1 \right) \right\rceil \tag{3.7}$$

where the first term represents the maximum check-bit delay without dependencies, and the second term represents the delay overhead introduced by these dependencies. For this calculation, MRW is the maximum row weight, and $r - L$ is the number of rows in the top portion of the **H**-matrix. Each of these rows create one dependency in the check-bit calculations for the bottom portion check-bits. These dependencies can be distributed amongst the L bottom portion check-bit calculations, and the set of dependencies within each can be calculated logarithmically. By ensuring no dependencies exist for the top portion rows, and the dependencies are distributed throughout the bottom portion rows, the additional encoder delay penalty can be limited to 1-2 logic levels for standard word lengths. The syndrome-bit generation logic in the decoder does not have this dependency issue. Hence, the maximum syndrome-bit generation logic depth is determined by Equation 2.10. This allows for negligible or even reduced delay for syndrome-bit generation in the proposed class of codes compared to the Hsiao or Dutta codes during a read operation.

3.2.6 Row Weight Balancing

Although the number of check-bits are fixed by the length of the codeword, optimizations on the encoder-decoder design can still be done by judicious selection of the top portion **H**-matrix columns. Optimizations can be performed by minimizing the total matrix weight and by balancing the matrix row weights. By minimizing the total matrix weight, the total number of XOR gates for implementing the check- and syndrome-bit generation logic circuits can be reduced, and by evenly distributing the matrix weight across the matrix rows, the maximum row weight can be minimized. As mentioned in Section 2.7.2, minimizing the maximum matrix row weight reduces the number of logic levels in the check and syndrome XOR logic trees, thereby reducing the encoder and decoder delays. The overall matrix weight can be minimized by selecting those columns with minimal weight. For example, in the case where there are three check-bits for the top portion of the parity check matrix, there are $2^3 = 8$ unique candidate columns to choose from. These columns are grouped by their column weight in Table 3.2.

Table 3.2: Set of 3-bit Columns with Various Column Weights

Column Weight	Elements in Set	Column Elements Base$_2$	Column Elements Base$_{10}$
0	1	$\{000\}_2$	$\{0\}_{10}$
1	3	$\{001, 010, 100\}_2$	$\{1, 2, 4\}_{10}$
2	3	$\{011, 101, 110\}_2$	$\{3, 5, 6\}_{10}$
3	1	$\{111\}_2$	$\{7\}_{10}$

There are four different column weights for the eight different columns. The zero column has a weight of 0, three columns exist with a weight of 1, three columns have a weight of 2, and one column has a weight of 3. By choosing those columns with minimum weight first, and the higher weighted columns only when necessary, the total matrix weight can be minimized. For example, for the (24, 16) $\mathbf{I}_5$ SEC-DED-DAEC-7AED code example where 5 unique columns must be selected from the 8 potential candidates, a minimum weight of 23 can be obtained for the top portion of the matrix by selecting all of the columns with weight 1 or less, and the one remaining column from the set of weight-2 columns. By careful selection of the weight-2 column for the even and odd sets of columns, the rows weights of the top portion of the matrix can be balanced.

Since the bottom portion of the **H**-matrix contains repeated $\mathbf{I}_L$ matrices, the weight of each column in the bottom portion of the **H**-matrix will always be 1. Further, the row weight for each bottom portion row is determined by the number of $\mathbf{I}_L$ sub-matrices. The concept of row weight balancing will be leveraged to minimized the syndrome-bit generation logic depth for the (24, 16) $\mathbf{I}_5$ code in Section 3.6.6.

3.2.7 Triple Adjacent-bit Error Correction

To construct an **H**-matrix capable of performing triple adjacent error correction the matrix must satisfy all of the constraints listed in Section 3.1.1. Constraint 6, the need to have the XOR result of any *three* adjacent columns unique from the XOR result of any other three adjacent columns and the other error correcting syndromes is what provides the TAEC capability. Due to the bottom portion $\mathbf{I}_L$ matrix, the triple adjacent-bit error syndromes are guaranteed to be distinct from the single and double adjacent-bit syndromes since the bottom portion of the **H**-matrix contributes a weight of 1, 2, or 3 to the syndrome for each single, double adjacent-bit, or triple adjacent-bit error respectively. TAEC **H**-matrices have been constructed by exploring the set of DAEC **H**-matrices for the TAEC property. An example (24, 16) $\mathbf{I}_5$ SEC-DED-TAEC-6AED code is shown in Figure 3.7, and its accompanying uniqueness table is shown in Table 3.3.

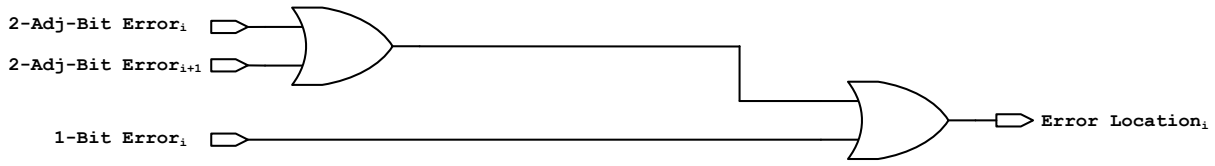
As mentioned in Section 3.2.1, the degree of AED depends on the degree of AEC. For the fixed amount of TAEC, the amount of AED is given by

```
(24, 16) I-5 Code
4 6, 0 4, 1 0, 2 1, 5 2

C             C   C   C       C C C                       C
1             4   6   8       5 3 7                       2
1 1 1 1 1 | 0 1 0 1 0 | 0 0 0 0 0 | 0 0 0 0 0 | 1 0 1 0 (9)
0 1 0 1 0 | 0 0 0 0 0 | 0 0 0 0 0 | 1 0 1 0 1 | 0 1 0 1 (7)
0 0 0 0 0 | 0 0 0 0 0 | 1 0 1 0 1 | 0 1 0 1 0 | 1 0 1 0 (7)
-----------------------------------------------------------
1 0 0 0 0 | 1 0 0 0 0 | 1 0 0 0 0 | 1 0 0 0 0 | 1 0 0 0 (5)
0 1 0 0 0 | 0 1 0 0 0 | 0 1 0 0 0 | 0 1 0 0 0 | 0 1 0 0 (5)
0 0 1 0 0 | 0 0 1 0 0 | 0 0 1 0 0 | 0 0 1 0 0 | 0 0 1 0 (5)
0 0 0 1 0 | 0 0 0 1 0 | 0 0 0 1 0 | 0 0 0 1 0 | 0 0 0 1 (5)
0 0 0 0 1 | 0 0 0 0 1 | 0 0 0 0 1 | 0 0 0 0 1 | 0 0 0 0 (4)
```

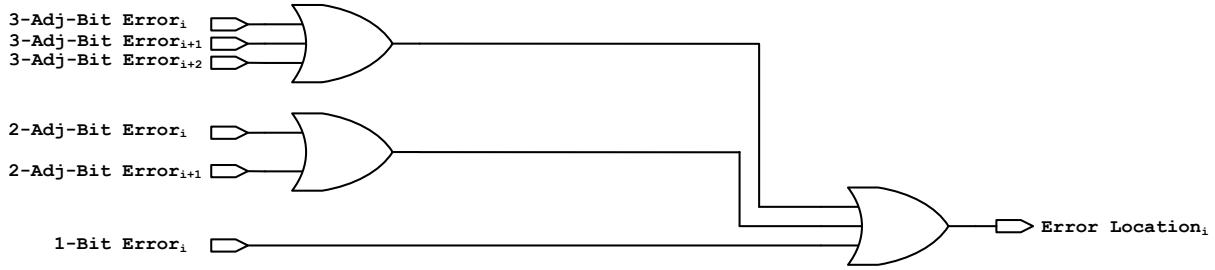
Figure 3.7: Example (24, 16) $\mathbf{I}_5$ SEC-DED-TAEC-6AED Parity Check Matrix

Table 3.3: (24, 16) $\mathbf{I}_5$ SEC-DED-TAEC-6AED Column Uniqueness Table

Consolidated Data	
Column Pairs, H_i (5)	{(4, 6), (0, 4), (1, 0), (2, 1), (5, 2)}
Phase 1 2-Adjacent Column Pair XOR, $h_{o_i} \oplus h_{e_i}$(5)	{2, 4, 1, 3, 7}
Phase 2 2-Adjacent Column Block XOR, $h_{o_i} \oplus h_{o_{i+1}}$ (4)	{6, 5, 2, 4}
3-Adjacent Column XOR, $h_i \oplus h_{i+1} \oplus h_{i+2}$ (22)	{220, 142, 199, 83, 25, 156, 14, 135, 179, 57, 28, 46, 7, 115, 89, 60, 78, 39, 211, 185, 92, 174}



(a) DAEC



(b) TAEC

Figure 3.8: For the DAEC and TAEC codes, each codeword bit can be involved in 3 and 6 different correctable error patterns respectively. By OR'ing the decoded syndrome value for each of these error patterns together, it can be determined if a particular bit was involved in an error. By implementing this circuitry for each codeword bit, and error location vector can be generated.

$$2L - (TAEC + 1) = 2L - 4 \; ; \; For \; L > 3 \tag{3.8}$$

This is shown in Figure 3.5. For TAEC codes, adjacent errors wider than $2L - 4$ may lead to miscorrections.

3.2.8 Encoder-Decoder Circuit

The encoding and decoding processes for the proposed codes use standard XOR logic for the check- and syndrome-bit generation in a manner identical to the Hamming and Hsiao SEC-DED schemes. The syndrome decoder logic however, has been modified to include correctable error matching signals for each of the adjacent bit upset syndrome patterns. For the SEC-DED-DAEC-yAED codes, each code-bit can be involved in three correctable error types: one single bit upset and two 2-adjacent bit upsets. By OR'ing the error matching signals together for each code-bit error location the syndrome decoder can indicate which particular bits have been involved in an upset. This modification is shown for a bitslice for the DAEC codes in Figure 3.8(a). For the SEC-DED-TAEC-yAED codes, the syndrome

$$
\begin{array}{c}
 & \underline{\mathbf{C_1}} & D_1 & D_2 & D_3 & D_4 & C_4 & D_5 & C_6 & D_6 & C_8 & D_7 & C_5 & \underline{\mathbf{C_3}} & C_7 & D_8 & D_9 & D_{10} & D_{11} & D_{12} & D_{13} & D_{14} & \underline{\mathbf{C_2}} & D_{15} & D_{16} \\
C_1 & \underline{\mathbf{0}} & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
C_2 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & \underline{\mathbf{0}} & 0 & 1 \\
C_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \underline{\mathbf{0}} & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0
\end{array}
$$

(a) Top Portion Generator Matrix, $\mathbf{G}_{Top}$

$$
\begin{array}{c}
 & C_1 & D_1 & D_2 & D_3 & D_4 & \underline{\mathbf{C_4}} & D_5 & \underline{\mathbf{C_6}} & D_6 & \underline{\mathbf{C_8}} & D_7 & \underline{\mathbf{C_5}} & C_3 & \underline{\mathbf{C_7}} & D_8 & D_9 & D_{10} & D_{11} & D_{12} & D_{13} & D_{14} & C_2 & D_{15} & D_{16} \\
C_4 & 1 & 0 & 0 & 0 & 0 & \underline{\mathbf{0}} & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
C_5 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \underline{\mathbf{0}} & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
C_6 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \underline{\mathbf{0}} & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
C_7 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \underline{\mathbf{0}} & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
C_8 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \underline{\mathbf{0}} & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0
\end{array}
$$

(b) Bottom Portion Generator Matrix, $\mathbf{G}_{Bottom}$

Figure 3.9: (24, 16) $\mathbf{I}_5$ SEC-DED-TAEC-6AED Encoder Generator Matrices

decoder circuit is modified as shown in Figure 3.8(b), since these codes are able to correct triple adjacent-bit errors, there are $n-2$ additional error correcting syndrome patterns to match. The i-th bit of the n-bit codeword will be corrected under one of any six conditions. These are if: 1. a single-bit error occurred at the i-th bit, 2. a double adjacent-bit error occurred at either the $(i-1, i)$, or 3. $(i, i+1)$ bits, or 4. a triple adjacent-bit error occurred at the $(i-2, i-1, i)$, 5. $(i-1, i, i+1)$, or 6. $(i, i+1, i+2)$ bits. By OR'ing the error matching signals corresponding to each of these syndrome patterns, the upset of a particular code-bit can be determined by the syndrome decoder. The area and performance implication of the additional decoder complexity is considered in Section 3.6.2 and Section 3.6.3 respectively.

3.3 Encoding Process

The encoding process can be represented using two generator matrices, $\mathbf{G}_{Top}$ and $\mathbf{G}_{Bottom}$, derived from the parity check matrix $\mathbf{H}$. The matrix $\mathbf{G}_{Top}$ is responsible for generating the check-bits that do not contain any check-bit data dependencies, and is derived from the top portion rows of the $\mathbf{H}$-matrix. The matrix $\mathbf{G}_{Bottom}$ is derived from the bottom portion rows of the $\mathbf{H}$-matrix and generates check-bits that may contain check-bit dependencies. The generator matrices $\mathbf{G}_{Top}$ and $\mathbf{G}_{Bottom}$ are respectively equal to the top and bottom portion rows of $\mathbf{H}$ with the exception that the i-th check-bit is removed from each of the i-th rows. This follows from the logic discussed in Section 2.6.3. The encoding generator matrices $\mathbf{G}_{Top}$ and $\mathbf{G}_{Bottom}$ for the (24, 16) $\mathbf{I}_5$ SEC-DED-TAEC-6AED code are shown in Figure 3.9.

The desired encoded codeword, $\mathbf{v}$, consists of the original dataword $\mathbf{d}$ and the calculated check-bits $\mathbf{c}$ distributed throughout the codeword. The arrangement of the (24, 16) $\mathbf{I}_5$ SEC-DED-TAEC-6AED codewords is as follows

$$\mathbf{v} \;=\; (\underline{\mathbf{c}_1} d_1 d_2 d_3 d_4 \; \underline{\mathbf{c}_4} d_5 \underline{\mathbf{c}_6} d_6 \underline{\mathbf{c}_8} \; d_7 \underline{\mathbf{c}_5 \mathbf{c}_3 \mathbf{c}_7} d_8 \; d_9 d_{10} d_{11} d_{12} d_{13} \; d_{14} \underline{\mathbf{c}_2} d_{15} d_{16})$$

The placement of the check-bits has been discussed in Section 3.2.5. For an input data vector, $\mathbf{d}$, the least significant check-bits, $\mathbf{c}_{LSBs}$, are calculated by setting all of the check-bit values in $\mathbf{v}$ to zero, and then multiplying $\mathbf{v}$ by the transpose of the $\mathbf{G}_{Top}$ generator matrix as follows

$$\mathbf{c}_{LSBs} \;=\; \mathbf{v} \cdot \mathbf{G}_{Top}^{T}.$$

The calculated $\mathbf{c}_{LSBs}$ check-bits can then be assigned to their respective positions in $\mathbf{v}$. The most significant check-bits, $\mathbf{c}_{MSBs}$, are then calculated using the generator matrix $\mathbf{G}_{Bottom}$ as follows

$$\mathbf{c}_{MSBs} \;=\; \mathbf{v} \cdot \mathbf{G}_{Bottom}^{T}.$$

The calculated $\mathbf{c}_{MSBs}$ check-bits are then assigned to their respective positions in $\mathbf{v}$, and the codeword can be stored in memory. Although this is mathematically represented using a multi-step process, it can easily be implemented in a combinational XOR logic circuit.

3.4 Encoding, Error Injection, Decoding Example

This section provides a full encoding and decoding example for the (24, 16) $\mathbf{I}_5$ SEC-DED-TAEC-6AED code in Figure 3.7 in the event of a 3-adjacent bit error. For this example we assume that the dataword $\mathbf{d}$ = (1010 1010 1010 1010) is to be transmitted, and the 3-adjacent error $\mathbf{e}$ = (00011 10000 00000 00000 0000) is injected into the codeword $\mathbf{v}$ during transmission.

3.4.1 Encoding

The $\mathbf{G}_{Top}$ matrix in Section 3.3 provides the following $\mathbf{c}_{LSBs}$ check-bit calculations

$$\begin{aligned} c_1 &= d_1 \oplus d_2 \oplus d_3 \oplus d_4 \oplus d_5 \oplus d_6 \oplus d_{14} \oplus d_{15} \\ c_2 &= d_1 \oplus d_3 \oplus d_9 \oplus d_{11} \oplus d_{13} \oplus d_{16} \\ c_3 &= d_7 \oplus d_8 \oplus d_{10} \oplus d_{12} \oplus d_{14} \oplus d_{15} \end{aligned}$$

while the $\mathbf{G}_{Bottom}$ matrix provides the following $\mathbf{c}_{MSBs}$ calculations

$$\begin{aligned} c_4 &= \underline{\mathbf{c}_1} \oplus d_7 \oplus d_9 \oplus d_{14} \\ c_5 &= d_1 \oplus d_5 \oplus d_{10} \oplus \underline{\mathbf{c}_2} \\ c_6 &= d_2 \oplus \underline{\mathbf{c}_3} \oplus d_{11} \oplus d_{15} \\ c_7 &= d_3 \oplus d_6 \oplus d_{12} \oplus d_{16} \\ c_8 &= d_4 \oplus d_8 \oplus d_{13}. \end{aligned}$$

The check-bit calculations are identical to the syndrome-bit calculations minus the check-bit being calculated. Notice the dependencies in the c_4, c_5, and c_6 equations on the check-bits c_1, c_2, and c_3 respectively. These bits must be calculated in the $\mathbf{c}_{LSBs}$ check-bit calculations before being used in the dependent $\mathbf{c}_{MSBs}$ equations. Encoding the dataword $\mathbf{d}$ = (1010 1010 1010 1010) produces the check-bits $\mathbf{c}$ =($\mathbf{c}_{LSBs}$ $\mathbf{c}_{MSBs}$) = (010 01011), and inserting the check- and data-bits into their appropriate locations for the codeword $\mathbf{v}$ provides

$$\begin{aligned} \mathbf{v} &= (\underline{\mathbf{c}_1} d_1 d_2 d_3 d_4 \ \underline{\mathbf{c}_4} d_5 \underline{\mathbf{c}_6} d_6 \underline{\mathbf{c}_8} \ d_7 \underline{\mathbf{c}_5 \mathbf{c}_3 \mathbf{c}_7} d_8 \ d_9 d_{10} d_{11} d_{12} d_{13} \ d_{14} \underline{\mathbf{c}_2} d_{15} d_{16}) \\ &= (\underline{0}1010 \ \underline{0}1\underline{0}0\underline{1} \ 1\underline{1}0\underline{1}0 \ 10101 \ 0\underline{1}10). \end{aligned}$$

This ends the encoding procedure, and the codeword $\mathbf{v}$ is written into memory.

3.4.2 Error Injection

When the SEU occurs, the 3-adjacent error is injected into the transmitted codeword $\mathbf{v}$. This can be represented by XORing the error vector $\mathbf{e}$ with the transmitted codeword $\mathbf{v}$.

This produces the received codeword **r**, and mathematically describes the data corruption process. At this point, the memory system has no knowledge of the data corruption. The received codeword is given by

$$\begin{aligned} \mathbf{r} &= \mathbf{v} \oplus \mathbf{e} \\ & \\ & \quad (01010\ 01001\ 11010\ 10101\ 0110) \\ & \oplus \ (00011\ 10000\ 00000\ 00000\ 0000) \\ & \quad \overline{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad} \\ & \quad (010\underline{01}\ \underline{1}1001\ 11010\ 10101\ 0110). \end{aligned}$$

Notice the data corruption in bits d_3, d_4, and c_4.

3.4.3 Decoding

During a read operation, the decoding process takes place. The decoder determines, if possible, if an error has occurred and the location of the error. It then performs the error correction. This process begins with the syndrome generation

$$\begin{aligned} \mathbf{S} &= \mathbf{r} \cdot \mathbf{H}^T \\ &= (0101\ 0011). \end{aligned}$$

The syndrome value matches the XOR combination of the fourth, fifth, and sixth columns in the **H**-matrix in Figure 3.7 (d_3, d_4, and c_4), and is the decimal equivalent, $(83)_{10}$, of the fourth entry in the 3-Adjacent Column XOR Results row in Table 3.3. The syndrome decoder translates the 8-bit syndrome pattern into the 24-bit error location vector, $\mathbf{E}_{Loc}$

$$\mathbf{E}_{Loc} = (00011\ 10000\ 00000\ 00000\ 0000).$$

The syndrome decoder has functioned correctly if the error location vector, $\mathbf{E}_{Loc}$, is equal to the injected error vector **e**. XORing the error location vector $\mathbf{E}_{Loc}$ bitwise with the received codeword **r** provides the corrected codeword, **u**

$$\begin{aligned} \mathbf{u} &= \mathbf{r} \oplus \mathbf{E}_{Loc} \\ &= (01010\ 01001\ 11010\ 10101\ 0110). \end{aligned}$$

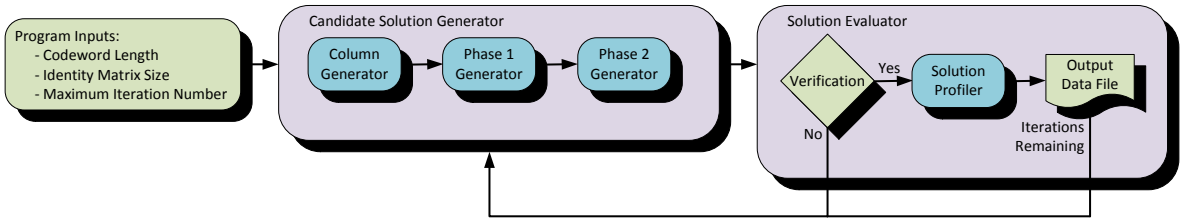


Figure 3.10: Matlab Solution Solver Flow Graph

Since the 3-adjacent error type is within the error handling capability of the (24, 16) $\mathbf{I}_5$ SEC-DED-TAEC-6AED code, the corrected codeword, $\mathbf{u}$, is identical to the originally transmitted codeword $\mathbf{v}$.

3.5 H-Matrix Generation and Verification

The two-phase solution method has been implemented in Matlab for constructing valid **H**-matrices. The solver, illustrated in Figure 3.10, is divided into two components. The first is a pseudo-greedy, **H**-matrix candidate solution generator, while the second component evaluates the candidate **H**-matrix solution for compliance with the desired **H**-matrix rules, discussed in Section 3.1.1. Provided that the candidate solution is a valid one, the matrix is then profiled in terms of the metrics discussed throughout Sections 2.7 and 3.2.5, and the data is output to a text file. In the event that the candidate solution is invalid, it is returned to the candidate solution generator for modification and resubmission. Modifications are performed by randomly swapping columns that violate the matrix construction constraints, and by reselecting candidate columns. This process is continued until the desired number of solution iterations have been performed. The solutions solver was run on each of the proposed matrix codes producing anywhere from over 100 to multiple thousands of solutions. For each of the proposed codes, the solution with the lowest total weight and miscorrection probabilities produced by the solution solver are presented in Section 3.6.

3.5.1 Construction Algorithm

The solver's code construction algorithm, as detailed in Figure 3.11, takes the codeword length, $\mathbf{I}_L$ size, and maximum iteration counters for each phase of the solution as inputs, and returns a set of valid **H**-matrices complete with a full performance evaluation as output (provided that a set of solutions is found). The solver initially selects a set of candidate H_i column vector pairs, $H(o, e)$, where o and e are top portion column vectors, either randomly,

or by minimum weight. This is considered to be Phase 0. Each of these pairs represent the odd and even column vectors in each of the H_i sub-matrices. To ensure overall **H**-matrix column uniqueness, all of the o elements in each pair are distinct, and likewise for the e elements. The XOR result of each H_i column vector pair is then calculated as $c = o \oplus e$. For the given set of column vectors, Phase 1 attempts to find sets of distinct c elements. Distinct sets are stored and passed to Phase 2. For sets with duplicate c elements, the e elements are randomly swapped, and the c elements are recalculated. This process is iterated p1Max number of times for p0Max sets of column vector pairs. Provided a valid set of Phase 1 solutions, Phase 2 attempts to order the list of H_i column vector pairs in such a manner that the XOR result, d, of the adjacent columns of any two adjacent H_i column vector pairs are distinct. For an odd $\mathbf{I}_L$ size this requires the set of $H(i).o \oplus H(i+1).o$ be unique, whereas for an even $\mathbf{I}_L$ size, the results of $H(i).e \oplus H(i+1).o$ must be unique. Each valid Phase 1 solution undergoes p2Max iteration attempts. All valid Phase 2 solutions make up a set of valid **H**-matrices for the given set of input parameters.

3.6 Evaluation

H-matrices for the proposed scheme have been implemented and verified for functionality using a custom test suite designed in Matlab for typical memory word sizes of 16, 32, and 64 data-bits. Relevant high-level performance metrics have been extracted directly from the codes' **H**-matrices and compared with Hsiao SEC-DED [31], Dutta SEC-DED-DAEC [44], BCH DEC [19], and Reed Solomon SbEC-DbED [11] codes. Both the BCH and RS codes have been implemented using parallelize, single-cycle, syndrome-based decoders. Further, synthesis results have been generated in Synopsys Design Compiler using Verilog Hardware Description Language (HDL) code implementations and a commercial 65 nm general purpose bulk CMOS technology standard cell library. The codes are compared in terms of their provided error correction and detection capabilities, required number of check-bits, encoder/decoder area, performance, and power consumption, as well as their miscorrection probabilities for non-adjacent errors. The values reported for each of the proposed codes are based on the **H**-matrix implementation found to have the minimum miscorrection probability for the minimum decoder logic gate count using the solution solver discussed in Section 3.5. The parity check matrices for all of the discussed codes are included in Appendix A. A summary of all high-level performance metrics and synthesis results are provided in Table 3.4 and Table 3.5 respectively in Section 3.6.5. For brevity, the plots presented throughout this section focus on the 16 data-bit implementations.

Input: n, L, p0Max, p1Max, p2Max
Output: setOf(p2Solutions)

Object: $H(i)$ Column vector pair, $H(o, e)$
  o = Column vector 1; e = Column vector 2
  $c = o \oplus e$
End Object

Object: List of $H(i)$ column vector pairs, H
  $length$ = ceiling(n/L)
  if L is Odd **then**
    $d = H(i).o \oplus H(i+1).o$ ▷ Odd $\mathbf{I}_L$ Size
  else
    $d = H(i).e \oplus H(i+1).o$ ▷ Even $\mathbf{I}_L$ Size
  end if
End Object

BEGIN CONSTRUCTION ALGORITHM
  for $p0 = 1$ **to** p0Max **do** ▷ Phase 0
    Create list of $H.length$ elements $H(o, e)$ via:
    1. Random Fill, 2. Minimum Weight,
    3. Modification of an existing list
    Calculate setOf($H(i).c$)

    for $p1 = 1$ **to** p1Max **do** ▷ Phase 1
      if setOf($H(i).c$).isUnique AND
        !setOf(p1Solutions).contains(H) **then**
        Add H to setOf(p1Solutions)
        Randomly swap a # of $H(i).e$ elements
      else
        for each duplicate in setOf($H(i).c$) **do**
          Randomly swap $dup - 1$ $H(i).e$ elements
        end for
      end if
      Recalculate setOf($H(i).c$)
    end for ▷ End Phase 1
  end for ▷ End Phase 0

  if !setOf(p1Solutions).isEmpty **then** ▷ Phase 2
    for each entry in setOf(p1Solutions) **do**
      Calculate setof($H.d$)

      for $p2 = 1$ **to** p2Max **do**
        if setOf($H.d$).isUnique AND
          !setOf(p2Solutions).contains(H) **then**
          Add H to setOf(p2Solutions)
          Randomly swap a # of $H(i)$'s
        else
          for each duplicate in setOf($H.d$) **do**
            Randomly swap $dup - 1$ $H(i)$'s
          end for
        end if
        Recalculate setOf($H.d$)
      end for
    end for
  end if ▷ End Phase 2
END CONSTRUCTION ALGORITHM

Figure 3.11: **H**-matrix construction algorithm

3.6.1 Error Correction and Detection Capabilities

In terms of error correction, the Hsiao SEC-DED code can correct all single-bit errors. The Dutta SEC-DED-DAEC codes extend this to provide double adjacent-bit error correction, while the proposed codes can provide either double or triple adjacent-bit error correction depending on implementation. The BCH DEC code provides double-bit error correction for all adjacent and non-adjacent double bit errors alike, while the degree of adjacent correction provided by the RS code depends on the location of the upset and can vary between 1 and b where b is the code's byte size. A double adjacent error that crosses over a byte boundary within a Reed Solomon code cannot be corrected; however, any error pattern contained entirely within a b-bit byte boundary can be corrected. The proposed code and RS code are the only ones capable of detecting errors of size greater than two bits. For both codes, this level of protection is a function of their embedded identity matrix or byte size, $\mathbf{I}_L$ or b, and is shown in Figure 3.12 in comparison with other codes. For the Reed Solomon code, an error of size $b + 2$ adjacent-bits may go undetected if it resides within three separate bytes; however, errors up to size $2b$ adjacent-bits can be detected provided they reside entirely within two adjacent byte locations.

While the SEC-DED-DAEC Dutta code (6 check-bits for 16 data-bits) and BCH DEC code (10 check-bits for 16 data-bits) each provide a maximum degree of 2-bit adjacent and 2-bit burst error detection, the proposed codes offer 5-11 bits of AED and 3-6 bits of BED for the DAEC implementations and between 4-10 bits AED for the TAEC implementations as the identity matrix is scaled from $\mathbf{I}_4$ up to $\mathbf{I}_7$. As the number of check-bits (and identity matrix size) increases, the degree of AED increases by 2-bits/check-bit, while the degree of BED increases by 1-bit/check-bit. For the cost of one additional check-bit over the (22,16) Dutta (or Hsiao) code, the proposed (23, 16) $\mathbf{I}_4$ DAEC code offers 5-bits AED and 3-bits BED, while for the same number of check-bits as the (26, 16) DEC BCH code the proposed (26, 16) $\mathbf{I}_7$ DAEC code offers 11-bits AED and 6-bits BED.

3.6.2 Implementation Area

The silicon area requirement for syndrome decoder based error correction circuits is dominated by the syndrome decoder circuit logic. This is made evident in Figure 3.13 which shows the 2-input XOR logic gate count for the syndrome generation logic and the synthesized area estimate for the complete ECC circuit (check/syndrome generator, syndrome decoder, and error corrector) for each of the proposed 16 data-bit DAEC codes. As the number of check-bits (and $\mathbf{I}_L$ matrix size) increases, fewer XOR gates are required for the check and syndrome bit generation logic. This reduces the size of the check/syndrome generator, but the additional check-bit adds an extra input signal to the syndrome decoder circuit, thereby increasing its complexity. This modification increases the syndrome de-

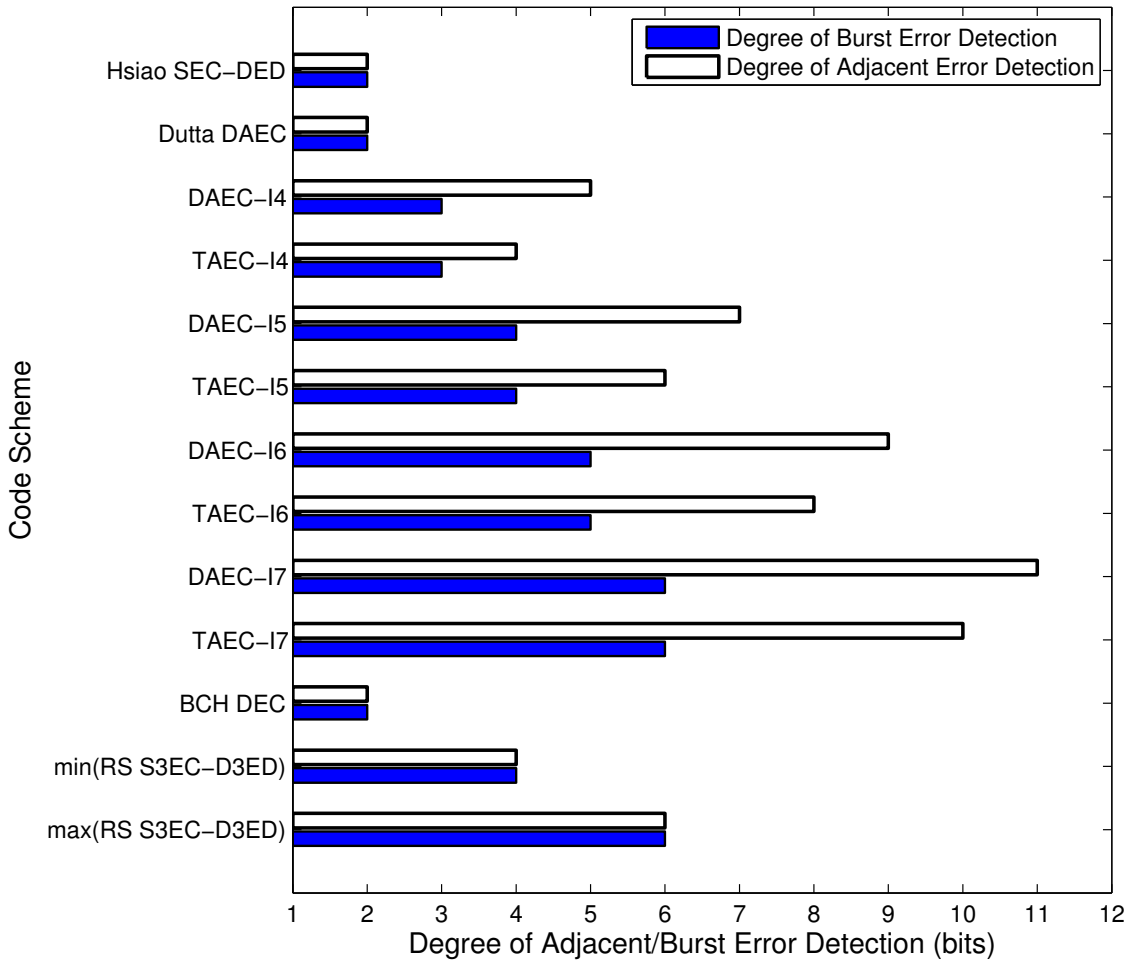


Figure 3.12: Adjacent and burst error detection for the proposed codes as a function of $\mathbf{I}_L$ matrix size compared with the SEC-DED-DAEC Dutta, BCH DEC, and RS S3EC-D3EC codes.

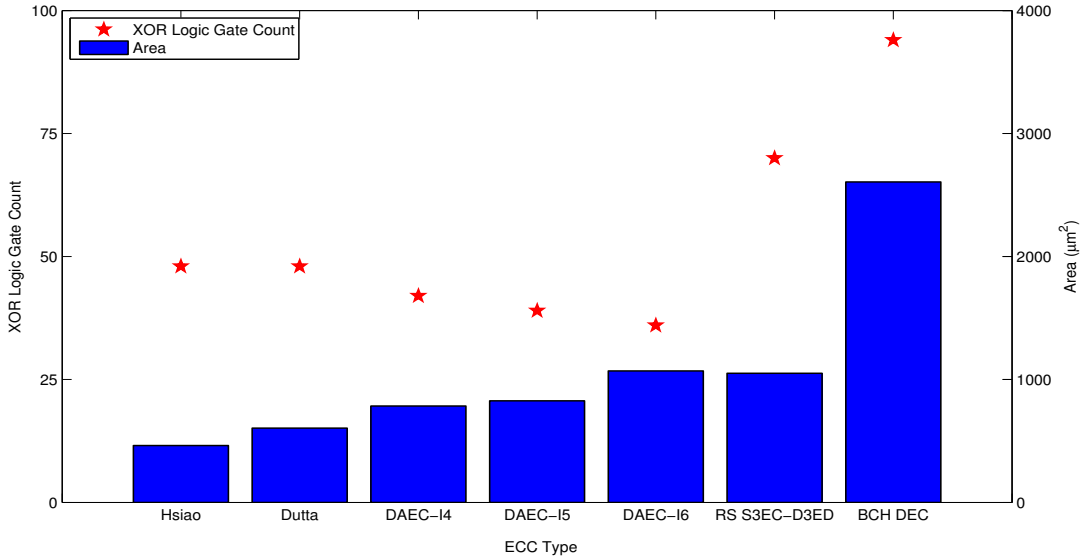


Figure 3.13: Check- and syndrome-bit generation XOR logic gate count and synthesized area for the proposed 16 data-bit SEC-DED-DAEC-yAED code implementations, (22, 16) SEC-DED-DAEC Dutta code, (25, 16) RS code, and (26, 16) DEC BCH code.

coder logic and in turn the total ECC circuit area as the number of check/syndrome bits is increased. This stresses the importance of implementing the ECC circuits to determine their required area as opposed to relying solely on a generator logic gate count estimate.

The proposed (23, 16) $\mathbf{I}_4$ code incurs a 69.5% increase in circuit area over the traditional SEC-DED implementation to achieve the double adjacent error correcting and triple adjacent error detecting functionality. This is significantly less than the 5.63x increase in area required for the (26, 16) BCH DEC code implementation. Further, the incremental area decreases as the memory word size increases; for the 64 data-bit codes, the proposed (73, 64) $\mathbf{I}_4$ code incurs only a 14.1% area overhead while the (78, 64) BCH DEC and (79, 64) RS codes incur 19.12x and 3.13x area penalties respectively over the traditional (72, 64) Hsiao SEC-DED scheme. This difference occurs because the addition of double adjacent error correction requires only $n-1$ additional correctable syndrome values, where n is the codeword length. To achieve the double random error correction offered by the DEC code requires an additional $\binom{n}{2}$ syndrome values, while the RS code requires $2^b - 1$ additional syndrome vectors for every b additional data-bits.

Adding the triple adjacent error correcting feature to the proposed set of codes inherently increases the number of error correcting syndromes. This increase is reflected in the circuit area as each syndrome pattern requires a syndrome-to-error-location circuit in the

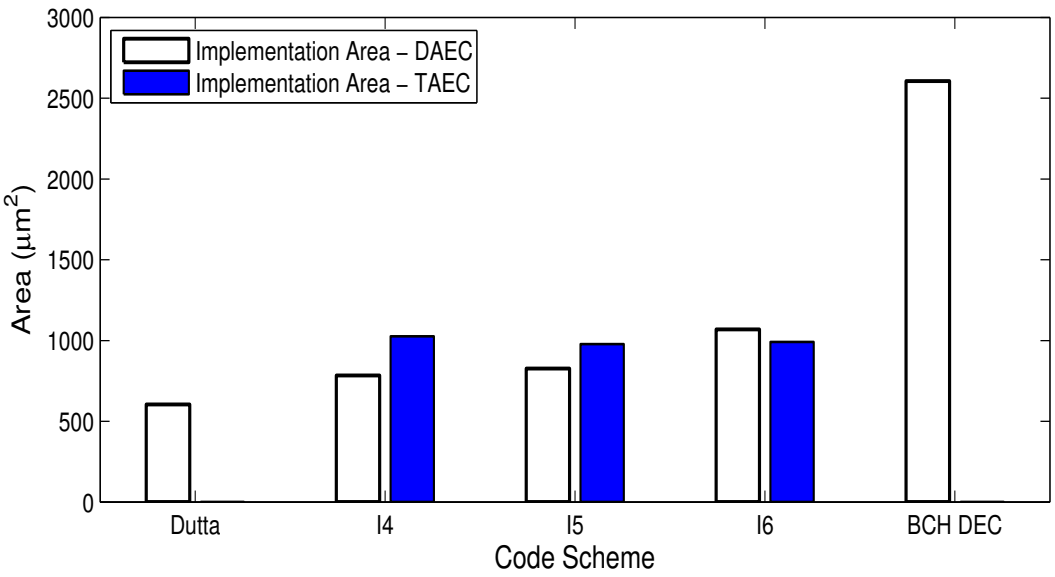


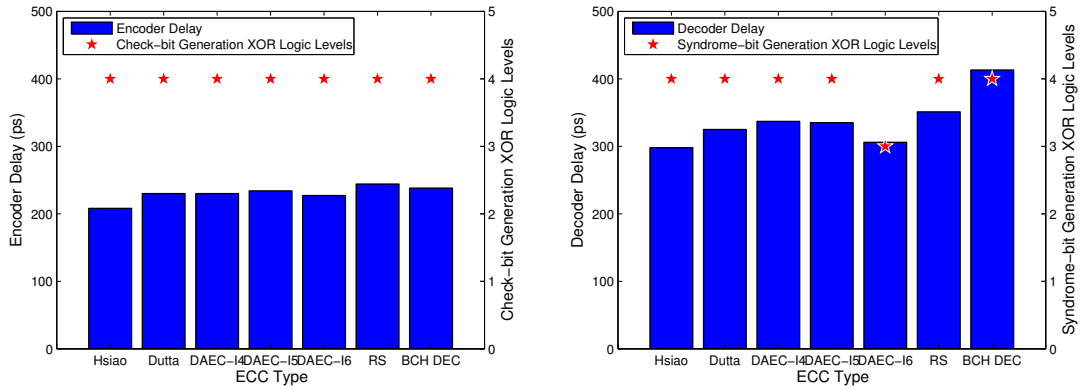
Figure 3.14: Synthesized area comparison between the proposed 16 data-bit DAEC and TAEC code implementations for the equivalent number of check bits.

syndrome decoder. Adding the TAEC feature to the (23, 16) $\mathbf{I}_4$ code requires an additional $n-1=21$ syndrome patterns, at a cost of 31% additional circuit area, as shown in Figure 3.14. The additional area impact incurred by the (25, 16) $\mathbf{I}_6$ DAEC code is a result of additional delay optimizations.

Regardless of the chosen ECC scheme, its circuit area can be amortized over the memory area. The ECC circuit area is governed by its error handling functionality and codeword size, and is independent of the capacity of the memory that it is protecting. Doubling the memory capacity will have a negligible impact on the ECC circuit area. Therefore, the additional circuit area required for the adjacent error correction functionality can have a minimal impact at the overall system level.

3.6.3 Encoder and Decoder Propagation Delay

For single-cycle memory, the ECC encoder and decoder propagation delays reside on the write and read critical paths respectively. The encoder delay includes the time required to calculate the check-bits for a write operation, while the decoder delay is the time required to generate the syndrome, decode it, and then perform any necessary corrections during a read operation. Minimizing the encoder and decoder delays is critical to maintaining high speed operation. Provided that SRAM performance is typically limited by its read delay,



(a) Encoder (b) Decoder

Figure 3.15: Synthesized propagation delay estimation for the proposed 16 data-bit SEC-DED-DAEC-yAED code implementations, (22, 16) SEC-DED-DAEC Dutta code, (25, 16) RS code, and (26, 16) DEC BCH code.

and the decoding process is a more complex operation compared to the encoding process, the synthesized ECC circuits have been optimized for decoder delay.

The encoder and decoder delays for the 16 data-bit codes is shown in Figure 3.15 and overlaid with the XOR logic depth of the check (encode) and syndrome (decode) generation logic. During the encoding process, the generation of the check-bits dominates the encoder propagation delay, and hence the delay is proportional to the encoder logic depth. For decoding, the decoder's propagation delay is comprised of syndrome generation, syndrome decoding, and error correction. Hence, the syndrome generation XOR logic depth and the syndrome decoder complexity both impact the decoding time.

As seen in Figure 3.15(a), each of the ECC schemes require the same number of XOR logic levels to encode the codeword's check-bits. Hence, the encoder delay percentage increase of even the (25, 16) RS code compared to the (22, 16) SEC-DED code is limited to only 17.3%. In terms of decoder delay, Figure 3.15(b), the delay slightly increases as a function of $\mathbf{I}_L$ matrix size. Although, this increase is limited to 16.7% for the (24, 16) $\mathbf{I}_5$ DAEC code over the (22, 16) Hsiao code. Further, for the (25, 16) $\mathbf{I}_6$ DAEC code where the XOR logic depth is reduced to 3 levels, the incremental delay is limited to only 2.6% over the baseline (22, 16) Hsiao code.

The TAEC feature (Figure 3.16) adds no more than a 10% increase in decoder delay relative to the (22,16) SEC-DED-DAEC code from [44] for any of the proposed 16 data-bit codes. While for the 32 data-bit implementations, the TAEC (40, 32)-$\mathbf{I}_4$ code requires only a 1.1% increase in decoder delay and 40.9% increase in area relative to the DAEC (40, 32)-$\mathbf{I}_4$ implementation.

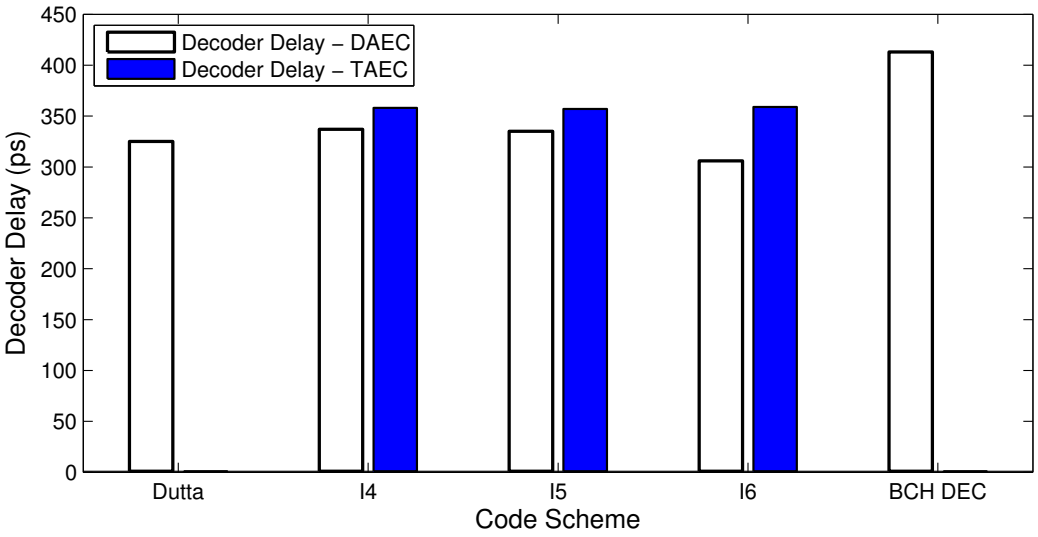


Figure 3.16: Synthesized propagation delay comparison between the proposed 16 data-bit DAEC code implementations and the TAEC implementations for the equivalent number of check bits.

3.6.4 Miscorrection Probability

For the proposed codes, the miscorrection probabilities are less than those presented in [44] for the equivalent number of data-bits and can be further reduced by increasing the number of check-bits. All miscorrection probabilities have been calculated using Equation 2.11 in Section 2.7.3. This reduction is due to the increased codeword length increasing the number of e-bit error combinations, and the code construction procedure reducing the number of sharable syndromes. As the number of check-bits increases, the number of possible non-zero syndromes increases while the number of error correcting syndromes for the code remains fixed. This is shown in Figure 3.17.

For the (22, 16) Dutta code there is a 64.3% probability of a double random bit error being misinterpreted as a 2-bit adjacent error, while this probability is reduced to 10.0% for the (26, 16) $\mathbf{I}_7$ proposed code. For 3-bit random errors, the miscorrection probability is reduced from 65.2% down to 14.0% for the same codes. The miscorrection probabilities for the Hsiao SEC-DED and BCH DEC codes are intrinsically zero for double-bit errors since the SEC-DED code does not perform any double error correction, and the DEC code is capable of correcting all adjacent and non-adjacent double-bit errors alike.

Since the bottom partition **H**-matrix columns in the proposed codes are restricted to a column weight of one, for any double bit error the bottom partition syndrome bits always have an even weight. Since the added TAEC syndrome patterns are the XOR result of

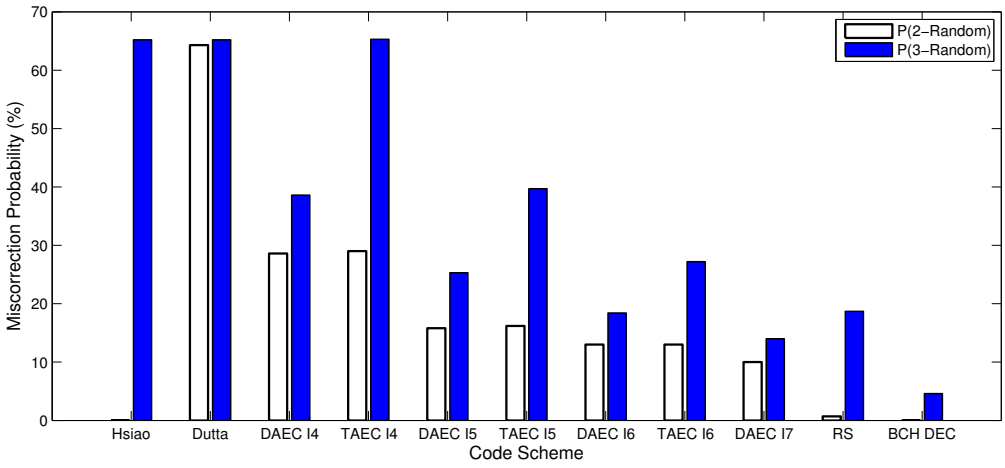


Figure 3.17: Miscorrection probabilities for 2-random and 3-random bit errors for the proposed 16 data-bit SEC-DED-DAEC-yAED code implementations, (22, 6) Hsiao SEC-DED code, (22, 16) SEC-DED-DAEC Dutta code, (25, 16) RS code, and (26, 16) DEC BCH code.

three adjacent columns, the bottom partition syndrome will always have a weight of three, thus preventing an overlap of the 2-bit random error and 3-bit adjacent error syndrome patterns. Hence, the TAEC feature in the proposed class of codes results in a minimal difference in the 2-bit random error miscorrection probability, P(2-Random). The slight differences in P(2-Random) seen in Table 3.4 between any of the DAEC and TAEC codes for the same $\mathbf{I}_L$ size and number of data-bits is due to its column selection and arrangement as opposed the added TAEC syndromes. For this same reason, the introduction of the TAEC syndromes does increase the three bit miscorrection probability since these error types can cause odd-column-weight syndrome patterns in the bottom partition sub-syndrome bits, however these three bit random errors are significantly more rare than their adjacent error counterparts.

3.6.5 Implementation Summary

The following section provides a comprehensive summary of results for the proposed codes for 16, 32, and 64 data-bit implementations. A summary of results is included for the Hsiao SEC-DED, Dutta SEC-DED-DAEC, BCH DEC, and Reed Solomon codes. Results have been divided into two tables, Table 3.4 summarizes all of the high-level characterization metrics extracted directly from the **H**-matrix, while Table 3.5 summarizes the synthesis results for each of the implemented circuits. For the 32 and 64 data-bit implementations,

$\mathbf{I}_3$ DAEC codes were found that use the same number of check-bits as the traditional SEC-DED codes. A $\mathbf{I}_3$ solution was not found for the 16 data-bit implementations.

H-Matrix Code Comparison

Table 3.4 summarizes the high-level characterization metrics extracted from the **H**-matrices of the proposed codes for 16, 32, and 64 data-bit memory word sizes. Metrics include: number of check-bits, error correction and detection ability, number of XOR logic levels and gate count required in the encoder check-bit generation logic and decoder syndrome-bit generation logic, and miscorrection probability for double and triple non-adjacent-bit errors being being identified and falsely corrected as otherwise correctable errors.

Synthesis Results

Table 3.5 summarizes the synthesis results of the HDL implementations for each of the proposed codes for 16, 32, and 64 data-bit memory word sizes. Synthesis results have been generated in Synopsys Design Compiler using Verilog and a commercial 65 nm general purpose bulk CMOS technology standard cell library. Metrics for each code include the number of check-bits and error correction and detection ability as reference, as well as the synthesized area, encoder/decoder propagation delay, and power estimations. Estimated power is dynamic switching power.

3.6.6 Modified Codes

In addition to the SEC-DED-DAEC-yAED and SEC-DED-TAEC-yAED codes presented in the previous sections, a variety of codes derived from these have been created as well. These include: increased identity matrix size (IIMS) codes, increased check-bit (ICB) codes, and optimized codes. Each of these slightly modify the code construction procedure in favour of one set of metrics at the expense of another.

Increased Identity Matrix Size Codes

Increased identity matrix size codes maximize the size of the repeated identity matrix, $\mathbf{I}_L$, in the bottom portion of the proposed code's **H**-matrix for an equivalent number of check-bits as one of the standard proposed SEC-DED-DAEC-yAED codes. This is done by minimizing the required number of check-bits in the top portion of the **H**-matrix, rather than simply using a one-to-one increasing correspondence between the number of check-bits for a code and its identity matrix size. These codes provide an increased level of adjacent and burst

Table 3.4: H-Matrix Code Comparison Summary

(n, k) I-Size	Code	Check-bits	xAEC	yAED	zBED	Generator XOR Logic			Miscorrection Probability	
						Logic Levels		Gate	P(2-Random)	P(3-Random)
						Encode	Decode	Count	(%)	(%)
16 Data-bit Codes										
(22, 16)	SEC-DED [31]	6	1	2	2	4	4	48	0.0	65.2
(22, 16)	DAEC [44]	6	2	2	2	4	4	48	64.3	65.2
(23, 16) $\mathbf{I}_4$	Proposed, DAEC	7	2	5	3	4	4	42	28.6	38.6
(24, 16) $\mathbf{I}_5$	Proposed, DAEC	8	2	7	4	4	4	39	15.8	25.3
(25, 16) $\mathbf{I}_6$	Proposed, DAEC	9	2	9	5	4	3	36	13.0	18.4
(26, 16) $\mathbf{I}_7$	Proposed, DAEC	10	2	11	6	4	3	35	10.0	14.0
(23, 16) $\mathbf{I}_4$	Proposed, TAEC	7	3	4	3	4	4	42	29.0	65.3
(24, 16) $\mathbf{I}_5$	Proposed, TAEC	8	3	6	4	4	4	39	16.2	39.7
(25, 16) $\mathbf{I}_6$	Proposed, TAEC	9	3	8	5	4	3	36	13.0	27.2
(26, 16)	BCH DEC [19]	10	2	2	2	4	4	94	0.0	4.6
(25, 16)	RS S3EC-D3ED [11]	9	1-3*	4-6*	4-6*	4	4	70	0.7	18.7
32 Data-bit Codes										
(39, 32)	SEC-DED [31]	7	1	2	2	4	4	96	0.0	59.6
(39, 32)	DAEC [44]	7	2	2	2	4	4	96	57.3	59.6
(39, 32) $\mathbf{I}_3$	Proposed DAEC	7	2	3	2	7	5	98	49.8	60.8
(40, 32) $\mathbf{I}_4$	Proposed DAEC	8	2	5	3	5	4	88	24.8	36.4
(41, 32) $\mathbf{I}_5$	Proposed, DAEC	9	2	7	4	5	4	84	21.4	29.0
(42, 32) $\mathbf{I}_6$	Proposed, DAEC	10	2	9	5	5	4	80	9.0	18.1
(40, 32) $\mathbf{I}_4$	Proposed, TAEC	8	3	4	3	5	4	88	25.8	60.3
(41, 32) $\mathbf{I}_5$	Proposed, TAEC	9	3	6	4	5	4	84	21.8	43.7
(42, 32) $\mathbf{I}_6$	Proposed, TAEC	10	3	8	5	5	4	80	9.3	25.7
(44, 32)	BCH DEC [19]	12	2	2	2	5	5	200	0.0	1.4
(44, 32)	RS S4EC-D4ED [11]	12	1-4*	5-8*	5-8*	5	5	157	0.0	4.1
64 Data-bit Codes										
(72, 64)	SEC-DED [31]	8	1	2	2	5	5	208	0.0	55.6
(72, 64)	DAEC [44]	8	2	2	2	5	5	208	54.6	55.6
(72, 64) $\mathbf{I}_3$	Proposed DAEC	8	2	3	2	7	5	211	47.8	57.0
(73, 64) $\mathbf{I}_4$	Proposed DAEC	9	2	5	3	7	5	191	25.8	35.4
(74, 64) $\mathbf{I}_5$	Proposed, DAEC	10	2	7	4	6	5	177	17.1	24.7
(75, 64) $\mathbf{I}_6$	Proposed, DAEC	11	2	9	5	6	5	172	11.0	18.0
(74, 64) $\mathbf{I}_5$	Proposed, TAEC	10	3	6	4	6	5	177	18.3	36.6
(75, 64) $\mathbf{I}_6$	Proposed, TAEC	11	3	8	5	6	5	172	11.4	25.3
(78, 64)	BCH DEC [19]	14	2	2	2	6	6	445	0.0	0.9
(79, 64)	RS S5EC-D5ED [11]	15	1-5*	6-10*	6-10*	6	6	364	0.3	1.3

*Correction and detection capabilities for each the Reed-Solomon codes depend on the error location

Table 3.5: Synthesis Results Comparison Summary

16 Data-bit Codes

(n, k) I-Size	Code	Check-bits	xAEC	yAED	zBED	Delay Optimized Synthesis Results			
						Area (μm^2)	Delay (ns)		Power (μW)
							Encode	Decode	
(22, 16)	SEC-DED [31]	6	1	2	2	462.24	0.208	0.298	192.2
(22, 16)	DAEC [44]	6	2	2	2	604.44	0.230	0.325	245.2
(23, 16) $\mathbf{I}_4$	Proposed, DAEC	7	2	5	3	783.72	0.230	0.337	263.5
(24, 16) $\mathbf{I}_5$	Proposed, DAEC	8	2	7	4	826.20	0.234	0.335	250.0
(25, 16) $\mathbf{I}_6$	Proposed, DAEC	9	2	9	5	1069.2	0.227	0.306	320.5
(26, 16) $\mathbf{I}_7$	Proposed, DAEC	10	2	11	6	836.28	0.243	0.348	242.2
(23, 16) $\mathbf{I}_4$	Proposed, TAEC	7	3	4	3	1026.0	0.237	0.358	325.7
(24, 16) $\mathbf{I}_5$	Proposed, TAEC	8	3	6	4	978.48	0.251	0.357	285.1
(25, 16) $\mathbf{I}_6$	Proposed, TAEC	9	3	8	5	990.72	0.237	0.359	288.3
(26, 16)	BCH DEC [19]	10	2	2	2	2606.4	0.238	0.413	732.6
(25, 16)	RS S3EC-D3ED [11]	9	1-3*	4-6*	4-6*	1050.44	0.244	0.351	355.4

32 Data-bit Codes

(n, k) I-Size	Code	Check-bits	xAEC	yAED	zBED	Delay Optimized Synthesis Results			
						Area (μm^2)	Delay (ns)		Power (μW)
							Encode	Decode	
(39, 32)	SEC-DED [31]	7	1	2	2	829.80	0.248	0.327	372.6
(39, 32)	DAEC [44]	7	2	2	2	1177.56	0.252	0.357	521.4
(39, 32) $\mathbf{I}_3$	Proposed DAEC	7	2	3	2	925.56	0.359	0.459	403.1
(40, 32) $\mathbf{I}_4$	Proposed DAEC	8	2	5	3	954.36	0.329	0.439	391.6
(41, 32) $\mathbf{I}_5$	Proposed, DAEC	9	2	7	4	1313.64	0.339	0.419	519.3
(42, 32) $\mathbf{I}_6$	Proposed, DAEC	10	2	9	5	1341.36	0.328	0.405	505.3
(40, 32) $\mathbf{I}_4$	Proposed, TAEC	8	3	4	3	1344.96	0.324	0.444	511.7
(41, 32) $\mathbf{I}_5$	Proposed, TAEC	9	3	6	4	1619.28	0.321	0.421	582.1
(42, 32) $\mathbf{I}_6$	Proposed, TAEC	10	3	8	5	1512.72	0.304	0.444	526.8
(44, 32)	BCH DEC [19]	12	2	2	2	9267.84	0.349	0.523	4393
(44, 32)	RS S4EC-D4ED [11]	12	1-4*	5-8*	5-8*	1781.64	0.262	0.427	684.8

64 Data-bit Codes

(n, k) I-Size	Code	Check-bits	xAEC	yAED	zBED	Delay Optimized Synthesis Results			
						Area (μm^2)	Delay (ns)		Power (μW)
							Encode	Decode	
(72, 64)	SEC-DED [31]	8	1	2	2	1611.0	0.307	0.395	779.7
(72, 64)	DAEC [44]	8	2	2	2	2386.8	0.309	0.424	1017.0
(72, 64) $\mathbf{I}_3$	Proposed DAEC	8	2	3	2	1623.2	0.388	0.530	721.3
(73, 64) $\mathbf{I}_4$	Proposed DAEC	9	2	5	3	1837.4	0.373	0.506	739.8
(74, 64) $\mathbf{I}_5$	Proposed, DAEC	10	2	7	4	2066.0	0.366	0.483	857.0
(75, 64) $\mathbf{I}_6$	Proposed, DAEC	11	2	9	5	2399.4	0.313	0.458	979.8
(74, 64) $\mathbf{I}_5$	Proposed, TAEC	10	3	6	4	3112.9	0.370	0.464	1207.0
(75, 64) $\mathbf{I}_6$	Proposed, TAEC	11	3	8	5	2784.2	0.369	0.494	985.1
(78, 64)	BCH DEC [19]	14	2	2	2	30811.6	0.386	0.598	14831.0
(79, 64)	RS S5EC-D5ED [11]	15	1-5*	6-10*	6-10*	5035.32	0.352	0.519	1642.0

*Correction and detection capabilities for each the Reed-Solomon codes depend on the error location

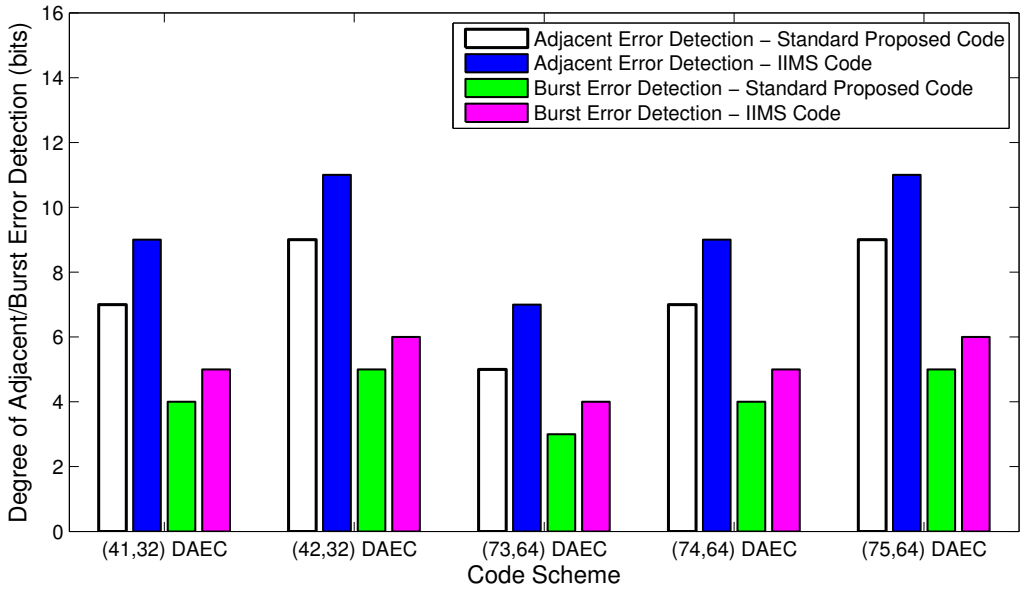


Figure 3.18: Degree of adjacent and burst error detection for Increased Identity Matrix Size codes compared to their equivalent SEC-DED-DAEC-yAED codes for the same number of check-bits.

error detection without an increase in the number of check-bits. Figure 3.18 compares the amount of provided error detection for the IIMS codes versus their equivalent SEC-DED-DAEC-yAED code for the same codeword size (i.e., same number of both check- and data-bits), while Table 3.6 compares the other metrics for the codes. The table includes both the IIMS codes and their SEC-DED-DAEC-yAED equivalents. IIMS codes have been created for the 32 and 64 data-bit DAEC codes.

Since each of the IIMS codes use an identity matrix one bit size larger than its SEC-DED-DAEC-yAED code with an equivalent number of check-bits, the IIMS code provides an additional two bits of AED and one additional bit of BED. This improvement comes at the cost of an increase in the code's miscorrection probabilities. The impact on silicon area and encoder/decoder performance is implementation dependent.

Increased Check-bit Codes

By increasing the number of check-bits beyond its minimum requirement for a given identity matrix size, the miscorrection probabilities of a code can be reduced. This can be performed for any type of code including Dutta, Hsiao, Hamming, BCH, or RS codes. An example is shown in Figure 3.19.

Table 3.6: Increased Identity Matrix Size Code Comparison

32 Data-bit Codes										
(n, k) I-Size	Code	Check-bits	xAEC	yAED	zBED	Synthesis Results			Miscorrection Probability	
						Area (μm^2)	Delay (ns)		P(2-Random) %	P(3-Random) %
							Encode	Decode		
(39, 32)	DAEC [44]	7	2	2	2	1177.56	0.252	0.357	57.3	59.6
(41, 32) $\mathbf{I}_5$	Proposed, DAEC	9	2	7	4	1313.64	0.339	0.419	21.4	29.0
(41, 32) $\mathbf{I}_6$	Proposed, IIMS	9	2	9	5	1591.20	0.307	0.391	24.5	33.5
(42, 32) $\mathbf{I}_6$	Proposed, DAEC	10	2	9	5	1341.36	0.328	0.405	9.0	18.1
(42, 32) $\mathbf{I}_7$	Proposed, IIMS	10	2	11	6	1316.88	0.261	0.379	16.5	24.5

64 Data-bit Codes										
(n, k) I-Size	Code	Check-bits	xAEC	yAED	zBED	Synthesis Results			Miscorrection Probability	
						Area (μm^2)	Delay (ns)		P(2-Random) %	P(3-Random) %
							Encode	Decode		
(72, 64)	DAEC [44]	8	2	2	2	2386.8	0.309	0.424	54.6	55.6
(73, 64) $\mathbf{I}_4$	Proposed DAEC	9	2	5	3	1837.4	0.373	0.506	25.8	35.4
(73, 64) $\mathbf{I}_5$	Proposed, IIMS	9	2	7	4	2542.68	0.361	0.48	34.6	45.3
(74, 64) $\mathbf{I}_5$	Proposed, DAEC	10	2	7	4	2066.0	0.366	0.483	17.1	24.7
(74, 64) $\mathbf{I}_6$	Proposed, IIMS	10	2	9	5	2107.08	0.363	0.477	23.0	32.0
(75, 64) $\mathbf{I}_6$	Proposed, DAEC	11	2	9	5	2399.4	0.313	0.458	11.0	18.0
(75, 64) $\mathbf{I}_7$	Proposed, IIMS	11	2	11	6	1898.64	0.409	0.515	16.2	23.9

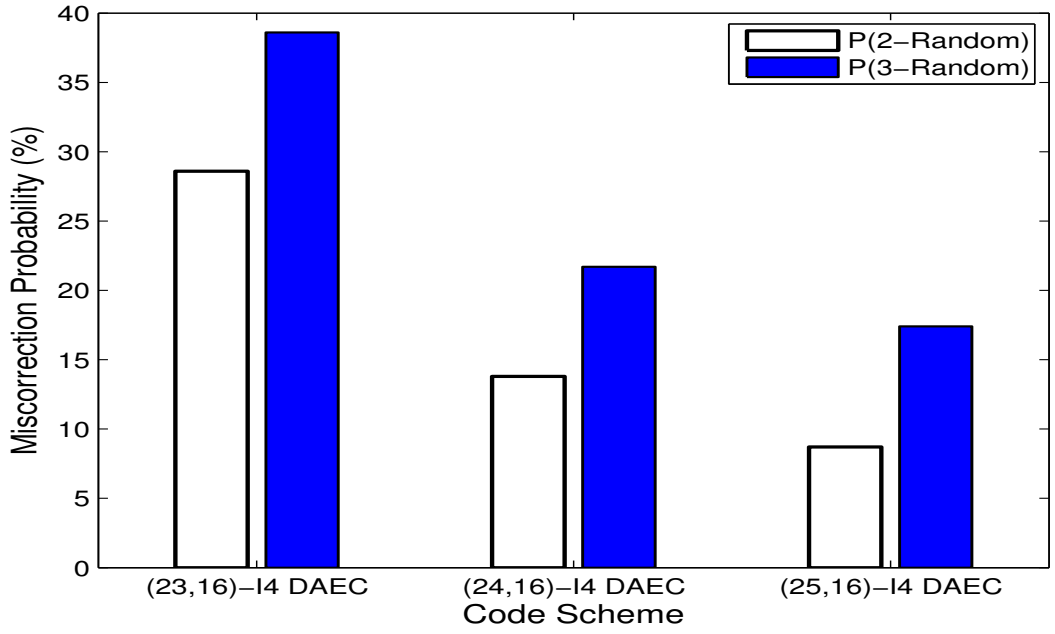


Figure 3.19: Miscorrection probabilities for 16 Data-bit Increased Check-bit codes.

By adding an additional check-bit to a (**23**, 16) $\mathbf{I}_4$ SEC-DED-DAEC-5AED code to make a (**24**, 16) $\mathbf{I}_4$ SEC-DED-DAEC-5AED code, the double-random-bit error miscorrection probability is reduced by 51.7% and the triple-random-bit error miscorrection probability is reduced by 43.8%. Since the size of the identity matrix is not changed, there is no added benefit to the degree of error detection. As the number of check-bits increases, the miscorrection probabilities progressively decrease. For the (**25**, 16) $\mathbf{I}_4$ SEC-DED-DAEC-5AED code, the 2-random and 3-random miscorrection probabilities decrease by 69.6% and 54.9% from the (**23**, 16) $\mathbf{I}_4$ DAEC-5AED code's 2-random and 3-random miscorrection probabilities respectively. ICB codes have practicality in fail-safe applications where miscorrection must be minimized.

Optimized Codes

By using the existing set of codes as an initial starting point, certain codes can be optimized to improve their performance. The (24, 16) $\mathbf{I}_5$ SEC-DED-DAEC-7AED has been found to be the best example of this. The (24, 16) $\mathbf{I}_5$ code in Figure 3.3 has row weights of 9, 7, and 7 in the top portion of its **H**-matrix. The calculation of their syndrome-bits require 4, 3, and 3 XOR logic levels respectively. If the first syndrome-bit with row weight 9 can be reduced to a row weight of 8, then it too can be calculated in $\log_2(8) = 3$ XOR logic levels. This can be achieved through row weight balancing to produce row weights of 8, 8, and 7 data-bits per syndrome-bit calculation. Unfortunately, this cannot be achieved while remaining in compliance with the column interleaving construction procedure discussed in Section 3.2.4, as by inspection, modifying a set of interleaved columns will affect at a minimum two columns, and simply shift the row weight of 9 from the first row of the **H**-matrix to either its second or third row. If instead, the interleaving guideline is relaxed and only a single column is modified, then the row weights can be balanced to minimize the maximum row weight. This column rearranging procedure has been performed on the parity check matrix in Figure 3.20 and the columns have been rearranged to ensure they still meet the DAEC criterion (Constraint 4).

Notice that this optimized parity check matrix has the same total weight as the previous implementation, and thus uses the same number of XOR gates for syndrome generation, but the row weights in the top portion of the matrix have been balanced to maintain a maximum row weight of 8. This allows for each of the top portion syndrome bits to be calculated within 3 XOR logic levels. This new optimized code is compared against the (22, 16) Dutta code and standard (24, 16) $\mathbf{I}_5$ SEC-DED-DAEC-7AED proposed code in Table 3.7.

Notice that the optimized code requires one less XOR logic level for its syndrome-bit generation logic as compared to the standard proposed (24, 16) $\mathbf{I}_5$ code and the (22, 16)

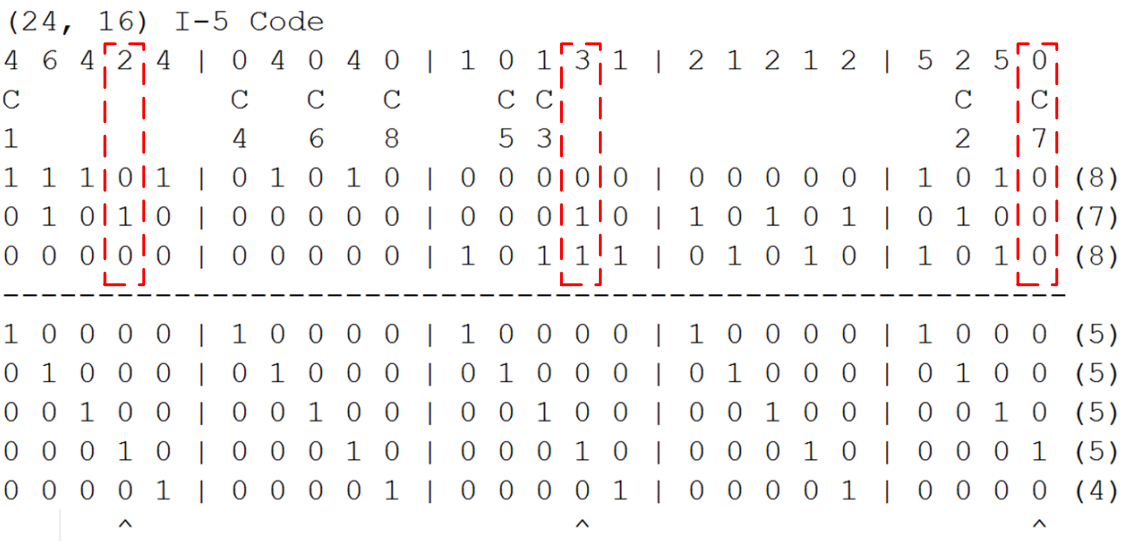


Figure 3.20: Optimized (24, 16) $\mathbf{I}_5$ code with maximum row weight of 8.

Table 3.7: (24, 16) $\mathbf{I}_5$ Optimized Code Example

(24, 16) I_5 Optimized Code Performance Example										
(n, k) I-Size	**Code**	**Check-bits**	**Generator XOR Logic**			**Synthesis Results**			**Miscorrection Probability**	
			Logic Levels		**Gate Count**	**Area (μm^2)**	**Delay (ns)**		**P(2-Random) %**	**P(3-Random) %**
			Encode	**Decode**			**Encode**	**Decode**		
(22, 16)	DAEC [44]	6	4	4	48	604.44	0.230	0.325	64.3	65.2
(24, 16) $\mathbf{I}_5$	Proposed, DAEC	8	4	4	39	783.72	0.230	0.337	28.6	38.6
(24, 16) $\mathbf{I}_5$	Proposed, Optimized	8	4	3	39	1031.4	0.223	0.312	24.5	33.5

Dutta code. This optimization yields a 7.4% improvement in synthesized decoder logic delay over the standard proposed scheme and a 4% improvement over the Dutta code.

Relaxing the strict column interleaving guideline can reduce the total number of XOR gates used in the syndrome generation logic for other codes; however, the (24, 16) $\mathbf{I}_5$ code is the only one that has been found for which the maximum syndrome-bit logic level depth can be reduced. This is due to the initial row weight values of the code being within such close proximity to a logic depth power-of-2 boundary. Other codes that show promise for a reduction in check-bit generation delay include the (39, 32) $\mathbf{I}_3$ SEC-DED-DAEC-3AED code and the (42, 32) $\mathbf{I}_7$ IIMS code.

3.7 Summary

In this chapter, the basic structure of the proposed error correction codes has been presented along with a set of constraints for constructing and showing each code's validity. An algorithm is presented for searching of acceptable codes. Finally, various high level and circuits-based performance metrics have been considered and compared against existing codes for 16, 32, and 64 data-bit implementations.

Chapter 4

Implementation, Verification, and Measurement

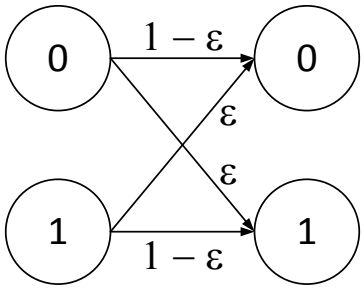
This chapter presents a soft error channel model and simulation results for a set of 64 data-bit error correcting codes in the presence of the error channel. The design of a SRAM+ECC test chip implemented in a 28 nm technology is then presented with neutron radiation test data. The radiation data is used in conjunction with SPICE-based critical charge simulations to estimate the soft error rates of a set of vendor designed SRAM bitcells.

4.1 Soft Error Rate Modeling

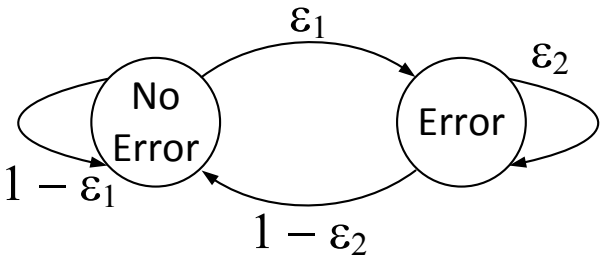
Although multi-cell upsets have been shown to make up over 35% of the total SER below the 32 nm process node, not all of these upsets necessarily result in errors [11]. The MCUs that can cause the most potential harm are the multi-bit upsets, those upsets that corrupt multiple bits within the same memory word. These upsets have been shown to make up approximately 3% of the total SER at the 32 nm node and, as shown in Table 1.1, increase with technology scaling [6]. In this section, a multi-bit upset error channel model is presented and applied against a collection of 64 data-bit error correction codes.

4.1.1 Error Channel Model

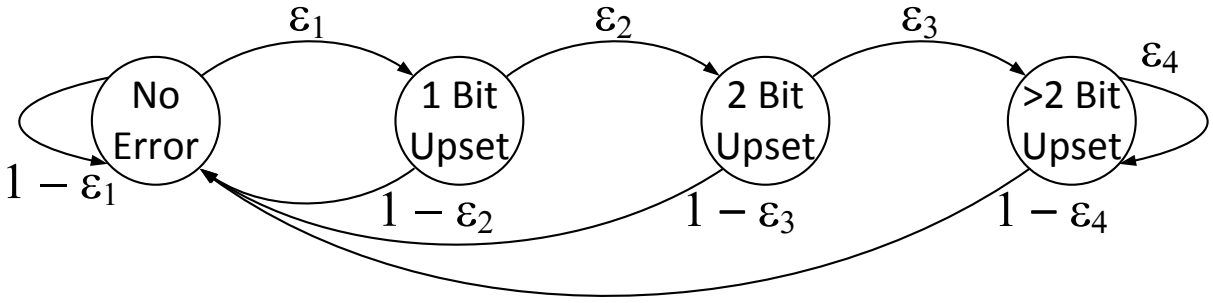
For an embedded memory system, radiation induced soft errors occur while the data resides in the memory's bitcells. The error channel exists temporally, between the time that data is written to a particular memory location, and when it is later read. The binary symmetric channel model, shown in Figure 4.1(a), provides a simple single-bit error model. It assumes



(a) Binary Symmetric Channel Model



(b) Gilbert-Elliott Channel Model



(c) 4-State Markov Chain Channel Model

Figure 4.1: Radiation Induced Noise Channel Models

an equal probability, ϵ, for a bit flipping its state from either $1 \rightarrow 0$ or $0 \rightarrow 1$, and probability $1 - \epsilon$ for maintaining its data within the channel. This is sufficient for SBUs; however, when an error can generate MBUs, a more sophisticated model is required to determine the number of bits upset per particle strike.

The Gilbert-Elliott model, shown in Figure 4.1(b), is a well known channel model used to represent burst errors [67]. It uses a 2-state Markov chain to indicate transitions between a 'good' error-free state and a 'bad' error state. This can be used for adjacent bit upsets where ϵ_1 is the probability of an error occurring and, ϵ_2 is the probability of successive adjacent bits also being upset. $1 - \epsilon_1$ and $1 - \epsilon_2$ are the probabilities of a bit not being upset based on the state of its adjacent bit. This can be used to inject error patterns into a transmitted codeword. First, an error pattern is generated by applying a binary symmetric channel to the transmitted codeword with $\epsilon = \epsilon_1$. Then, for each error, its size is determined by iterating through its adjacent bits using ϵ_2 as the probability of successively increasing its error size.

In this work, the Gilbert-Elliott model is extended using the 4-state Markov chain shown in Figure 4.1(c). The two additional states are added to provide a more accurate representation of the MBU behavior observed in [6]. Like the Gilbert-Elliott channel model,

Table 4.1: Example Error Distribution Before and After Error Correction Using the SEC-DED-TAEC-8AED Code

Error Size	Before Correction		After Correction	
(adjacent-bits)	Count	%	Count	%
1	9635	96.30	0	0.00
2	317	3.16	0	0.00
3	19	0.19	0	0.00
4	15	0.15	15	44.12
5	10	0.10	10	29.41
6	3	0.03	3	8.82
7	2	0.02	2	5.88
8	2	0.02	2	5.88
9	2	0.02	2	5.88
Total	10005	100.0	34	100.0

$\epsilon_1 = 10^{-6}, \epsilon_2 = 0.036, \epsilon_3 = 0.15, \epsilon_4 = 0.6$

ϵ_1 represents the occurrence of an error and ϵ_2 the probability of the upset affecting multiple bits, ϵ_3 and ϵ_4 are then used to determine the size of the upset. $1-\epsilon_2$, $1-\epsilon_3$, and $1-\epsilon_4$ are the probabilities of a given bit not being upset when the respective number of its adjacent bits have been upset.

The 4-state Markov chain error channel model is used to compare the effectiveness of each of the ECCs discussed in Section 4.1.2 using a custom SER test suite developed in Matlab. The simulations show each code's corrected-SER as a function of its raw-SER in terms of errors/bit-year after being exposed to the error channel. All simulations generate a minimum of 10^4 errors per datapoint to ensure statistical significance of the results. The channel parameters have been tuned to resemble the data presented in [6] for a 22 nm technology using ϵ_1 equal to the raw-SER, $\epsilon_2 = 0.036$ the MBU percentage of the total SER, $\epsilon_3 = 0.15$, and $\epsilon_4 = 0.6$. Table 4.1 shows an example distribution of the number of errors before and after the ECC correction process is performed by the TAEC code for a raw-SER of 10^{-6} errors/bit-year, and Table 4.2 shows this same raw error distribution in comparison to the 22 nm technology in [6]. In this example, MBUs make up 3.69% of the total SER, and all of the single, double-adjacent, and triple-adjacent bit errors have been corrected, the distributions are within 5% of the distribution in [6] for error sizes up to 8 bits. By increasing the simulation run time, a greater number of the larger, lower probability upsets will occur, and potentially increase the maximum width upset.

While failures in time (FIT)/Mb is more commonly used to measure SER, the error/bit-year unit lends itself more readily to the simulation environment (1 FIT = 1 error per 10^9 device hours, and 1 FIT/Mb = 8.354x10^{-12} errors/bit-year). In Section 4.1.2, results are

Table 4.2: MCU bit width and percentage of the total SER comparison between the 4-state Markov chain model and 22 nm technology in [6]

Error Channel	% of Total SER	Bit width				
		2	3	4-8	>8	Max (bits)
4-state Markov Chain Model	3.69	3.16	0.19	0.30	0.02	9
22 nm technology [6]	3.6	3.0	0.2	0.3	0.1	18

Table 4.3: ECC Check-bit Overheads

Code	n (bits)	r (bits)	k (bits)	Overhead (%)
(72, 64) SEC-DED	72	8	64	12.5
(72, 64)-I_3 SEC-DED-DAEC-3AED	72	8	64	12.5
(75, 64)-I_6 SEC-DED-TAEC-8AED	75	11	64	17.2
(78, 64) BCH DEC	78	14	64	21.9
(79, 64) RS S5EC-D5ED	79	15	64	23.4

first presented in terms of errors/bit-year, then in terms of FIT/Mb in the International Technology Roadmap for Semiconductors (ITRS) anticipated sub-2000 range, and finally at the system level in terms of FIT/device.

4.1.2 Simulation Results - Corrected-SER vs. Raw-SER

The check-bit overheads for the 64 data-bit codes considered in this work are shown in Table 4.3. Of these codes, the SEC-DED code has the lowest overhead, but is only capable of correcting single-bit errors and detecting two-bit random errors. The SEC-DED-DAEC-3AED (DAEC) and SEC-DED-TAEC-8AED (TAEC) codes presented in Chapter 3, are designed for small adjacent error correction and detection, and provide 2-bit adjacent correction/3-bit adjacent detection, and 3-bit adjacent correction/8-bit adjacent detection respectively. The BCH DEC code is capable of correcting any random two-bit error pattern, and hence requires roughly twice the number of check-bits as the SEC-DED code, while the Reed Solomon (RS) code is capable of single 5-bit-byte error correction and double 5-bit-byte error detection (S5EC-D5ED). Depending on implementation, both the BCH and RS codes can require multiple clock cycles to perform their error correction/detection procedure.

In Figure 4.2 the solid line indicates the situation where no ECC is applied. In this case, the corrected-SER is equal to the raw-SER. The SEC-DED and BCH DEC codes provide roughly one to two orders of magnitude SER improvement over the unprotected base case respectively. For the same check-bit overhead as the basic SEC-DED code, the

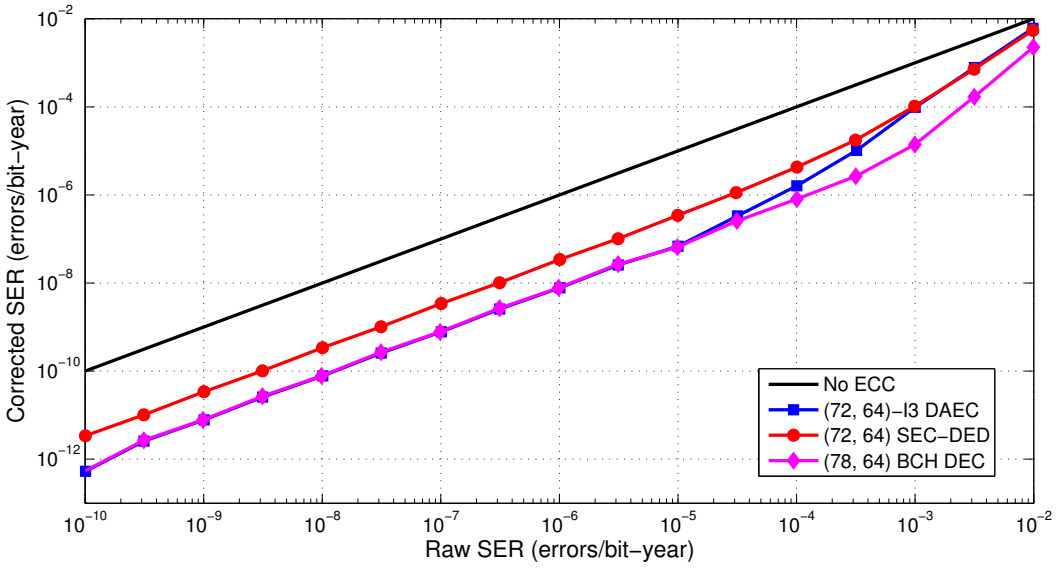


Figure 4.2: Corrected-SER vs. raw-SER for (72, 64)-I_3 SEC-DED-DAEC-3AED, (72, 64) SEC-DED, and (78, 64) BCH DEC codes using a 4-state Markov chain error channel model.

DAEC code provides a SER improvement between that of the SEC-DED and BCH DEC codes. For error rates greater than 10^{-3} errors/bit-year, the probability of multiple non-adjacent upsets occurring within the same codeword is still relatively high, and hence the DAEC code performs roughly equal to the SEC-DED code. For lower error rates, the probability of multiple independent upsets occurring within the same word is reduced, and hence adjacent bit upsets dominate the number of MBUs. This allows the corrected-SER of the DAEC code to approach that of the BCH DEC code for a 42.8% reduction in check-bit overhead.

In Figure 4.3 the TAEC code is compared against the MBU correcting BCH DEC and Reed Solomon S5EC-D5ED codes. Again, for higher SERs, the BCH code provides better random error correcting performance relative to the adjacent error correcting TAEC and RS codes since the probability of multiple non-adjacent upsets occurring within the same codeword is higher. For lower SERs however, adjacent MBUs are more prominent and the TAEC code outperforms the BCH DEC code providing a 1.5x-2.35x improvement in corrected-SER.

According to the ITRS guidelines, SRAM bit-level SER is expected to be below 2000 FIT/Mb over the next decade [10]. Translating the simulated error rates into units of FIT/Mb provides the data shown in Figure 4.4. Data is shown for raw (uncorrected) SERs between 0 and 2000 FIT/Mb. Again, the solid line indicates the situation where no ECC is applied. Applying the SEC-DED code provides an approximate 30x reduction

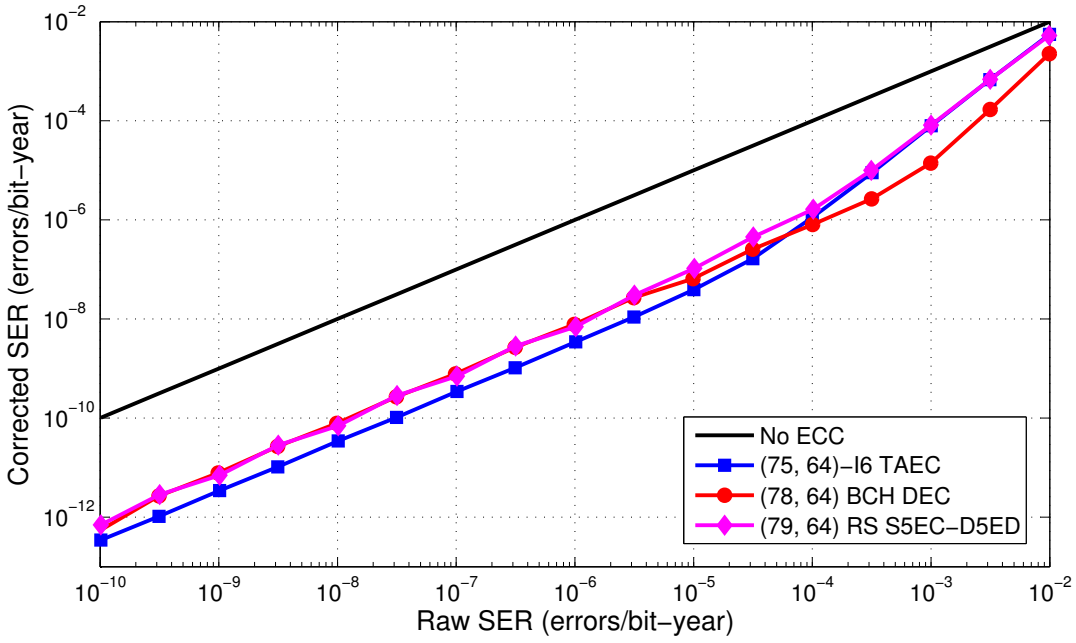


Figure 4.3: Corrected-SER vs. raw-SER for (75, 64)-I_6 SEC-DED-TAEC-8AED, (78, 64) BCH DEC, and (79, 64) Reed Solomon S5EC-D5ED codes using a 4-state Markov chain error channel model.

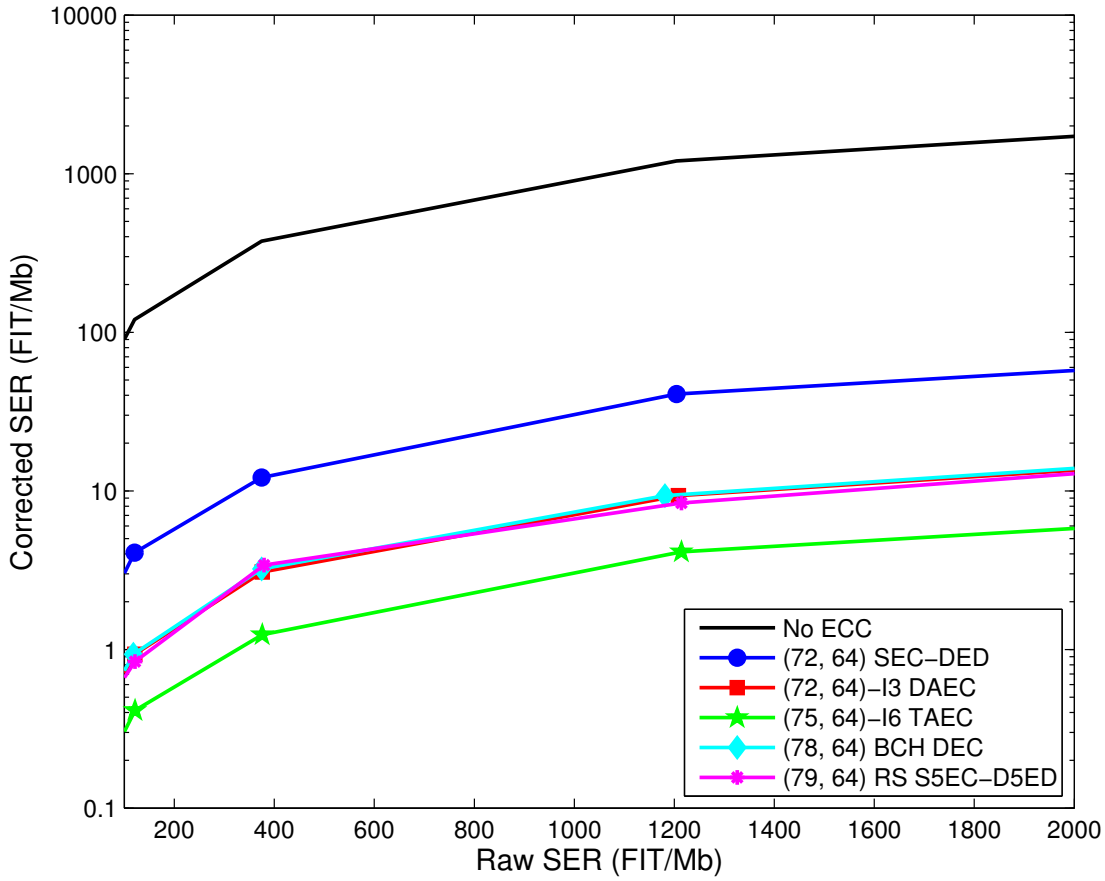
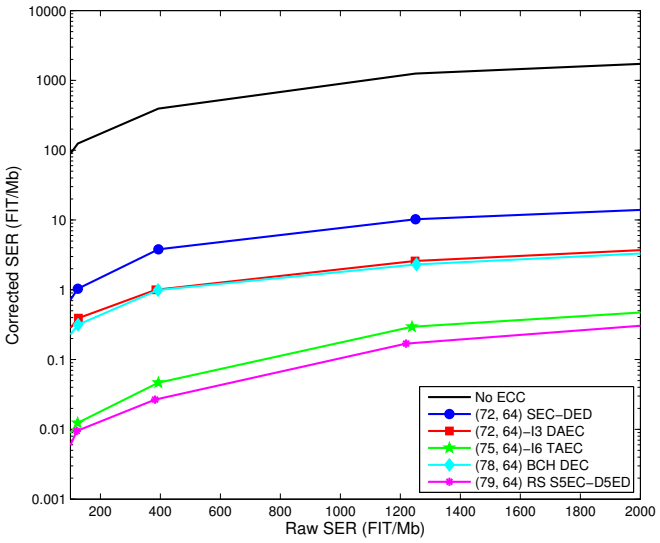


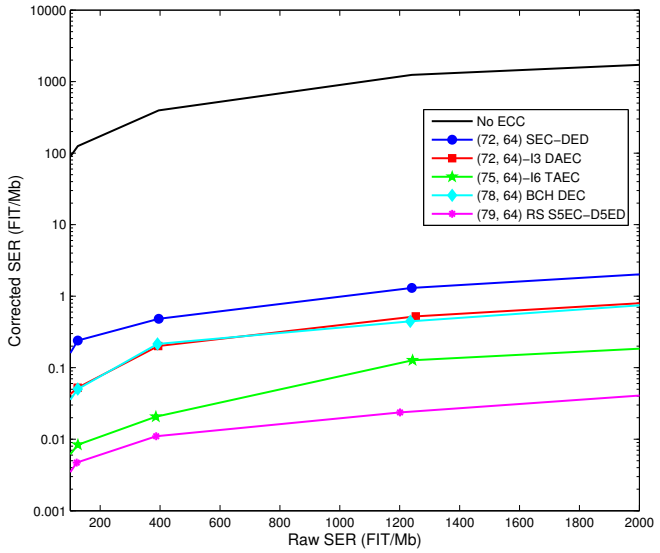
Figure 4.4: Simulated corrected-SER vs. raw-SER for various ECC schemes using 1-way interleaving.

in SER, while the BCH DEC and RS codes provide roughly 120x SER improvement over the unprotected baseline. For the same check-bit overhead as the basic SEC-DED scheme, the DAEC code provides a SER improvement comparable to that of the BCH DEC and RS codes, while for a cost of three additional check-bits, the TAEC code provides a SER improvement of approximately 300x over the raw-SER, and 2.25-2.5x over the BCH DEC and RS codes. For an unprotected raw-SER of 1200 FIT/Mb, the SEC-DED protection reduces the SER to 40.83 FIT/Mb, while the BCH DEC, DAEC, and TAEC codes provide corrected-SERs of 9.33, 9.32, and 4.13 FIT/Mb respectively.

Applying interleaving in conjunction with the various ECC schemes provides the corrected-SERs shown in Figure 4.5. Figure 4.5(a) provides the 2-way interleaving data, while Fig-



(a) 2-way Interleaving



(b) 4-way Interleaving

Figure 4.5: Simulated corrected-SER vs. raw-SER for various ECC schemes using 2-, and 4-way interleaving.

ure 4.5(b) shows the situation when 4-way interleaving is applied. Applying additional degrees of interleaving improves the corrected-SER for each coding scheme. A comparison of the corrected-SERs for each code as a function of interleaving is shown in Figure 4.6 using a fixed raw-SER of 1200 FIT/Mb. The corrected-SER of the $(72, 64)$-$\mathbf{I}_3$ DAEC code is within 0.3 FIT/Mb of the $(78, 64)$ BCH code for each interleaving scheme while requiring 6 fewer check-bits. This occurs since the adjacent upsets dominate the error channel causing the added random error correction protection to provide only a margin benefit.

The Reed-Solomon code benefits most in terms of improved corrected-SER by increasing the degree of interleaving. This is due to the code's arrangement and byte-wise nature of its interleaving scheme. Without any interleaving (1-way interleaving), errors as small as two adjacent bits can cause uncorrectable upsets if these errors occur within two separate bytes of the RS scheme. This causes the corrected-SER for the RS code to be on the same order as the $\mathbf{I}_3$ DAEC and BCH codes while requiring the greatest number of check-bits. By adding one additional degree of interleaving however, the same susceptible two adjacent bits are separated by an entire b-bit byte distance. For the presented $(79, 64)$ S5EC-D5ED code, $b = 5$, and as such a minimum 7-adjacent bit upset would be required to upset two bytes within the same codeword; hence, the code can provide a minimum 6-adjacent bit error correction using 2-way interleaving.

The $(75, 64)$-$\mathbf{I}_6$ TAEC code is able to provide a SER of approximately half that of the RS code with no interleaving applied, and within approximately 0.1 FIT/Mb for 2- and 4-way interleaving while requiring four fewer check-bits. Using 2-way interleaving, the $(75, 64)$-$\mathbf{I}_6$ TAEC code can correct errors of up to 6 adjacent-bits, or using 4-way interleaving, 12 adjacent-bits. This shows that for an equivalent degree of interleaving, the proposed codes can provide a cost effective solution in terms of check-bits relative to their multi-bit correcting code counterparts, while still providing an adequate amount of soft error protection.

The bit-level SER can be translated into a device or system-level SER by multiplying the bit-level SER by the total capacity of the memory system. The memory capacity for modern microprocessor embedded L2 and L3 SRAM cache is on the order of megabytes (MB), where 1 MB = 8 Mb. For instance, AMD's two-core x86, 64-bit Steamroller CPU (fabricated in a 28 nm HK+MG process) contains a 2 MB L2-cache shared between the two cores [68], while Intel's 22 nm Haswell processors use an 8 MB L3-cache paired with a 128 MB L4 in-package embedded DRAM module [69]. Further, the high performance variant of Intel's 65 nm Xeon processor family contains a 16 MB shared L3-cache [62]. In Figure 4.7, the 1200 raw-SER/Mb data with 2-way interleaving is mapped to the system level for 2, 4, 8, and 16 MB data capacities. For an 8 MB data capacity, the SER/device for the BCH DEC, DAEC, RS, and TAEC schemes are respectively 179.4, 185.7, 13.4, and 21.75 FIT/device; this is down from 76800 FIT/device (1200 FIT/Mb $\times$ 8Mb/MB $\times$ 8MB/device) for the case of the unprotected array without ECC.

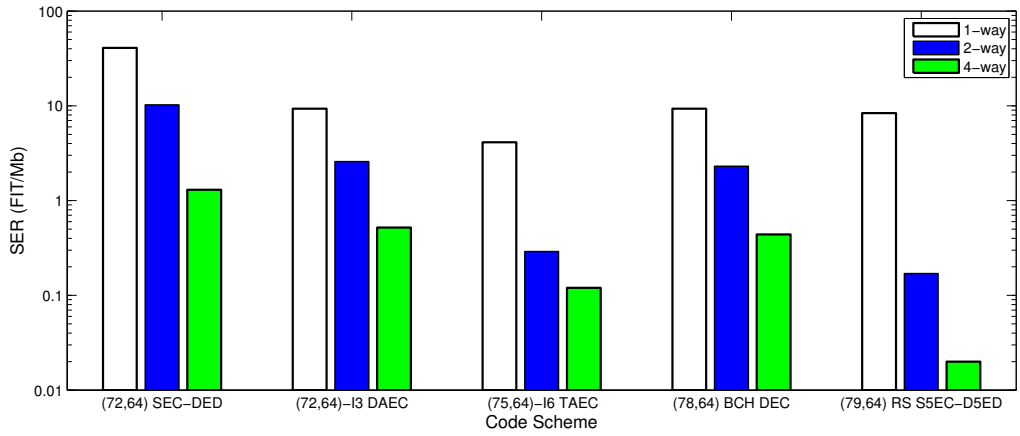


Figure 4.6: Simulated corrected-SER as a function of word interleaving for various ECC schemes at a raw-SER of 1200 FIT/Mb.

4.2 Test Chip Design

A test chip has been fabricated in a low power, 28 nm HK+MG bulk CMOS process using a full-custom design flow. The design includes a 75 kb 6T SRAM macro protected using a (75, 64)-I_6 SEC-DED-TAEC-8AED code based on the theory presented in Chapter 3 and leverages the MCU shaping strategies discussed in Section 4.2.3. A block diagram of the circuit is shown in Figure 4.8. The ECC decoder and column multiplexing circuits have been modified such that the degree of adjacent error correction, C, and interleaving, I, are programmable between 0- (off), 1- (SEC), 2- (DAEC), and 3- (TAEC) bits, and 1-, 2-, and 4-way interleaving. The maximum MCU width that can be completely corrected by the memory is the selected interleaving-correction product, $I \cdot C$. Enabling the TAEC feature in conjunction with the 4-way interleaving scheme allows for the correction of up to 12 adjacent-bits. The test chip features seven operating modes listed in Table 4.4. The circuit operating mode is determined by the Read/Write (R/W) and ECC input control signals. The Read and Write Bypass modes allow the SRAM read and write operations to be tested independent of the ECC, while the ECC Test Decode and Encode modes allow for isolated testing of the ECC circuit independent of the memory. These modes have been included for the initial chip ramp up testing. The Read ECC, Read ECC with Writeback, and Write ECC operating modes are for full-functionality operation, using the memory and ECC together. The Writeback feature allows read data to be written back into the SRAM array following an error correction operation. This can be used to prevent the accumulation of bit errors over time. Further, the chip includes two timing configurations, one for high-speed testing up to 700 MHz at 1 V, and one for low-speed testing down to

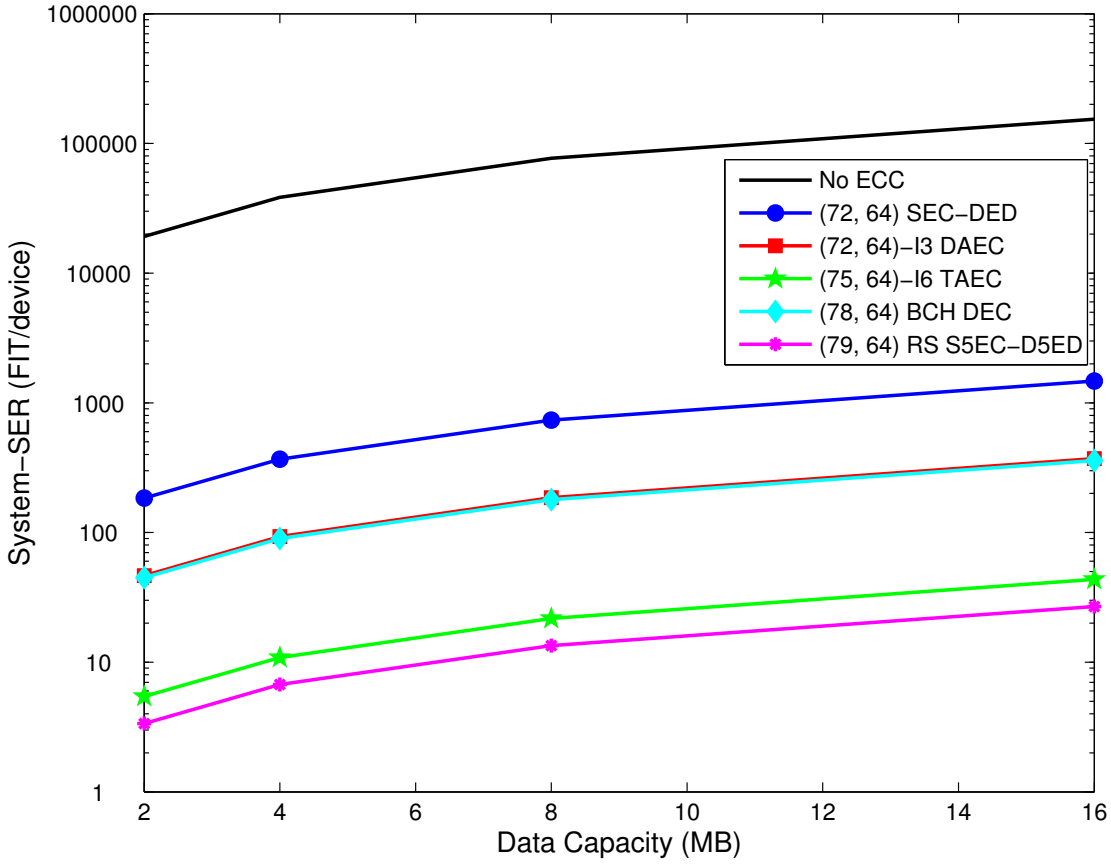


Figure 4.7: System level SER/device for 2 to 16 MB data capacity for various ECC schemes at a raw-SER of 1200 FIT/Mb using 2-way interleaving.

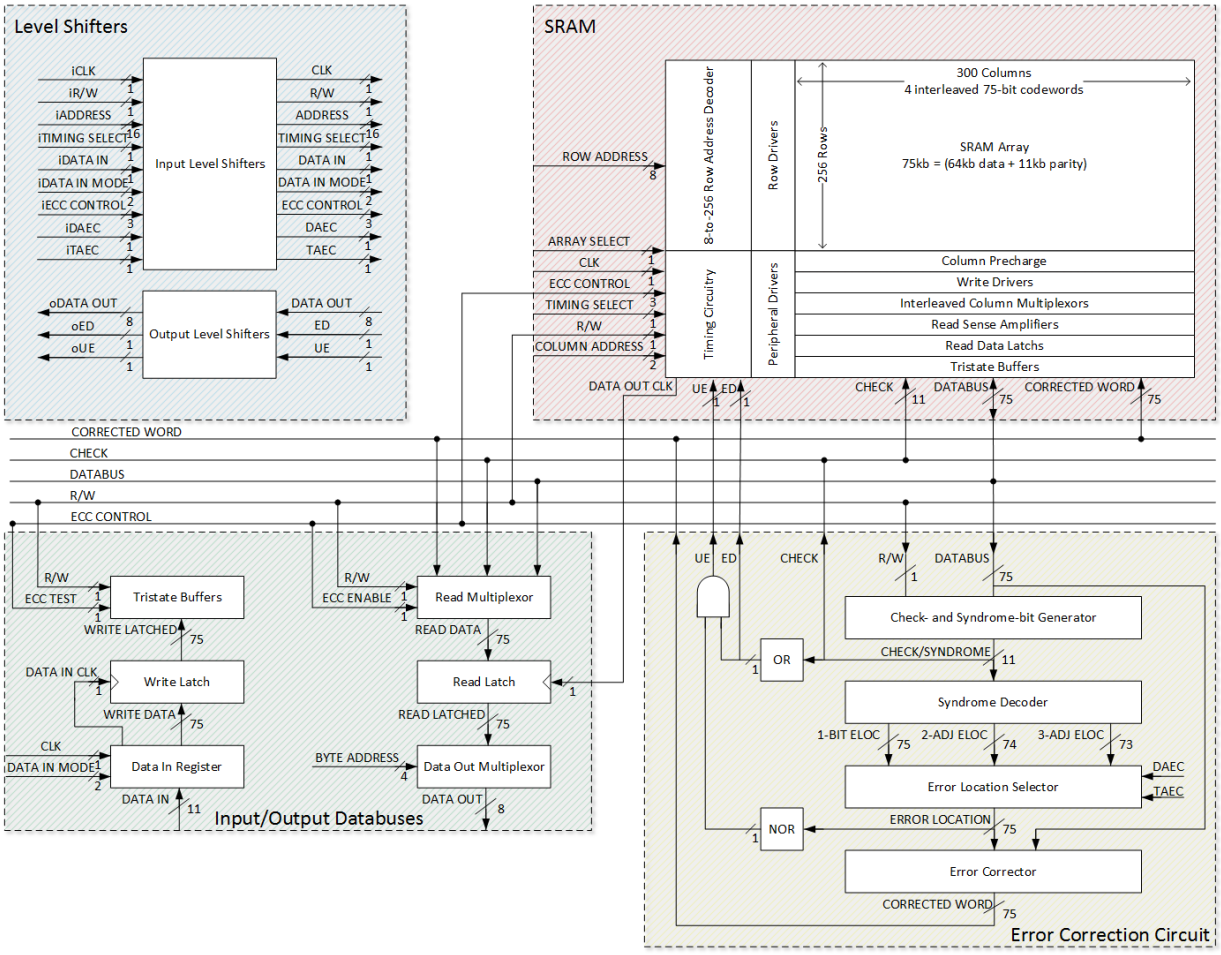


Figure 4.8: 6T SRAM macro and error correction circuit block diagram with input/output databus interface and voltage domain level shifters.

Table 4.4: Test Chip Operating Modes

Mode	Input Signals				Mode Description
	R/W	ECC Test	ECC Enable	ECC Writeback	
Read Bypass	1	0	0	X	Memory read test, ECC disabled
Write Bypass	0	0	0	X	Memory write test, ECC disabled
ECC Test Decode	1	1	1	X	ECC decoder test, memory disabled
ECC Test Encode	0	1	1	X	ECC encoder test, memory disabled
Read ECC	1	0	1	0	Read/decode operation
Read ECC with Writeback	1	0	1	1	Read/decode operation writeback to array
Write ECC	0	0	1	X	Write/encode operation

1 - On, 0 - Off, X - Don't Care

920 kHz at 400 mV.

To elicit the largest possible number of multi-cell upsets, the circuit has been implemented in a commercial low power, 28 nm bulk CMOS process. At the time of design, this was the most advanced technology available to Canadian academia through the Canadian Microelectronics Corporation (CMC) [70]. To the best of our knowledge, no other Canadian university had fabricated in this technology for purposes other than initial kit characterization. This posed a number of unique challenges, but was necessary to design the smallest bitcell possible.

4.2.1 Memory Organization

The memory, shown in Figure 4.9, has been designed as a single 75 kb SRAM macro (64 kb data, 11 kb parity) organized in a 256 row by 300 column bitcell arrangement. The memory capacity was constrained by the 1.23 mm x 1.23 mm die area allotment by CMC, and the inclusion of another project on the chip. Increasing the memory capacity would allow for more radiation data at the increased monetary cost for additional die area. The macro block can be tiled to facilitate a rapid increase in memory capacity. A 64 data-bit word size was chosen to allow for single cycle memory access if the memory were to be implemented in a 64-bit processor. Recently, 64 data-bit processors have been emerging in the mobile market space [71]. Four codewords have been placed per memory row, and can be logically accessed to achieve 1-, 2-, or 4-way interleaving. This has been done for comparison purposes. Each arrangement is shown in Figure 4.10 where each square represents a bitcell in the memory row, and 'W' and 'B' are used to indicate the word and bit address information for each bitcell. Combined with the 11 ECC check-bits, each 64-bit dataword consists of 75 code-bits, and hence arranging four codewords per row yields the 300-bit row size. Limiting the number of rows to 256 entries allows the array to be accessed using an 8-bit row address decoder and maintains read and write memory

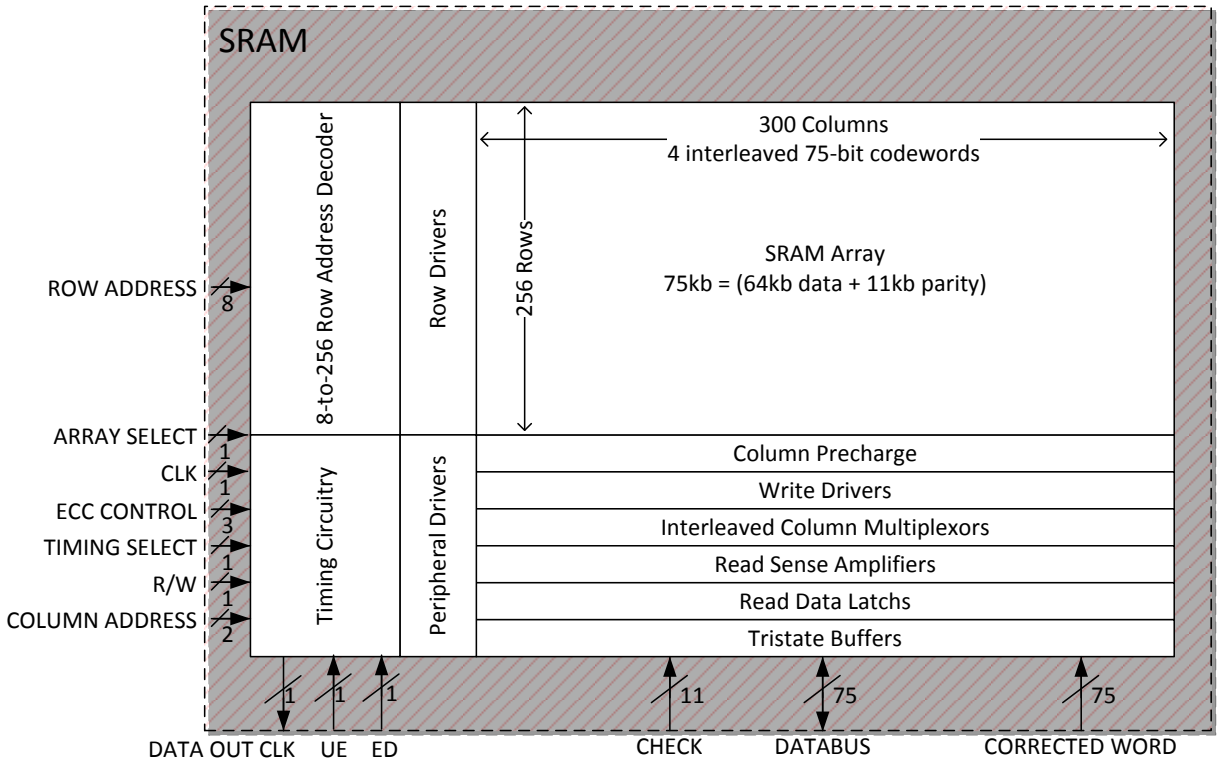
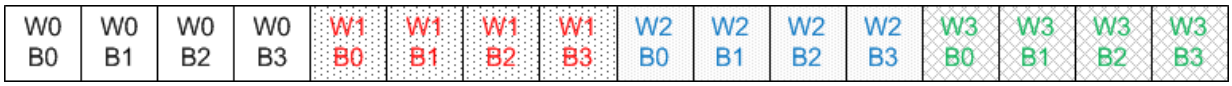


Figure 4.9: SRAM Macro Block Diagram

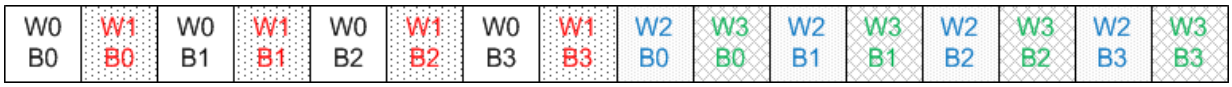
operations above 1 GHz at the nominal 1 V supply voltage. Following the original circuit design for 1.25 GHz operation, additional timing guard bands were added limiting the final nominal supply voltage operating speed to 700 MHz. This was done to ensure functional yield since this was the first time fabricating using this design process technology.

4.2.2 Memory Address Space

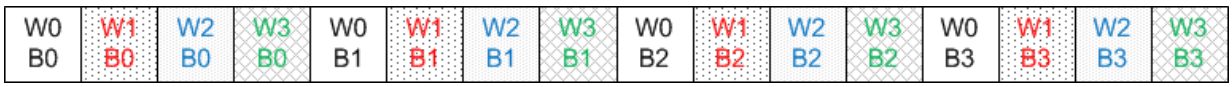
The implemented test chip uses a 16-bit memory address space shared with another memory design on the chip. As shown in Table 4.5, the address space has been partitioned into four main sections: array select (2 bits), row select (8 bits), word select (2 bits), and byte select (4 bits). The address signals are available to both memories at all times; however, only one array is accessed at a time. The array select signals act as a one-hot encoding to control which memory array on the test chip is being accessed and has control of the shared output data bus. When the address bits A(15:14) = '01', the memory array discussed in this thesis is active and controls the output data bus. The row select signals, A(13:6),



(a) 1-way



(b) 2-way



(c) 4-way

Figure 4.10: Row organization for 1-, 2-, and 4-way interleaving.

Table 4.5: Test Chip 16-bit Memory Address Space

A(15:14)			A(13:6)		A(5:4)			A(3:0)	
Array Select			Row Select		Word Select			Byte Select	
2 Arrays			256 Rows		4 Words			10 Bytes	
A(15)	A(14)	Select	MSB A(13) .. LSB A(6)		A(5)	A(4)	Select	MSB A(3) .. LSB A(0)	
0	0	-	0000 0000	Row 0	0	0	Word 0	0000 .. 0111	Byte 0-7
0	1	Yes	0000 0001	Row 1	0	1	Word 1	1xx0	Byte 8
1	0	-	1111 1110	Row 254	1	0	Word 2	1xx1	Byte 0
1	1	-	1111 1111	Row 255	1	1	Word 3	-	

provide the 8-bit input address to the 8-to-256 row decoder to select which of the 256 rows in the memory array to access. The word select signals, A(5:4), determine which of the four 75-bit memory words is accessed per 300-bit row selected by the row address decoder. Finally, since the chip is a pin-limited design (as can be seen in Figure 4.28, where the 120-pin pad frame complete with I/O ring takes up a considerable amount of the allocated die area) it was not feasible for each bit on the 75-bit output data bus to have a dedicated output pin. As such, the 75-bit output data had to be muxed-down to an 8-bit bus using a set of four byte select control signals, for a total of 12 pins (8 data + 4 control). The byte select control signals, A(3:0), make up the four least most significant bits in the memory's address space. Under this output bus sizing constraint, a complete 75-bit word can still be accessed within a single cycle operation; however, to retrieve the 75 bits of read data from the array's output data bus requires 10 clock cycles for the data to be read one 8-bit byte at a time on the chip-level output data bus.

The partitioning of the 75-bit codewords into 8-bit bytes is shown in Table 4.6. Notice that for Byte 9, only the 3 least significant bits contain codeword data, and the remaining 5 more significant bits have been hardcoded to return zero. Further, since 75 does not fit evenly within a power of two boundary, when cycling over the four byte select address

Table 4.6: 8-bit Byte Divisions for 75-bit Codewords

Byte 9	Byte 8	Byte 7	Byte 6	Byte 5	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0
00000 + Bit(74:72)	Bit(71:64)	Bit(63:56)	Bit(55:48)	Bit(47:40)	Bit(39:32)	Bit(31:24)	Bit(23:16)	Bit(15:8)	Bit(7:0)
75-bit codeword = 1 3-bit byte + 9 8-bit bytes									

space from A(3:0) = (1010) to (1111), these bytes alternate between the data being stored in Byte 8 and Byte 9 for A(0) = 0 and 1 respectively. This issue could be resolved by placing only the 64 data-bits on the output data bus and using only 3 bits for byte select (2^3 bytes $\times$ 8 bits/byte = 64 bits); however, for functional test purposes retrieving the codeword's check- and syndrome-bits is vital. For practical lab measurements, the entire address space is cycled and any redundant or unused address locations are pruned during post processing. Finally, this bus muxing architecture is only necessary for functional chip verification. If used in an embedded environment, the entire 75-bit codeword could use a complete 75-bit internal data bus for routing data within a single cycle.

4.2.3 Memory Bitcell

Modern process development kits include reduced geometry SRAM design rules and vendor supplied bitcells for improving the SRAM bitcell density beyond that possible using standard logic design rules. The technology we were using had these cells available for pre-layout, schematic based simulations, but prohibitive additional cost and licensing prevented their use for post-layout extracted netlists and fabrication. The implemented SRAM bitcell was therefore designed using logic design rules and made to mimic the stability performance of the vendor supplied dense cell (smallest area) configuration. This came at the cost of additional bitcell area and cell leakage current. Figure 4.11 provides a comparison between the vendor supplied dense, high speed (higher read current), and high performance (very high read current) cells and two cells designed using the logic design rules. All simulations were performed under the worst case process corner for the given measurement, nominal supply voltage (1.0 V), and nominal operating temperature (27 ^{o}C). All results have been normalized to the vendor dense cell values. One cell was designed as a proportionally scaled-up version of the dense cell configuration such that the minimum logic design rule geometry was used for the minimum feature size in the vendor supplied dense cell. This configuration requires a 2.47x increase in estimated cell area and 70% additional leakage current over the dense cell configuration. Since these are pre-layout implementations, the area estimate is the sum of each cell's individual transistor $W \times L$ products. The bitcell implemented in the test chip was sized in an effort to match the stability margins (read static noise margin, and write margin) for the dense cell, while requiring less area than the scaled-up bitcell design. Final bitcell sizing is shown in Table 4.7. The implemented bitcell's stability metrics are within 5% of the vendor supplied dense cell configuration,

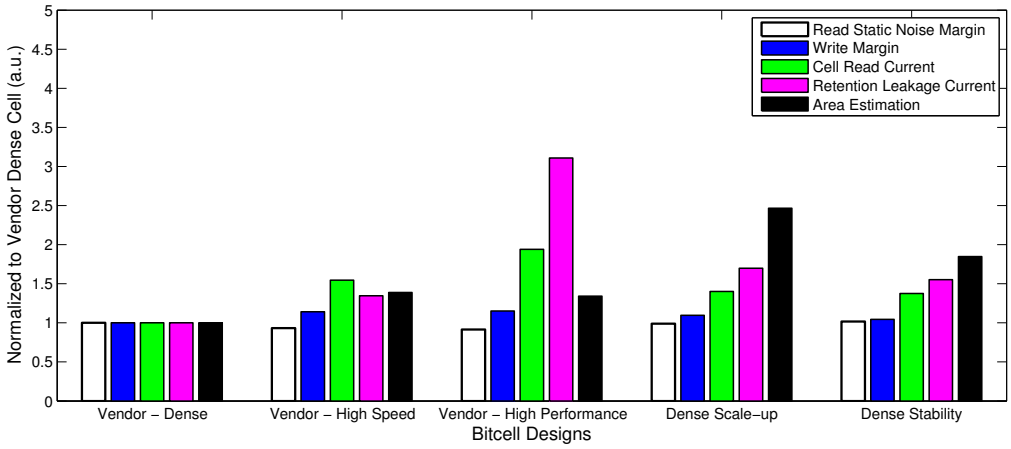


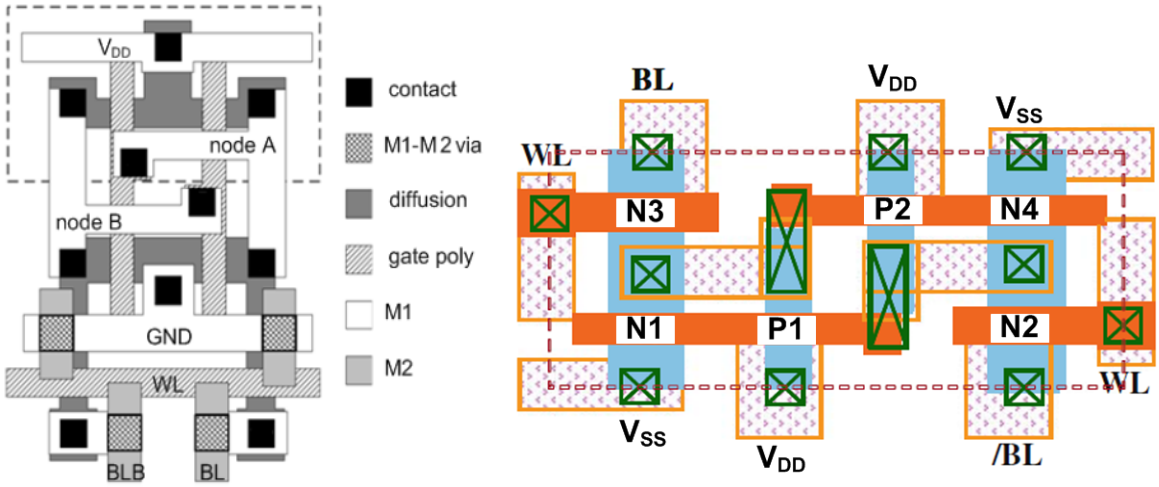
Figure 4.11: SRAM bitcell design comparison with vendor supplied bitcells

Table 4.7: Implemented 6T Bitcell Sizing

Device Pair	Width (nm)	Length (nm)
Access Transistors	115	45
Driver Transistors	165	45
Load Transistors	80	77

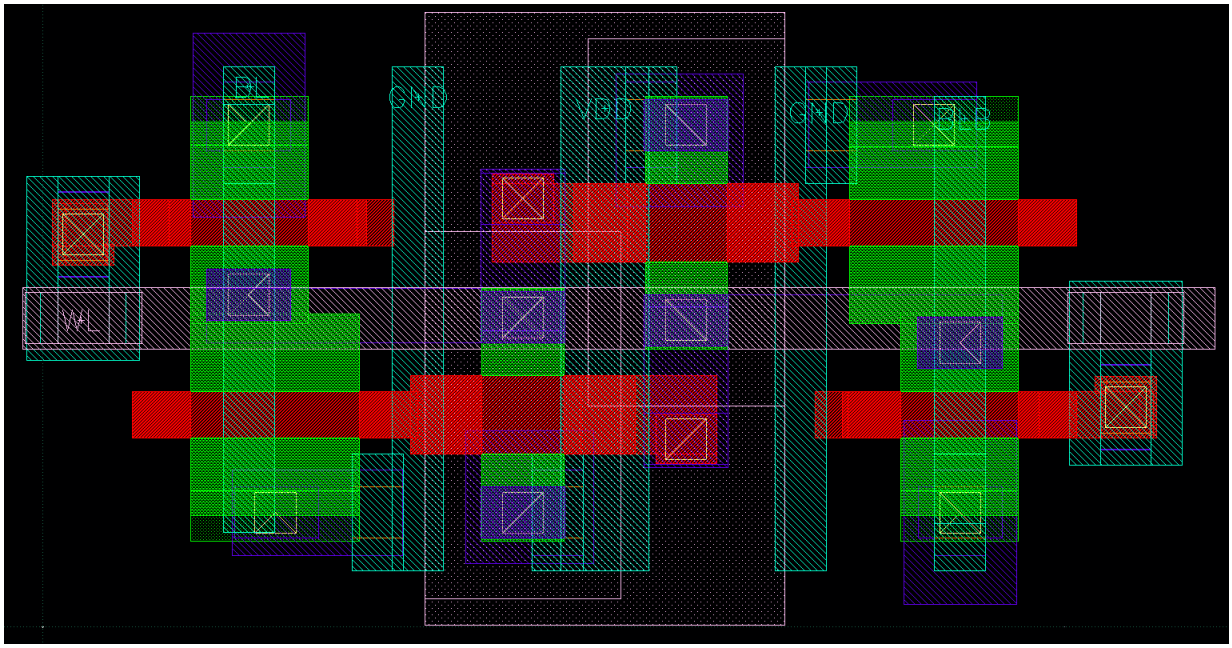
and come at a cost of 84.8% additional estimated bitcell area and 55.3% additional cell leakage current. Actual implemented layout area is considered in Section 4.5.3.

To help limit the number of MBUs caused by MCU_{WL} and $\text{MCU}_{Cluster}$, the 6T bitcell has been designed using a wide cell layout as opposed to a tall cell configuration. These two layout configurations are shown in Figure 4.12. Not only does this reduce the height of the bitline columns, leading to lower bitline capacitance and in turn better performance, but it increases the physical distance between adjacent bitcells in each row [72]. Using this topology, the bitcell wells and $\text{V}_{DD}/\text{V}_{SS}$ supply rails are also run vertically down each column. Since charge has a greater tendency to stay within a well or substrate boundary, the probability of upsetting row-wise adjacent cells is limited [73]. Although using these techniques do increase the MCU_{BL} probability, the upset cells reside in different words, and can thus be corrected via ECC. The effect of this MCU shaping will be seen in the radiation data presented in Section 4.4.2. The implemented SRAM bitcell is shown in Figure 4.12(c), and its dimensions are 1.02 μm by 0.379 μm giving a bitcell area of 0.387 μm^2. This bitcell area is between those of industry reported, vendor supplied cells in 65 nm (0.495 μm^2 [74]) and 45 nm (0.315 μm^2 [75]) technologies using the SRAM design rules.



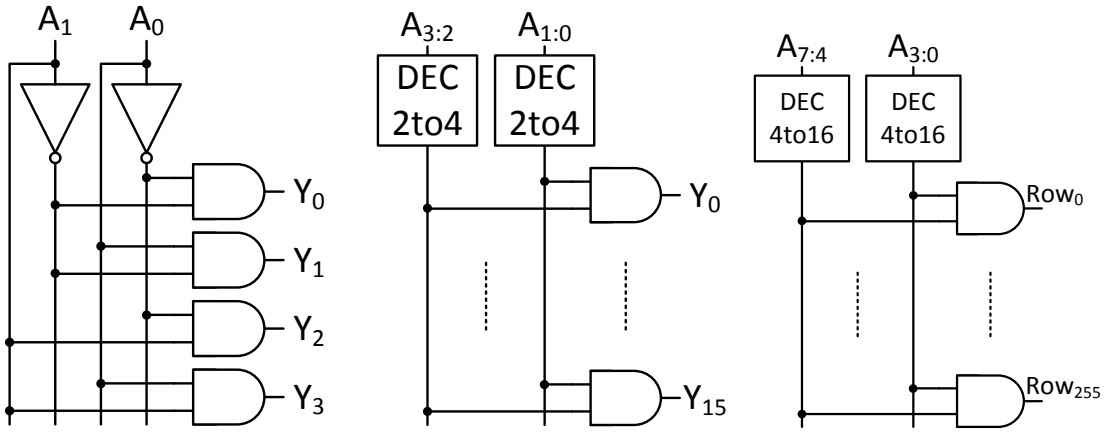
(a) Tall bitcell configuration [30]

(b) Wide bitcell configuration [13]



(c) Implemented wide bitcell

Figure 4.12: SRAM bitcell layout configurations



(a) 2-to-4 bit Decoder (b) 4-to-16 bit Decoder (c) 8-to-256 bit Decoder

Figure 4.13: Hierarchical 2-input AND Gate-based 8-to-256 Address Decoder Unit

4.2.4 Row Conditioning Circuitry

The row conditioning circuitry consists of an 8-to-256 address decoder and a series of wordline drivers for each row in the array. The address decoder is responsible for functionally selecting one of the 256 rows in the SRAM array based on the 8-bit input row address, while the wordline driver serves two purposes, it: 1). gates the row signal such that the row's wordline is only accessed when necessary based on the generated timing signals, and 2). provides a driver such that the functional row signal can drive the large wordline load. The decoder is implemented using a conventional, hierarchical pre- and post-decode architecture, logarithmically dividing the input address. The 8-to-256 decoder is hierarchically constructed from smaller decoder units by successively dividing the address equally in half and using a series of 2-input AND gates act as gaters for subportions of the full address space. This is shown in Figure 4.13.

The 2-to-4 bit address decoder unit, shown in Figure 4.13(a), takes a two bit binary address (A_1A_0) and selects the appropriate output signal ($Y_{3..0}$). Two of these units can the be used as a pre-decoder unit for a 2-input AND gate plane post-decoder, shown in Figure 4.13(b), to create a 4-to-16 bit address decoder. This process is repeated using two 4-to-16 bit decoder units as the pre-decoder for the final 8-to-256 bit address decoder. In this implementation, AND gates are used rather than a series of NAND/NOR gates in order to drive the large intermediate loads presented from long metal lines and large capacitive fanout per signal. For example, each of the 32 pre-decoder outputs in the 8-to-256 decoder has to drive 16 AND gates in the post-decoder as well as a long interconnect spanning the full height of the decoder (approximately equal to the 256 row height of the

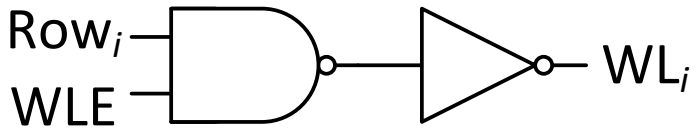


Figure 4.14: Wordline Driver Circuit

SRAM macro array).

The address decode process has been designed to be performed in parallel with the bitline precharging phase of the cell access operation. This reduces overall latency of the system, but requires the row select signal to be gated since its contamination delay may be less than the bitline precharge duration. The wordline driver, shown in Figure 4.14, serves two functions: it provides a drive buffer for the row select signal to drive the large capacitance on the wordlines, and it acts as a gating element for the row select signal. The wordline control signal is gated by the wordline enable signal (WLE) generated by the timing control block. The wordline is thus only high if both its corresponding row select signal and the WLE signal is high. This prevents the unintentional activation of the WL signal during other operating phases (e.g., bitline precharge, bitline discharge for write, etc...).

If a correctable error is detected by the error correction circuit and the ECC writeback functionality is enabled, the wordline signal will be activated and deactivated twice during a read operation (once during the initial read, then again during the writeback operation). Since data will be written back to the same address that it was just read from, the address does not need to be recalculated, but rather the WLE signal can just be reactivated allowing the row signal to pass through to the wordline. This results in a reduction in switching activity in the address decoder, and in turn a reduction in switching power.

4.2.5 Column Conditioning Circuitry

While the row conditioning circuitry is responsible for selecting the appropriate row for data access, the column conditioning circuitry is responsible for preparing and performing the bitline functionality for both read and write operations. This includes: precharging the bitlines to the supply voltage prior to each read or write operation, applying the appropriate data to the bitlines prior to a write operation, and sensing a sufficiently resolvable differential voltage during a read operation. Further, by storing more than one memory word per row, the x-y memory array aspect ratio can be made closer to unity (more square). This allows for data to be stored across more columns using fewer cells per column. This allows for a significant reduction in the capacitance per bitline, significantly improving

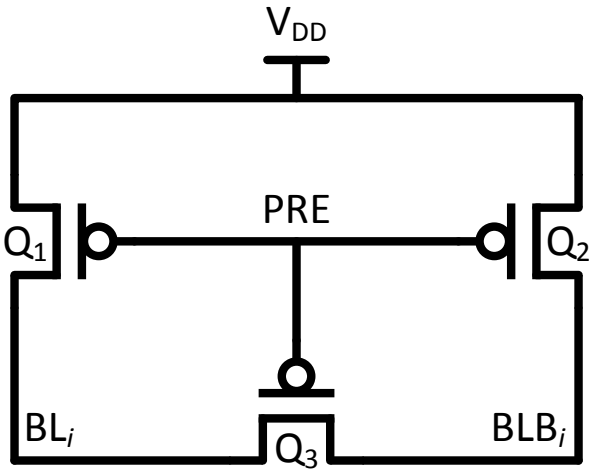


Figure 4.15: Precharge and Equalize Circuit

read access times where bitlines are being discharged through near minimum sized bitcells, at the expense of a larger wordline capacitance which may be overcome using a larger peripheral wordline driver circuit. By storing multiple memory words per row, additional word selection column multiplexer circuitry is included per column. In this section, each of the column conditioning circuits are considered in turn.

Precharge and Equalize

One precharge and equalize circuit, shown in Figure 4.15, is placed within each of the 300 columns within the implemented SRAM array. When the precharge control signal is low, both bitlines are charged up to the supply voltage V_{DD}. At the same time, transistor Q3 equalizes the potential of the two bitlines. The equalize device serves two purposes. First, in the even that one of the bitlines is already high while the other bitline is low (e.g., after a write operation) it provides a second charging path to bring the low bitline up to V_{DD} by sharing charge between the two bitlines. This allows for faster precharge times. Second, by equalizing each bitline pair, any precharge voltage mismatch due to process variability can be eliminated, removing one more potential source of offset voltage necessary to be overcome by the read bitcell.

Column Multiplexer

In the implemented SRAM macro four codewords are stored per memory row. Since only one word is accessed at any given time, the read/write circuitry can be shared across four columns of data. To facilitate the selection of which column should be passed to the read

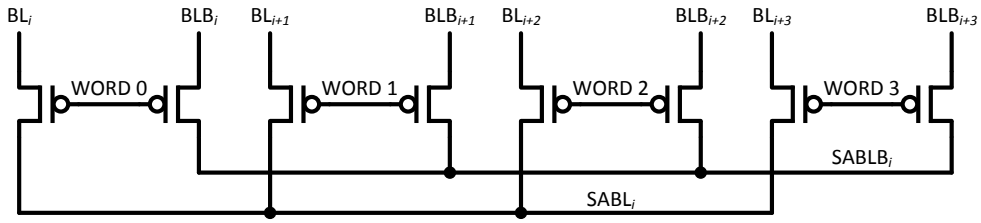


Figure 4.16: 4-to-1 Differential Column Multiplexer Circuit

circuitry (or which 'way' the data comes from), four columns worth of bitline pairs are passed to a differential multiplexer, shown in Figure 4.16, and one is fed through to a sense amplifier circuit based on the state of the input word address. The two bit word address is decoded to a one-hot-of-four, active-low, select signal using a 2-to-4 bit address decoder. Since during a read operation, one of the bitlines is at V_{DD} while the other bitlines is at V_{DD} minus some small delta, both bitlines are near V_{DD} and as such a single PMOS pass transistor can be used as the gating element for each bitline. By gating the column select signal, the column multiplexer can also acts as an isolation circuit between the bitline and read circuitry. A write driver circuit is used to perform the four to one column selection function for the write circuitry.

Write Driver

The write driver's basic functionality is to selectively discharge one of the bitlines per column based on the input data to condition the bitlines for a write operation. To reduce discharge time, discharging can be performed through a single large NMOS transistor with the control circuitry driving the transistor's gate. The write driver circuit implemented in the test chip is shown in Figure 4.17. Similar to the column multiplexer, a one-hot column select signal (active high) is used to only discharge the bitlines for the columns of the codeword being written. This signal is gated with a write enable (WE) signal generated by the timing control block such that bitlines are only discharged during the proper phase of the access operation. Finally, to facilitate the ECC writeback functionality, a control signal is used to choose the data written by the write driver. For a standard write operation, data is taken from the input data bus, while for a writeback operation following an error correction, the data comes from the ECC's corrected output data bus.

Sense Amplifier and Read Data Latch

Sense amplifiers are an important component within the SRAM memory macro unit. The primary function of the sense amplifier is to amplify the small analog differential voltage,

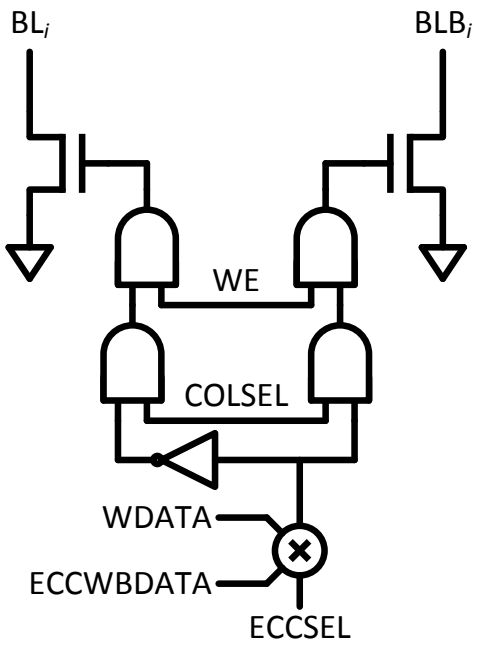


Figure 4.17: Write Driver Circuit

ΔV_{BL}, developed across the highly capacitive bitlines during a read-access operation to a full swing digital output signal. The time required to fully discharge the large capacitance on the bitlines themselves through the near minimum sized bitcells is extremely prohibitive to high speed operation, and as such a sense amplifier is employed to accelerate the read operation. A current latch-based sense amplifier, shown in Figure 4.18, has been implemented for each of the 75 codeword bits within the SRAM macro. The current latch-based sense amplifier was chosen due to its favorably low input offset variability, $\sigma_{SA_{offset}}$, at low supply voltage compared to other sense amplifier topologies. In this configuration, the input bitlines are decoupled from the digital outputs, preventing the need for the amplifier to assist in fully discharging one of the bitlines. This serves two benefits: first, a speed improvement is realized by only needing to discharge a much smaller output capacitance with the assistance of amplifier's gain, and second a power saving is realized by not completely discharging large bitline capacitance.

The circuit works by precharging the internal Q and Qb nodes of the bistable storage element to V_{DD}. This forces the Q and Qb nodes into a metastable state, where any slight deviation from this point will allow the forward feedback action of the cell to rapidly transition it into a bi-stable state. By enabling the sense amplifier through the SAE signal, and applying the differentially sensed bitline voltage, ΔV_{BL}, across the gates of the input transistors, the device with the higher voltage input (V_{DD}) will discharge its internal node

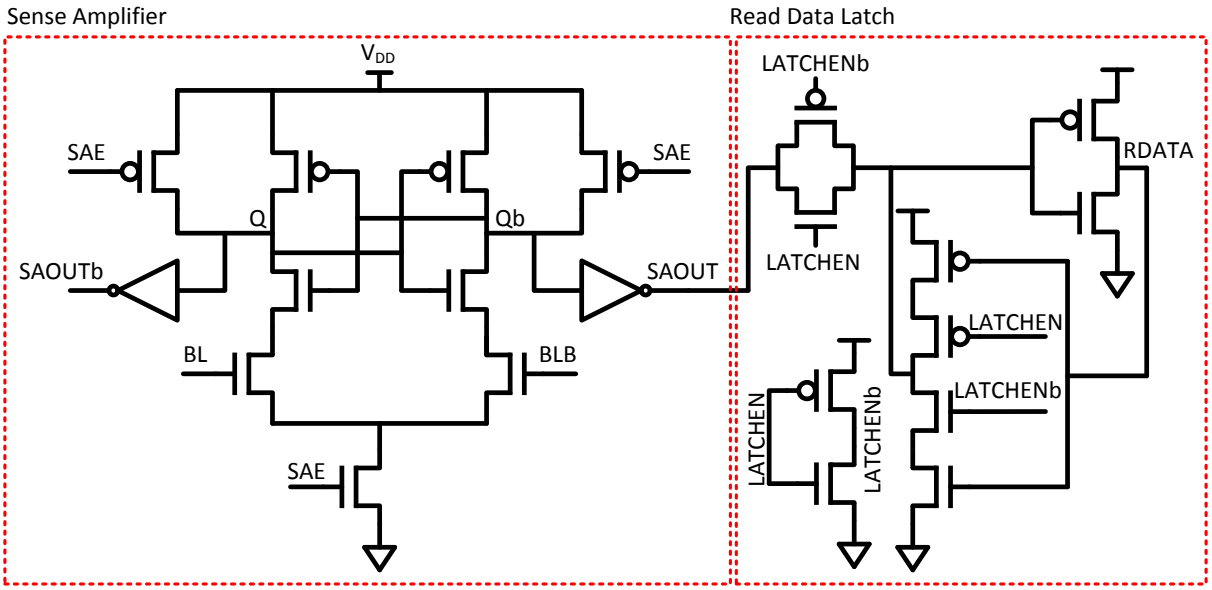


Figure 4.18: Current Latch-Based Sense Amplifier Circuit with Read Data Latch

more quickly than the side controlled by the lower input voltage ($V_{DD} - \Delta V$). This will drive the storage element to a set of full swing outputs. The full swing output value is then passed to a latching element. This latch is transparent when the sense amplifier is enabled allowing it to sample the new data. The latch holds its data when the sense amplifier is disabled, thus the read output data is held within the latching element until new data is read.

Each sense amplifier is shared via the column multiplexer circuitry across four columns. This relaxes the area constraint of the sense amplifier allowing it to be fit within the column-pitch of four bitcells rather than constrained within the width of a single cell. In doing so the input offset voltage variability, $\sigma_{SA_{offset}}$, can be minimized since the transistor threshold variability $\sigma_{V_{th}}$ is proportional to $1\sqrt{W \cdot L}$. By having a larger area footprint in which to layout the cell, larger devices with lower variability can be used, and hence reduce the variability in sense amplifier input offset voltage needed to be overcome.

4.2.6 ECC Circuit

Rather than implementing multiple error correcting circuits, increasing circuit complexity and requiring additional die area, a modified version of the (75, 64) $\mathbf{I}_6$ SEC-DED-TAEC-8AED code was implemented for on-chip error correction and detection. The check- and syndrome-bit generation XOR logic equations are dictated by the **H**-matrix in Fig-

$$\mathbf{H} = \begin{pmatrix}
000000\ 000000\ 010101\ 000000\ 101010\ 000000\ 000000\ 000000\ 111111\ 010101\ 000000\ 101010\ 111 \\
010101\ 010101\ 010101\ 000000\ 010101\ 101010\ 101010\ 000000\ 000000\ 101010\ 000000\ 101010\ 000 \\
000000\ 010101\ 000000\ 101010\ 000000\ 010101\ 111111\ 101010\ 101010\ 010101\ 000000\ 000000\ 000 \\
000000\ 101010\ 000000\ 101010\ 000000\ 000000\ 010101\ 010101\ 000000\ 000000\ 101010\ 010101\ 000 \\
010101\ 000000\ 101010\ 010101\ 000000\ 000000\ 000000\ 000000\ 000000\ 101010\ 101010\ 010101\ 111 \\
\\
100000\ 100000\ 100000\ 100000\ 100000\ 100000\ 100000\ 100000\ 100000\ 100000\ 100000\ 100000\ 100 \\
010000\ 010000\ 010000\ 010000\ 010000\ 010000\ 010000\ 010000\ 010000\ 010000\ 010000\ 010000\ 010 \\
001000\ 001000\ 001000\ 001000\ 001000\ 001000\ 001000\ 001000\ 001000\ 001000\ 001000\ 001000\ 001 \\
000100\ 000100\ 000100\ 000100\ 000100\ 000100\ 000100\ 000100\ 000100\ 000100\ 000100\ 000100\ 000 \\
000010\ 000010\ 000010\ 000010\ 000010\ 000010\ 000010\ 000010\ 000010\ 000010\ 000010\ 000010\ 000 \\
000001\ 000001\ 000001\ 000001\ 000001\ 000001\ 000001\ 000001\ 000001\ 000001\ 000001\ 000001\ 000
\end{pmatrix}$$

Figure 4.19: Implemented (75, 64) $\mathbf{I}_6$ SEC-DED-TAEC-8AED Code **H**-matrix

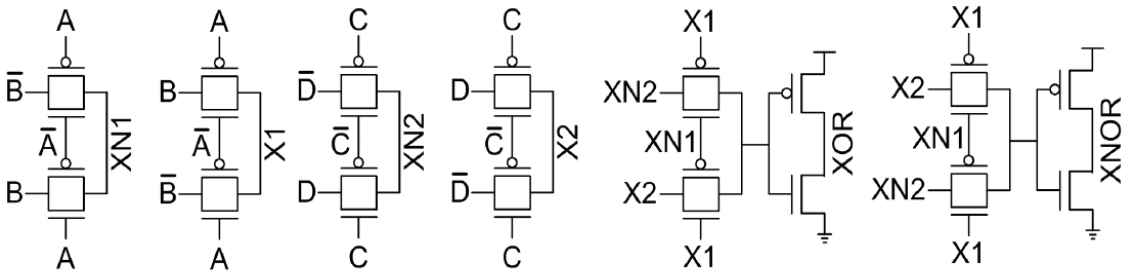
Table 4.8: Programmable Error Correction Controller

Input Signals			
ECC Enable	DAEC	TAEC	Error Correction Enabled
0	X	X	None, ECC disabled
1	0	0	Single
1	1	0	Double adjacent
1	0	1	Don't use case
1	1	1	Triple adjacent

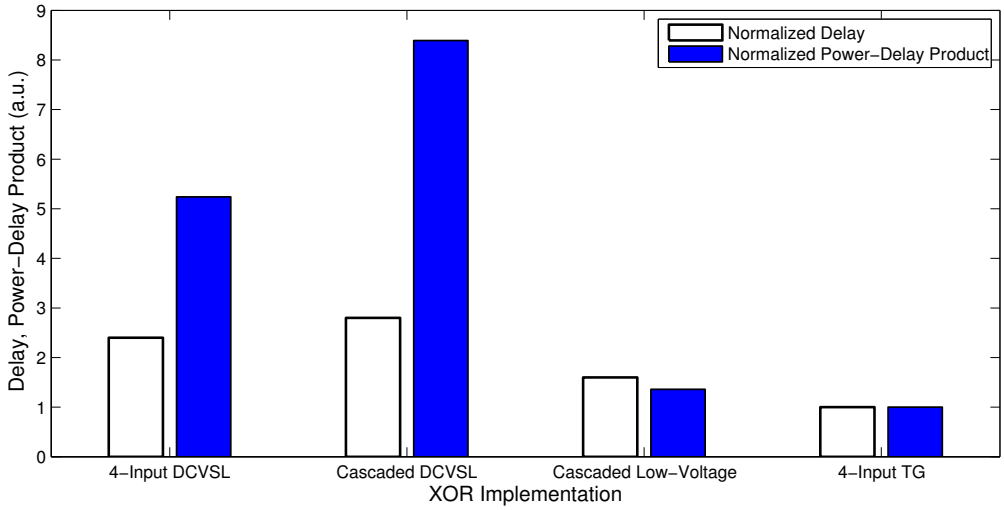
1 - On, 0 - Off, X - Don't Care

ure 4.19 and the XOR logic gates have been implemented using fully-differential, 4-input transmission-gate (TG) based XOR's optimized for power-delay product. The circuit and a comparison with other gates is shown in Figure 4.20. The circuit has been designed for low voltage functional operation down to 300 mV and shows faster propagation delay and lower power-delay product compared to a 4-input differential cascode voltage switch logic (DCVSL) XOR, cascaded 2-input DCVSL XORs, and the low voltage/low delay XOR gate in [76].

The syndrome decoder has been modified to use different sets of syndrome vectors based on the configuration of a set of error correction control signals. Based on the control signal settings, shown in Table 4.8, the decodable syndrome vectors, and in turn the degree of provided error correction can be programmed. This allows a variable degree of error correction capabilities to be compared within a single circuit test vehicle. The syndrome decoder consists of an 11-to-222 decoder, similar to an address decoder, cascaded with the syndrome selector circuit. The decoder takes as input the 11 calculated syndrome-bits and determines which, if any, of the 222 distinct correctable error patterns have occurred. These outputs are fed into the programmable syndrome selector circuit. Each codeword bit can be involved in three distinct triple adjacent bit error patterns, two double adjacent bit error patterns, and one single bit error pattern. These bit error pattern decoded syndromes are gated by the DAEC and TAEC control signals as shown in the selector circuit bit-slice in Figure 4.21. The selector gates the multiple adjacent bit error decoded syndromes and then, for each codeword bit location, an OR gate is used to determine if the particular bit



(a) Implemented 4-input transmission-gate based XOR schematic



(b) Comparison

Figure 4.20: 4-input transmission-gate based XOR circuit schematic and XOR gate comparision.

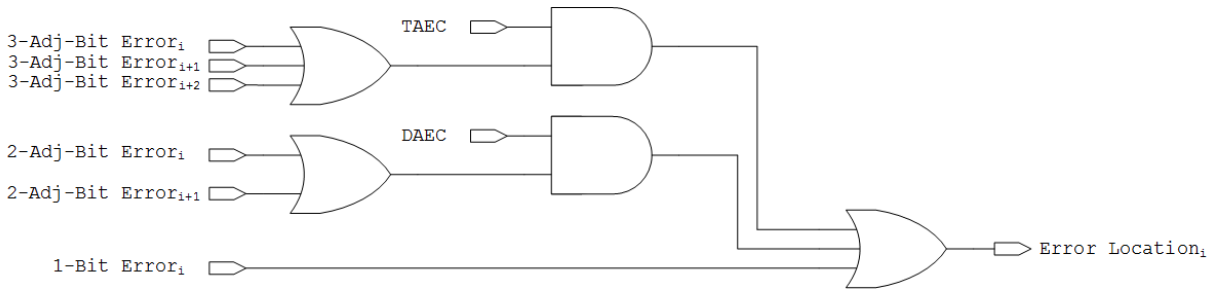


Figure 4.21: Programmable Syndrome Decoder Selector Logic Bit-slice

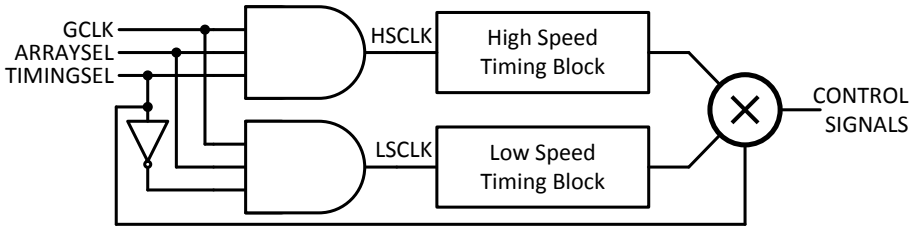


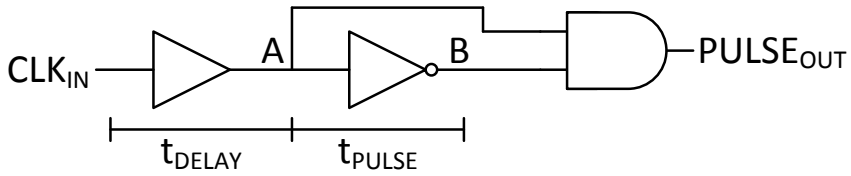
Figure 4.22: Timing Block Select Circuitry

has been involved in any of the three error types. Once the bit error locations have been established, the error corrector circuit performs a bitwise XOR operation on the original 75-bit read codeword and the error location vector produced by the syndrome decoder. This process is similar to that shown in Figure 1.7. The error corrector circuit has been implemented using 75 2-input transmission gate based XOR gates. The corrected codeword output is forwarded to the output databus where it can optionally be written back to the memory array. Writing the data back to the memory overwrites any corrupted data from the processed codeword.

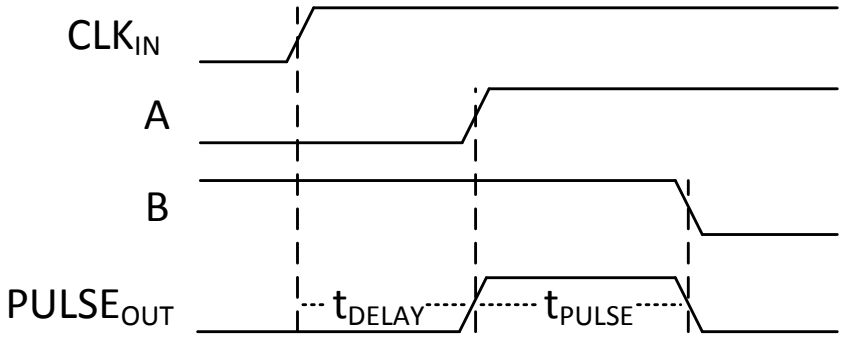
4.2.7 Timing and Control Circuitry

The intended purpose of the designed memory system is for embedded dynamic voltage and frequency scaling (DVFS) applications in which operating frequencies could potentially be in the gigahertz range. Externally generating these signal speeds for circuit board-level testing is however infeasible barring very sophisticated test equipment. As such, the memory system has been designed using an edge-triggered, self-timed timing mechanism for generating all of the memory circuit's internal timing signals based solely on the rising edge of the input clock and the functional operating mode signals. Both a high and low speed timing block have been included in the design. This is shown in Figure 4.22. The high-speed block is intended for nominal supply voltage (1.0 V), full-speed operation, while the low-speed block has been included for functional test purposes and low voltage (400 mV) operation. The selection between timing signals is determined by an external timing select (TIMINGSEL) signal, while the entire timing block unit is gated by an array select (ARRAYSEL) signal. The rising edge of the global clock signal (GCLK) is used to initiate the sequence of timing events given that the ARRAYSEL signal is true.

The timing control circuit is responsible for generating the memory array's row and column conditioning signaling. For the row circuitry, this includes the wordline enable (WLE) signal while for the column circuitry this includes the: column and sense amplifier precharge signals (PRE and SAPRE), write enable (WRITEEN), sense amplifier enable (SAE), and sense amplifier latch enable (LATCHEN) signals. The sequencing of each



(a) Schematic



(b) Timing Diagram

Figure 4.23: An inverter-delay-line-based pulse generator is used for generating internal timing signals based solely on the rising edge transition of the input clock signal (CLK).

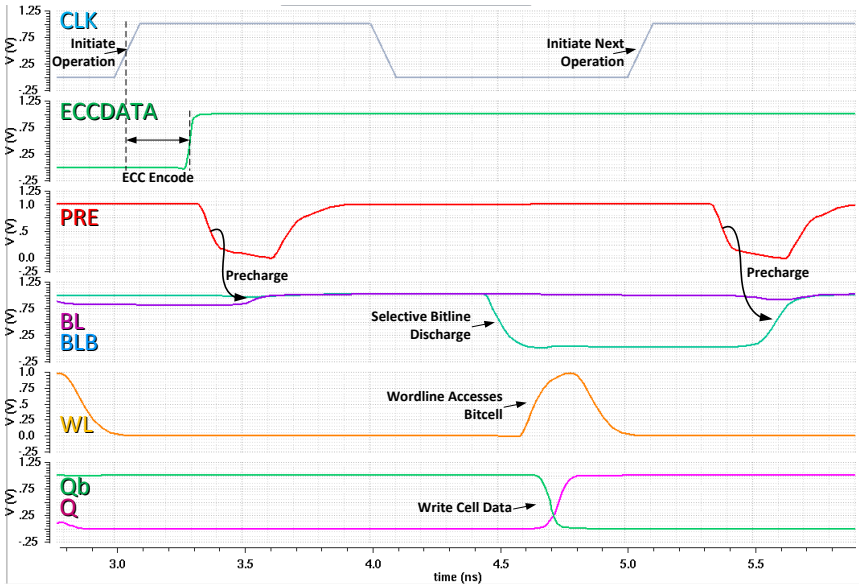
signal is determined by: 1). whether a read or write operation is being performed (R/W), 2). the ECC mode determined by the ECC test (ECCTEST), ECC enable (ECCEN), and ECC writeback (ECCWB) signals, and 3). the status of the error correction circuit's error detected (ED) and uncorrectable error (UE) signals. These signals (not shown in Figure 4.22) act as control inputs to the timing block units.

All timing signals are generated using an inverter-delay-line-based pulse generator circuit derived from the rising edge of the input clock signal. This is shown in Figure 4.23. The input clock rising edge is propagated through a non-inverting delay element to node A. This is then passed to one of the inputs of an AND gate. The arrival of a rising edge at node A raises the output of the AND gate. Node B, acting as the second input to the gate, is a delayed, inverted copy of A. The difference between the rising transition at node A and the falling transition at node B determines the pulse width of the AND gate's output high phase. By varying the clock-to-A delay, the low phase of the output signal can be varied, while by varying the A-to-B delay, the output high phase can be varied. Since the delay is generated using only the rising edge of the input clock, the output signal is independent of the input signal's frequency. This allows for full-speed testing while using a low-speed external input clock. By using this inverter-delay-line-based pulse generator architecture the internal SRAM timing signaling can be generated using the just the input

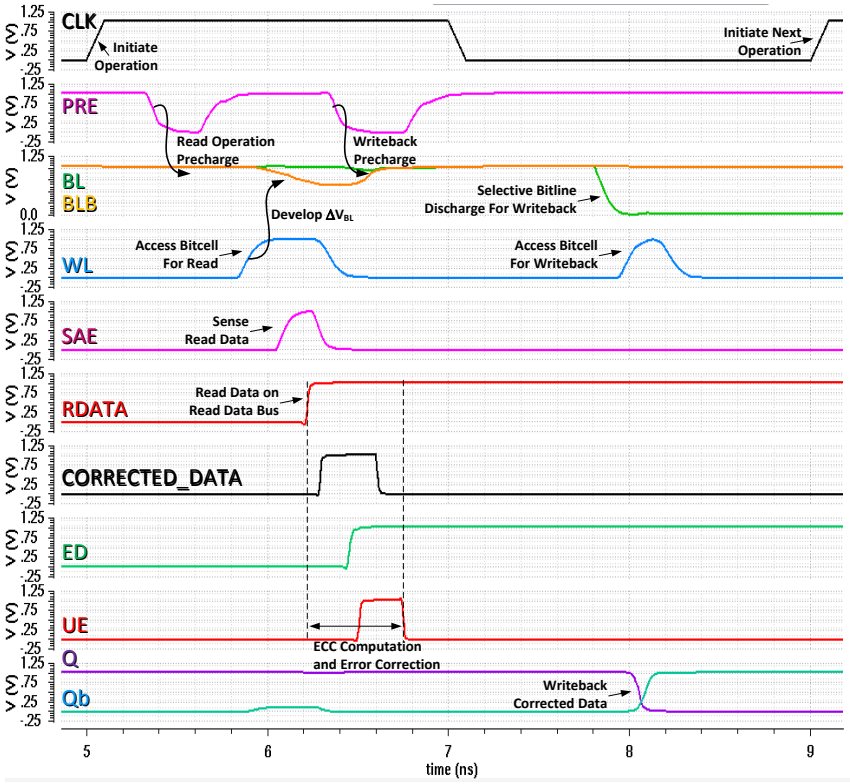
clock (either HSCLK or LSCLK) and a series of clock-shapers (AND gates) based on the functional operating mode signals. The write with ECC encoding signaling is shown in Figure 4.24(a) and read with ECC decoding with error correction and writeback is shown in Figure 4.24(b).

During a write operation (Figure 4.24(a)), the arrival of the input clock signal (CLK) initiates the generation of the write timing control signals. Write data is placed on the ECC circuit's input data bus and the additional check-bits to be written (ECCDATA) are first calculated. The bitlines (BL/BLB) are then precharged to V_{DD}. This operation is control via the active-low precharge signal (PRE). Next, the write enable (WE) signal, not shown, selectively discharges one of the bitlines to '0' based on the write data. In this example, the check-bit signal (ECCDATA) is a '1', so BLB is pulled low, while BL remains high. The bitcell is then exposed to the bitlines by raising the wordline (WL). This allows the cell's internal data (Q/Qb) to take on the new data value. Finally, after the data is written, the WL is then lowered. In this example, we can see that during the next clock cycle, the bitlines are reset to V_{DD} during the next precharge phase. If the ECC circuit is disabled, no check-bits are calculated, and only the data-bits are written to the array.

The read operation including ECC writeback (Figure 4.24(b)) is a more involved operation. Similar to the write operation, all read timing control signals are generated based on the rising edge of the input clock (CLK). The active-low precharge signal (PRE) ensures the bitlines (BL/BLB) are both initialized to V_{DD}. In this example, this is already the case. The address decoding operation, not shown, happens concurrently with the precharge operation. Following the precharge phase, the bitlines are left to float at V_{DD}, and the read evaluation phase begins. By enabling the wordline signal (WL), one of the floating bitlines begins to slowly discharge through the accessed bitcell. After a sufficient differential bitline voltage, ΔV_{BL}, has been developed to overcome the sense amplifier's input offset voltage, V_{offset}, the sense amplifier and read data latch are enabled via the sense amplifier enable (SAE - shown) signal and the read latch enable (LATCHEN - not shown) signal respectively. Following the successful reading of data, the read latch enters hold mode, and both the wordline and sense amplifier are turned off. This would mark the end of a traditional read operation without error correction. With the ECC unit enabled however, read data is then passed to the ECC's input data bus where it is inspected for errors. The ECC, after a fixed propagation delay, then computes the error detected (ED) and uncorrectable error (UE) status signals, and the corrected output data (CORRECTEDDATA). Provided that an uncorrectable error has not occurred (i.e., UE = 0), the output data can then be read off of the corrected data's output data bus. Finally, if the ECC writeback functionality is enabled, the circuit is prepared for writing in the event that the ECC detects a correctable error, (i.e., ED = 1 and UE = 0). To facilitate this, the precharge circuit charges the bitlines back up to V_{DD} immediately after the read operation, and a write operation is performed with the corrected data after it has been calculated. This process is identical to



(a) Write signaling with ECC encoding



(b) Read signaling with ECC correction and writeback

Figure 4.24: Write and read timing control signals with ECC functionality.

the standard write operation in which one of the bitlines are selectively discharged based on the writeback data. The cell is then exposed to the bitlines via raising the wordline and the corrected data is written into the cell.

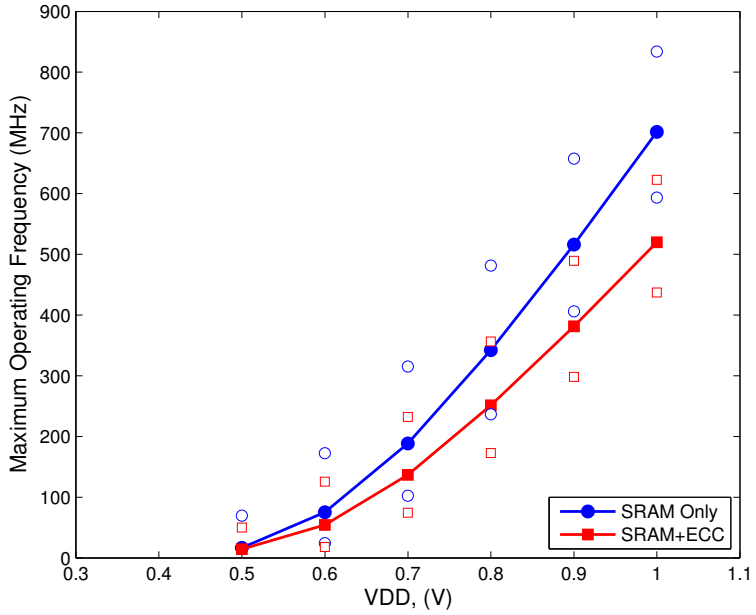
4.2.8 Performance Simulations

By using an edge-triggered, self-timed timing-control circuit, all read/write memory operations can be generated and verified at full-speed while using a low-speed, externally generated test clock. Although this allows for full-speed functional operation, it prohibits in-lab performance timing measurements. In light of this, Figure 4.25 provides Fast Simulation Program with Integrated Circuit Emphasis (SPICE)-based, full-chip performance data as a function of supply voltage across global process corners. Included in the figure is the maximum memory operating frequency for both the high-speed (Figure 4.25(a)) and low-speed (Figure 4.25(b)) test modes with the ECC circuit enabled and disabled.

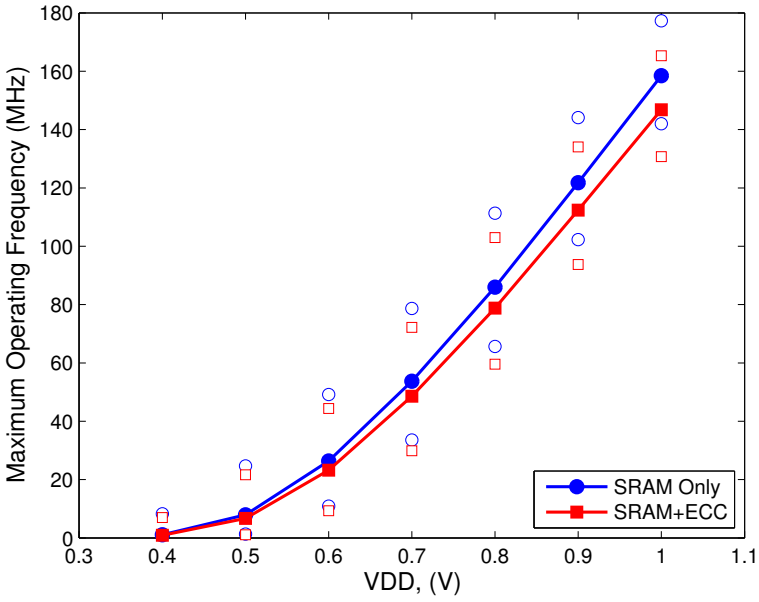
The main curve data is simulated at TT/27 ^{o}C, while the upper and lower bounds are simulated at FF/85 ^{o}C and SS/-25 ^{o}C respectively; these are the best and worst case timing corners. Initially, this may seem counter intuitive since at nominal supply voltage timing performance degrades as a function of temperature. At lower supply voltages however, in sub-65 nm technologies, increasing temperature has a positive impact on device performance. This is due to the temperature induced threshold voltage shift enhancing transistor drain current due to the increase in overdrive voltage ($|V_{GS} - Vth|$). At lower supply voltages, this effect outweighs the temperature induced carrier mobility degradation, and as such, drain current, and in turn device speeds, improve with increased temperature [77]. The high-speed test mode provides memory only operation between 701.7 MHz at 1.0 V and 16.7 MHz at 0.5 V. Enabling the ECC invokes a 15-25% performance penalty, reducing the operating frequencies to 519.75 MHz and 14.34 MHz respectively. The low-speed timing mode has been included for operation below 0.5V. It was designed to operate down to 200 mV, however functional memory operation ceases below 400 mV. The low-speed timing mode operates between 158.5 MHz at 1.0 V and 1.1 MHz at 0.4 V without ECC. Enabling the ECC invokes a 7-16% performance penalty and reduces the maximum operating frequency to 146.8 MHz at 1.0 V and 920 kHz at 0.4 V respectively.

4.2.9 Printed Circuit Board Design

A four layer printed circuit board (PCB) has been designed to perform various measurements on the test chip. The top and bottom layers are for signal routing, while the inner layers are power and ground planes. The fully populated PCB is shown in Figure 4.26. The test chip has been packaged and wire bonded in a 120 pin grid array (PGA) package



(a) High-Speed Timing Mode



(b) Low-Speed Timing Mode

Figure 4.25: Simulated memory timing performance with and without ECC enabled as a function of supply voltage. Simulations performed at (TT/27 ^{o}C), (SS/-25 ^{o}C), and (FF/85 ^{o}C).

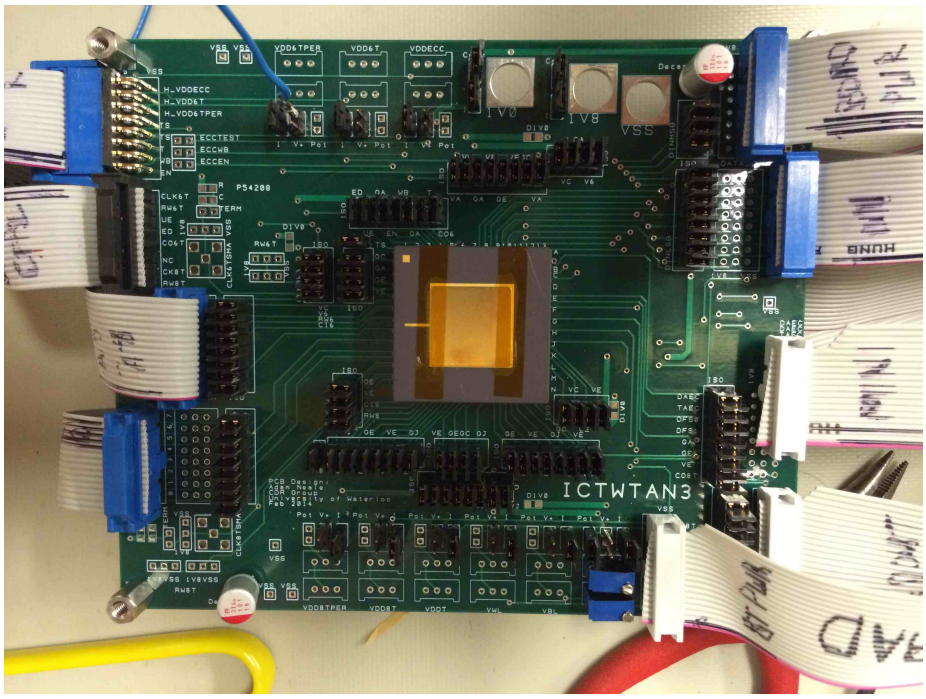


Figure 4.26: PCB for test chip measurements

using a double tier wire bonding configuration. A total of ten dies were packaged: five by the vendor Quik-Pak in San Diego, CA [78] using long, single-wire wire bonds, and another five by Corwil in Milpitas, CA [79] using two short wires and conductive interposers. Multiple vendors were contracted due to the technical complexity of the tight pitched 22.5 μm staggered, double-tiered bonding requirements. The chip layout, bonding diagrams, and pin description is provided in Appendix B. The core supply voltages for the memory array (VDD-6T), ECC (VDD-ECC), and peripheral circuits (VDD-PER) have been supplied independently to facilitate current measurements for each voltage region. Additional voltage supplies are provided for the 1.8 V and 1.0 V level converters, buffers, and ESD circuits within the input and output drivers circuitry. Address, data, and control signals can be driven by both jumpers and off-board ribbon cables. The off-board socket connections allow for the connection of remote test equipment. A separate controller card, shown in Figure 4.27, was designed to control the test chip in conjunction with a data generator through the ribbon cables during the radiation testing procedure.

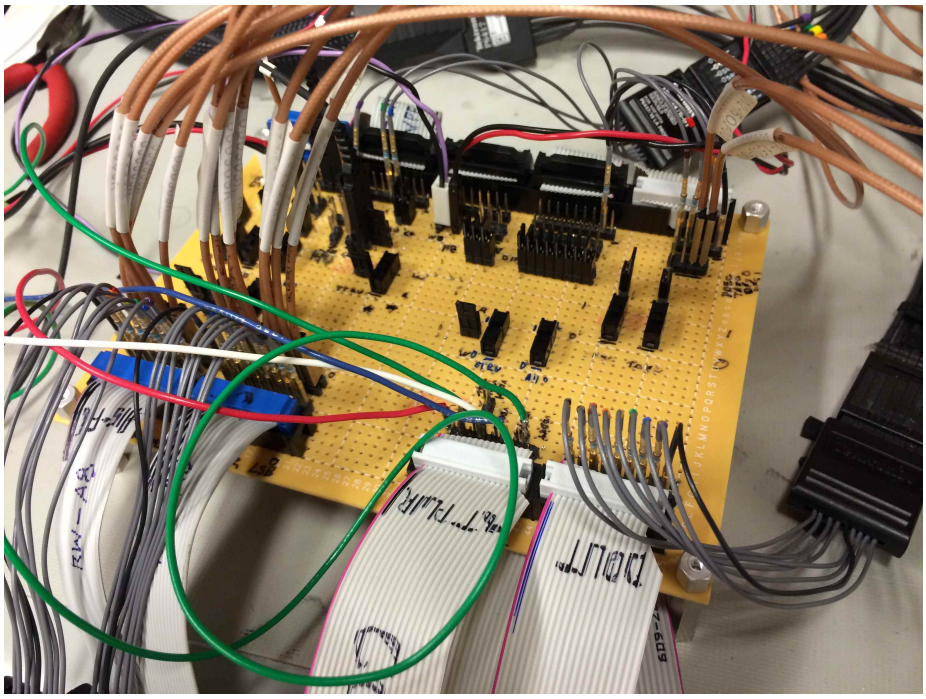


Figure 4.27: PCB for external controller card

4.3 Silicon Measurement

Measurements of the silicon test chip have been performed in two phases. First, functional and performance testing was conducted at the CMOS Design and Reliability Group research test lab at the University of Waterloo. This was followed by radiation induced soft error measurements at the Tri-University Meson Facility (TRIUMF) Canada's National Laboratory for Particle and Nuclear Physics, located in Vancouver, Canada.

The top-level chip layout and die photo for the silicon test chip is shown in Figure 4.28 and Figure 4.29 respectively and its features are listed in Table 4.9. A total of ten dies were packaged; however, two chips were found to have shorted wire bonds. The potential for this was expected due to the complexity of the fine pitched wire bonding requirements. Both of these shorts were deemed to be show stopping, resulting in a 20% chip-level yield loss. One short was between a pair of V_{DD}/V_{SS} supply rails making the chip completely inoperable, while the other was between a V_{DD} supply rail and an address line, thus creating a stuck-at-1 fault on the address bus effectively halving the memory address space. In total eight dies were deemed sufficiently operational for test.

The minimum operating voltage (VDD_{MIN}) distribution for the set of eight functional, packaged chips is shown in Table 4.10 in 50 mV increments with and without the SEC-DED

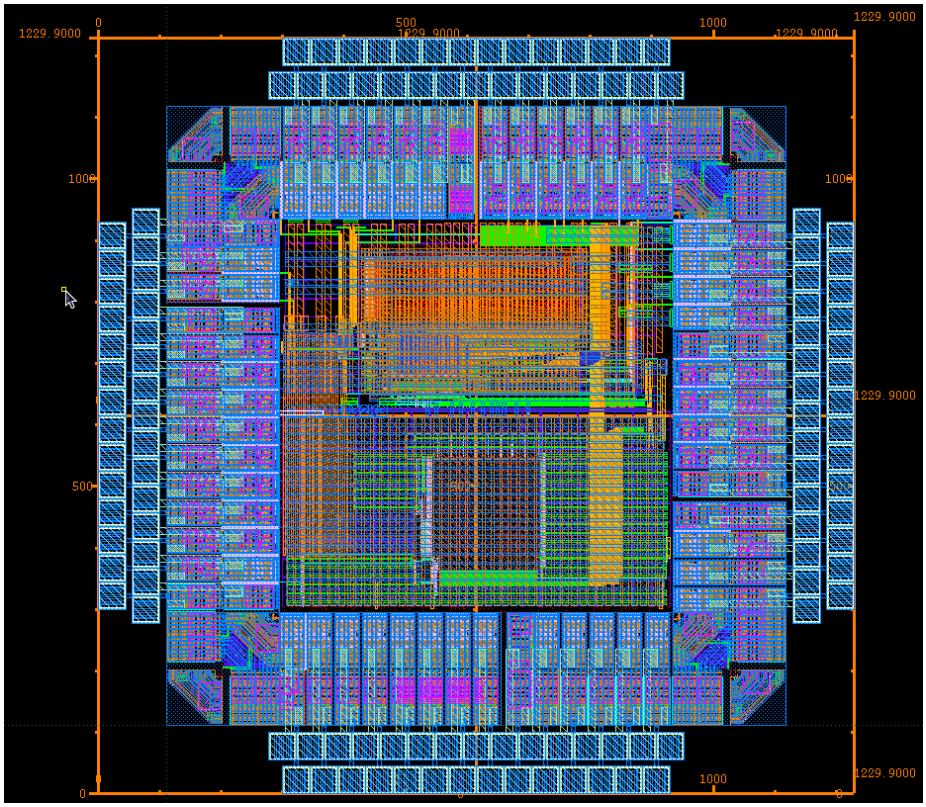


Figure 4.28: 75 kb SRAM Macro with ECC Full Chip Layout

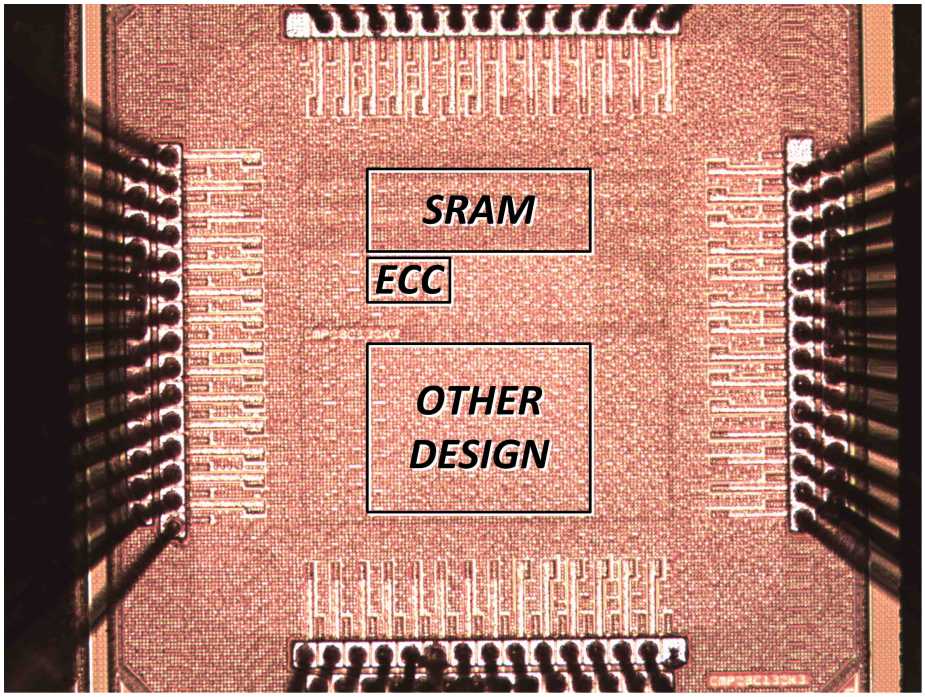


Figure 4.29: 75 kb SRAM Macro with ECC Die Photo

Table 4.9: Features of Fabricated 28 nm Test Chip

Process	28 nm LP HK+MG Bulk CMOS
Nominal Supply Voltage	1.0 V
Memory Capacity	75 kb = 64 kb data + 11 kb parity
Configuration	256 rows x 300 columns x 1 bank 75 bit codeword = 64 data bits + 11 parity bits 4-way interleaving
Physical Size	Die: 1.23 mm x 1.23 mm SRAM: 366.1 μm x 146.6 μm ECC: 139.4 μm x 89.3 μm
Bitcell Area	1.02 μm x 0.379 μm = 0.387 μm^2 (logic design rules)
Performance/Energy	0.92 MHz, 0.015 fJ/bit @ 0.4 V
ECC	(75, 64)-I_6 SEC-DED-TAEC-8AED
ECC Features	Selectable single, double, or triple adjacent error correction
Minimum Operating Voltage	400 mV with ECC, 500 mV without ECC

Table 4.10: VDD_{MIN} Measurement for 8 Test Chips

	Number of Chips				
VDD_{MIN} (mV)	400	450	500	550	600
Without ECC	0	0	3	1	4
With ECC	2	1	4	1	0

Table 4.11: Average Power Measurement for 2 Test Chips Capable of 400 mV Operation

Operating	SRAM		ECC		ECC/(ECC+SRAM)		Total = SRAM+ECC		
Voltage	Active	Leakage	Active	Leakage	Active	Leakage	Active	Leakage	Leakage
(V)	(μW)	(μW)	(μW)	(μW)	(%)	(%)	(μW)	(μW)	(%)
1.0	16.97	9.33	0.76	0.66	4.29	6.60	17.73	9.99	36.0
0.4	0.83	0.24	0.07	0.07	7.78	22.6	0.90	0.31	25.6

Low-Speed Test Mode, Input Clock Frequency 500 kHz

ECC feature enabled. Enabling the ECC reduces VDD_{MIN} by an average of 80 mV over the chipset, and allows for operation as low as 400 mV on two of the eight packaged chips. Since weak cell failure is a random failure mechanism, enabling the additional adjacent error correction modes provides negligible improvement in VDD_{MIN}.

To measure power consumption, an ammeter was connected in series with the SRAM and ECC supply voltage pins (VDD-6T and VDD-ECC) to monitor the average current as the address space was continually cycled. Multiplying the average current by the supply voltage provides the average power consumption. Under the same conditions, all individual read and write power measurements were within 6% of one another, and as such have been averaged into an single active power value. The leakage power is the power consumed while the memory remains in the idle/unaccessed mode. The SRAM and ECC average measured power consumption for the two chips capable of operating down to 400 mV is shown for supply operating voltages of 1.0 V and 0.4 V in Table 4.11. Measurement was performed using a 500 kHz clock in the low-speed test mode. For the ECC circuit, the active and leakage power consumption measurements were comparable since only the memory circuit was being gated in the retention mode; new input data was still being passed to the ECC as the address space was being cycled in the measurement test mode. At 1 V, the total average active power consumption was 17.73 μW with the ECC contributing 4.29% of this power. Reducing the supply voltage to 0.4 V reduced the active power consumption 94.9% to 0.9 μW. By removing the configurable interleaving and ECC circuitry, the chip's power consumption could be further reduced.

The degree of adjacent error correction is programmable using an off-chip configuration register with setting described in Table 4.8. In Figure 4.30 an ECC test mode is used to inject a progressively increasing sized error (1-3 adjacent-bits) into a codeword. For each error, the codeword is read using the circuit's ECC operating modes (No ECC, single-

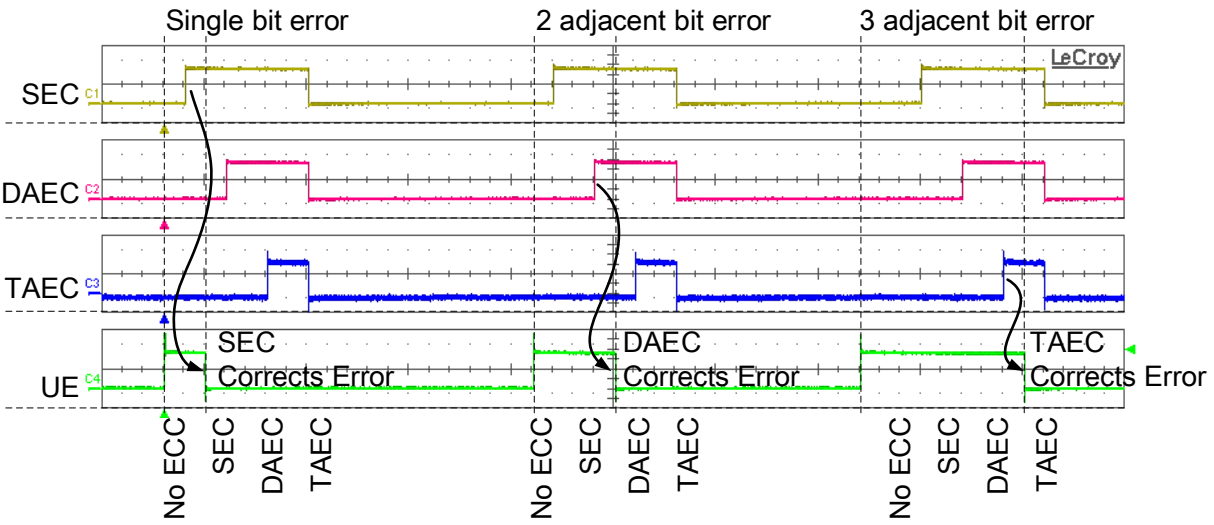


Figure 4.30: Measured error injection example. Additional ECC functionality is required to perform error corrections as the injected error size increases.

Table 4.12: Measured Error Injection Example

Uncorrectable Error (UE)	Correction Type			
Error Type	No ECC	SEC	DAEC	TAEC
Single	X	✓	✓	✓
2-Adjacent	X	X	✓	✓
3-Adjacent	X	X	X	✓

✓- Indicates a correctable error
X - Indicates an uncorrectable error

error-correction (SEC), DAEC, and TAEC). An oscilloscope is used to capture the error correction register and the ECC's Uncorrectable Error (UE) status signal for each read cycle. As the size of the error increases, more advanced ECC modes are required to correct the error. This is detailed in Table 4.12. When the ECC is disabled, none of the injected errors are correctable. Enabling the basic ECC allows for the single-bit error to be corrected, while enabling the DAEC feature allows for both the single-bit and 2 adjacent bit error to be corrected. Finally, enabling the TAEC feature allows for the 3 adjacent bit error to be corrected.

Table 4.13 provides a comparison of this work with other ECC protected SRAM arrays. These memories are each protected using SEC-DED schemes. Since the arrays are of different sizes, an effort has been made to create a fair comparison by quantifying average and leakage energy per bit. Due to differences in array architectures and process tech-

Table 4.13: Design Comparison

	This work	JSSC [48]	JSSC [29]	CICC [80]
Year	2014	2006	2009	2013
Technology	28 nm LP	130 nm	90 nm	28 nm HP
VDD (V)	0.4	0.4	0.8	1.0
Topology	6T+ECC	6T+Hidden ECC	6T+ECC	8T, no ECC
Memory Size (kb)	75	38	68.5	512
Speed (MHz)	0.92	27	100	-
Leakage (pA/bit)	10.1	91.8	230	353
Average Energy (pJ)	1.16	1.55 @ 0.3 V	5.34	9.58
Average Energy / bit (fJ/bit)	0.015	0.04 @ 0.3 V	0.08	0.018

nologies however, direct chip-to-chip measurement comparison is difficult. Nevertheless, normalizing to the bit-level provides a more objective measure for comparison than directly between arrays. We see that for the same V_{DD}, the proposed SRAM consumes 88.9% less leakage current and 62.5% less average access (read and write) energy per bit compared to [48]. [48] is able to maintain high access speed even at low voltage by removing the ECC circuit from the critical path. This is done by performing the error correction process at periodic intervals rather than as the data is read, and can lead to issues regarding data integrity if corrupted data is read prior to a correction cycle. Compared to [80], designed in a similar technology node (28 nm), the proposed SRAM consumes 16.6% less energy per bit for read and write access and has 97.1% less leakage current per bit. Scaling the supply voltage on the proposed SRAM to the nominal 1.0 V supply increases the leakage current to 121 pA/bit, which is still 2.92x less than the high performance (HP) cell in [80]. The leakage current ratio between the simulated vendor supplied, high performance SRAM cell and the proposed cell in the same process development kit shown in Figure 4.11 is 2.00x. This provides some insight into the differences between different vendor's provided bitcells. The leakage current for the High Performance cell in the used low-power kit, is still low relative to the cell in [80] using a high performance kit.

4.4 Radiation Testing

Accelerated high-energy neutron beam testing was conducted at the Tri-University Meson Facility's (TRIUMF) Neutron Irradiation Facility (NIF) in Vancouver, Canada [81]. All testing was conducted in compliance with the Joint Electron Device Engineering Council (JEDEC) JESD89A measurement standard: Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray Induced Soft Errors in Semiconductor Devices [82]. The

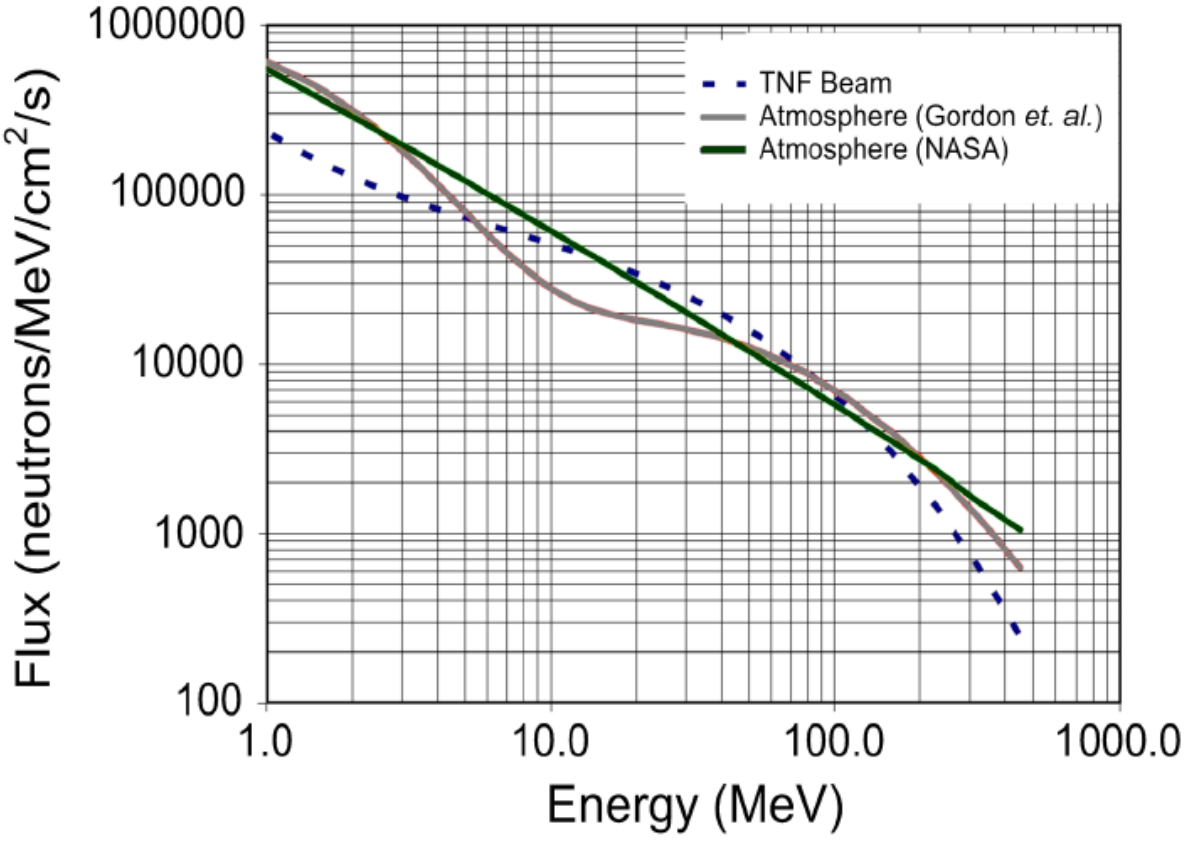


Figure 4.31: NIF neutron beam spectrum compared with the atmospheric spectrum [81].

neutron beam's energy spectrum, as shown in Figure 4.31, has been shaped to mimic that of atmospheric neutrons, while its fluence has been accelerated to reduce irradiation time. The NIF average beam fluence during the experiment was 2.08x10^6 neutrons/cm^2-s, which is approximately 3.75x10^8 times the average neutron fluence at sea level in New York City. New York City is commonly used as a normalization location for SER measurement data [7]. All SER measurements have been converted to FIT/Mb according to the following acceleration factor scaling equation,

$$\text{SER} = \text{Number of Upsets} \times \frac{10^9}{a_t a_n} \times \frac{1 \text{ Mb}}{\text{Array Size}} \tag{4.1}$$

where a_n is the neutron fluence acceleration factor, (e.g., 3.75x10^8), a_t is the irradiation time or time acceleration factor, and 1Mb/Array Size is the memory array capacity scaling

factor. The neutron fluence acceleration factor, a_n, varies between experiments based on the number of counted neutrons, and the neutron absorption ratio. The number of neutrons that the circuit is exposed to during the experiment is given by

$$\text{Neutron Exposure} = \text{CF} \times \frac{\text{Monitor Count without PCB}}{\text{Monitor Count with PCB}} \times \text{Counted Neutrons} \tag{4.2}$$

where the calibration factor (CF) is approximately 2.7x10^3, the neutron monitor count with and without the PCB in place is the neutron absorption ratio and is measured by counting the number of neutrons without and with the presence of the device under test each over a 10 second snapshot. Finally, Counted Neutrons is the total number of neutrons that pass through the chip and are collected by the neutron monitor counter throughout the duration of the experiment. The neutron exposure provides a cross section in terms of neutrons/cm^2. Dividing this value by the time duration provides the neutron fluence at the test facility, and normalizing this rate by the standardized fluence at New York City gives the acceleration factor, a_n. That is

$$a_n = \frac{\text{Neutron Exposure}}{\text{Duration of Experiment} \times \text{Fluence}_{NYC}} \tag{4.3}$$

4.4.1 Experimental Setup

The test facility and equipment setup at the TRIUMF NIF is shown in Figure 4.32. A 500 MeV cyclotron is used to establish a high energy 450 MeV proton beam line. The proton beam is targeted at an aluminum plate beam dump inside a water tank. As protons collide with the aluminum beam dump, neutrons are emitted and flow through a channel guide connected to the tank. The neutron beam channel guide has a vertical access aperture in the shielding at a distance of 2.6 m from the beam dump, with an access point 5 m above the channel. This vertical access point through the concrete shielding is shown in Figure 4.32(b). The device under test is mounted on a track and lowered down the access point where it is bombarded by the neutron beam. Cables connect the device under test to a monitoring station above the vertical access point shown in Figure 4.32(c). Testing the memory consists of the following test procedure:

1. Write a known data pattern to the memory array with the neutron beam on, but the circuit out of the beam path.
2. Read the memory array and compare the data pattern against the known written pattern. This ensures proper functionality of the memory and indicates the location of any weak bit-cells.

3. Measure the neutron beam fluence without the device under test in the beam path.
4. Lower the circuit into the neutron beam path.
5. Re-write the known data pattern into the memory array.
6. Allow the device under test to be exposed to the neutron beam for a measured period of time.
7. Measure the neutron beam fluence. The ratio between the beam fluence with and without the device under test in the beam line is the neutron absorption ratio.
8. Read the entire address space. Any differences between the read data and originally written known pattern is recorded as an error.
9. Repeat steps 5-8 until the total desired measurement time is reached. Measurements are done in multiple write/read operations to prevent the accumulation of errors over time.

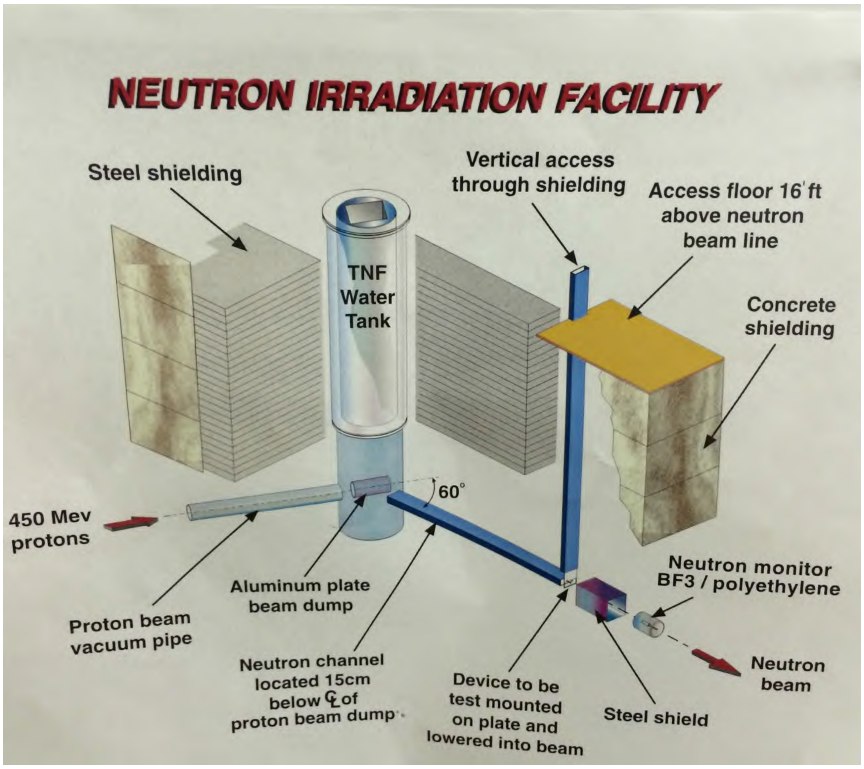
Radiation test data has been collected across a supply voltage range of 1.0 V down to 375 mV using the All-0, All-1, and Checker Board test data patterns. The duration of each experiment ranged from 2 to 10 hours of irradiation time. Data sampling was performed with the ECC Writeback feature disabled such that multiple degrees of error correction could be used to process the same data sets.

4.4.2 Measurement Results

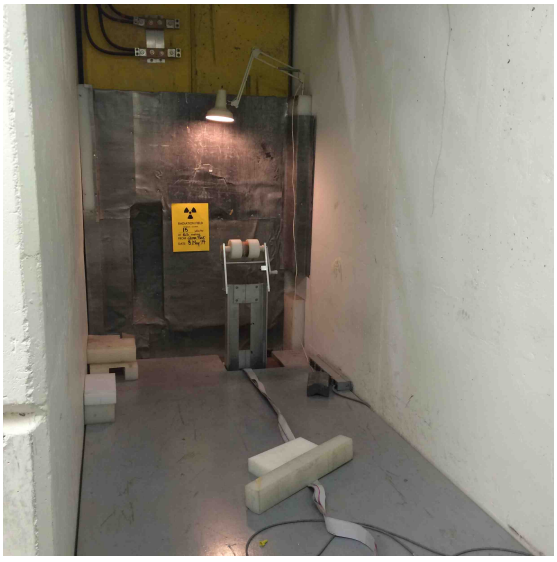
In this section, the soft error rate measurement data is presented as functions of supply voltage (V_{DD}), written data pattern, error correction mode, and error correction mode with bit-interleaving. Figure 4.33 shows a visualization of the superposition of all of the soft error measurement error location bitmaps for V_{DD} = 500 mV. The figure shows an x-y representation of the memory array's bitcells with 256 rows and 300 columns. Upset bitcells are represented with a black square, and the upset patterns in rows 1-64 and columns 225-300 has been enhanced to show the different error types.

SER vs. $\mathbf{V}_{DD}$

Table 4.14 and Figure 4.34 presents the radiation test data and raw-SER performance without applied ECC as a function of supply voltage. SER acceleration coefficients vary for each test case due to differences in irradiation times and variability in the neutron beam fluence at the test facility. At nominal supply voltage (1.0 V), MCUs make up 12.5% of



(a) TRIUMF Neutron Irradiation Facility setup



(b) PCB mounting station above the neutron beam

(c) Test equipment monitoring station

Figure 4.32: TRIUMF NIF test facility and test station equipment setup.

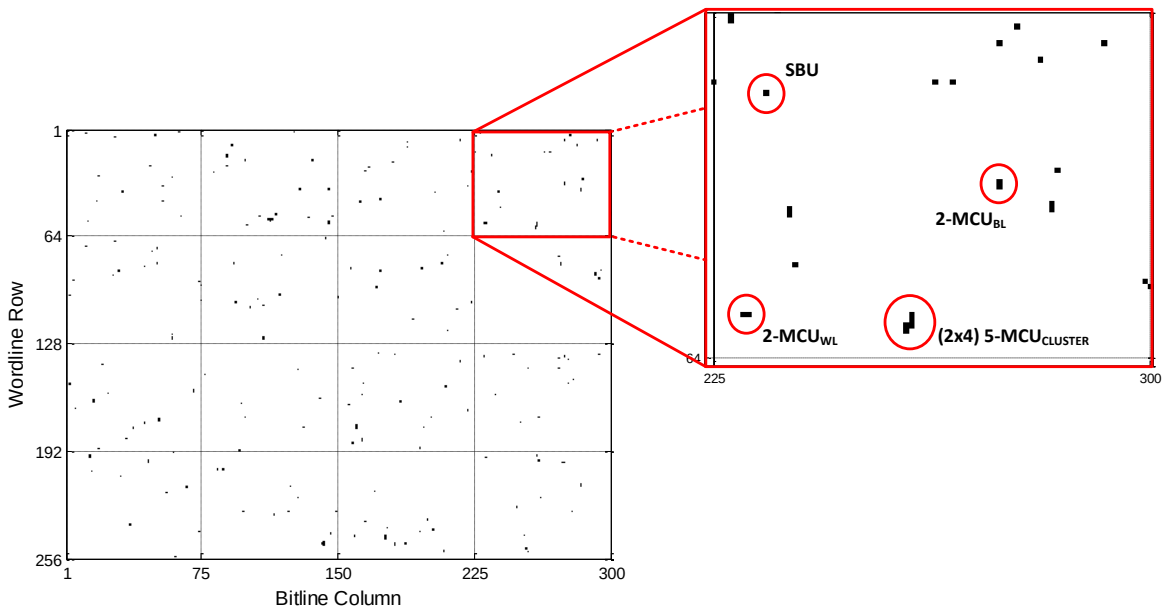


Figure 4.33: Superposition of all soft error measurement error location bitmaps for V_{DD} = 500 mV.

Table 4.14: Soft Error Rate Calculation from Radiation Test Data for 0.4 V to 1.0 V

V_{DD} (V)	Time (hr)	Total Errors	SBU	MCU	MCU_{BL}	MCU_{WL}	$MCU_{Cluster}$	MCU (%)	$MCU_{Cluster}$ + MCU_{WL} (%)	Fluence (n/cm$^2 \cdot h$)	$\frac{10^9}{a_n a_t}$	SER (FIT/Mb)
1.0	3.25	16	14	2	2	0	0	12.5	0.00	8.53x10^9	0.7215	157.62
0.8	2	16	14	2	2	0	0	12.5	0.00	7.36x10^9	1.3578	296.60
0.6	10	123	106	17	14	0	3	13.8	2.44	7.45x10^9	0.2684	450.75
0.5	10	189	148	41	35	2	4	21.7	3.17	6.96x10^9	0.2872	741.10
0.4	2	68	58	10	9	1	0	14.7	1.47	7.90x10^9	1.2658	1174.85

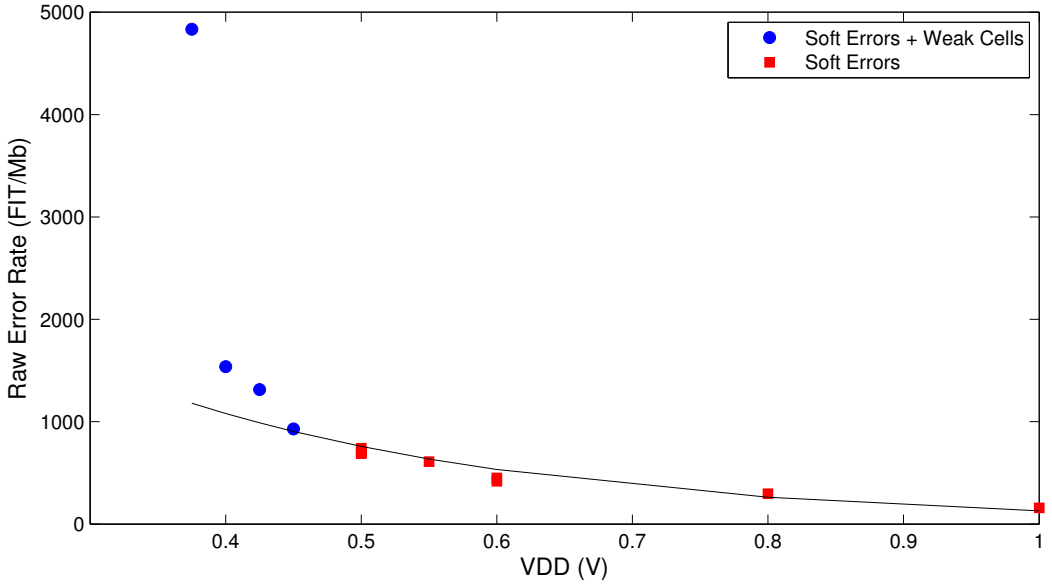


Figure 4.34: Raw error rate (radiation induced soft errors plus weak cells) vs. V_{DD} without ECC protection.

the total SER with none of the upsets belonging to the MCU_{WL} or $\text{MCU}_{Cluster}$ categories. By scaling the supply voltage, the SER increases exponentially. At 500 mV MCUs make up 21.7% of the total SER and the MCU_{WL} + $\text{MCU}_{Cluster}$ total accounts for 3.17% of the total SER. These values are consistent with those predicted in [6] and the SER model in Section 4.1. Additionally, MCU_{BL} makes up over 80% of the measured MCUs across the entire supply voltage range, this can be attributed to the MCU shaping strategies discussed in Section 4.2.3. By reducing the supply voltage below 500 mV, random single-bit weak cells begin to emerge and influence the measured error rate. These weak cells, making up approximately 25% of the total error rate, can be observed prior to irradiation, and have been filtered out of the SER data in Table 4.14. Due to their random, as opposed to adjacent bit upset behavior, they do have an impact on the ECC correction process and are therefore shown in Figure 4.34 and the ECC mode comparison shown in Figure 4.35. In Figure 4.34 the raw error rate is shown as a function of supply voltage for a series of radiation experiments for different time intervals and data patterns. The memory is fully functional, and the error rate consists of only radiation induced soft errors when the supply voltage is scaled from 1 V down to 500 mV. Below this voltage, weak bitcells influence the number of measured errors. This can be seen in the increased error rate above the fitted curve. Below 400 mV, weak cells and other peripheral circuit failures dominate the error rate yielding the memory inoperable. This is seen in the extreme increase in the number

Table 4.15: SER using Different Data Patterns for V_{DD} = 0.5 V, time = 2 hours

Pattern	Total Errors	SBU	MCU	% MCU	Fluence ($n/cm^2 \cdot h$)	SER (FIT/Mb)
All 0's	40	32	8	20.0	7.83x10^9	697.56
All 1's	34	25	9	26.5	6.64x10^9	698.84
CB	37	29	8	21.6	7.13x10^9	708.53

of measured errors.

SER vs. Data Pattern

Experiments were performed using All-0, All-1, and Checker Board (CB) data patterns for irradiation times of 120 minute each. This data is summarized in Table 4.15. Each of these data patterns provide SER's within 1.5% of each other and a total MCU count within one upset for all data patterns. The error rate showed minimal dependence on data pattern.

SER vs. ECC Mode

By applying an increasing amount of adjacent ECC correction, the corrected-SER can be reduced for a given supply voltage. In Figure 4.35, the corrected-SER is plotted for 0.6 V, 0.5 V, and 0.4 V using 1-way interleaving for each of the four ECC correction modes (No ECC, SEC-DED, DAEC, and TAEC). When no ECC is applied the corrected-SER is equal to the raw-SER. By applying even the basic SEC-DED scheme, the SER is reduced by 37x ($\sim$97%) at 500 mV. The remaining errors are those MBUs beyond the error correcting capabilities of the SEC-DED code. By applying the DAEC feature, the double adjacent-bit upsets are removed (a 5x SER reduction over SEC-DED, 189x reduction over no ECC, or $\sim$99.5% of the total raw-SER) and only those upsets affecting more than two adjacent bits remain. Since all upsets in the 500 mV dataset are less than or equal to three adjacent bits, enabling the TAEC feature is able to remedy all of the upsets. Each of these corrected-SERs are on the same order as those predicted in Section 4.1.2 for the given raw-SER. Similar benefits can be seen by using additional interleaving, however this option may not always be available depending on the memory's specifications. For the 400 mV dataset, one radiation induced upset occurred in a non-adjacent bit location in the same memory word as a corrupted weak bitcell. Although detectable, the added error mechanism adds to the overall error rate experienced at the 400 mV setting. Additional random bit ECC protection would be necessary to correct both types of upsets simultaneously. This would allow for a further potential reduction in VDD_{MIN}.

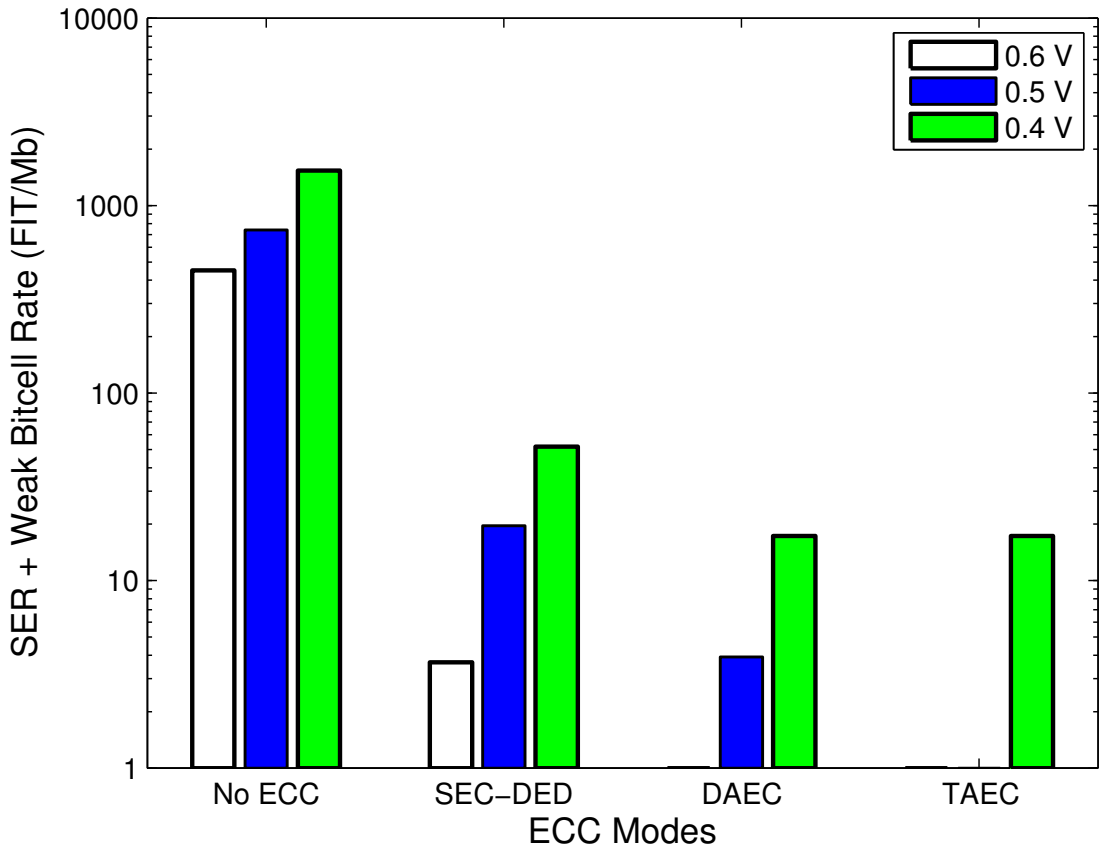


Figure 4.35: Radiation induced SER plus VDD induced weak bitcell rates for each ECC mode. Weak bitcell upsets only occur at the 400 mV datapoints.

SER vs. ECC Mode with Bit-Interleaving

Figure 4.36 shows how interleaving combined with ECC impacts the SER. The dataset shows the radiation induced SER for each ECC mode at V_{DD} = 500 mV using 1-, 2-, and 4-way interleaving. Without ECC applied, interleaving on its own has no impact on correcting upsets. By applying the basic SEC-DED scheme, the SER is reduced by 37x using 1-way interleaving, 195x for 2-way interleaving, and eliminated completely using 4-way interleaving. Applying the DAEC code corrects all errors for 2-way interleaving, while the TAEC feature is required to correct all errors for 1-way interleaving. Provided a larger capacity memory circuit, increased irradiation time, or smaller bit cell (as will be discussed for a set of vendor cells in Section 4.5.3) more of the lower-probability, larger errors would emerge, providing richer datasets and the potential to more accurately model the error rates for the higher correction-interleaving protected configurations.

4.5 Vendor Cell SER Estimation

In this section, estimates are made regarding the anticipated SER performance of the supplied vendor cells for the 28 nm process used for fabrication. These cells are more likely to be used in commercial products. Critical charge, Q_{crit}, for the implemented bitcell and vendor cells is first determined as a function of supply voltage by SPICE simulations. The results are then used in conjunction with the radiation test data to extract relevant process parameters. These parameters are used to estimate the bit- and system-level SER performance for the vendor cells.

4.5.1 Critical Charge SPICE Estimations

For estimating SRAM bitcell critical charge, Q_{crit}, a two component exponential function, shown in Equation 4.4 [30] is used to model a current pulse being injected into one of the bitcell's internal storage nodes, shown in Figure 4.37. This models the transient current pulse shown in Figure 1.1.

$$i_{\text{injected}}(t) \;=\; \frac{Q}{\tau_f - \tau_r}\left(e^{-t/\tau_f} - e^{-t/\tau_r}\right) \tag{4.4}$$

Here, Q is the peak charge deposited by the current pulse, and τ_r and τ_f are the pulse's rise and fall time constants. The pulse has a short rise time and a long fall time. The critical charge is calculated through successive SPICE simulation in which the peak charge, Q, is successively increased until the bitcell data changes state. This is shown in Figure 4.38. For

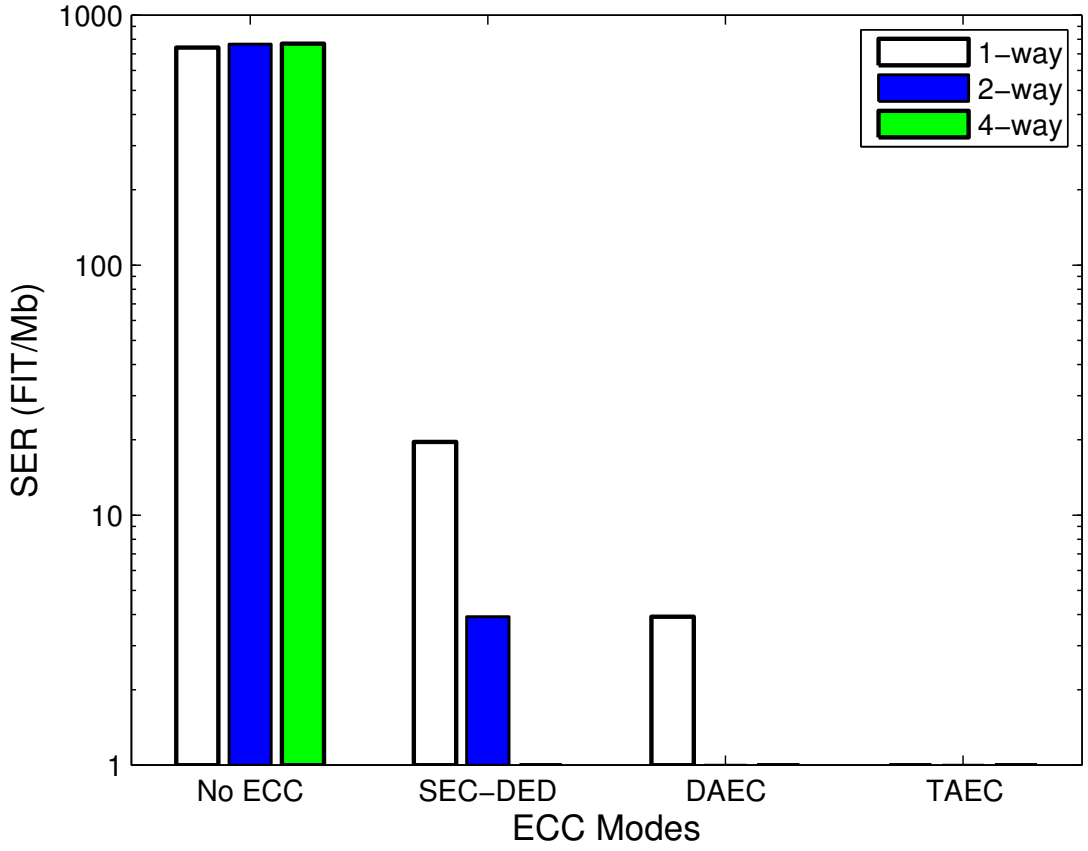


Figure 4.36: Radiation induced SER for each ECC mode at $V_{DD} = 500$ mV for 1-, 2-, and 4-way interleaving.

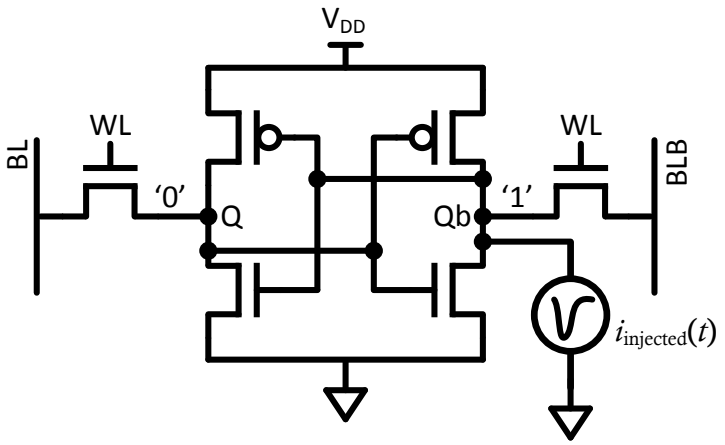


Figure 4.37: 6T SRAM Bitcell Critical Charge Testbench

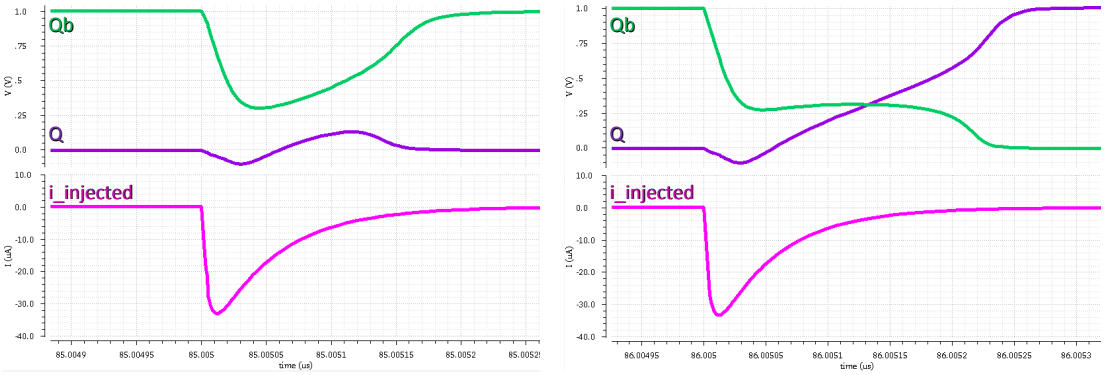
the implemented bitcell, the cell is able to maintain its data (Figure 4.38(a)) after exposure to a particle strike modeled by the exponential current pulse. While in Figure 4.38(b), for an small incremental increase in the peak of the current pulse, the cell data is corrupted. By numerical integration of the injected current, the critical charge can be determined. Critical charge for the implemented bitcell and three vendor supplied cells is shown in Figure 4.39 as a function of supply voltage at (TT/27 ^{o}C).

The implemented cell, designed using logic design rules, uses larger device sizes compared to the vendor cells, and as such has a larger Q_{crit} across the entire supply voltage range. Of the four cells, the high speed vendor cell has the lowest critical charge consistently equal to 60-70% of the implemented cell's Q_{crit}. At the nominal 1.0 V supply, the implemented bitcell has a critical charge of 2.13 fC, while the vendor dense, high speed, and high performance cells have critical charges of 1.46 fC, 1.39 fC, and 1.71 fC respectively. This data in conjunction with the data in the next section will be used to estimate the vendor cell's SER.

4.5.2 Parameter Extraction

In [83] an empirical model is developed for relating the bit-level SER to the critical charge, Q_{crit}. The model is described using the following exponential relationship between SER and Q_{crit}:

$$SER \;=\; k \times F \times A \times \exp\left(-\frac{Q_{crit}}{Q_s}\right). \tag{4.5}$$



(a) No Data Corruption (b) Data Corruption

Figure 4.38: By incrementally increasing the peak charge, Q, deposited by an exponential current pulse, $i_{injected}(t)$, the critical charge, Q_{crit}, necessary to corrupt a bitcell can be determined.

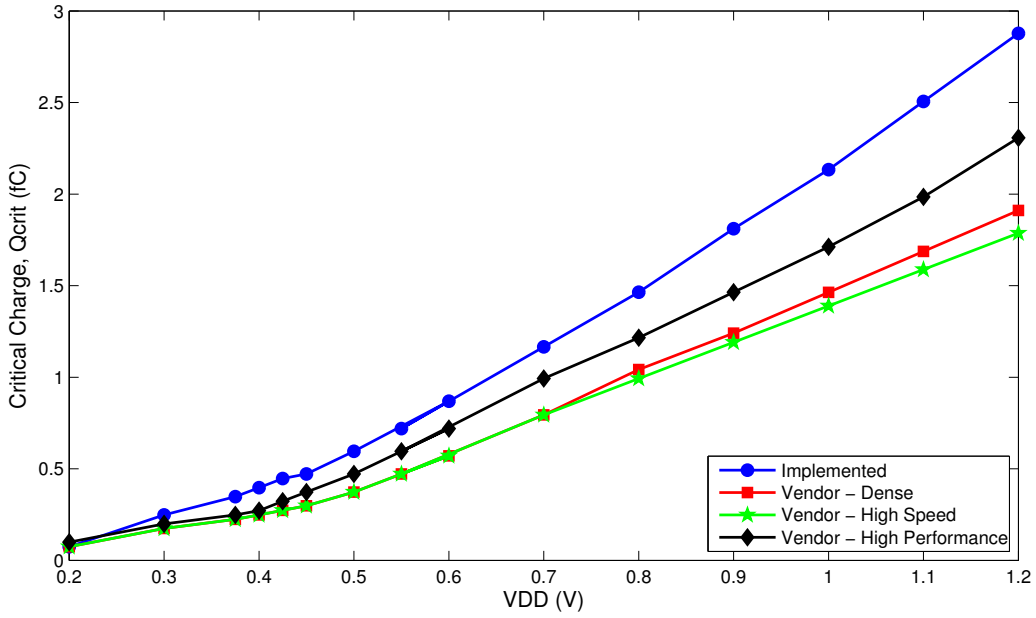


Figure 4.39: Critical charge as a function of supply voltage V_{DD} for the implemented bitcell and vendor supplied cells.

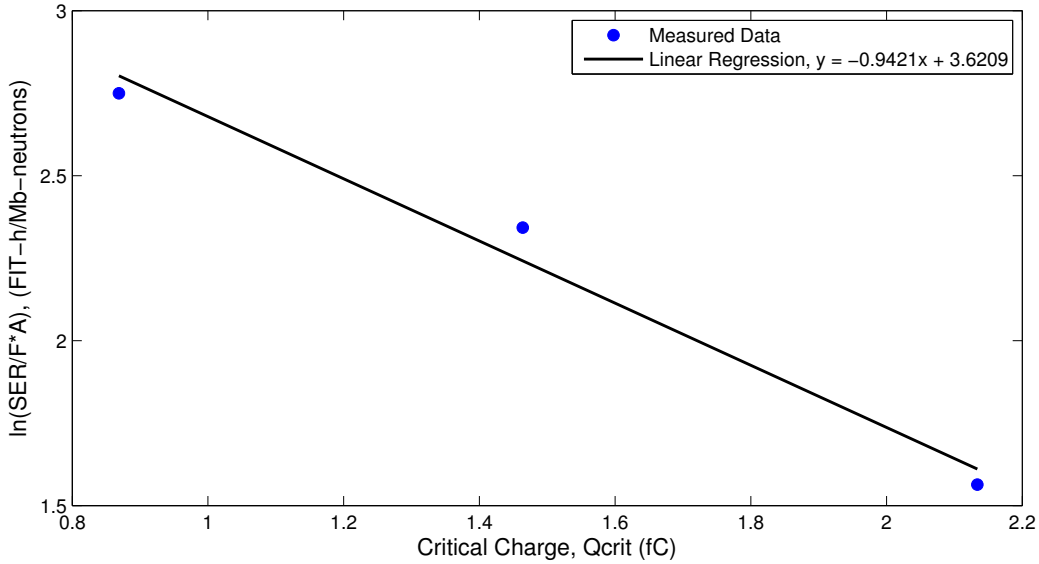


Figure 4.40: Extraction of charge collection efficiency, Q_s, and proportionality constant, k.

In the expression, k is a proportionality constant, F is the high energy neutron beam fluence in particles/cm^2-h, A is the sensitive area of the circuit in cm^2 (in this case we use the bitcell area), and Q_s is the charge collection efficiency of the device in fC. The charge collection efficiency is process dependent and depends on both the substrate doping and carrier mobility. If k and Q_s can be determined, the SER can be calculated for a given bitcell since F is a controllable parameter, A is known at design time, and the Q_{crit} can be calculated using the SPICE simulation method discussed in Section 4.5.1.

Both k and Q_s can be extracted from the radiation test data by using the measured SER and high energy neutron beam fluence from the radiation experiment. By rearranging Equation 4.5 and taking the natural logarithm, the following linear equation can be produced as a function of Q_{crit}:

$$\ln\left(\frac{SER}{F \times A}\right) \;=\; \left(-\frac{1}{Q_s}\right) Q_{crit} \;+\; \ln\, k. \tag{4.6}$$

By plotting the known data on the left-hand side of Equation 4.6 for a number of Q_{crit} values at different supply voltages, a linear regression will produce a line with slope $-1/Q_s$ and y-intercept $\ln(k)$. This is shown in Figure 4.40 for supply voltages of 0.6 V, 0.8 V, and 1.0 V. From this, extracted values of $k = 37.3711$ FIT-h/Mb-neutron and $Q_s = 1.0614$ fC can be obtained, and substituted into Equation 4.6 to give:

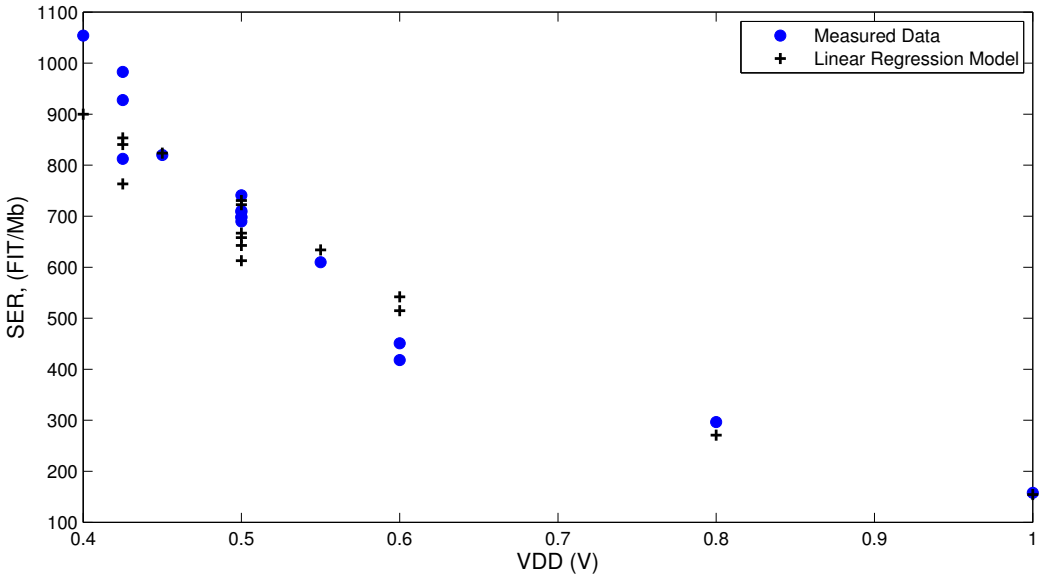


Figure 4.41: Measured and modeled SER vs. V_{DD}. Modeled data is within 15% of measured data for all data points except for one point at $V_{DD} = 0.6$ V, Measured = 417.95 FIT/Mb, Model = 541.80 FIT/Mb, Percent Difference = 29.6%.

$$SER \;=\; 37.3711 \times F \times A \times \exp\left(-\frac{Q_{crit}}{1.0616}\right). \tag{4.7}$$

By using the Q_{crit} values calculated across the supply voltage range, Equation 4.7 can be compared against the measured radiation data. This is shown in Figure 4.41. The model is within 15% of all but one of the measured SER data points across the entire 0.4 V to 1.0 V voltage range. The model gives a reasonable estimate for predicting SER performance.

4.5.3 SER Performance

Since the vendor cells can be fabricated in the same process as the radiation tested implemented bitcell, the extracted k and Q_s parameters from Section 4.5.2 can be used with the vendor cell Q_{crit} data from Section 4.5.1 to estimate the vendor cell's SER performance. This is summarized in Table 4.16.

Using Equation 4.5, the bit-level SER for each bitcell can be calculated in terms of FIT/Mb. The layout area A for each bitcell is provided in Table 4.16. As to be expected,

Table 4.16: 28 nm SRAM Bitcell SER Comparison at Nominal (1.0 V) Supply Voltage

Cell	Bitcell Area (μm^2)	Array Area (mm^2)	Capacity (MB)	Bit-SER (FIT/Mb)	System-SER (FIT/Device)
Implemented	0.387	29.721	9.375	165.24	12 393
Vendor - Dense	0.120	29.721	29.52	96.66	22 827
Vendor - High Speed	0.152	29.721	23.31	130.78	24 387
Vendor - High Performance	0.152	29.721	23.31	96.74	18 040

Array Area = Implemented bitcell area × 75 kb capacity × 1000 macro tiles

the bit-level SER decreases for the smaller bitcells despite their reduced Q_{crit}. This is similar to the scaling trend data shown in Figure 1.4. The array area is limited to the area of the bitcell array and does not include the address decoder and additional row/column peripheral circuitry associated with the SRAM macro. It is calculated by multiplying the implemented bitcell's area by its 75 kb capacity then by 1000 macro tiles to achieve a 9.375 MB capacity (8 MB data plus 1.375 MB parity). This array area is then used to calculate the potential array capacity provided by each of the vendor cells for the same given die area. The dense vendor cell has a bitcell area that is 31% of the implemented cell and hence can yield a capacity 3.15x greater than that of the logic design rule implemented cell. Considering each of the bitcell's fixed area capacities as their device area, the system-level SER in terms of FIT/device can be calculated by multiplying each cell's bit-level SER by its capacity. Again, much like the the scaling trends seen in Figure 1.4, despite their reduced per-bit SER, the system-level SER of the smaller (or scaled) cells increases due to their increased capacity per fixed unit area relative to the larger cells. This data can be used to provide bit- and system-level SER estimates for memories designed using the vendor bitcells at design time. For the vendor dense cell to have the same system-SER as the implemented cell, the vendor dense cell's capacity can be reduced to 16.02 MB. This would require 16.133 mm^2 of die area, 54.3% that of the logic design rule cell, while achieving a 1.71x greater capacity. Applying any of the ECC schemes presented in this chapter will reduce the bit-level SER, and in turn produce a reduction in the system-level SER. For instance, reducing the power supply to 0.5 V for the implemented cells would produce a system-SER of 55 582.5 FIT/device for the 9.375 MB capacity cache. Enabling the DAEC ECC could reduce this to 294 FIT/device.

Finally, an estimate can be made for the maximum MCU_{WL} upset size for each of the vendor cells by considering the implemented cell's maximum MCU_{WL} upset size in conjunction with each cell's critical charge and bitcell area. For the implemented cell, a 3-adjacent bit same row upset was observed during one of the radiation experiments when the supply voltage was equal to 500 mV. Assuming the high energy particle directly struck the cell's most sensitive diffusion, the sensitive diffusions in the adjacent most cells within the same row are each separated by a distance of one bitcell width, w, from the upset's

point of origin. Assuming that the charge dissipates exponentially as a function of distance from the strike location, which has been observed in [84], we can determine the collected charge as a function of distance from the strike location. That is:

$$Q_{col}(d) \;=\; Q_o \times e^{-d}, \tag{4.8}$$

where $Q_{col}(d)$ is the collected charge at distance d from the strike origin and Q_o is the charge deposited at the origin. If $Q_{col}(d) \geq Q_{crit}$ at the adjacent bit's sensitive node, then we can assume that the cell will be upset. We can determine a lower and upper bound, with the aid of Figure 4.42, on the maximum number of adjacent upset cells by considering the case where $Q_{col}(d) = Q_{crit}$ when 1). d is equal to the width of one bit, i.e., $d = w$, and, 2). when d is just less than the distance necessary to corrupt the next most adjacent bit, i.e., $d = 2w - \delta$. This models the cases when Q_o is just large enough for a 3-bit upset to occur, and when Q_o is just shy of creating a larger upset. Assuming a small quantity for δ (0.01 μm), width $w = 1.02$ μm, and using the implemented cell's Q_{crit} at the voltage at which the upset occurred, (Q_{crit} at 500 mV = 0.595 fC), Q_o is calculated for the lower bound as $Q_{o|LB} = 1.652$ fC and for the upper bound as $Q_{o|UB} = 4.536$ fC respectively.

By rearranging Equation 4.8, the distance d at which $Q_{col} = Q_{crit}$, given a charge Q_o deposited at the strike origin, can be determined for each vendor cell. This is given by:

$$d \;=\; -\ln\left(\frac{Q_{crit}}{Q_o}\right). \tag{4.9}$$

Assuming the vendor cells have the same aspect ratio as the implemented cell, each of their respective widths, w_{cell}, can be determined. By normalizing the distance d at which $Q_{col} = Q_{crit}$ by each cell's width and taking the floor of this value, the number of upset cells to one side of the originally upset cell can be determined. Multiplying this value by 2 and adding one gives the total number of adjacent upset bitcells for the given particle strike.

$$\text{MCU}_{WL}\text{ Size} \;=\; 2 \times \left\lfloor \frac{d}{w_{cell}} \right\rfloor + 1. \tag{4.10}$$

A summary of this process is provided in Table 4.17 for each of the vendor cells. As to be expected, due to its lowest critical charge and bitcell area, the dense cell has the largest anticipated MCU_{WL} size ranging from 5 to 9 bitcells. Of the implemented ECC and interleaving schemes, the only configuration that could correct the 9-adjacent MCU_{WL} upset would be the TAEC ECC combined with 4-way interleaving, no other configuration could correct an upset of this size. This emphasizes the need for additional adjacent error correction and detection capabilities as semiconductor technology continues to scale.

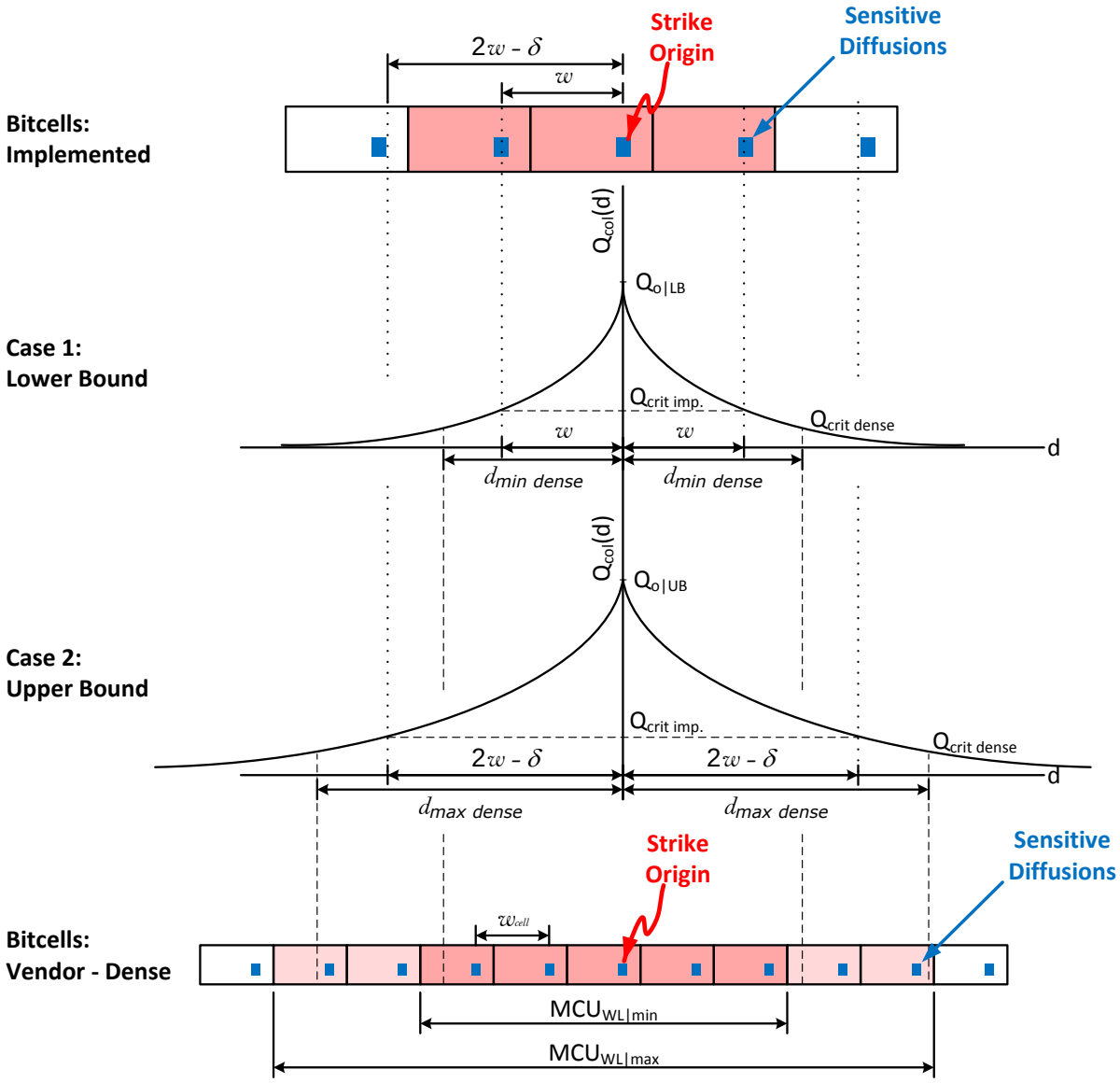


Figure 4.42: MCU width bounds for vendor supplied cells can be determined from their Q_{crit}, bitcell width, and Q_o upper and lower bound parameter extracted from the implemented bitcell's simulated and radiation test data. In this example, a particle strike causing a 3-MCU$_{WL}$ using the implemented bitcell could create an upset ranging from a 5-MCU$_{WL}$ to a 9-MCU$_{WL}$ for an array implemented using the vendor's dense bitcell.

Table 4.17: 28 nm SRAM Bitcell MCU Width Comparison at 500 mV Supply Voltage

Cell	Bitcell Area (μm^2)	w_{cell} (μm)	Q_{crit} @ 500mV (fC)	d_{min} (μm)	d_{max} (μm)	MCU$_{WL\|min}$ (Bitcell)	MCU$_{WL\|max}$ (Bitcell)
Implemented	0.387	1.020	0.595	1.020	2.030	3	3
Vendor - Dense	0.120	0.568	0.372	1.490	2.500	5	9
Vendor - High Speed	0.152	0.640	0.372	1.490	2.500	5	7
Vendor - High Performance	0.152	0.640	0.471	1.254	2.264	3	7

$Q_{o|LB} = 1.652$ fC and $Q_{o|UB} = 4.536$ fC

4.6 Summary

In this chapter, an error channel model has been presented for estimating size and frequency of multi-bit upsets in scaled technologies. Simulations have been performed to determine the corrected soft error rates produced by various 64 data-bit error correcting codes in the presence of this error channel. Further, a silicon test chip designed and fabricated in a 28 nm process technology is discussed, and presented with simulation and radiation measurement data. This data is then used to estimate the bit- and system-level soft error rates for a set of vendor designed SRAM bitcells in the same 28 nm technology.

Chapter 5

Conclusions and Future Work

This chapter summarizes the contributions of this research and outlines a potential research direction moving forward.

5.1 Contributions to the Field

As external radiation sources remain constant and semiconductor technology scales deep into the sub-45 nm regime, radiation induced multi-bit soft error vulnerability continues to increase in semiconductor devices. With on-chip embedded SRAM using near-minimum feature-size devices and consuming over 50% of the die area in state-of-the-art microprocessors and system-on-chip designs, SRAM is particularly susceptible to upsets. Increasingly reliable SRAM designs utilizing modern soft error mitigation techniques are becoming a necessity. In this research, the influence of soft errors in SRAM is investigated at the architectural level, and the utility of adjacent bit error correcting codes is explored. Further, techniques have been experimentally validated through test chip implementation and accelerated neutron irradiation tests. The main contributions and from this research include:

5.1.1 A New Class of Error Correcting Codes for Adjacent Multi-bit Upsets

We have proposed a new error correcting code sub-class for modern soft error reliability. The codes are targeted toward mitigating the adjacent multi-bit upsets most prominent in scaled SRAM technologies, and provide a middle ground between the currently favoured SEC-DED ECC schemes and the more robust, but more complex, multi-bit correcting schemes. By using a number of check-bits between the two codes, a greater degree of

error reliability targeted specifically toward the error patterns found in aggressively scaled technologies is provided compared to the SEC-DED codes for an area overhead less than the traditional multi-bit codes. The code's design procedure and full performance analysis is provided complete with encoding and decoding examples.

5.1.2 Soft Error Rate Modeling

A multi-bit upset error channel model has been developed to estimate the frequency and distribution of radiation induced error types. The error model is used to estimate the corrected soft error rates for various error correcting codes as a function of raw error rate after being exposed to the channel model. The proposed model is in agreement with the error size distributions presented in [6]. The model has been used to provide corrected-SER estimates for Hsiao SEC-DED, BCH DEC, and S5EC-D5ED Reed-Solomon codes, as well as DAEC and TAEC versions of the proposed code class using 1-, 2-, and 4-way interleaving.

5.1.3 28 nm Test Chip Design and Implementation

A test chip has been successfully designed and implemented in a 28 nm bulk CMOS technology to provide silicon validation for the proposed set of codes across a wide range of input parameters and to establish the full-custom design flow for the process development kit. This kit was chosen to allow for the smallest possible SRAM bitcell area and largest possible number of multi-cell upsets during radiation testing. At the time of design, this 28 nm technology was the most advanced process available to Canadian academia through the Canadian Microelectronics Corporation, and to the best of our knowledge no other university had fabricated in this technology for purposes other than initial kit characterization. The chip contains a 75 kb 6T SRAM macro with a configurable implementation of the proposed ECC class allowing for selectable degrees of adjacent error correction and interleaving. The chip was designed for dynamic voltage and frequency scaling, providing full functional operation from 1 V at 700 MHz down to 500 mV at 16.5 MHz with no ECC, and 400 mV at 0.92 MHz with the ECC enabled. Having successfully completed the full development cycle and created a series of design automation scripts for this technology, design time can be reduced and yield/performance confidence increased for future chip implementation development cycles.

Additionally, we have performed an industry standard accelerated neutron irradiation test on the fabricated test chip. This has verified the soft error behaviour for the process technology, and shown the applicability of the proposed ECC schemes for mitigating multiple adjacent-bit soft errors. Further, all soft error data has been presented in raw,

non-normalized form. This data is typically closely guarded by industry developers and obfuscated through normalization. Thus, any group interested in soft error rate measurement should be able to benefit from this work.

5.1.4 Soft Error Rate Estimations for Vendor Cells

Since it was not possible to perform irradiation tests on the vendor supplied SRAM bitcells, process specific parameters were extracted from the radiation test data using the bitcell implemented using logic design rules, and used to extrapolate SER estimates for the vendor cells. This data provides raw, non-normalized SER estimates for near-state-of-the-art industry-designed embedded SRAM bitcells.

5.1.5 Publications

This work has been published in one IEEE journal article and has appeared in one IEEE international conference proceeding. Two additional IEEE journal articles are currently under peer review. A full list of these works, and other publications written throughout the duration of this Ph.D., has been provided in Appendix C.

5.2 Future Work

As semiconductor technology continues to scale, it is anticipated that both the percentage and size of multi-cell and multi-bit upsets will continue to grow. As such, the need for adjacent-cell soft error mitigation will increase. Furthermore, the demand for reduced power consumption will continue to push SRAM toward lower voltage designs. This will not only further exacerbate soft error rates, but lead to an increase in the number of V_{DD}-induced weak bitcells. To address this, a potential future research direction would be the investigation of light weight error correction codes capable of many multiple-adjacent-bit correction. For example, the extension of a BCH double error correcting code to include adjacent-bit error correction for one (or both) of its bit errors. Or, another alternative would be to consider multiple-byte or multiple-burst Reed-Solomon error correcting codes. This would allow for reliable soft error and weak bitcell protection in the ultra-low voltage SRAM domain while ensuring minimal area and performance overhead.

Another consideration for future research is in the validation of the assumptions made regarding the SERs for the vendor supplied SRAM bitcells. By obtaining access to the extracted layout views for these cells, larger capacity arrays could be implemented, irradiated, and compared to the data sets collected in this work.

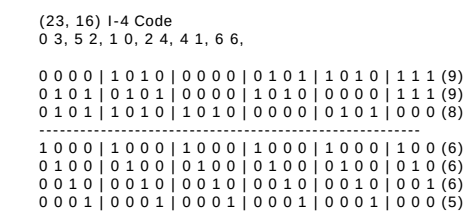
Finally, having successfully gone through the 28 nm full-custom test chip development cycle (and having developed several design automation components in the process) other designs can be implemented in this technology. This is useful for showing how a particular design behaves in light of aggressive technology scaling.

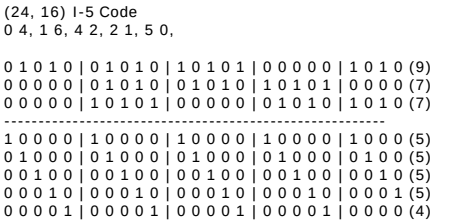
Appendix A

Implemented H-Matrices

A.1 16 Data-bit Codes

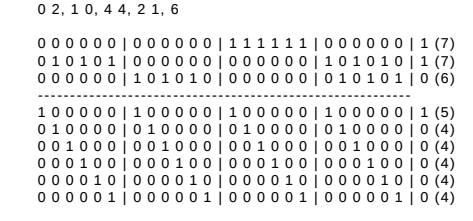
A.1.1 SEC-DED-DAEC-yAED

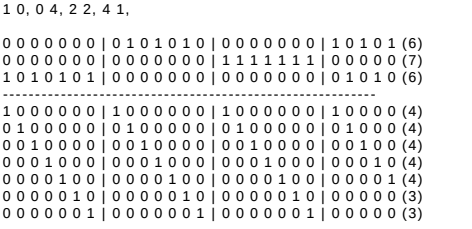




(a) SEC-DED-DAEC-5AED (23, 16) I-4 Code Check-bits(1-7) = Columns{17 15 20 1 10 3 12}

(b) SEC-DED-DAEC-7AED (24, 16) I-5 Code Check-bits(1-8) = Columns{11 12 8 1 22 3 24 5}





(c) SEC-DED-DAEC-9AED (25, 16) I-6 Code Check-bits(1-9) = Columns{13 2 9 1 8 3 10 5 12}

(d) SEC-DED-DAEC-11AED (26, 16) I-7 Code Check-bits(1-10) = Columns{22 16 3 8 2 10 4 12 6 14}

Figure A.1: 16 Data-bit SEC-DED-DAEC-yAED Codes

A.1.2 SEC-DED-TAEC-yAED

```
(23, 16) I-4 Code
0 3, 2 4, 4 1, 5 2, 1 0, 6 6,

0 0 0 0 | 0 1 0 1 | 1 0 1 0 | 1 0 1 0 | 0 0 0 0 | 1 1 1 (9)
0 1 0 1 | 1 0 1 0 | 0 0 0 0 | 0 1 0 1 | 0 0 0 0 | 1 1 1 (9)
0 1 0 1 | 0 0 0 0 | 0 1 0 1 | 1 0 1 0 | 1 0 1 0 | 0 0 0 (8)
-------------------------------------------------------
1 0 0 0 | 1 0 0 0 | 1 0 0 0 | 1 0 0 0 | 1 0 0 0 | 1 0 0 (6)
0 1 0 0 | 0 1 0 0 | 0 1 0 0 | 0 1 0 0 | 0 1 0 0 | 0 1 0 (6)
0 0 1 0 | 0 0 1 0 | 0 0 1 0 | 0 0 1 0 | 0 0 1 0 | 0 0 1 (6)
0 0 0 1 | 0 0 0 1 | 0 0 0 1 | 0 0 0 1 | 0 0 0 1 | 0 0 0 (5)
```

(a) SEC-DED-TAEC-4AED (23, 16) I-4 Code Check-bits(1-7) = Columns{9 14 19 1 18 3 20}

```
(24, 16) I-5 Code
4 6, 0 4, 1 0, 2 1, 5 2,

1 1 1 1 1 | 0 1 0 1 0 | 0 0 0 0 0 | 0 0 0 0 0 | 1 0 1 0 (9)
0 1 0 1 0 | 0 0 0 0 0 | 0 0 0 0 0 | 1 0 1 0 1 | 0 1 0 1 (7)
0 0 0 0 0 | 0 0 0 0 0 | 1 0 1 0 1 | 0 1 0 1 0 | 1 0 1 0 (7)
-------------------------------------------------------
1 0 0 0 0 | 1 0 0 0 0 | 1 0 0 0 0 | 1 0 0 0 0 | 1 0 0 0 (5)
0 1 0 0 0 | 0 1 0 0 0 | 0 1 0 0 0 | 0 1 0 0 0 | 0 1 0 0 (5)
0 0 1 0 0 | 0 0 1 0 0 | 0 0 1 0 0 | 0 0 1 0 0 | 0 0 1 0 (5)
0 0 0 1 0 | 0 0 0 1 0 | 0 0 0 1 0 | 0 0 0 1 0 | 0 0 0 1 (5)
0 0 0 0 1 | 0 0 0 0 1 | 0 0 0 0 1 | 0 0 0 0 1 | 0 0 0 0 (4)
```

(b) SEC-DED-TAEC-6AED (24, 16) I-5 Code Check-bits(1-8) = Columns{1 22 13 6 12 8 14 10}

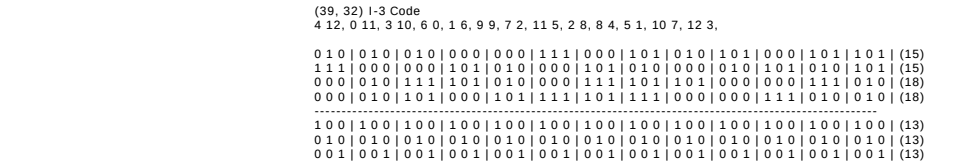
```
(25, 16) I-6 Code
2 0, 4 4, 1 2, 0 1, 6

0 0 0 0 0 0 | 1 1 1 1 1 1 | 0 0 0 0 0 0 | 0 0 0 0 0 0 | 1 (7)
1 0 1 0 1 0 | 0 0 0 0 0 0 | 0 1 0 1 0 1 | 0 0 0 0 0 0 | 1 (7)
0 0 0 0 0 0 | 0 0 0 0 0 0 | 1 0 1 0 1 0 | 0 1 0 1 0 1 | 0 (6)
-----------------------------------------------------------
1 0 0 0 0 0 | 1 0 0 0 0 0 | 1 0 0 0 0 0 | 1 0 0 0 0 0 | 1 (5)
0 1 0 0 0 0 | 0 1 0 0 0 0 | 0 1 0 0 0 0 | 0 1 0 0 0 0 | 0 (4)
0 0 1 0 0 0 | 0 0 1 0 0 0 | 0 0 1 0 0 0 | 0 0 1 0 0 0 | 0 (4)
0 0 0 1 0 0 | 0 0 0 1 0 0 | 0 0 0 1 0 0 | 0 0 0 1 0 0 | 0 (4)
0 0 0 0 1 0 | 0 0 0 0 1 0 | 0 0 0 0 1 0 | 0 0 0 0 1 0 | 0 (4)
0 0 0 0 0 1 | 0 0 0 0 0 1 | 0 0 0 0 0 1 | 0 0 0 0 0 1 | 0 (4)
```

(c) SEC-DED-TAEC-8AED (25, 16) I-6 Code Check-bits(1-9) = Columns{7 14 15 19 2 21 4 23}

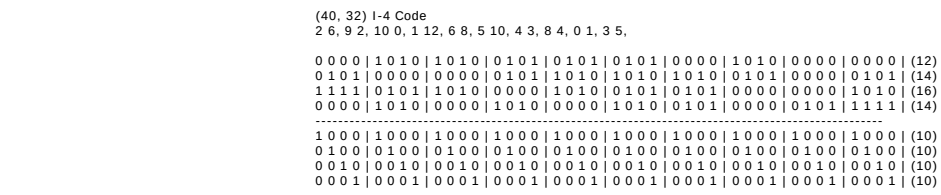
Figure A.2: 16 Data-bit SEC-DED-TAEC-yAED Codes

A.2 32 Data-bit Codes

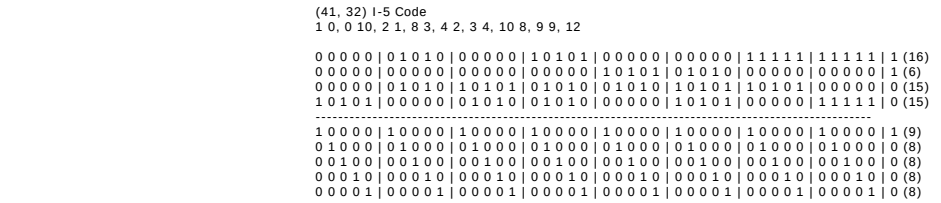
A.2.1 SEC-DED-DAEC-yAED



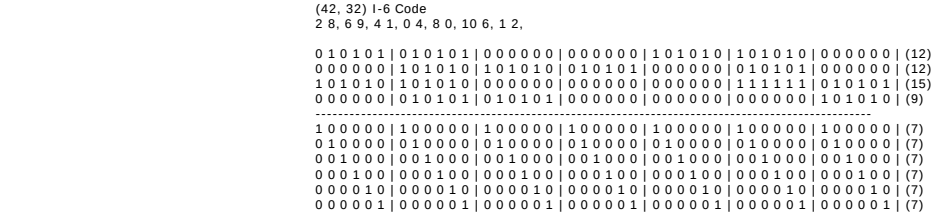
(a) SEC-DED-DAEC-3AED (39, 32) I-3 Code Check-bits(1-7) = Columns{28 29 27 13 4 11 6}



(b) SEC-DED-DAEC-5AED (40, 32) I-4 Code Check-bits(1-8) = Columns{29 30 3 36 33 10 35 12}



(c) SEC-DED-DAEC-7AED (41, 32) I-5 Code Check-bits(1-9) = Columns{32 23 24 5 6 2 8 4 10}



(d) SEC-DED-DAEC-9AED (42, 32) I-6 Code Check-bits(1-10) = Columns{25 20 3 16 19 26 21 28 23 30}

Figure A.3: 32 Data-bit SEC-DED-DAEC-yAED Codes

A.2.2 SEC-DED-TAEC-yAED

```
(40, 32) I-4 Code
2 10, 9 2, 8 5, 1 0, 6 8, 3 12, 5 3, 4 1, 0 4, 10 6,

0 1 0 1 | 1 0 1 0 | 1 0 1 0 | 0 0 0 0 | 0 1 0 1 | 0 1 0 1 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 1 0 1 0 | (12)
0 0 0 0 | 0 0 0 0 | 0 1 0 1 | 0 0 0 0 | 1 0 1 0 | 0 1 0 1 | 1 0 1 0 | 1 0 1 0 | 0 1 0 1 | 0 1 0 1 | (14)
1 1 1 1 | 0 1 0 1 | 0 0 0 0 | 0 0 0 0 | 1 0 1 0 | 1 0 1 0 | 0 1 0 1 | 0 0 0 0 | 0 0 0 0 | 1 1 1 1 | (16)
0 0 0 0 | 1 0 1 0 | 0 1 0 1 | 1 0 1 0 | 0 0 0 0 | 1 0 1 0 | 1 1 1 1 | 0 1 0 1 | 0 0 0 0 | 0 0 0 0 | (14)
-----------------------------------------------------------------------------------------------
1 0 0 0 | 1 0 0 0 | 1 0 0 0 | 1 0 0 0 | 1 0 0 0 | 1 0 0 0 | 1 0 0 0 | 1 0 0 0 | 1 0 0 0 | 1 0 0 0 | (10)
0 1 0 0 | 0 1 0 0 | 0 1 0 0 | 0 1 0 0 | 0 1 0 0 | 0 1 0 0 | 0 1 0 0 | 0 1 0 0 | 0 1 0 0 | 0 1 0 0 | (10)
0 0 1 0 | 0 0 1 0 | 0 0 1 0 | 0 0 1 0 | 0 0 1 0 | 0 0 1 0 | 0 0 1 0 | 0 0 1 0 | 0 0 1 0 | 0 0 1 0 | (10)
0 0 0 1 | 0 0 0 1 | 0 0 0 1 | 0 0 0 1 | 0 0 0 1 | 0 0 0 1 | 0 0 0 1 | 0 0 0 1 | 0 0 0 1 | 0 0 0 1 | (10)
```

(a) SEC-DED-TAEC-4AED (40, 32) I-4 Code Check-bits(1-8) = Columns{9 34 3 32 33 14 35 16}

```
(41, 32) I-5 Code
0 8, 5 5, 8 4, 3 0, 1 3, 6 1, 2 6, 4 2, 12

0 1 0 1 0 | 0 0 0 0 0 | 1 0 1 0 1 | 0 0 0 0 0 | 0 0 0 0 0 | 0 0 0 0 0 | 0 0 0 0 0 | 0 0 0 0 0 | 1 (6)
0 0 0 0 0 | 1 1 1 1 1 | 0 1 0 1 0 | 0 0 0 0 0 | 0 0 0 0 0 | 1 0 1 0 1 | 0 1 0 1 0 | 1 0 1 0 1 | 1 (16)
0 0 0 0 0 | 0 0 0 0 0 | 0 0 0 0 0 | 1 0 1 0 1 | 0 1 0 1 0 | 1 0 1 0 1 | 1 1 1 1 1 | 0 1 0 1 0 | 0 (15)
0 0 0 0 0 | 1 1 1 1 1 | 0 0 0 0 0 | 1 0 1 0 1 | 1 1 1 1 1 | 0 1 0 1 0 | 0 0 0 0 0 | 0 0 0 0 0 | 0 (15)
-----------------------------------------------------------------------------------------------
1 0 0 0 0 | 1 0 0 0 0 | 1 0 0 0 0 | 1 0 0 0 0 | 1 0 0 0 0 | 1 0 0 0 0 | 1 0 0 0 0 | 1 0 0 0 0 | 1 (9)
0 1 0 0 0 | 0 1 0 0 0 | 0 1 0 0 0 | 0 1 0 0 0 | 0 1 0 0 0 | 0 1 0 0 0 | 0 1 0 0 0 | 0 1 0 0 0 | 0 (8)
0 0 1 0 0 | 0 0 1 0 0 | 0 0 1 0 0 | 0 0 1 0 0 | 0 0 1 0 0 | 0 0 1 0 0 | 0 0 1 0 0 | 0 0 1 0 0 | 0 (8)
0 0 0 1 0 | 0 0 0 1 0 | 0 0 0 1 0 | 0 0 0 1 0 | 0 0 0 1 0 | 0 0 0 1 0 | 0 0 0 1 0 | 0 0 0 1 0 | 0 (8)
0 0 0 0 1 | 0 0 0 0 1 | 0 0 0 0 1 | 0 0 0 0 1 | 0 0 0 0 1 | 0 0 0 0 1 | 0 0 0 0 1 | 0 0 0 0 1 | 0 (8)
```

(b) SEC-DED-TAEC-6AED (41, 32) I-5 Code Check-bits(1-9) = Columns{11 12 33 29 1 17 3 19 5}

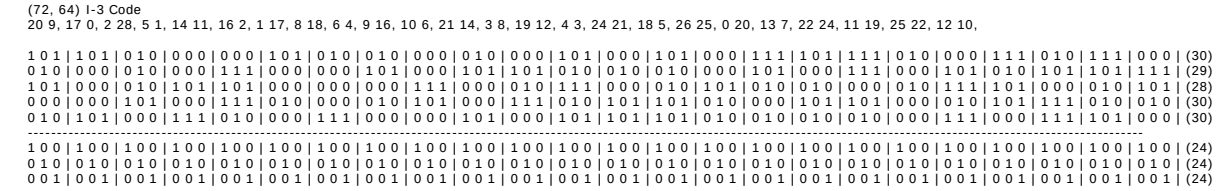
```
(42, 32) I-6 Code
0 4, 1 0, 2 8, 6 9, 4 1, 8 6, 10 2,

0 0 0 0 0 0 | 0 0 0 0 0 0 | 0 1 0 1 0 1 | 0 1 0 1 0 1 | 0 0 0 0 0 0 | 1 0 1 0 1 0 | 1 0 1 0 1 0 | (12)
0 1 0 1 0 1 | 0 0 0 0 0 0 | 0 0 0 0 0 0 | 1 0 1 0 1 0 | 1 0 1 0 1 0 | 0 1 0 1 0 1 | 0 0 0 0 0 0 | (12)
0 0 0 0 0 0 | 0 0 0 0 0 0 | 1 0 1 0 1 0 | 1 0 1 0 1 0 | 0 0 0 0 0 0 | 0 1 0 1 0 1 | 1 1 1 1 1 1 | (15)
0 0 0 0 0 0 | 1 0 1 0 1 0 | 0 0 0 0 0 0 | 0 1 0 1 0 1 | 0 1 0 1 0 1 | 0 0 0 0 0 0 | 0 0 0 0 0 0 | (9)
-----------------------------------------------------------------------------------------------
1 0 0 0 0 0 | 1 0 0 0 0 0 | 1 0 0 0 0 0 | 1 0 0 0 0 0 | 1 0 0 0 0 0 | 1 0 0 0 0 0 | 1 0 0 0 0 0 | (7)
0 1 0 0 0 0 | 0 1 0 0 0 0 | 0 1 0 0 0 0 | 0 1 0 0 0 0 | 0 1 0 0 0 0 | 0 1 0 0 0 0 | 0 1 0 0 0 0 | (7)
0 0 1 0 0 0 | 0 0 1 0 0 0 | 0 0 1 0 0 0 | 0 0 1 0 0 0 | 0 0 1 0 0 0 | 0 0 1 0 0 0 | 0 0 1 0 0 0 | (7)
0 0 0 1 0 0 | 0 0 0 1 0 0 | 0 0 0 1 0 0 | 0 0 0 1 0 0 | 0 0 0 1 0 0 | 0 0 0 1 0 0 | 0 0 0 1 0 0 | (7)
0 0 0 0 1 0 | 0 0 0 0 1 0 | 0 0 0 0 1 0 | 0 0 0 0 1 0 | 0 0 0 0 1 0 | 0 0 0 0 1 0 | 0 0 0 0 1 0 | (7)
0 0 0 0 0 1 | 0 0 0 0 0 1 | 0 0 0 0 0 1 | 0 0 0 0 0 1 | 0 0 0 0 0 1 | 0 0 0 0 0 1 | 0 0 0 0 0 1 | (7)
```

(c) SEC-DED-TAEC-8AED (42, 32) I-6 Code Check-bits(1-10) = Columns{31 2 15 28 1 8 3 10 5 12}

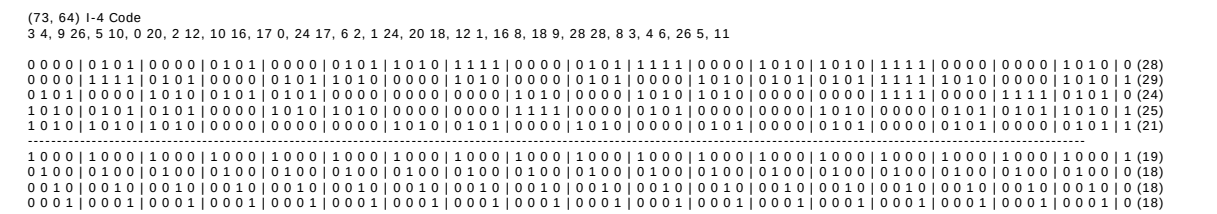
Figure A.4: 32 Data-bit SEC-DED-TAEC-yAED Codes

A.3 64 Data-bits

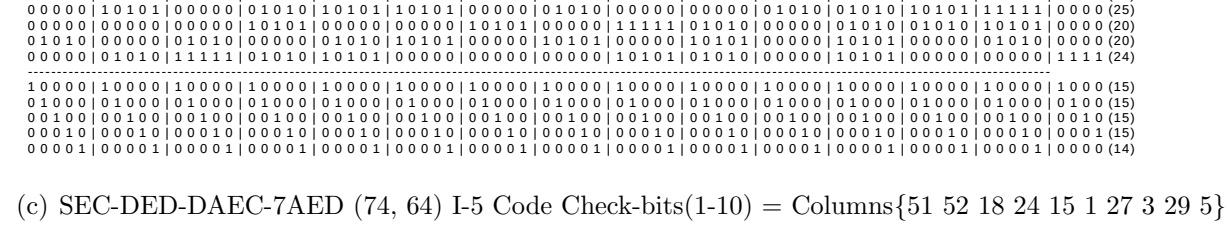
A.3.1 SEC-DED-DAEC-yAED



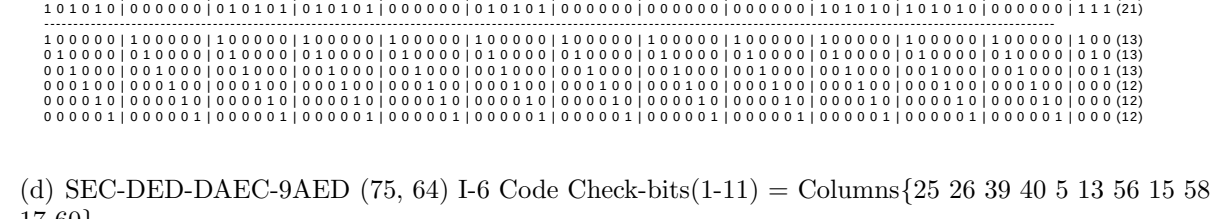
(a) SEC-DED-DAEC-3AED (72, 64) I-3 Code Check-bits(1-8) = Columns{16 38 45 7 11 55 5 57}



(b) SEC-DED-DAEC-5AED (73, 64) I-4 Code Check-bits(1-9) = Columns{49 50 67 36 37 13 26 15 28}



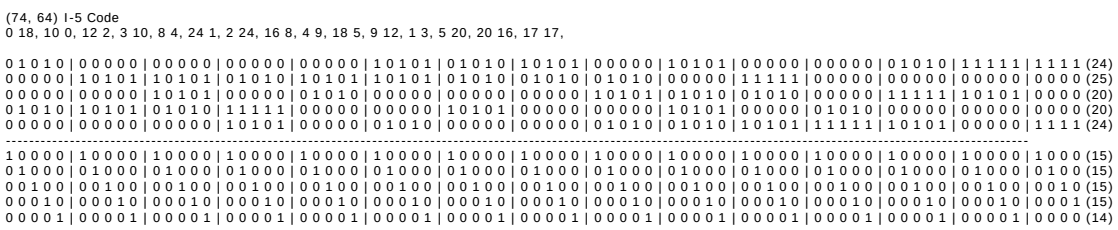
(c) SEC-DED-DAEC-7AED (74, 64) I-5 Code Check-bits(1-10) = Columns{51 52 18 24 15 1 27 3 29 5}



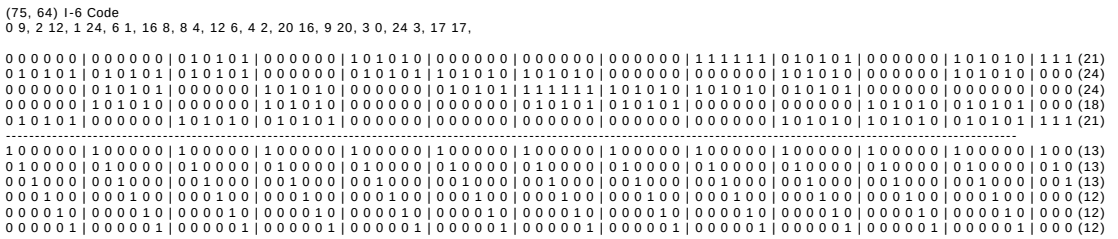
(d) SEC-DED-DAEC-9AED (75, 64) I-6 Code Check-bits(1-11) = Columns{25 26 39 40 5 13 56 15 58 17 60}

Figure A.5: 64 Data-bit SEC-DED-DAEC-yAED Codes

A.3.2 SEC-DED-TAEC-yAED



(a) SEC-DED-TAEC-6AED (74, 64) I-5 Code Check-bits(1-10) = Columns{36 37 43 14 60 1 7 3 9 5}



(b) SEC-DED-TAEC-8AED (75, 64) I-6 Code Check-bits(1-11) = Columns{25 26 45 11 24 1 62 3 64 5 66}

Figure A.6: 64 Data-bit SEC-DED-TAEC-yAED Codes

A.4 Increased Identity Matrix Size Codes

A.4.1 32 Data-bits

```
(41, 32) I-6 Code
3 0, 2 6, 7 1, 1 4, 0 2, 4 3, 6 7,

0 0 0 0 0 0 | 0 1 0 1 0 1 | 1 0 1 0 1 0 | 0 1 0 1 0 1 | 0 0 0 0 0 0 | 1 0 1 0 1 0 | 1 1 1 1 1 (17)
1 0 1 0 1 0 | 1 1 1 1 1 1 | 1 0 1 0 1 0 | 0 0 0 0 0 0 | 0 1 0 1 0 1 | 0 1 0 1 0 1 | 1 1 1 1 1 (23)
1 0 1 0 1 0 | 0 0 0 0 0 0 | 1 1 1 1 1 1 | 1 0 1 0 1 0 | 0 0 0 0 0 0 | 0 1 0 1 0 1 | 0 1 0 1 0 (17)
-----------------------------------------------------------------------------------------------
1 0 0 0 0 0 | 1 0 0 0 0 0 | 1 0 0 0 0 0 | 1 0 0 0 0 0 | 1 0 0 0 0 0 | 1 0 0 0 0 0 | 1 0 0 0 0 (7)
0 1 0 0 0 0 | 0 1 0 0 0 0 | 0 1 0 0 0 0 | 0 1 0 0 0 0 | 0 1 0 0 0 0 | 0 1 0 0 0 0 | 0 1 0 0 0 (7)
0 0 1 0 0 0 | 0 0 1 0 0 0 | 0 0 1 0 0 0 | 0 0 1 0 0 0 | 0 0 1 0 0 0 | 0 0 1 0 0 0 | 0 0 1 0 0 (7)
0 0 0 1 0 0 | 0 0 0 1 0 0 | 0 0 0 1 0 0 | 0 0 0 1 0 0 | 0 0 0 1 0 0 | 0 0 0 1 0 0 | 0 0 0 1 0 (7)
0 0 0 0 1 0 | 0 0 0 0 1 0 | 0 0 0 0 1 0 | 0 0 0 0 1 0 | 0 0 0 0 1 0 | 0 0 0 0 1 0 | 0 0 0 0 1 (7)
0 0 0 0 0 1 | 0 0 0 0 0 1 | 0 0 0 0 0 1 | 0 0 0 0 0 1 | 0 0 0 0 0 1 | 0 0 0 0 0 1 | 0 0 0 0 0 (6)
```

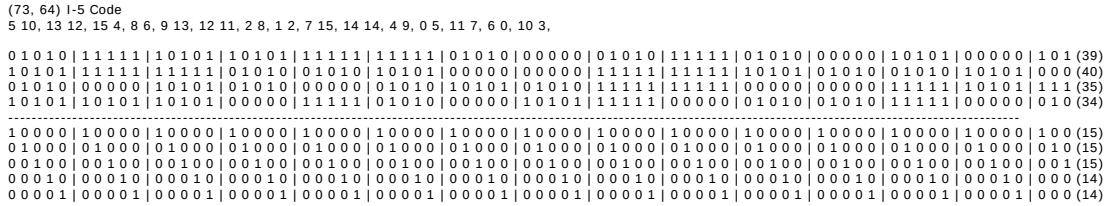
(a) SEC-DED-DAEC-9AED (41, 32) I-6 IIMS Code Check-bits(1-9) = Columns{31 26 21 25 2 27 4 29 6}

```
(42, 32) I-7 Code
1 0, 6 3, 5 2, 4 4, 2 1, 0 6,

0 0 0 0 0 0 0 | 1 0 1 0 1 0 1 | 1 0 1 0 1 0 1 | 1 1 1 1 1 1 1 | 0 0 0 0 0 0 0 | 0 1 0 1 0 1 0 | (18)
0 0 0 0 0 0 0 | 1 1 1 1 1 1 1 | 0 1 0 1 0 1 0 | 0 0 0 0 0 0 0 | 1 0 1 0 1 0 1 | 0 1 0 1 0 1 0 | (17)
1 0 1 0 1 0 1 | 0 1 0 1 0 1 0 | 1 0 1 0 1 0 1 | 0 0 0 0 0 0 0 | 0 1 0 1 0 1 0 | 0 0 0 0 0 0 0 | (14)
-----------------------------------------------------------------------------------------------
1 0 0 0 0 0 0 | 1 0 0 0 0 0 0 | 1 0 0 0 0 0 0 | 1 0 0 0 0 0 0 | 1 0 0 0 0 0 0 | 1 0 0 0 0 0 0 | (6)
0 1 0 0 0 0 0 | 0 1 0 0 0 0 0 | 0 1 0 0 0 0 0 | 0 1 0 0 0 0 0 | 0 1 0 0 0 0 0 | 0 1 0 0 0 0 0 | (6)
0 0 1 0 0 0 0 | 0 0 1 0 0 0 0 | 0 0 1 0 0 0 0 | 0 0 1 0 0 0 0 | 0 0 1 0 0 0 0 | 0 0 1 0 0 0 0 | (6)
0 0 0 1 0 0 0 | 0 0 0 1 0 0 0 | 0 0 0 1 0 0 0 | 0 0 0 1 0 0 0 | 0 0 0 1 0 0 0 | 0 0 0 1 0 0 0 | (6)
0 0 0 0 1 0 0 | 0 0 0 0 1 0 0 | 0 0 0 0 1 0 0 | 0 0 0 0 1 0 0 | 0 0 0 0 1 0 0 | 0 0 0 0 1 0 0 | (6)
0 0 0 0 0 1 0 | 0 0 0 0 0 1 0 | 0 0 0 0 0 1 0 | 0 0 0 0 0 1 0 | 0 0 0 0 0 1 0 | 0 0 0 0 0 1 0 | (6)
0 0 0 0 0 0 1 | 0 0 0 0 0 0 1 | 0 0 0 0 0 0 1 | 0 0 0 0 0 0 1 | 0 0 0 0 0 0 1 | 0 0 0 0 0 0 1 | (6)
```

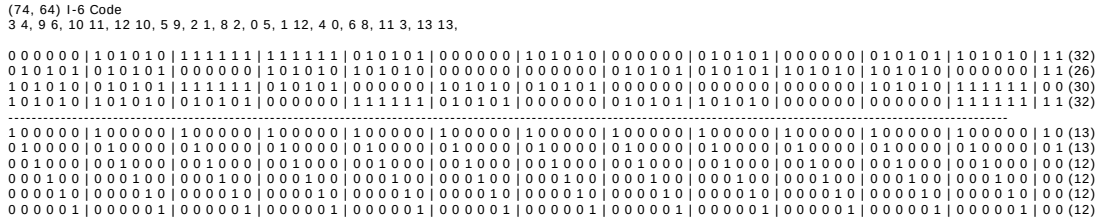
(b) SEC-DED-DAEC-11AED (42, 32) I-7 IIMS Code Check-bits(1-10) = Columns{22 16 3 36 2 38 4 40 6 42}

Figure A.7: 32 Data-bit SEC-DED-DAEC-yAED IIMS Codes

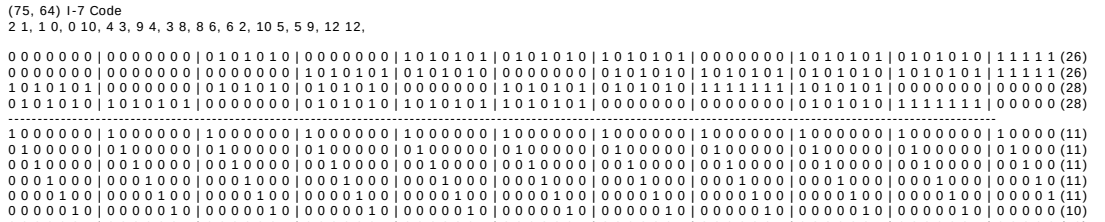
A.4.2 64 Data-bits



(a) SEC-DED-DAEC-7AED (73, 64) I-5 Code Check-bits(1-9) = Columns{16 12 33 40 56 67 58 69 60}



(b) SEC-DED-DAEC-9AED (74, 64) I-6 Code Check-bits(1-10) = Columns{37 2 33 34 43 56 45 58 47 60}



(c) SEC-DED-DAEC-11AED (75, 64) I-7 Code Check-bits(1-11) = Columns{42 30 3 4 15 9 17 11 19 13 21}

Figure A.8: 64 Data-bit SEC-DED-DAEC-yAED IIMS Codes

A.5 Increased Check-bit Codes

```
(24, 16) I-4 Code
0 8, 3 12, 2 4, 8 2, 4 0, 1 1,

0 1 0 1 | 0 1 0 1 | 0 0 0 0 | 1 0 1 0 | 0 0 0 0 | 0 0 0 0 | (6)
0 0 0 0 | 0 1 0 1 | 0 1 0 1 | 0 0 0 0 | 1 0 1 0 | 0 0 0 0 | (6)
0 0 0 0 | 1 0 1 0 | 1 0 1 0 | 0 1 0 1 | 0 0 0 0 | 0 0 0 0 | (6)
0 0 0 0 | 1 0 1 0 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 1 1 1 1 | (6)
-----------------------------------------------------------
1 0 0 0 | 1 0 0 0 | 1 0 0 0 | 1 0 0 0 | 1 0 0 0 | 1 0 0 0 | (6)
0 1 0 0 | 0 1 0 0 | 0 1 0 0 | 0 1 0 0 | 0 1 0 0 | 0 1 0 0 | (6)
0 0 1 0 | 0 0 1 0 | 0 0 1 0 | 0 0 1 0 | 0 0 1 0 | 0 0 1 0 | (6)
0 0 0 1 | 0 0 0 1 | 0 0 0 1 | 0 0 0 1 | 0 0 0 1 | 0 0 0 1 | (6)
```

(a) SEC-DED-DAEC-5AED (24, 16) I-4 ICB Code Check-bits(1-8) = Columns{13 10 11 24 1 18 3 20}

```
(25, 16) I-4 Code
16 8, 1 0, 2 16, 4 1, 0 4, 8 2, 24

1 0 1 0 | 0 0 0 0 | 0 1 0 1 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 1 (5)
0 1 0 1 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 1 0 1 0 | 1 (5)
0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 1 0 1 0 | 0 1 0 1 | 0 0 0 0 | 0 (4)
0 0 0 0 | 0 0 0 0 | 1 0 1 0 | 0 0 0 0 | 0 0 0 0 | 0 1 0 1 | 0 (4)
0 0 0 0 | 1 0 1 0 | 0 0 0 0 | 0 1 0 1 | 0 0 0 0 | 0 0 0 0 | 0 (4)
-----------------------------------------------------------
1 0 0 0 | 1 0 0 0 | 1 0 0 0 | 1 0 0 0 | 1 0 0 0 | 1 0 0 0 | 1 (7)
0 1 0 0 | 0 1 0 0 | 0 1 0 0 | 0 1 0 0 | 0 1 0 0 | 0 1 0 0 | 0 (6)
0 0 1 0 | 0 0 1 0 | 0 0 1 0 | 0 0 1 0 | 0 0 1 0 | 0 0 1 0 | 0 (6)
0 0 0 1 | 0 0 0 1 | 0 0 0 1 | 0 0 0 1 | 0 0 0 1 | 0 0 0 1 | 0 (6)
```

(b) SEC-DED-DAEC-5AED (25, 16) I-4 ICB Code Check-bits(1-9) = Columns{1 2 15 24 5 17 6 19 8}

Figure A.9: 16 Data-bit SEC-DED-DAEC-yAED ICB Codes

A.6 Optimized Code

```
(24, 16) I-5 Code
4 6 4 2 4 | 0 4 0 4 0 | 1 0 1 3 1 | 2 1 2 1 2 | 5 2 5 0

1 1 1 0 1 | 0 1 0 1 0 | 0 0 0 0 0 | 0 0 0 0 0 | 1 0 1 0 (8)
0 1 0 1 0 | 0 0 0 0 0 | 0 0 0 1 0 | 1 0 1 0 1 | 0 1 0 0 (7)
0 0 0 0 0 | 0 0 0 0 0 | 1 0 1 1 1 | 0 1 0 1 0 | 1 0 1 0 (8)
-----------------------------------------------------------
1 0 0 0 0 | 1 0 0 0 0 | 1 0 0 0 0 | 1 0 0 0 0 | 1 0 0 0 (5)
0 1 0 0 0 | 0 1 0 0 0 | 0 1 0 0 0 | 0 1 0 0 0 | 0 1 0 0 (5)
0 0 1 0 0 | 0 0 1 0 0 | 0 0 1 0 0 | 0 0 1 0 0 | 0 0 1 0 (5)
0 0 0 1 0 | 0 0 0 1 0 | 0 0 0 1 0 | 0 0 0 1 0 | 0 0 0 1 (5)
0 0 0 0 1 | 0 0 0 0 1 | 0 0 0 0 1 | 0 0 0 0 1 | 0 0 0 0 (4)
```

Figure A.10: Optimized (24, 16) I-5 Code with maximum row weight of 8 Check-bits(1-8) = Columns{1 4 13 6 12 8 24 10}

A.7 Dutta Codes

(22,16) SEC-DED-DAEC Dutta Code

```
1 1 0 0 1 1 0 0 0 1 1 1 0 1 0 0 | 1 0 0 0 0 0 (9)
0 0 0 1 0 1 0 0 1 1 0 1 1 0 0 1 | 0 1 0 0 0 0 (8)
1 0 1 0 1 0 0 1 1 1 1 0 0 0 1 1 | 0 0 1 0 0 0 (10)
1 0 0 1 0 0 1 1 1 0 0 1 0 1 1 0 | 0 0 0 1 0 0 (9)
0 1 1 0 1 0 1 1 0 0 0 0 1 1 0 1 | 0 0 0 0 1 0 (9)
0 1 1 1 0 1 1 0 0 0 1 0 1 0 1 0 | 0 0 0 0 0 1 (9)
```

(a) SEC-DED-DAEC (22, 16) Dutta Code Check-bits(1-6) = Columns{17 18 19 20 21 22}

(39,32) SEC-DED-DAEC Dutta Code

```
1 0 1 1 1 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 1 0 1 0 0 1 1 1 0 0 0 0 | 1 0 0 0 0 0 0 (15)
0 1 0 1 0 1 0 0 1 0 1 1 0 0 0 1 0 0 0 1 1 1 0 0 0 1 1 0 0 0 0 1 | 0 1 0 0 0 0 0 (14)
1 0 1 0 1 0 1 0 0 1 1 0 0 0 1 0 1 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 | 0 0 1 0 0 0 0 (15)
0 1 0 1 0 0 0 1 1 1 0 0 0 1 0 1 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 1 | 0 0 0 1 0 0 0 (15)
0 0 1 0 0 1 1 0 0 0 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0 0 0 0 1 1 1 0 | 0 0 0 0 1 0 0 (14)
1 0 0 0 0 1 0 0 1 0 0 1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0 1 0 0 1 1 | 0 0 0 0 0 1 0 (15)
0 1 0 0 1 0 1 1 0 0 1 0 1 1 0 0 1 0 1 0 0 1 0 1 0 0 1 1 1 0 0 0 | 0 0 0 0 0 0 1 (15)
```

(b) SEC-DED-DAEC (39, 32) Dutta Code Check-bits(1-7) = Columns{33 34 35 36 37 38 39}

(72, 64) SEC-DED-DAEC Dutta Code

```
1 0 0 1 0 1 0 1 1 1 0 1 0 1 1 0 0 1 0 1 0 0 1 0 0 1 0 0 0 0 1 0 1 1 0 0 0 1 0 0 1 1 0 0 1 0 0 0 1 1 0 1 0 0 0 0 1 1 1 0 0 0 0 0 | 1 0 0 0 0 0 0 0 (27)
1 0 1 1 1 0 0 1 1 0 0 0 0 0 1 0 1 0 1 0 0 1 0 0 1 0 0 0 0 1 0 1 1 0 0 0 1 0 0 1 1 0 0 1 0 0 0 1 1 0 1 0 0 0 0 1 1 1 0 0 0 0 0 1 | 0 1 0 0 0 0 0 0 (26)
0 1 1 0 1 0 1 0 1 0 1 1 1 0 1 0 1 1 0 0 1 0 0 1 0 0 0 0 1 0 1 1 0 0 0 1 0 0 1 1 0 0 1 0 0 0 1 1 0 1 0 0 0 0 1 1 1 0 0 0 0 0 1 1 | 0 0 1 0 0 0 0 0 (29)
1 1 1 1 0 1 1 0 0 1 0 1 1 0 0 1 1 0 0 1 0 0 1 0 1 0 0 1 0 1 1 0 0 0 1 0 0 1 1 0 0 1 0 0 0 1 1 0 1 0 0 0 0 1 1 0 0 0 0 0 0 1 1 1 | 0 0 0 1 0 0 0 0 (30)
1 0 0 1 0 1 1 0 0 0 1 1 1 1 0 1 1 0 1 0 0 1 0 1 0 0 1 0 1 1 0 0 0 1 0 0 1 1 0 0 1 0 0 0 1 1 0 0 0 0 0 0 1 1 0 1 0 0 0 0 1 1 1 0 | 0 0 0 0 1 0 0 0 (29)
1 1 1 0 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 0 0 1 0 1 0 0 1 0 1 1 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 1 1 0 1 0 0 0 0 1 1 1 0 0 | 0 0 0 0 0 1 0 0 (32)
0 1 1 0 1 0 1 1 0 1 1 0 1 1 1 1 0 0 0 1 0 1 0 0 1 0 1 1 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 1 1 0 0 1 0 0 0 1 1 0 1 0 0 0 0 1 1 1 0 0 0 | 0 0 0 0 0 0 1 0 (29)
0 1 0 1 1 1 1 0 1 1 1 0 0 1 1 1 0 0 1 0 1 0 0 0 1 0 1 1 0 0 0 0 1 0 1 1 0 0 0 1 0 0 1 1 0 0 1 0 0 0 1 1 0 1 0 0 0 0 1 1 1 0 0 0 0 | 0 0 0 0 0 0 0 1 (30)
```

(c) SEC-DED-DAEC (72, 64) Dutta Code Check-bits(1-8) = Columns{65 66 67 68 69 70 71 72}

Figure A.11: Various Length Dutta Codes

A.8 BCH Codes

BCH2 (26, 16)

1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 0 1 0 0 0 0 1 0 1 (8)
0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 0 1 0 0 0 0 1 0 (7)
0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 0 1 0 0 0 0 1 (7)
0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 0 1 0 0 0 0 (6)
0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 1 1 0 1 1 0 1 (9)
0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 1 1 0 1 1 0 1 1 0 0 1 1 (11)
0 0 0 0 0 0 1 0 0 0 1 1 1 0 1 0 0 0 0 1 0 1 1 1 0 0 (9)
0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 0 1 0 0 0 0 1 0 1 1 1 0 (9)
0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 0 1 0 0 0 0 1 0 1 1 1 (9)
0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 0 1 0 0 0 0 1 0 1 1 (8)

(a) (26, 16) BCH DEC Code Check-bits(1-10) = Columns{1 2 3 4 5 6 7 8 9 10}

BCH DEC (44, 32)

1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 1 0 0 1 1 0 1 0 0 0 1 1 0 0 0 1 1 0 1 1 1 1 1 1 0 0 (19)
0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 1 0 0 1 1 0 1 0 0 0 1 1 0 0 0 1 1 0 1 1 1 1 1 1 1 0 (19)
0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 0 1 1 1 1 1 0 0 1 0 1 1 1 1 0 1 1 1 0 1 0 0 0 0 1 1 (21)
0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 0 1 1 1 1 1 0 0 1 0 1 1 1 1 0 1 1 1 0 1 0 0 0 0 1 (20)
0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 1 0 1 0 0 0 1 1 1 1 0 1 1 0 0 1 0 1 1 0 0 (15)
0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 1 0 1 0 0 0 1 1 1 1 0 1 1 0 0 1 0 1 1 0 (15)
0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 1 0 1 0 0 0 1 1 1 1 0 1 1 0 0 1 0 1 1 (15)
0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 1 1 0 1 0 1 0 0 1 0 0 1 0 0 1 1 0 0 1 1 0 0 1 (14)
0 0 0 0 0 0 0 0 1 0 0 0 1 1 0 1 1 0 1 0 0 1 0 1 1 1 0 1 1 1 1 0 0 0 1 0 0 0 1 1 0 0 0 0 (17)
0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 1 0 0 1 1 0 1 0 1 0 0 0 1 1 0 0 0 1 1 0 1 1 1 1 1 1 1 0 0 1 0 0 (19)
0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 1 0 0 1 1 0 1 0 1 0 0 0 1 1 0 0 0 1 1 0 1 1 1 1 1 1 1 0 0 1 0 (19)
0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 1 0 0 1 1 0 1 0 1 0 0 0 1 1 0 0 0 1 1 0 1 1 1 1 1 1 1 0 0 1 (19)

(b) (44, 32) BCH DEC Code Check-bits(1-12) = Columns{1 2 3 4 5 6 7 8 9 10 11 12}

BCH DEC (78, 64)

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 0 0 0 1 0 1 0 1 0 0 0 0 1 1 0 1 1 0 1 0 0 1 0 1 0 1 0 1 0 1 0 0 1 0 1 0 1 0 1 0 0 0 1 0 1 1 (32)
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 0 0 0 1 0 1 0 1 0 0 0 0 1 1 0 1 1 0 1 0 0 1 0 1 0 1 0 1 0 1 0 0 1 0 1 0 1 0 1 0 0 0 1 0 1 (31)
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 0 0 0 1 0 1 0 1 0 0 0 0 1 1 0 1 1 0 1 0 0 1 0 1 0 1 0 1 0 1 0 0 1 0 1 0 1 0 1 0 0 0 1 0 (30)
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 0 0 0 1 0 1 0 1 0 0 0 0 1 1 0 1 1 0 1 0 0 1 0 1 0 1 0 1 0 1 0 0 1 0 1 0 1 0 1 0 0 0 1 (30)
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 0 0 0 1 0 1 0 1 0 0 0 0 1 1 0 1 1 0 1 0 0 1 0 1 0 1 0 1 0 1 0 0 1 0 1 0 1 0 1 0 0 0 (29)
0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 1 0 0 1 0 1 1 1 1 1 0 1 1 0 0 0 0 0 1 1 0 0 1 0 0 1 0 1 0 1 1 1 0 0 1 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 1 1 0 1 1 1 1 1 (34)
0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 1 1 0 1 0 1 1 0 0 1 1 1 1 1 1 1 0 1 0 0 1 0 0 1 0 0 1 1 1 1 1 1 0 1 0 0 0 0 0 1 0 1 1 0 1 0 1 1 1 0 1 0 1 1 1 0 0 1 0 0 (38)
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 1 1 0 1 1 0 0 1 1 1 1 1 1 1 0 1 0 0 1 0 0 1 0 0 1 1 1 1 1 1 0 1 0 0 0 0 0 1 0 1 1 0 1 0 1 1 1 0 1 0 1 1 1 0 0 1 0 (38)
0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 1 1 1 0 0 1 0 1 0 0 1 0 1 0 0 1 1 1 0 0 1 0 1 1 0 0 0 0 1 0 1 0 1 0 0 1 1 1 0 1 0 1 0 1 0 0 0 0 1 1 1 0 0 1 0 0 0 0 1 1 0 0 1 0 (31)
0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 1 0 1 0 0 1 1 1 0 0 1 0 1 1 0 0 0 0 0 1 0 1 0 1 0 0 1 1 1 0 1 0 1 0 1 0 0 0 0 1 1 1 0 0 1 0 0 0 0 1 1 0 0 1 0 (32)
0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 1 1 1 0 1 0 0 0 0 1 0 0 1 0 1 1 0 1 1 0 0 1 0 0 0 0 0 0 1 1 1 1 0 1 0 1 0 1 0 1 1 1 1 0 1 0 1 1 0 1 1 0 0 0 0 1 0 (31)
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 1 1 1 0 1 0 1 0 0 1 0 0 1 0 1 1 0 0 1 1 1 0 0 0 0 0 0 1 1 1 1 0 1 0 1 0 1 0 1 0 1 1 1 0 1 0 1 1 0 0 1 0 0 0 0 1 (31)
0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 0 1 1 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 0 1 1 1 1 0 0 0 1 0 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 1 1 1 0 1 1 (41)
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 0 0 0 1 0 1 0 0 0 0 1 1 0 1 1 0 1 0 0 1 0 1 0 1 0 1 0 1 0 0 1 0 1 0 1 0 0 0 1 0 1 1 0 (31)

(c) (78, 64) BCH DEC Code Check-bits(1-14) = Columns{1 2 3 4 5 6 7 8 9 10 11 12 13 14}

Figure A.12: Various Length BCH Codes

A.9 Reed Solomon Codes

RS(25,16) S3EC-D3ED

```
1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 (7)
0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 (6)
0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 (6)
1 0 0 0 0 1 0 1 0 1 0 1 0 1 1 1 0 0 0 1 0 0 0 0 0 (9)
0 1 0 1 0 1 0 1 1 1 1 1 1 1 0 1 0 0 0 0 1 0 0 0 0 (12)
0 0 1 0 1 0 1 0 1 0 1 1 1 1 1 1 0 0 0 0 0 1 0 0 0 (11)
1 0 0 0 1 0 0 1 1 1 1 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 (9)
0 1 0 0 1 1 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0 0 0 1 0 (10)
0 0 1 1 0 1 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 (9)
```

(a) S3EC-D3ED (25, 16) RS Code Check-bits(1-9) = Columns{17 18 19 20 21 22 23 24 25}

RS(44,32) S4EC-D4ED

```
1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 (9)
0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 (9)
0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 (9)
0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 (9)
1 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 (14)
0 1 0 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 0 0 1 0 1 1 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 (18)
0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 (16)
0 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 (15)
1 0 0 0 0 0 1 0 1 0 0 1 0 1 1 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 (18)
0 1 0 0 0 0 1 1 1 0 1 0 1 0 1 0 1 1 1 1 1 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 (17)
0 0 1 0 1 0 0 1 0 1 1 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 (18)
0 0 0 1 0 1 0 0 0 0 1 1 1 1 0 1 0 1 0 1 0 1 1 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 (17)
```

(b) S4EC-D4ED (44, 32) RS Code Check-bits(1-12) = Columns{33 34 35 36 37 38 39 40 41 42 43 44}

RS(79,64) S5EC-D5ED

```
1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 (14)
0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 (14)
0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 (14)
0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 (14)
0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 (13)
1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 1 0 0 1 0 0 0 1 0 1 0 1 0 1 1 1 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 (27)
0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 1 0 0 1 0 0 0 1 0 1 0 1 0 1 1 1 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 (25)
0 0 1 0 0 0 1 0 0 1 1 0 0 1 0 0 0 1 0 1 0 1 0 1 1 1 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 (38)
0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 1 0 0 1 0 0 0 0 1 0 1 0 1 0 1 1 1 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 1 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 (34)
0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 1 0 0 1 0 0 0 1 0 1 0 1 0 1 1 1 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 1 0 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 (30)
1 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 1 0 1 1 0 1 1 0 1 1 0 0 1 0 0 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0 0 1 0 0 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 (30)
0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 1 1 0 1 1 0 0 1 0 0 1 1 0 1 1 1 1 1 1 1 1 0 1 1 0 0 0 0 0 0 0 1 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 (27)
0 0 1 0 0 1 0 0 1 0 0 1 0 1 1 1 0 1 1 0 0 1 0 0 1 1 0 1 1 1 1 1 1 1 1 0 0 1 0 0 0 0 0 0 1 1 0 1 1 0 1 1 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 (32)
0 0 0 1 0 0 1 0 0 1 0 0 1 0 1 1 1 0 1 1 0 1 1 0 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0 0 1 0 0 1 1 0 1 1 1 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 (31)
0 0 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 1 1 1 0 1 1 0 0 1 0 0 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 0 0 0 0 0 0 0 1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 (29)
```

(c) S5EC-D5ED (79, 64) RS Code Check-bits(1-15) = Columns{65 66 67 68 69 70 71 72 73 74 75 76 77 78 79}

Figure A.13: Various Length Reed Solomon Codes

Appendix B

Details of Test Chip

75 kb 6T SRAM Macro with a Modified (75, 64)-I_6 SEC-DED-TAEC-8AED Error Correction Code

Technology: 28nm HK+MG bulk LP-CMOS

CMC Run Code: 1302CT

Design Name: ICTWTAN3

Die Area Proper: 1.230mm * 1.230mm = 1.513mm^2

Die Area with 57μm Scribe Lines: 1.287mm * 1.287mm = 1.656mm^2

Tape-out Date: May 28th, 2013

Packaging by: Corwil Technology Corporation and Quik-Pak

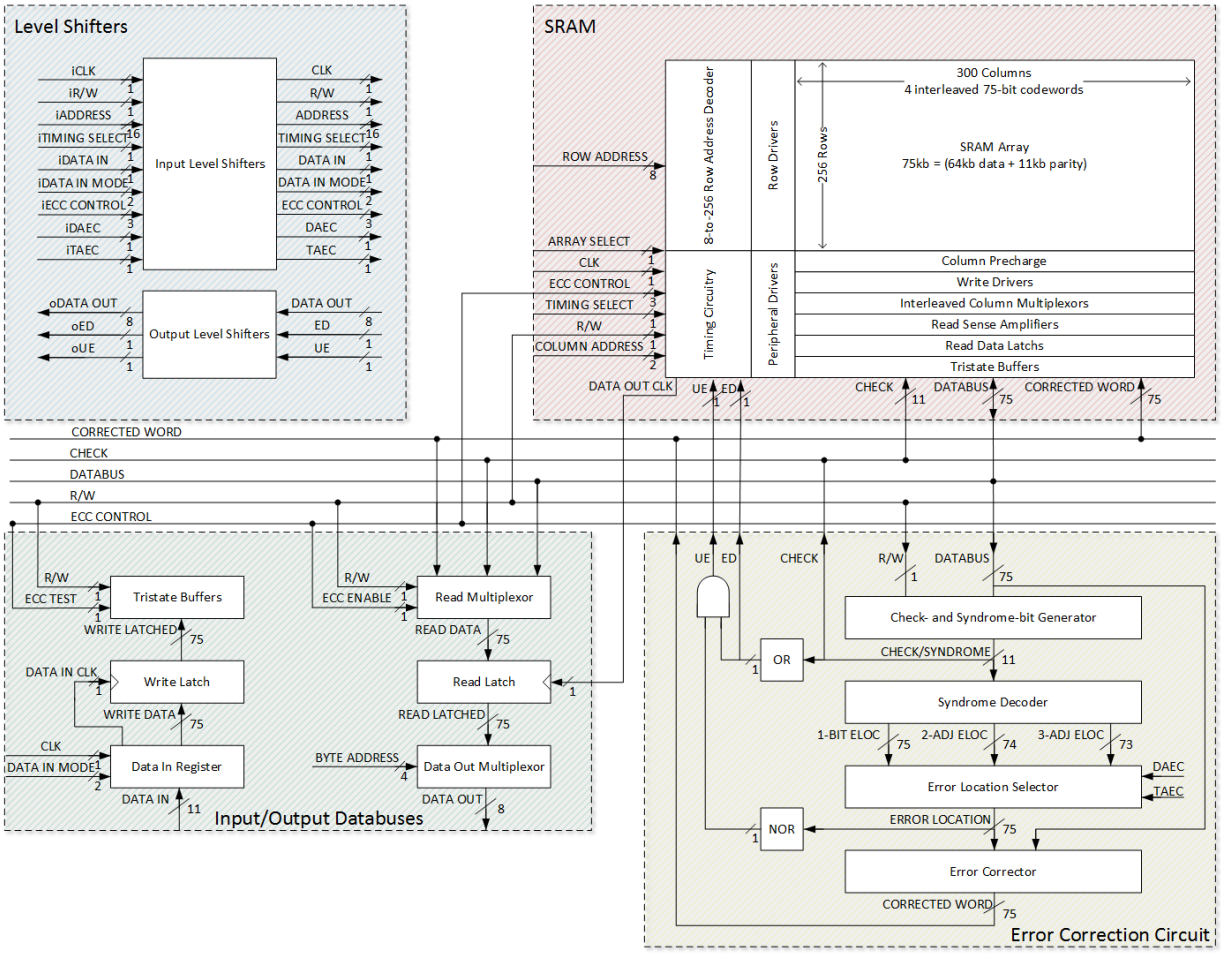


Figure B.1: ICTWTAN3 - 6T SRAM + ECC + I/O + Level Shifter Circuit Level Block Diagram

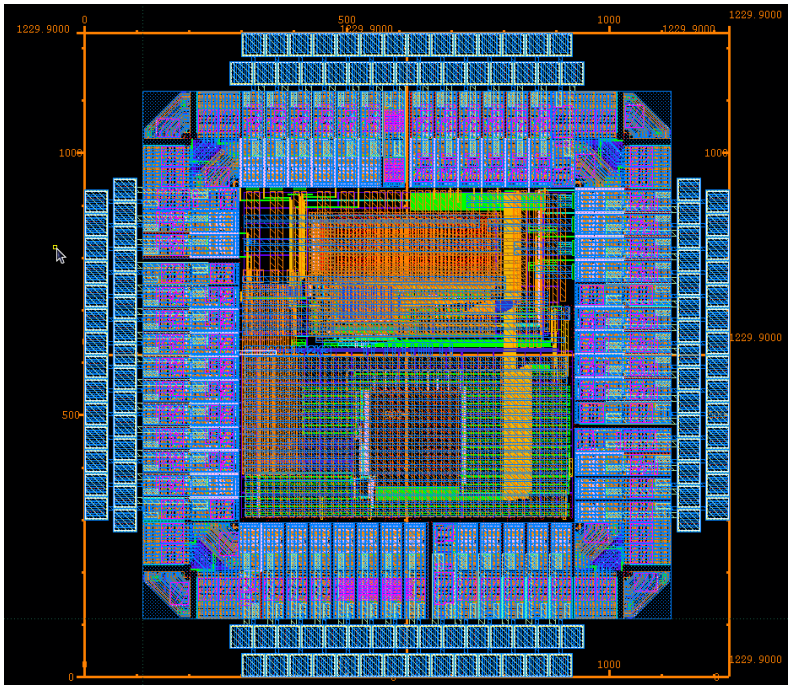


Figure B.2: ICTWTAN3 - Full Chip Layout

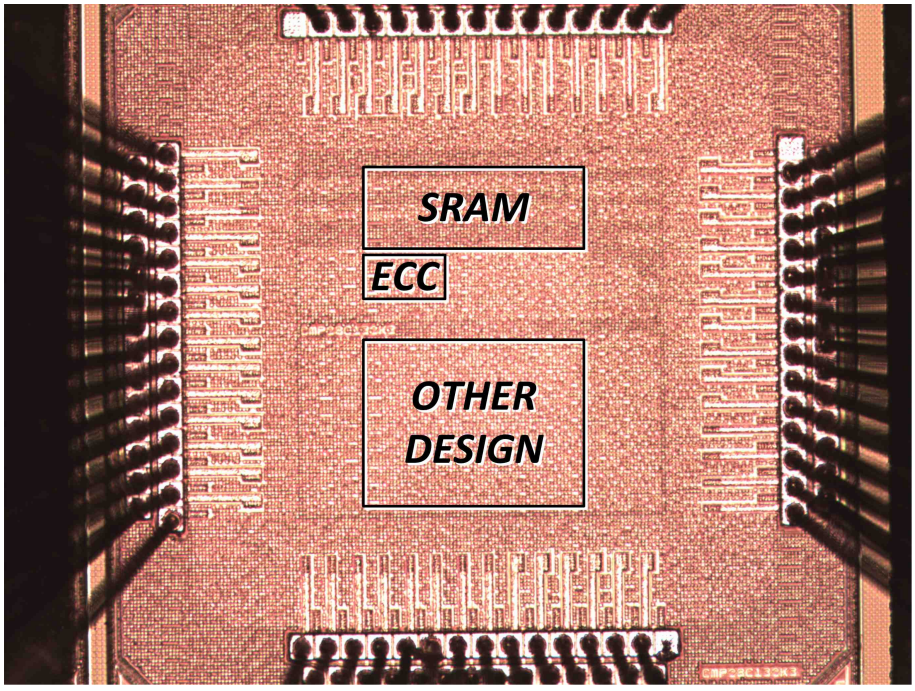


Figure B.3: ICTWTAN3 - Die Photo

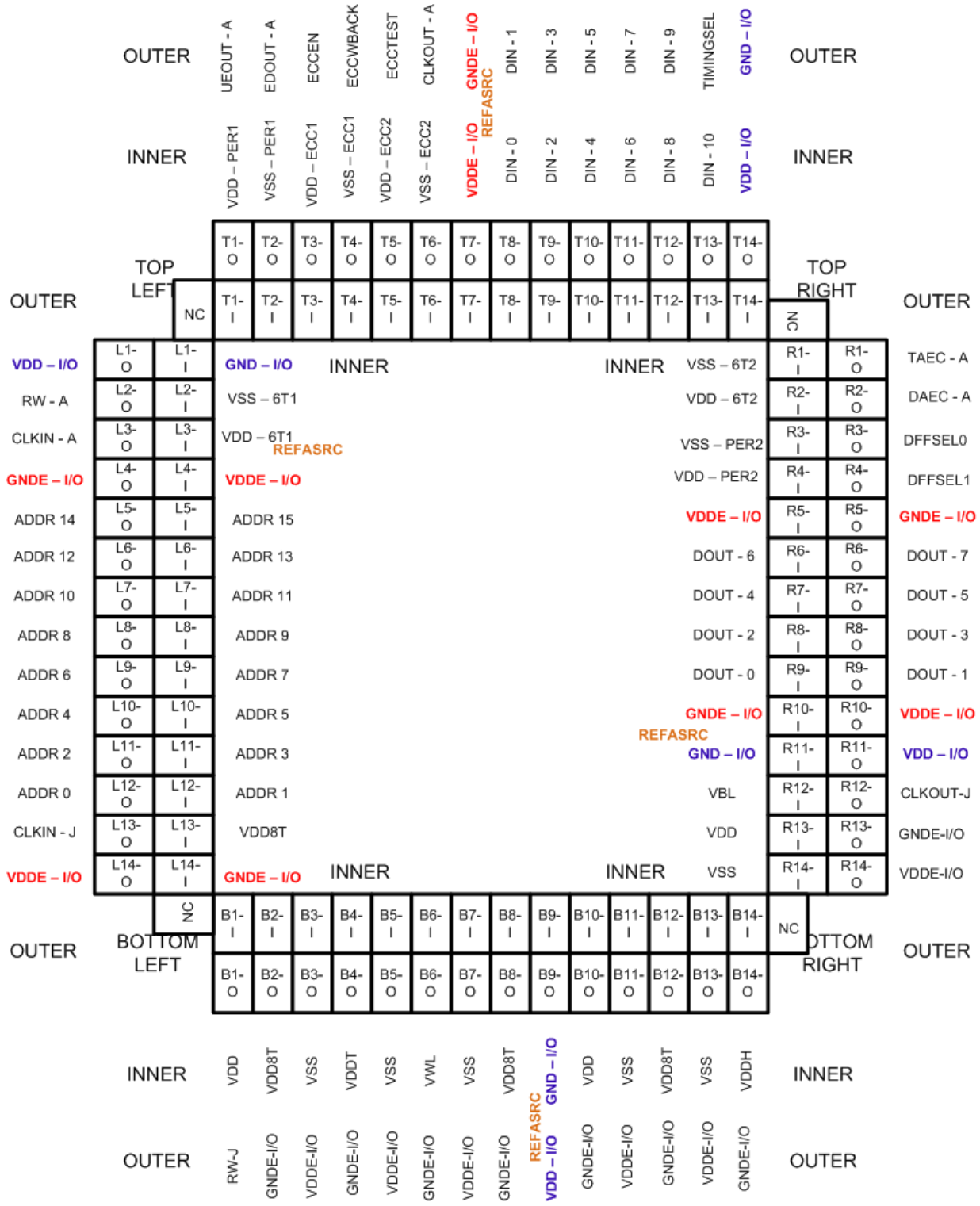


Figure B.4: ICTWTAN3 - Die Pad Frame Connectivity

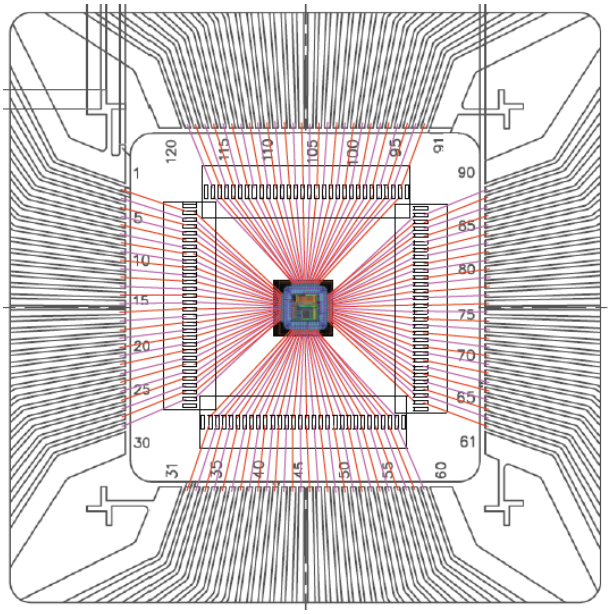


Figure B.5: ICTWTAN3 - Bonding Diagram - Two tiered bonding using two short wires and conductive interposers, die in center of package cavity. Bonding by Corwil Technology Corportation [79]

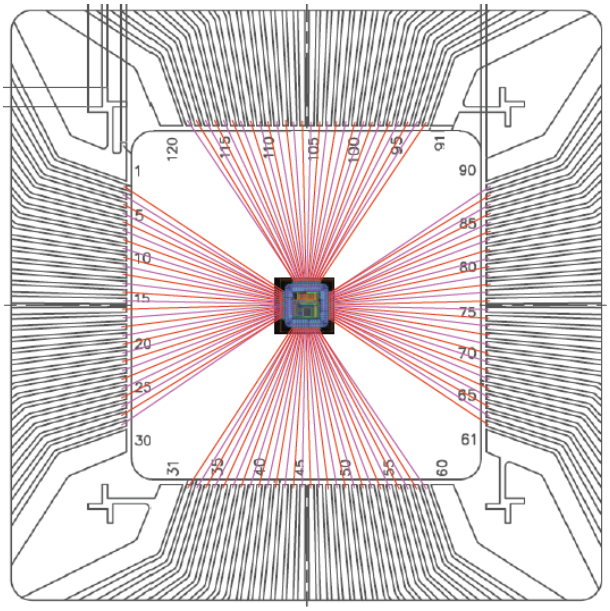


Figure B.6: ICTWTAN3 - Bonding Diagram - Two tiered bonding using single long wire, die in center of package cavity. Bonding by Quik-Pak [78]

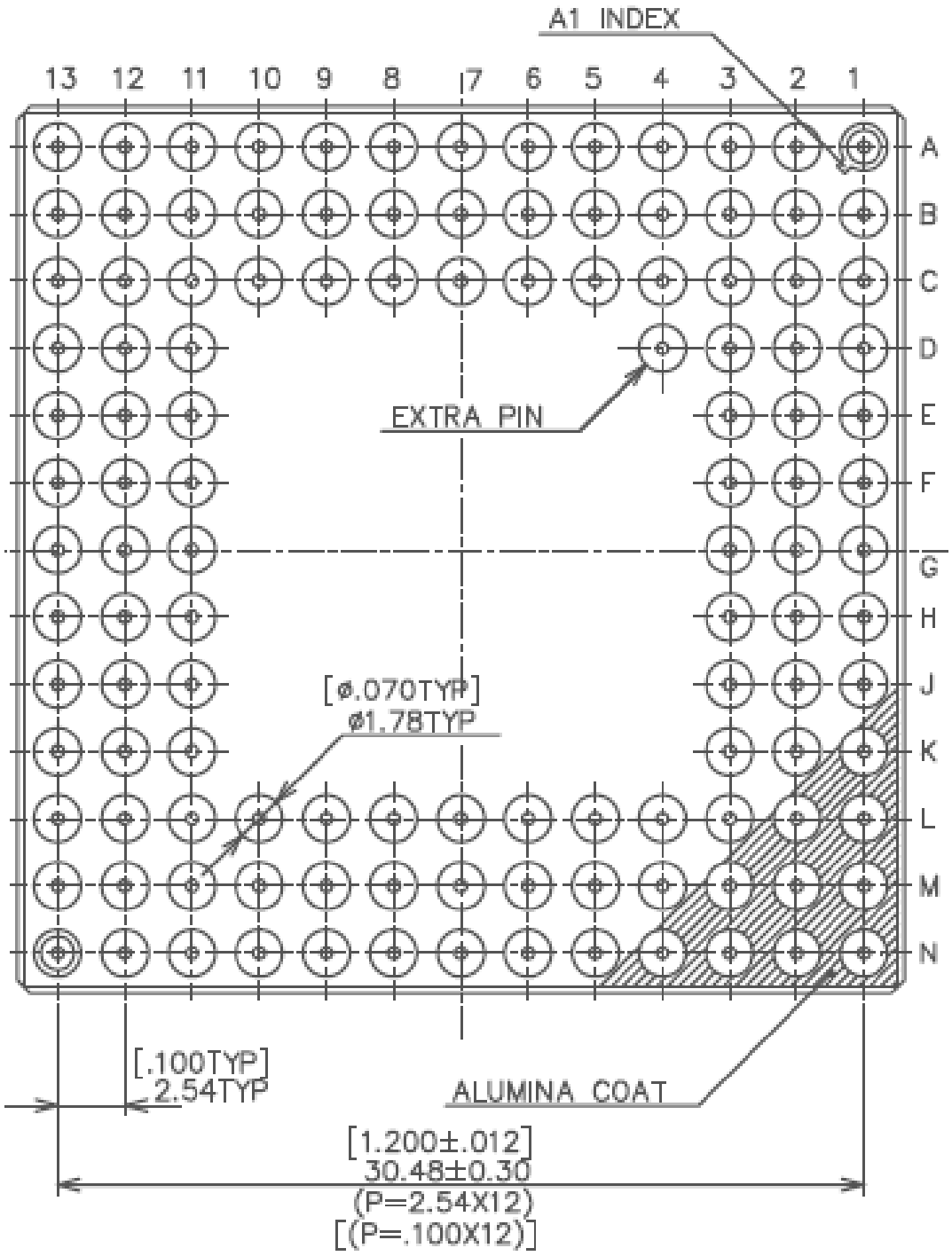


Figure B.7: ICTWTAN3 - 120 Pin Grid Array (PGA) Package Pin Configuration

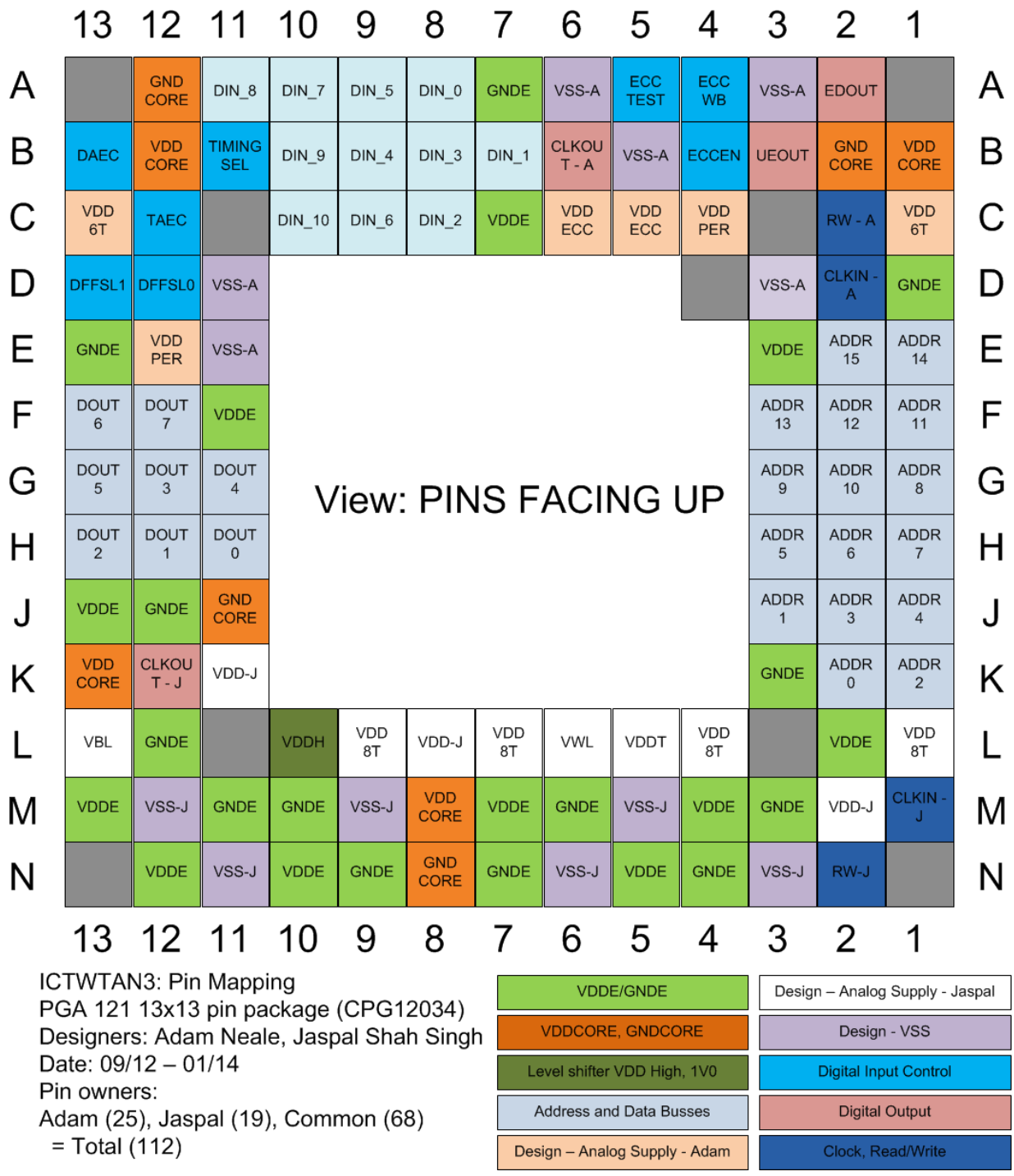


Figure B.8: ICTWTAN3 - Pin Map

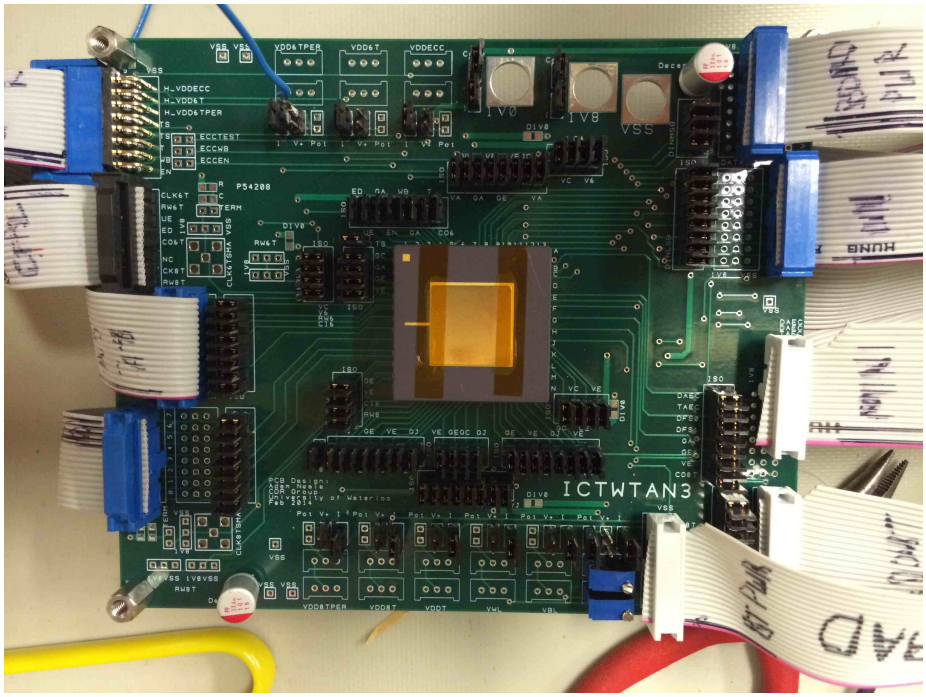


Figure B.9: ICTWTAN3 - Testboard for Device Under Test

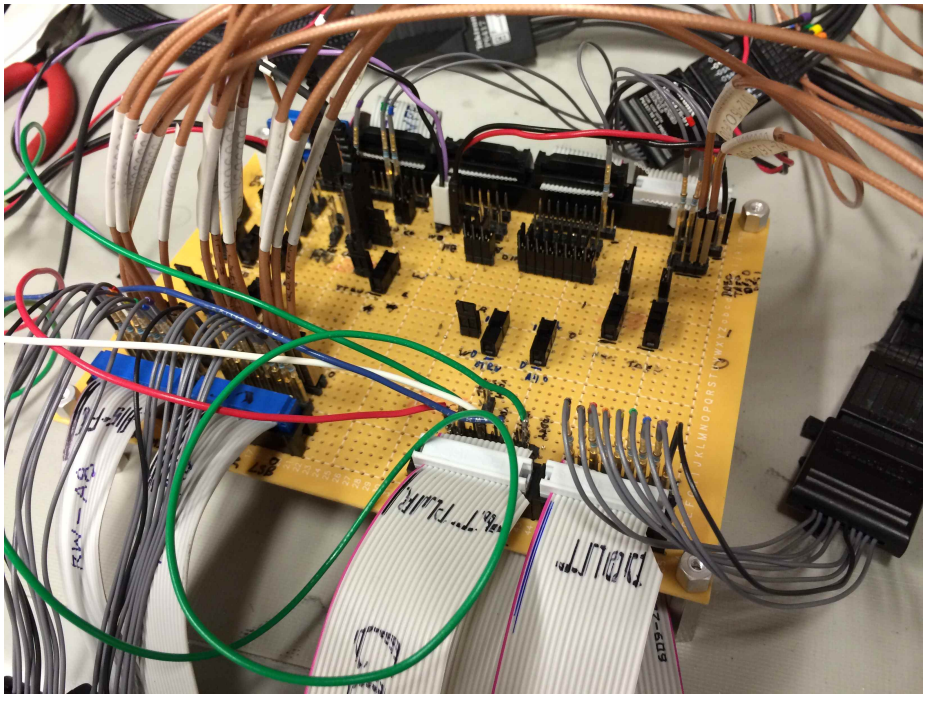


Figure B.10: ICTWTAN3 - External Controller Card for Device Under Test

Table B.1: Pin Description for ICTWTAN3 120 PGA Test Chip - Pins 120-61

Pin Location	Pin Number	Pad Side	Pad Ring	Pad Number	Pin Name	Description	Direction	Type
A1	120	-	-	-	NC	No Connect	-	-
B3	119	Top	Outer	1	UEOUT	Uncorrectable Error Flag	Output	Digital
C4	118	Top	Inner	1	VDDPER	VDD Periphreal Circuits, 1V0 Nominal	Power	Analog
A2	117	Top	Outer	2	EDOUT	Error Detected Flag	Output	Digital
A3	116	Top	Inner	2	VSS-A	Adam Ground	Power	Digital
B4	115	Top	Outer	3	ECCEN	ECC Enable	Input	Digital
C5	114	Top	Inner	3	VDDECC	VDD Error Correction Block	Power	Analog
A4	113	Top	Outer	4	ECCWRITEBACK	ECC Writeback Enable	Input	Digital
B5	112	Top	Inner	4	VSS-A	Adam Ground	Power	Digital
A5	111	Top	Outer	5	ECCTEST	ECC Unit Test Mode Enable	Input	Digital
C6	110	Top	Inner	5	VDDECC	VDD Error Correction Block, 1V0 Nominal	Power	Analog
B6	109	Top	Outer	6	CLKOUT-A	Clock Out - Adam	Output	Digital
A6	108	Top	Inner	6	VSS-A	Adam Ground	Power	Digital
A7	107	Top	Outer	7	GNDE-I/O	IO Ring Ground	Power	Digital
C7	106	Top	Inner	7	VDDE-I/O	IO Ring VDD 1V8	Power	Digital
B7	105	Top	Outer	8	DIN< 1 >	Data Input < 1 >	Input	Digital
A8	104	Top	Inner	8	DIN< 0 >	Data Input < 0 >	Input	Digital
B8	103	Top	Outer	9	DIN< 3 >	Data Input < 3 >	Input	Digital
C8	102	Top	Inner	9	DIN< 2 >	Data Input < 2 >	Input	Digital
A9	101	Top	Outer	10	DIN< 5 >	Data Input < 5 >	Input	Digital
B9	100	Top	Inner	10	DIN< 4 >	Data Input < 4 >	Input	Digital
A10	99	Top	Outer	11	DIN< 7 >	Data Input < 7 >	Input	Digital
C9	98	Top	Inner	11	DIN< 6 >	Data Input < 6 >	Input	Digital
B10	97	Top	Outer	12	DIN< 9 >	Data Input < 9 >	Input	Digital
A11	96	Top	Inner	12	DIN< 8 >	Data Input < 8 >	Input	Digital
B11	95	Top	Outer	13	TIMINGSEL	TIMING SELECT (1V0, 400mV)	Input	Digital
C10	94	Top	Inner	13	DIN< 10 >	Data Input < 10 >	Input	Digital
A12	93	Top	Outer	14	GND-I/O	Core Ground in I/O	Power	Digital
B12	92	Top	Inner	14	VDD-I/O	Core VDD in I/O, 1V0	Power	Digital
C11	91	-	-	-	NC	No Connect	-	-
A13	90	-	-	-	NC	No Connect	-	-
C12	89	Right	Outer	1	TAEC	Triple Adjacent Error Correction Select	Input	Digital
D11	88	Right	Inner	1	VSS-A	Adam Ground	Power	Digital
B13	87	Right	Outer	2	DAEC	Double Adjacent Error Correction Select	Input	Digital
C13	86	Right	Inner	2	VDD6T	VDD 6T Array, 1V0 Nominal	Power	Analog
D12	85	Right	Outer	3	DINFFSEL< 0 >	Data Input FF Mux Select < 0 >	Input	Digital
E11	84	Right	Inner	3	VSS-A	Adam Ground	Power	Digital
D13	83	Right	Outer	4	DINFFSEL< 1 >	Data Input FF Mux Select < 1 >	Input	Digital
E12	82	Right	Inner	4	VDDPER	VDD Periphreal Circuits, 1V0 Nominal	Power	Analog
E13	81	Right	Outer	5	GNDE-I/O	IO Ring Ground	Power	Digital
F11	80	Right	Inner	5	VDDE-I/O	IO Ring VDD 1V8	Power	Digital
F12	79	Right	Outer	6	DOUT< 7 >	Data Output < 7 >	Output	Digital
F13	78	Right	Inner	6	DOUT< 6 >	Data Output < 6 >	Output	Digital
G13	77	Right	Outer	7	DOUT< 5 >	Data Output < 5 >	Output	Digital
G11	76	Right	Inner	7	DOUT< 4 >	Data Output < 4 >	Output	Digital
G12	75	Right	Outer	8	DOUT< 3 >	Data Output < 3 >	Output	Digital
H13	74	Right	Inner	8	DOUT< 2 >	Data Output < 2 >	Output	Digital
H12	73	Right	Outer	9	DOUT< 1 >	Data Output < 1 >	Output	Digital
H11	72	Right	Inner	9	DOUT< 0 >	Data Output < 0 >	Output	Digital
J13	71	Right	Outer	10	VDDE-I/O	IO Ring VDD 1V8	Power	Digital
J12	70	Right	Inner	10	GNDE-I/O	IO Ring Ground	Power	Digital
K13	69	Right	Outer	11	VDD-I/O	Core VDD in I/O, 1V0	Power	Digital
J11	68	Right	Inner	11	GND-I/O	Core Ground in I/O	Power	Digital
K12	67	Right	Outer	12	CLKOUT-J	Clock Out - Jaspal	Output	Digital
L13	66	Right	Inner	12	VBL	Bitline Voltage	Input	Analog
L12	65	Right	Outer	13	GNDE-I/O	IO Ring Ground	Power	Digital
K11	64	Right	Inner	13	VDD-J	VDD for array peripheral	Power	Analog
M13	63	Right	Outer	14	VDDE-I/O	IO Ring VDD 1V8	Power	Digital
M12	62	Right	Inner	14	VSS-J	Jaspal Ground	Power	Digital
L11	61	-	-	-	NC	No Connect	-	-

Table B.2: Pin Description for ICTWTAN3 120 PGA Test Chip - Pins 60-1

Pin Location	Pin Number	Pad Side	Pad Ring	Pad Number	Pin Name	Description	Direction	Type
N13	60	-	-	-	NC	No Connect	-	-
M11	59	Bottom	Outer	14	GNDE-I/O	IO Ring Ground	Power	Digital
L10	58	Bottom	Inner	14	VDDH	Level Converter VDD High 1V0	Power	Digital
N12	57	Bottom	Outer	13	VDDE-I/O	IO Ring VDD 1V8	Power	Digital
N11	56	Bottom	Inner	13	VSS-J	Jaspal Ground	Power	Digital
M10	55	Bottom	Outer	12	GNDE-I/O	IO Ring Ground	Power	Digital
L9	54	Bottom	Inner	12	VDD8T	VDD for 8T Array	Power	Analog
N10	53	Bottom	Outer	11	VDDE-I/O	IO Ring VDD 1V8	Power	Digital
M9	52	Bottom	Inner	11	VSS-J	Jaspal Ground	Power	Digital
N9	51	Bottom	Outer	10	GNDE-I/O	IO Ring Ground	Power	Digital
L8	50	Bottom	Inner	10	VDD-J	VDD for array peripheral	Power	Analog
M8	49	Bottom	Outer	9	VDD-I/O	Core VDD in I/O, 1V0	Power	Digital
N8	48	Bottom	Inner	9	GND-I/O	Core Ground in I/O	Power	Digital
N7	47	Bottom	Outer	8	GNDE-I/O	IO Ring Ground	Power	Digital
L7	46	Bottom	Inner	8	VDD8T	VDD for 8T Array	Power	Analog
M7	45	Bottom	Outer	7	VDDE-I/O	IO Ring VDD 1V8	Power	Digital
N6	44	Bottom	Inner	7	VSS-J	Jaspal Ground	Power	Digital
M6	43	Bottom	Outer	6	GNDE-I/O	IO Ring Ground	Power	Digital
L6	42	Bottom	Inner	6	VWL	Wordline Voltage	Input	Analog
N5	41	Bottom	Outer	5	VDDE-I/O	IO Ring VDD 1V8	Power	Digital
M5	40	Bottom	Inner	5	VSS-J	Jaspal Ground	Power	Digital
N4	39	Bottom	Outer	4	GNDE-I/O	IO Ring Ground	Power	Digital
L5	38	Bottom	Inner	4	VDDT	VDD for the Timing Block	Input	Analog
M4	37	Bottom	Outer	3	VDDE-I/O	IO Ring VDD 1V8	Power	Digital
N3	36	Bottom	Inner	3	VSS-J	Jaspal Ground	Power	Digital
M3	35	Bottom	Outer	2	GNDE-I/O	IO Ring Ground	Power	Digital
L4	34	Bottom	Inner	2	VDD8T	VDD for 8T Array	Power	Analog
N2	33	Bottom	Outer	1	RW-J	Read/Write - Jaspal	Input	Digital
M2	32	Bottom	Inner	1	VDD-J	VDD for array peripheral	Power	Analog
L3	31	-	-	-	NC	No Connect	-	-
N1	30	-	-	-	NC	No Connect	-	-
L2	29	Left	Outer	14	VDDE-I/O	IO Ring VDD 1V8	Power	Digital
K3	28	Left	Inner	14	GNDE-I/O	IO Ring Ground	Power	Digital
M1	27	Left	Outer	13	CLKIN-J	Clock In - Jaspal	Input	Digital
L1	26	Left	Inner	13	VDD8T	VDD for 8T Array	Power	Analog
K2	25	Left	Outer	12	ADDR< 0 >	Address Input < 0 >	Input	Digital
J3	24	Left	Inner	12	ADDR< 1 >	Address Input < 1 >	Input	Digital
K1	23	Left	Outer	11	ADDR< 2 >	Address Input < 2 >	Input	Digital
J2	22	Left	Inner	11	ADDR< 3 >	Address Input < 3 >	Input	Digital
J1	21	Left	Outer	10	ADDR< 4 >	Address Input < 4 >	Input	Digital
H3	20	Left	Inner	10	ADDR< 5 >	Address Input < 5 >	Input	Digital
H2	19	Left	Outer	9	ADDR< 6 >	Address Input < 6 >	Input	Digital
H1	18	Left	Inner	9	ADDR< 7 >	Address Input < 7 >	Input	Digital
G1	17	Left	Outer	8	ADDR< 8 >	Address Input < 8 >	Input	Digital
G3	16	Left	Inner	8	ADDR< 9 >	Address Input < 9 >	Input	Digital
G2	15	Left	Outer	7	ADDR< 10 >	Address Input < 10 >	Input	Digital
F1	14	Left	Inner	7	ADDR< 11 >	Address Input < 11 >	Input	Digital
F2	13	Left	Outer	6	ADDR< 12 >	Address Input < 12 >	Input	Digital
F3	12	Left	Inner	6	ADDR< 13 >	Address Input < 13 >	Input	Digital
E1	11	Left	Outer	5	ADDR< 14 >	Address Input < 14 >	Input	Digital
E2	10	Left	Inner	5	ADDR< 15 >	Address Input < 15 >	Input	Digital
D1	9	Left	Outer	4	GNDE-I/O	IO Ring Ground	Power	Digital
E3	8	Left	Inner	4	VDDE-I/O	IO Ring VDD 1V8	Power	Digital
D2	7	Left	Outer	3	CLKIN-A	Clock In - Adam	Input	Digital
C1	6	Left	Inner	3	VDD6T	VDD 6T Array, 1V0 Nominal	Power	Analog
C2	5	Left	Outer	2	RW-A	Read/Write - Adam	Input	Digital
D3	4	Left	Inner	2	VSS-A	Adam Ground	Power	Digital
B1	3	Left	Outer	1	VDD-I/O	Core VDD in I/O, 1V0	Power	Digital
B2	2	Left	Inner	1	GND-I/O	Core Ground in I/O	Power	Digital
C3	1	-	-	-	NC	No Connect	-	-

Appendix C

Publications From This Work

- **A. Neale**, M. Jonkman, and M. Sachdev, "Adjacent-MBU Tolerant SEC-DED-TAEC-yAED Codes for Embedded SRAMs", to appear in *Circuits and Systems II: Express Briefs, IEEE Transactions on*, DOI: 10.1109/TCSII.2014.2368262.

- **A. Neale** and M. Sachdev, "A 0.4 V 75 kbit SRAM Macro in 28 nm CMOS Featuring a 3-Adjacent MBU Correcting ECC", in *Proc. IEEE 2014 Custom Integrated Circuits Conference (CICC)*, San Jose, CA, 15-17 September 2014.

- **A. Neale** and M. Sachdev, "A New SEC-DED Error Correction Code Subclass for Adjacent MBU Tolerance in Embedded Memory," *Device and Materials Reliability, IEEE Transactions on* , vol.13, no.1, pp.223,230, March 2013.
  DOI: 10.1109/TDMR.2012.2232671.

Other Accepted Publications During Ph.D.

- **A. Neale**, and M. Sachdev, "Success Mechanisms for STEM Student-Athletes, in *Opportunities and New Directions (OND '12)*, University of Waterloo, Waterloo, Ontario, Canada, April 2012.

- **A. Neale**, O. Grant, and M. Sachdev, "The Road to Success for STEM Student-Athletes, in *Proc. of the 119th Annual American Society for Engineering Education Conference and Exposition (ASEE '12)*, Austin, TX, June 2012, pp. 1-25.

- **A. Neale**, and M. Sachdev, "Learning to Juggle: The Challenges and Benefits to being an Engineering Graduate Student and Collegiate Athlete", in *Proc. of the 119th Annual American Society for Engineering Education Conference and Exposition (ASEE '12)*, Austin, TX, June 2012.

- **A. Neale** and M. Sachdev, "Digitally programmable SRAM timing for nano-scale technologies," in *Proc. Int. Symp. on Quality Electronic Design (ISQED), 2011*, San Jose, CA pp.1,7, 14-16 March 2011 doi: 10.1109/ISQED.2011.5770776

References

[1] Taejoong Song, Woojin Rim, Jonghoon Jung, Giyong Yang, Jaeho Park, Sunghyun Park, Kang-Hyun Baek, Sanghoon Baek, Sang-Kyu Oh, Jinsuk Jung, Sungbong Kim, Gyuhong Kim, Jintae Kim, Youngkeun Lee, Kee Sup Kim, Sang-Pil Sim, Jong Shik Yoon, and Kyu-Myung Choi. 13.2 A 14nm FinFET 128Mb 6T SRAM with VMIN-Enhancement Techniques for Low-Power Applications. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, pages 232–233, Feb 2014.

[2] Yen-Huei Chen, Wei-Min Chan, Wei-Cheng Wu, Hung-Jen Liao, Kuo-Hua Pan, Jhon-Jhy Liaw, Tang-Hsuan Chung, Quincy Li, G.H. Chang, Chih-Yung Lin, Mu-Chi Chiang, Shien-Yang Wu, S. Natarajan, and J. Chang. 13.5 A 16nm 128Mb SRAM in High-k; Metal-Gate FinFET Technology with Write-Assist Circuitry for Low-VMIN Applications. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, pages 238–239, Feb 2014.

[3] R. Schenker and V. Singh. Foundations for Scaling Beyond 14nm. In *Custom Integrated Circuits Conference (CICC), 2013 IEEE*, pages 1–4, Sept 2013.

[4] R.C. Baumann. Radiation-Induced Soft Errors in Advanced Semiconductor Technologies. *Device and Materials Reliability, IEEE Transactions on*, 5(3):305 – 316, Sept. 2005.

[5] C. Slayman. Soft Error Trends and Mitigation Techniques in Memory Devices. In *Reliability and Maintainability Symposium (RAMS), 2011 Proceedings - Annual*, pages 1 –5, Jan. 2011.

[6] E. Ibe, H. Taniguchi, Y. Yahagi, K.-i. Shimbo, and T. Toba. Impact of Scaling on Neutron-Induced Soft Error in SRAMs From a 250 nm to a 22 nm Design Rule. *Electron Devices, IEEE Transactions on*, 57(7):1527 –1538, July 2010.

[7] J.F. Ziegler and H. Puchner. *SER - History, Trends and Challenges - A Guide for Designing with Memory ICs*. Cypress Semiconductor Corporation, 1st edition, 2004.

[8] Timothy C. May and Murray H. Woods. A New Physical Mechanism for Soft Errors in Dynamic Memories. In *Reliability Physics Symposium, 1978. 16th Annual*, pages 33 –40, April 1978.

[9] J. Ziegler and W. Lanford. The Effect of Sea Level Cosmic Rays on Electronic Devices. In *Solid-State Circuits Conference. Digest of Technical Papers. 1980 IEEE International*, volume XXIII, pages 70–71, Feb 1980.

[10] International Roadmap for Semiconductors. http://www.itrs.net/, 2011.

[11] E. Fujiwara. *Code Design for Dependable Systems - Theory and Practical Applications*. John Wiley and Sons, Inc., 2006.

[12] Wei Chen, Szu-Liang Chen, Siufu Chiu, R. Ganesan, V. Lukka, W.W. Mar, and S. Rusu. A 22nm 2.5MB Slice On-die L3 Cache for the Next Generation Xeon Processor. In *VLSI Circuits (VLSIC), 2013 Symposium on*, pages C132–C133, June 2013.

[13] H. Yamauchi. *Embedded Memories for Nano-Scale VLSIs*, chapter Chapter 3 - Embedded SRAM Design in Nanometer-Scale Technologies. Springer Science and Business Media, LLC.

[14] P.E. Dodd and L.W. Massengill. Basic Mechanisms and Modeling of Single-Event Upset in Digital Microelectronics. *Nuclear Science, IEEE Transactions on*, 50(3):583 – 602, June 2003.

[15] D.J. Rennie, T. Shakir, and M. Sachdev. Design Challenges in Nanometric Embedded Memories. In *Signals, Circuits and Systems (SCS), 2009 3rd International Conference on*, pages 1 –8, Nov. 2009.

[16] G. Georgakos, P. Huber, M. Ostermayr, E. Amirante, and F. Ruckerbauer. Investigation of Increased Multi-Bit Failure Rate Due to Neutron Induced SEU in Advanced Embedded SRAMs. In *VLSI Circuits, 2007 IEEE Symposium on*, pages 80–81, June 2007.

[17] G.C. Cardarilli, A. Leandri, P. Marinucci, M. Ottavi, S. Pontarelli, M. Re, and A. Salsano. Design of a Fault Tolerant Solid State Mass Memory. *Reliability, IEEE Transactions on*, 52(4):476 – 491, Dec. 2003.

[18] J. Maiz, S. Hareland, K. Zhang, and P. Armstrong. Characterization of Multi-bit Soft Error Events in Advanced SRAMs. In *Electron Devices Meeting, 2003. IEDM '03 Technical Digest. IEEE International*, pages 21.4.1 – 21.4.4, Dec. 2003.

[19] R. Naseer and J. Draper. Parallel Double Error Correcting Code Design to Mitigate Multi-bit Upsets in SRAM s. In *Solid-State Circuits Conference, 2008. ESSCIRC 2008. 34th European*, pages 222 –225, Sept. 2008.

[20] A.D. Tipton, J.A. Pellish, J.M. Hutson, R. Baumann, X. Deng, A. Marshall, M.A. Xapsos, H.S. Kim, M.R. Friendlich, M.J. Campola, C.M. Seidleck, K.A. LaBel, M.H. Mendenhall, R.A. Reed, R.D. Schrimpf, R.A. Weller, and J.D. Black. Device-Orientation Effects on Multiple-Bit Upset in 65 nm SRAMs. *Nuclear Science, IEEE Transactions on*, 55(6):2880 –2885, Dec. 2008.

[21] H. Kobayashi, N. Kawamoto, J. Kase, and K. Shiraish. Alpha Particle and Neutron-induced Soft Error Rates and Scaling Trends in SRAM. In *Reliability Physics Symposium, 2009 IEEE International*, pages 206 –211, April 2009.

[22] Sai-Wai Fu, A.M. Mohsen, and T.C. May. Alpha-Particle-Induced Charge Collection Measurements and the Effectiveness of a Novel p-well Protection Barrier on VLSI Memories. *Electron Devices, IEEE Transactions on*, 32(1):49 – 54, Jan 1985.

[23] D. Burnett, C. Lage, and A. Bormann. Soft-Error-Rate Improvement in Advanced BiCMOS SRAMs. In *Reliability Physics Symposium, 1993. 31st Annual Proceedings., International*, pages 156 –160, March 1993.

[24] J.D. Hayden, R.C. Taft, P. Kenkare, C. Mazure, C. Gunderson, B.-Y. Nguyen, M. Woo, C. Lage, B.J. Roman, S. Radhakrishna, R. Subrahmanyan, A.R. Sitaram, P. Pelley, J.-H. Lin, K. Kemp, and H. Kirsch. A Quadruple Well, Quadruple Polysilicon BiCMOS Process for Fast 16 Mb SRAM's. *Electron Devices, IEEE Transactions on*, 41(12):2318 –2325, Dec 1994.

[25] E.H. Cannon, D.D. Reinhardt, M.S. Gordon, and P.S. Makowenskyj. SRAM SER in 90, 130 and 180 nm Bulk and SOI Technologies. In *Reliability Physics Symposium Proceedings, 2004. 42nd Annual. 2004 IEEE International*, pages 300 – 304, April 2004.

[26] S. E. Diehl, A. Ochoa, P. V. Dressendorfer, R. Koga, and W. A. Kolasinski. Error Analysis and Prevention of Cosmic Ion-Induced Soft Errors in Static CMOS RAMs. *Nuclear Science, IEEE Transactions on*, 29(6):2032 –2039, Dec. 1982.

[27] F. Ootsuka, M. Nakamura, T. Miyake, S. Iwahashi, Y. Ohira, T. Tamaru, K. Kikushima, and K. Yamaguchi. A Novel 0.20 um Full CMOS SRAM Cell Using Stacked Cross Couple with Enhanced Soft Error Immunity. In *Electron Devices Meeting, 1998. IEDM '98 Technical Digest., International*, pages 205 –208, Dec 1998.

[28] T. Calin, M. Nicolaidis, and R. Velazco. Upset Hardened Memory Design for Submicron CMOS Technology. *Nuclear Science, IEEE Transactions on*, 43(6):2874 –2878, Dec 1996.

[29] S.M. Jahinuzzaman, J.S. Shah, D.J. Rennie, and M. Sachdev. Design and Analysis of A 5.3-pJ 64-kb Gated Ground SRAM With Multiword ECC. *Solid-State Circuits, IEEE Journal of*, 44(9):2543–2553, Sept 2009.

[30] S.M. Jahinuzzaman. *Modeling and Mitigation of Soft Errors in Nanoscale SRAMs*. PhD thesis, University of Waterloo, 2008.

[31] M. Y. Hsiao. A Class of Optimal Minimum Odd-weight-column SEC-DED Codes. *IBM Journal of Research and Development*, 14(4):395 –401, July 1970.

[32] S.S. Mukherjee, J. Emer, and S.K. Reinhardt. The Soft Error Problem: An Architectural Perspective. In *High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on*, pages 243 – 247, Feb. 2005.

[33] R.W. Hamming. Error Correcting and Error Detecting Codes. *Bell Sys. Tech. Journal*, 29:147–160, April 1950.

[34] C. L. Chen and M. Y. Hsiao. Error-Correcting Codes for Semiconductor Memory Applications: A State-of-the-Art Review. *IBM Journal of Research and Development*, 28(2):124 –134, March 1984.

[35] D. Rossi, N. Timoncini, M. Spica, and C. Metra. Error Correcting Code Analysis for Cache Memory High Reliability and Performance. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, pages 1 –6, March 2011.

[36] Xinmiao Zhang and K.K. Parhi. High-Speed Architectures for Parallel Long BCH Encoders. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 13(7):872 –877, July 2005.

[37] Y.S. Kavian, A. Falahati, A. Khayatzadeh, and M. Naderi. High Speed Reed-Solomon Decoder with Pipeline Architecture. In *Wireless and Optical Communications Networks, 2005. WOCN 2005. Second IFIP International Conference on*, pages 415 – 419, March 2005.

[38] V. Pless. Decoding the Golay Codes. *Information Theory, IEEE Transactions on*, 32(4):561 – 567, Jul 1986.

[39] H. Naeimi and A. DeHon. Fault Secure Encoder and Decoder for NanoMemory Applications. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 17(4):473 –486, April 2009.

[40] Ming Zhu, Liyi Xiao, Shuhao Li, and Yanjing Zhang. Efficient Two-Dimensional Error Codes for Multiple Bit Upsets Mitigation in Memory. In *Defect and Fault Tolerance in VLSI Systems (DFT), 2010 IEEE 25th International Symposium on*, pages 129 –135, Oct. 2010.

[41] Ching-Yi Chen and Cheng-Wen Wu. An Adaptive Code Rate EDAC Scheme for Random Access Memory. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pages 735 –740, March 2010.

[42] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1. In *Communications, 1993. ICC '93 Geneva. Technical Program, Conference Record, IEEE International Conference on*, volume 2, pages 1064–1070 vol.2, May 1993.

[43] Robert G. Gallager. *Low Density Parity Check Codes*. PhD thesis, Massachusetts Institute of Technology, 1963.

[44] A. Dutta and N.A. Touba. Multiple Bit Upset Tolerant Memory Using a Selective Cycle Avoidance Based SEC-DED-DAEC Code. In *VLSI Test Symposium, 2007. 25th IEEE*, pages 349 –354, May 2007.

[45] Sanghyeon Baeg, ShiJie Wen, and R. Wong. SRAM Interleaving Distance Selection With a Soft Error Failure Model. *Nuclear Science, IEEE Transactions on*, 56(4):2111 –2118, Aug. 2009.

[46] S.M. Abbas, Sanghyeon Baeg, and Sungju Park. Multiple Cell Upsets Tolerant Content-Addressable Memory. In *Reliability Physics Symposium (IRPS), 2011 IEEE International*, pages SE.1.1 –SE.1.5, April 2011.

[47] S.S. Mukherjee, J. Emer, T. Fossum, and S.K. Reinhardt. Cache Scrubbing in Microprocessors: Myth or Necessity? In *Dependable Computing, 2004. Proceedings. 10th IEEE Pacific Rim International Symposium on*, pages 37 – 42, March 2004.

[48] T. Suzuki, Y. Yamagami, I. Hatanaka, A. Shibayama, H. Akamatsu, and H. Yamauchi. A Sub-0.5-V Operating Embedded SRAM Featuring a Multi-bit-Error-Immune Hidden-ECC Scheme. *Solid-State Circuits, IEEE Journal of*, 41(1):152–160, Jan 2006.

[49] F.G. Stremler. *Introduction to Communication Systems*. Addison-Wesley Publishing Company, 3rd edition, 1990.

[50] W.W. Peterson and Jr. E.J. Weldon. *Error-Correcting Codes*. MIT Press, 2nd edition, 1972.

[51] S. Papaharalabos, D. Benmayor, P.T. Mathiopoulos, and Pingzhi Fan. Performance Comparisons and Improvements of Channel Coding Techniques for Digital Satellite Broadcasting to Mobile Users. *Broadcasting, IEEE Transactions on*, 57(1):94–102, March 2011.

[52] D.J.C. MacKay and R.M. Neal. Near Shannon limit performance of low density parity check codes. *Electronics Letters*, 32(18):1645–, Aug. 1996.

[53] Sang Hoon Lee, Jia Ae Seok, and Eon Kyeong Joo. Serial concatenation of LDPC and turbo code for the next generation mobile communications. In *Wireless and Optical Communications Networks, 2005. WOCN 2005. Second IFIP International Conference on*, pages 425–427, March 2005.

[54] Xin-Yu Shih, Cheng-Zhou Zhan, Cheng-Hung Lin, and An-Yeu Wu. An 8.29 mm^2 52 mW Multi-Mode LDPC Decoder Design for Mobile WiMAX System in 0.13 mum CMOS Process. *Solid-State Circuits, IEEE Journal of*, 43(3):672–683, March 2008.

[55] J.C. Porcello. Designing and implementing Low Density Parity Check (LDPC) Decoders using FPGAs. In *Aerospace Conference, 2014 IEEE*, pages 1–7, March 2014.

[56] M.A. Jordan and R.A. Nichols. The effects of channel characteristics on turbo code performance. In *Military Communications Conference, 1996. MILCOM '96, Conference Proceedings, IEEE*, volume 1, pages 17–21 vol.1, Oct 1996.

[57] 3GPP - 3rd Generation Partnership Project. http://www.3gpp.org, 2014.

[58] IEEE Standard for WirelessMAN-Advanced Air Interface for Broadband Wireless Access Systems. *IEEE Std 802.16.1-2012*, pages 1–1090, Sept 2012.

[59] Digital Video Broadcasting Project. https://www.dvb.org, 2014.

[60] IEEE Standard for Ethernet - Section 6. *IEEE Std 802.3-2012 (Revision to IEEE Std 802.3-2008)*, pages 1–0, Dec 2012.

[61] IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE Std 802.11-2007 (Revision of IEEE Std 802.11-1999)*, pages 1–1076, June 2007.

[62] Simon Tam, S. Rusu, Jonathan Chang, S. Vora, B. Cherkauer, and D. Ayers. A 65nm 95W Dual-Core Multi-Threaded Xeon Processor with L3 Cache. In *Solid-State Circuits Conference, 2006. ASSCC 2006. IEEE Asian*, pages 15–18, Nov 2006.

[63] J. M. Rabaey, A. Chandrakasan, and B. Nikolic. *Digital Integrated Circuits*. Prentice Hall, 2nd edition, 2003.

[64] S. Lin and Jr. D.J. Costello. *Error Control Coding: Fundamentals and Applications*. Prentice Hall, 2nd edition, 2004.

[65] I.S. Reed and G. Solomon. Polynomial Codes Over Certain Finite Fields. In *SIAM Journal of Applied Mathematics*, pages 300–304, June 1960.

[66] A. Vanstone and P. C. van Oorschot. *An Introduction to Error Correcting Codes with Applications*. Kluwer Academic Publishers, 1st edition, 2001.

[67] H. Labiod. Performance of Reed Solomon Error-Correcting Codes on Fading Channels. In *Personal Wireless Communication, 1999 IEEE International Conference on*, pages 259–263, 1999.

[68] K. Gillespie, H.R. Fair, C. Henrion, R. Jotwani, S. Kosonocky, R.S. Orefice, D.A Priore, J. White, and K. Wilcox. 5.5 Steamroller: An x86-64 Core Implemented in 28nm Bulk CMOS. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, pages 104–105, Feb 2014.

[69] N. Kurd, M. Chowdhury, E. Burton, T.P. Thomas, C. Mozak, B. Boswell, M. Lal, A Deval, J. Douglas, M. Elassal, A Nalamalpu, T.M. Wilson, M. Merten, S. Chennupaty, W. Gomes, and R. Kumar. Haswell: A Family of IA 22nm Processors. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, pages 112–113, Feb 2014.

[70] Canadian Microelectronics Corporation. http://www.cmc.ca, 2014.

[71] Qualcomm - Snapdragon. http://http://www.qualcomm.com/snapdragon, 2014.

[72] R.W. Mann, T.B. Hook, P.T. Nguyen, and B.H. Calhoun. Nonrandom Device Mismatch Considerations in Nanoscale SRAM. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 20(7):1211–1220, July 2012.

[73] K. Osada, K. Yamaguchi, Y. Saitoh, and T. Kawahara. SRAM Immunity to Cosmic-Ray-Induced Multi Errors Based on Analysis of an Induced Parasitic Bipolar Effect. *Solid-State Circuits, IEEE Journal of*, 39(5):827–833, May 2004.

[74] K. Utsumi, E. Morifuji, M. Kanda, S. Aota, T. Yoshida, K. Honda, Y. Matsubara, S. Yamada, and F. Matsuoka. A 65nm Low Power CMOS Platform with 0.495 um2 SRAM for Digital Processing and Mobile Applications. In *VLSI Technology, 2005. Digest of Technical Papers. 2005 Symposium on*, pages 216–217, June 2005.

[75] H. Pilo, V. Ramadurai, G. Braceras, J. Gabric, S. Lamphier, and Yue Tan. A 450ps Access-Time SRAM Macro in 45nm SOI Featuring a Two-Stage Sensing-Scheme and Dynamic Power Management. In *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International*, pages 378–621, Feb 2008.

[76] Chien-Cheng Yu, Wei-Ping Wang, and Bin-Da Liu. A 3-input XOR/XNOR for Low-Voltage Low-Power Applications. In *Circuits and Systems, 2000. IEEE APCCAS 2000. The 2000 IEEE Asia-Pacific Conference on*, pages 505–508, 2000.

[77] R. Kumar and V. Kursun. Temperature-Adaptive Energy Reduction for Ultra-Low Power-Supply-Voltage Subthreshold Logic Circuits. In *Electronics, Circuits and Systems, 2007. ICECS 2007. 14th IEEE International Conference on*, pages 1280–1283, Dec 2007.

[78] Quik-Pak. http://www.icproto.com, 2014.

[79] Corwil Technology Corporation. http://www.corwil.com, 2014.

[80] M. Yabuuchi, H. Fujiwara, Y. Tsukamoto, M. Tanaka, S. Tanaka, and K. Nii. A 28nm High Density 1R/1W 8T-SRAM Macro with Screening Circuitry Against Read Disturb Failure. In *Custom Integrated Circuits Conference (CICC), 2013 IEEE*, pages 1–4, Sept 2013.

[81] E.W. Blackmore, P. E. Dodd, and M.R. Shaneyfelt. Improved Capabilities for Proton and Neutron Irradiations at TRIUMF. In *Radiation Effects Data Workshop, 2003. IEEE*, pages 149–155, July 2003.

[82] Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices. In *JEDEC Test Standard 89A*, Sep. 2006.

[83] P. Hazucha and C. Svensson. Impact of CMOS Technology Scaling on the Atmospheric Neutron Soft Error Rate. *Nuclear Science, IEEE Transactions on*, 47(6):2586–2594, Dec 2000.

[84] N. Seifert, V. Ambrose, B. Gill, Q. Shi, R. Allmon, C. Recchia, S. Mukherjee, N. Nassif, J. Krause, J. Pickholtz, and A Balasubramanian. On the Radiation-Induced Soft Error Performance of Hardened Sequential Elements in Advanced Bulk CMOS Technologies. In *Reliability Physics Symposium (IRPS), 2010 IEEE International*, pages 188–197, May 2010.