Fault Tolerant, Low Voltage SRAM Design

by

Yildiz Sinangil

B.A.Sc. in Electrical & Electronics Engineering
Bogazici University, Istanbul 2008

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering and Computer Science

at the MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2010

© Massachusetts Institute of Technology 2010. All rights reserved.

Author
Department of Electrical Engineering and Computer Science
May 21, 2010

Certified by
Anantha P. Chandrakasan
Joseph F. and Nancy P. Keithley Professor of Electrical Engineering
Thesis Supervisor

Accepted by
Terry P. Orlando
Chairman, Department Committee on Graduate Students
Fault Tolerant, Low Voltage SRAM Design

by

Yildiz Sinangil

Submitted to the Department of Electrical Engineering and Computer Science on May 21, 2010, in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering and Computer Science

Abstract

Scaling of process technologies has made power management a significant concern for circuit designers. Moreover, denser integration and shrinking geometries also have a negative impact on circuit reliability. Therefore, fault tolerance is becoming a more challenging problem. Static Random Access Memories (SRAMs) play a significant role in circuit power consumption and reliability of digital circuits.

This thesis focuses on fault tolerant and low voltage SRAM design. A double error correcting binary BCH codec is chosen to mitigate reliability problems. Different decoding schemes are compared in terms of their synthesized power, area and latency. An alternative decision-tree based decoder is analyzed. This decoder requires 16ns for error correction and 5ns for error detection at 1.2V using 65nm CMOS. Compared to conventional iterative decoding scheme in which error correction takes more than 100 clock cycles for 128-bit word length, the analyzed decoder has a significant latency advantage. Meanwhile, compared to the look-up table (LUT) decoder, the decision-tree based architecture has 2X area and power savings. Hence, the tree-based decoder is an alternative design which does not have the extreme power and area consumption of a LUT decoder and does not have the extreme latency of an iterative implementation.

An 8T SRAM block is designed in 65nm CMOS low-power, high $V_T$ process for the on-chip caches of a low-voltage processor. This SRAM is designed for the array voltage range of 1.2V to 0.4V. It provides more than 4 orders of magnitude performance scaling and 10X power savings.

Thesis Supervisor: Anantha P. Chandrakasan
Title: Joseph F. and Nancy P. Keithley Professor of Electrical Engineering
Acknowledgments

Three years ago when I was an exchange student at UT Texas, I sent an email to Anantha to introduce myself. Honestly, I was not hopeful to hear back from him. Why would he spare his busy time for me? I was just a junior student from the other side of the world and I was not even graduating soon. Not only did he give me an appointment, but also he invited me as a summer internship student to his group. After that day, I always felt myself extremely lucky to be working with him. Thanks Anantha for believing in me and supporting me right from the beginning. It is a real privilege to be a part of your group.

Anantha’s group is very special for me since I started dating with my husband at MIT. Although this made our house a second office with white boards, computers and scattered papers everywhere, I love living in it with him. Ersin, thank you very much for being my best friend, helping me through the way and your endless love.

My dear parents and brother, I cannot express how much I love you. You trusted in me more than I trusted in myself. Dear dad, you are the most farsighted person I know. I remember you talking about the fascinating projects that are being done at MIT when I was graduating from high school. My dear mom, you always told me to relax and enjoy the life. Thank you for our chats on the phone, for balancing my life and calming me down. My dear brother thanks a lot for being such a perfect brother in every way. You are my number one.

My dear Havva and Sukru parents, thank you very much for being there for me. Hearing your voice over the phone has always made me realize how lucky I am to have such a loving new family.

Nathan, thank you very much for your collaboration in the ReISC project. You are a professional in almost all areas of electronics. Thanks a lot for sharing your knowledge with me. Francesco, thanks for your help in making our lives easier. You were always very genuine, helpful and kind. I also thank Ananthagroup for great discussions and for their friendship. I feel like I have a second big family in the USA. Lastly, thanks Margaret for making my day with your warm smile every morning.
Contents

1 Introduction .................................................. 13
  1.1 Fault Types .................................................. 15
  1.2 Failure Rate Calculations For Semiconductors .............. 16
  1.3 Error Correcting Codes ..................................... 17
     1.3.1 Block Codes ........................................... 17
     1.3.2 Convolutional Codes .................................... 19
  1.4 Related Work .............................................. 19
  1.5 Thesis Contribution ......................................... 20

2 Low Voltage SRAM Design in 65nm CMOS ....................... 23
  2.1 6T Bitcell at Low Supply Voltages ........................ 23
     2.1.1 Read, Write and Hold Operation ....................... 23
     2.1.2 6T Bitcell Challenges at Low Supply Voltages ......... 24
  2.2 8T Bitcell at Low Supply Voltages ........................ 25
  2.3 Architecture of the 8T SRAM Design in 65nm CMOS .......... 27
     2.3.1 Power Supply Voltages Used in the Design ............. 27
     2.3.2 Word Length (WLen) Optimization ..................... 29
     2.3.3 Peripheral Circuit Design Considerations ............. 31
  2.4 Simulation Results of the SRAM ........................... 35
  2.5 Test Chip Architecture .................................... 36

3 Error Correction Coding For SRAM Design ..................... 37
  3.1 Definition of BCH Codes ................................... 38
3.2 Decoding of BCH Codes .............................................. 39
3.3 Hardware Implementation ............................................. 42
   3.3.1 Encoder and Syndrome Generator Implementation ....... 43
   3.3.2 Decoder Implementation ........................................ 46
3.4 Using ECC to Increase SRAM Yield versus Redundancy ........ 48

4 Conclusions ................................................................. 51
4.1 Summary of Contributions ............................................ 51
4.2 Low Voltage SRAM Design in 65nm CMOS ......................... 51
4.3 Error Correction Coding for SRAM ................................. 52
4.4 Future Work ............................................................. 53

A Overview of Galois Fields and Linear Block Codes ............... 55
   A.1 Galois Field Definition and Algebra ........................... 55
   A.1.1 Definition of Galois Fields .................................... 55
   A.1.2 Binary Field Arithmetic ....................................... 56
   A.2 Linear Block Codes ................................................ 58
   A.2.1 Generation of The Code ......................................... 58
   A.2.2 Syndrome and Error Detection ................................. 60
List of Figures

1-1 Block diagram for the ReISC processor. ........................................ 14
1-2 (a) SRAM Reliability without ECC vs. with SECDED ECC. (b) SRAM reliability in detail for commonly used word lengths and possible scrubbing periods. ............................... 16
1-3 Block code types. ........................................................................... 18

2-1 Schematic for a standard 6T bitcell. ............................................... 24
2-2 Schematic for an 8T bitcell. .......................................................... 25
2-3 Read and hold margin distribution at $V_{DD}$=500mV for 22nm predictive technology. The model is developed by the Nanoscale Integration and Modeling (NIMO) Group at ASU. .......................... 26
2-4 Layouts for (a) a 6T and (b) an 8T bitcell. ................................. 27
2-5 Architecture diagram of the 32kb 8T SRAM in 65nm CMOS. ........ 28
2-6 WLen of the SRAM is chosen to be 128 bits considering the power at idle and active states. ............................................................... 29
2-7 The configurations for (a) WLen=32 and (b) WLen=64. .......... 30
2-8 Schematics for the WWL driver including an asynchronous level converter based on DCVSL gate. ................................................. 31
2-9 Schematics for the NMOS input strong-arm sense amplifier and the following NOR-based latch. ..................................................... 32
2-10 Input offset distribution for our sense amplifier. ......................... 33
2-11 Internal timing control signals of the 8T SRAM design for $V_{DDP}$ at 600mV. ................................................................. 34
2-12 Layout of the 32kb low-voltage SRAM designed in 65nm. ....... 35
2-13 Layout of the low-voltage ReISC microprocessor chip. On-chip SRAM blocks are highlighted. ......................... 36

3-1 The block diagram for an SRAM protected by ECC. ............. 37
3-2 Number of Parity Bits and Area Overhead vs. Data Word Length. 38
3-3 The architecture of a generic BCH codec. ......................... 42
3-4 The schematics of a (15,7) encoder circuit. ....................... 45
3-5 The block diagram for the three decoding blocks: (a) LUT decoder, (b) Decision-tree based decoder, and (c) Iterative decoder. ....... 46
3-6 For a 65nm, 32kB SRAM with 95% yield and 128 bits of word length
   (a) The minimum supply voltage drops when error correction capability is increased (b) There is an optimum point for the energy (parity bit overhead is considered only) ......................... 48
3-7 For a 65nm, 32kB SRAM with 95% yield Redundancy vs. ECC analysis results. (a) Word Length = 256 bits (b) Word Length = 128 bits ..... 49

A-1 Systematic format of a codeword. ................................. 59
List of Tables

1.1 On-chip memory sizes in years. ................................. 14
1.2 A (15,7) binary block code example .......................... 18

2.1 65nm CMOS SRAM design properties and simulation results. 35
2.2 On-chip memories used in low-voltage ReISC processor chip 36

3.1 Synthesis results for the encoder. ............................. 45
3.2 Synthesis results for the syndrome generator. ................. 45
3.3 Synthesis results for the LUT decoder. ......................... 47
3.4 Synthesis results for the Tree decoder. ......................... 47

A.1 Modulo-2 addition and multiplication. ......................... 56
A.2 Three representations for the elements of GF(2^4) generated by primitive polynomial p(X) = 1+X+X^4 ......................... 57
Chapter 1

Introduction

Advances in integrated circuits enabled improvements in various areas such as biomedical electronics. Portable medical monitors, wearable electronics, and body implants are some popular research topics in this area. A low-voltage, fault-tolerant system on chip (SoC) design, which is called the ReISC (Reduced Energy Instruction Set), is being developed for medical applications in 65nm low power, high $V_T$ CMOS process. The block diagram of ReISC is pictured in Figure 1-1. Including numerous blocks such as the CPU, a custom SRAM, couple of real time clocks, serial ports, an A/C, on-chip caches and off-chip memory interfaces, this is a very complex system with around 43k gates. ReISC design trade-offs require a balance of very low-power consumption in order to decrease battery weight and size and high-reliability for longer device lifetimes without a fault. Constituting around 25% of the chip area, instruction and data caches are the building blocks that are focused in this thesis. These SRAM blocks need to accommodate low-power and high-reliability.

Integrated circuits still scale according to Moore’s Law [1] which states that using process scaling, the number of transistors that can be placed on a single die will approximately double every two years. Although process scaling has enabled denser and more complex system design, it produced a number of problems. Most notably, both the active and leakage power are increasing exponentially [2]. As a result, the total power consumed has also increased to levels that impose fundamental limitation to functionality and performance [3]. This makes low-power circuit design a very active
and important research area. In order to achieve lower power consumption, many
different design techniques are investigated and supply voltage scaling has become a
very effective method.

<table>
<thead>
<tr>
<th>Reference Size</th>
<th>POWER4</th>
<th>Itanium</th>
<th>Nehalem</th>
<th>POWER7</th>
</tr>
</thead>
</table>

Today’s complex integrated circuits include significant sizes of on-chip memories.
In Table 1.1 it is clearly seen that on-chip memory size is increasing substantially
over the years. In order to decrease total power consumption of integrated circuits,
designing low-voltage SRAMs is very important.

The reliability of a system is defined as the ability to perform its required functions
under stated conditions for a specific time [8]. Shrinking geometries, lower supply
voltages, higher frequencies and denser integration have a negative impact on circuit
reliability.

Due to scaling, the critical charge to flip one bit reduces since the capacitances
decrease. On the other hand, the probability of corrupting data in a particular bitcell
also decreases because of smaller bitcell areas. Hence, soft error rate per bitcell is
projected to remain the same over the next technology generations [9]. However, on-chip SRAM sizes are rising as in line with Table 1.1. Hence, SRAM reliability is becoming a more challenging design problem.

1.1 Fault Types

The main source of reliability degradation is system faults which can be categorized into three groups according to their duration and occurrence: permanent, intermittent, and transient faults. [10]

1. **Permanent faults**: These are irreversible physical changes. Once occurred, these faults do not vanish. The most common source of them is manufacturing process.

2. **Intermittent faults**: These faults are periodic bursts that usually repeat themselves. However, they are not continuous as permanent faults. They take place in unstable or marginal hardware and become active when certain environmental changes occur such as higher or lower temperatures.

3. **Transient errors**: These are momentary single malfunctions caused by temporary environmental conditions such as neutron and alpha particles, interconnect noise, and electrostatic discharge. Another term used for a transient error is a single event upset (SEU) or a soft error. The occurrence of them is commonly random and therefore difficult to detect.

The improvement in semiconductor design and manufacturing techniques has significantly reduced the number of permanent faults. On the other hand, occurrence of intermittent and transient faults grow in advanced technologies [11]. For instance, lower clock periods increase the number of errors generated by the violations of timing margins. Therefore, the intermittent faults that take place due to process variations are rising. Similarly, smaller transistors and lower supply voltages result in higher sensitivity to neutron and alpha particles which means that the number of particle induced transients is growing [12],[13].
1.2 Failure Rate Calculations For Semiconductors

Understanding a product’s reliability requires an understanding of failure rate calculation. The traditional method of determining a product’s failure rate is through the use of accelerated operating life tests performed on a sample of randomly selected devices. The definitions of some important terms that are used in failure rate calculations are summarized below [14].

- **Failure Rate** ($\lambda$): Occurrence of failures per unit time.
- **Failure in Time (FIT)**: Measure of failure rate in $10^9$ device hours; e.g. 1 FIT = 1 failure in $10^9$ device hours.
- **Soft Error Rate (SER)**: Rate at which a device or system encounters soft errors. Typically expressed in FIT.
- **Multiple Bit Upset (MBU)**: Multiple bit errors that can occur when adjacent bits fail due to a single strike.
- **Memory Scrubbing**: A process of detecting and correcting bit errors in memory by using error correction coding (ECC). The memory controller scans systematically through the memory, in order to detect and correct the errors.

Figure 1-2: (a) SRAM Reliability without ECC vs. with SECDED ECC. (b) SRAM reliability in detail for commonly used word lengths and possible scrubbing periods.
Single-error correction, double-error detection (SECDED) is very effective in enhancing SRAM reliability. For a FIT=500 and 1MB SRAM size, the reliability of SRAM with and without ECC is shown in Figure 1-2(a). It is clearly seen that if ECC is not used, the probability to encounter no faults due to soft errors in 10 years is very low. However, a simple SECDED code and scrubbing once in a few years is very effective in increasing reliability to very high probabilities. Figure 1-2(b) shows that for common SRAM word lengths, scrubbing the memory once in a few years results into very reliable systems.

In this analysis, we used the traditional model that estimates the reliability of memories suffering SEUs only. Actually, MBUs are becoming more and more important as technology scales [15]. Therefore, we focused on ways to implement double error correcting (DEC) codes rather than a simple SECDED code.

1.3 Error Correcting Codes

In recent years, there has been a strong demand for reliable systems. In order to increase system reliability, ECC algorithms are developed. By definition, error correcting codes add redundant data to the message that is being sent so that even if a certain number of errors are introduced to the message over the channel, the original message still can be recovered [14].

In 1948, Shannon demonstrated in a landmark paper that if the signaling rate of a system is less than the channel capacity, reliable communication can be achieved if one chooses proper encoding and decoding techniques [16]. The design of good codes and of efficient decoding methods was initiated by Hamming, Golay, and others in late 1940s and still is an important research topic. Two structurally different types of codes are in common use today: block codes and convolutional codes.

1.3.1 Block Codes

The encoder for a block code divides the information sequence into message blocks of \( k \) information bits or symbols. There are a total of \( 2^k \) different possible messages. The
Table 1.2: A (15,7) binary block code example

<table>
<thead>
<tr>
<th>Messages</th>
<th>Codewords</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0000)</td>
<td>(0000000)</td>
</tr>
<tr>
<td>(1000)</td>
<td>(1101000)</td>
</tr>
<tr>
<td>(0100)</td>
<td>(0110100)</td>
</tr>
<tr>
<td>(1100)</td>
<td>(1011100)</td>
</tr>
<tr>
<td>(0010)</td>
<td>(1110010)</td>
</tr>
<tr>
<td>(1010)</td>
<td>(0011010)</td>
</tr>
<tr>
<td>(0110)</td>
<td>(1001110)</td>
</tr>
<tr>
<td>(1110)</td>
<td>(0101110)</td>
</tr>
<tr>
<td>(0001)</td>
<td>(1010011)</td>
</tr>
<tr>
<td>(1001)</td>
<td>(0111011)</td>
</tr>
<tr>
<td>(0101)</td>
<td>(1100111)</td>
</tr>
<tr>
<td>(1101)</td>
<td>(0001101)</td>
</tr>
<tr>
<td>(0011)</td>
<td>(0100011)</td>
</tr>
<tr>
<td>(1011)</td>
<td>(1001011)</td>
</tr>
<tr>
<td>(0111)</td>
<td>(0010111)</td>
</tr>
<tr>
<td>(1111)</td>
<td>(1111111)</td>
</tr>
</tbody>
</table>

encoder transforms each message independently into an n-bit codeword. Therefore, corresponding to the $2^k$ different possible messages, there are $2^k$ different possible codewords at the encoder output. This set of $2^k$ codewords of length n is called an (n,k) block code and $k \leq n$.

![Figure 1-3: Block code types.](image)

An example of a binary block code with (n,k)=(15,7) is given in Table 1.2. Since the n-symbol output codeword depends only on the corresponding k-bit input mes-
sage, the encoder is *memoryless*. Since SRAMs have blocks of words in their nature, block codes are chosen to be worked with.

There are many different block code types. Most important ones are Hamming, BCH, and Reed Solomon Codes given in Figure 1-3. Hamming codes are capable of correcting single errors. These codes are still very popular due to their easy encoding and decoding. On the other hand, for multiple error correction, BCH and Reed Solomon codes are commonly used. Reed Solomon codes are more suitable for burst error correction whereas BCH codes are appropriate for random error correction. For instance, in CD and DVDs Reed Solomon codes are widely used. Due to random nature of soft errors in SRAMs, BCH codes are investigated in more detail in this thesis.

### 1.3.2 Convolutional Codes

The encoder for a convolutional code also accepts k-bit blocks of the information sequence and produces an n-bit encoded sequence. However, each encoded block depends not only on the corresponding k-bit message, but also on the previous message blocks. Hence, the encoder has *memory*. Convolutional codes are more suitable for data stream processing.

### 1.4 Related Work

Since energy and power are directly related to each other, low-$V_{DD}$ operation have been an important research topic for low-power design. A sub-$V_T$ microprocessor operational down to 180mV was presented in [17]. This design uses a mux-based memory which is the first example of a sub-$V_T$ memory reported. [18] demonstrated a sub-$V_T$ SRAM operational down to 400mV in 65nm CMOS. This SRAM uses 10T bitcell design to have a high density in the array and uses peripheral assist circuits to enable sub-$V_T$ operation. The work in [19] proposed an SRAM operating down to 350mV in 65nm CMOS. This design uses 8T bitcells and sense amplifier redundancy to improve yield. In [20] an 8T SRAM operational down to 250mV is presented. This
SRAM is designed in 65nm CMOS process and uses hardware reconfigurability as a solution to power and area overheads due to peripheral assist circuitry. However, none of these designs are focusing on reliability and they do not use ECC.

A SECDED code is capable of correcting one error and detecting all possible double errors. It is commonly used in memories. Some examples are the L2 and L3 caches of Itanium processor [21], L2 cache of Power4 [22], and L2 cache of UltraSparc processor. More recent designs are increasingly using multiple error correction. For instance, L3 cache of 8-core Xeon Processor uses double error correction, triple error detection (DECTED) ECC [6]. Also in [23] an SRAM in 65nm is designed using multi-bit ECC.

There are multiple ways to implement the BCH decoders. A sequential decoding scheme is used in [24] for a Flash memory using 5-bit binary BCH ECC. The time overhead for this design is 500ns for error detection and 250μs for error correction. Clearly, error correction takes 500X more time than detection which might be tolerable for a flash memory but not tolerable for an SRAM. In [25], a look-up table based decoder is used for a faster implementation. However, this design results in a much bigger area.

1.5 Thesis Contribution

This thesis presents a low voltage SRAM design in 65nm CMOS process which is designed as the instruction and data caches for a ReISC microprocessor chip. Moreover, ECC design considerations suitable for SRAM design are investigated. A double error correcting binary BCH code is chosen as the ECC scheme. Three different binary BCH decoders are compared in terms of their power, area and latency. A decision-tree based decoder is analyzed. Lastly, ECC and redundancy are compared for their ability to correct hard errors.

The first part of this thesis presents an SRAM that is designed for both sub-Vt and above-Vt operation. The target operational supply voltage ranges from 400mV which is in sub-$V_T$ region to 1.2V. An 8T bitcell is used to construct a high density
array. Different voltage supplies are provided for array, periphery and write word line driver for leakage and performance optimization, better integration to the ReISC microprocessor and low voltage operation. On-chip cache design of this SRAM is also presented.

The second part of the thesis focuses on the fault tolerance design considerations suitable for SRAM applications. Binary BCH ECC is investigated in both theory and implementation. Three different decoders for this ECC is compared in terms of their synthesized leakage, area, and latency. An alternative to the conventional iterative and look-up table (LUT) decoders is presented. Lastly, ECC and redundancy are compared for their ability to correct hard errors for different SRAM word lengths.
Chapter 2

Low Voltage SRAM Design in 65nm CMOS

2.1 6T Bitcell at Low Supply Voltages

To provide background for understanding the basics of SRAM operation, a brief overview of the traditional 6T SRAM bitcell and its operation is presented in this section. Figure 2-1 shows a schematic for the basic 6T bitcell. This bitcell is made up of back-to-back inverters that store the cell state ($M_1, M_3, M_4,$ and $M_6$) and access transistors for write and read operations ($M_2$ and $M_5$).

2.1.1 Read, Write and Hold Operation

For a read access, the first phase is precharging the bitlines ($BL$ and $BLB$) to high. Afterwards, bitlines are allowed to float and the wordline ($WL$) is asserted. In this second phase the storage nodes of the bitcell ($Q$ or $QB$) that holds a '0' will pull its bitline low through its access transistor. Lastly, a sense amplifier detects this differential voltage on the bitlines and the output is latched.

For a write access, WL is asserted to turn the access transistors on. The bitlines are driven to the correct differential value that will be written in the bitcell. If the sizing is done correctly, data held by the back-to-back inverters is driven to the new
data.

For hold operation the wordlines are low and the access transistors are off. Because of the positive feedback between the back-to-back inverters, storage nodes can hold the data indefinitely in a stable state.

2.1.2 6T Bitcell Challenges at Low Supply Voltages

The ability of the back-to-back inverters to maintain their state is measured by the bitcell’s static noise margin (SNM) [26]. SNM quantifies the amount of voltage noise required at the internal nodes of a bitcell to flip its data.

SNM during hold shows the fundamental limit of a bitcell functionality, therefore it is a very important metric. However, cell stability during write and read presents a more significant SNM limitation. During a read operation, bitlines are precharged to '1' and the word lines are asserted. The internal node of the bitcell that stores a zero gets pulled upward through the access transistors. This increase in voltage severely degrades the SNM at read operation (RSNM). Similarly, at write operation access transistors need to overpower the stored data hold by back-to-back inverters. This ability to overpower the internal feedback can be quantified by SNM for write operation (WSNM).

6T bitcell provides a good density, performance, and stability balance. Hence, this
cell became an industry standard over many years. However, it fails to operate in low
voltages [19] due to the RSNM and WSNM problems. This phenomenon motivated
the design of new bitcell topologies.

2.2 8T Bitcell at Low Supply Voltages

![8T bitcell schematic](image)

Figure 2-2: Schematic for an 8T bitcell.

In order to achieve low-voltage operation, a new 8T bitcell topology is introduced.
This bitcell is firstly used by [27] in 2005 and since then it has gained substantial
popularity. For instance, in 2007 [28] presented an 8T SRAM in 65nm CMOS which
is functional down to 350mV. Two years later, another 8T SRAM in 65nm CMOS
operational in a wide supply voltage range of 250mV to 1.2V is presented by [29].

The schematics for this bitcell is given in Figure 2-2. The 8T bitcell has two
extra transistors ($M_7$, $M_8$) that generate an independent read port. This new port
provides a disturb-free read mechanism. Read and write port separation allows each
operation to be independently optimized. This permits improvement of the cell write
margin, which, combined with excellent read stability, enables a variation-tolerant
SRAM cell. Without RSNM and WSNM problem, the worst-case stability condition
for an 8T bitcell is determined by its hold static noise margin (HSNM). A dramatic
stability improvement can thus be achieved without a tradeoff in performance since
a read access is still performed by two stacked NMOS transistors.
Figure 2-3: Read and hold margin distribution at $V_{DD}=500$ mV for 22nm predictive technology. The model is developed by the Nanoscale Integration and Modeling (NIMO) Group at ASU.

A Monte Carlo analysis has been performed on a predictive 22nm bitcell. Figure 2-3 shows the RSNM and HSNM distributions for this model. At 500mV supply voltage, hold stability is preserved for each of the 10,000 data points. However, almost half of the bitcells fail to operate due to RSNM problem. WSNM violations appear in the same manner (not shown in Figure 2-3). Hence, in advanced technologies, 6T bitcell which is limited by RSNM and WSNM is not a proper choice for low-voltage operation. On the contrary, 8T bitcell is limited by HSNM and it can be a better choice at low-voltages.

One disadvantage of 8T bitcell is that it is not suitable for column-interleaved designs. During a write operation, un-accessed bits on the accessed row experience a condition that is equivalent to a read disturb on a 6T cell. Effectively, separating read and write ports does not diminish RSNM problem anymore. Column-interleaved architectures are preferred for better soft-error immunity. Lack of column-interleaving is yet another motivation for investigating multi-bit ECC schemes for SRAMs.

The sensing network is an important part of SRAM design. In SRAMs, sensing can be single-ended or differential depending on the array architecture and bitcell design. Due to the single ended read port, single-ended sense amplifiers are used with the 8T bitcell. On the other hand, write operation is performed the same way it is
done in 6T bitcell.

One other advantage of 8T bitcell is its compact layout which results into only a 30% area increase. Figure 2-4 shows the layouts of a standard 6T and an 8T bitcell. The read port extends the layout without adding any significant complication to the layout.

## 2.3 Architecture of the 8T SRAM Design in 65nm CMOS

An 8T SRAM is designed in 65nm low-power, high $V_T$ CMOS process. Each SRAM block contains 256 rows and 128 columns, therefore the memory capacity is 32kb. 4 macros of SRAM design are used for instruction and data caches of the ReISC microprocessor chip. The architecture of the SRAM block can be seen in Figure 2-5.

### 2.3.1 Power Supply Voltages Used in the Design

The SRAM uses three different power supply voltages:

1. $V_{DDW}$: Write word line (WWL) voltage.
2. $V_{DDA}$: Array supply voltage.
3. $V_{DDP}$: Periphery supply voltage that is used in row, column, and timing circuitries as well as address decoder.

For an 8T bitcell, RSNM problem is diminished since the read port is separated from the write port. In order to achieve low voltage operation, WSNM of 8T bitcell should also be in acceptable limits in low-voltages. The conventional way is to size the standard 6T part of the 8T bitcell for better write-ability. In order to do that, the width of $M2$ and $M5$ given in Figure 2-2 should be increased. For the specific 65nm CMOS technology, our analysis showed that a significant area increase is needed for the target minimum supply voltage operation ($400$ mV).

Other techniques are investigated in the literature to enhance write margin. One possible solution is to use a relatively higher supply voltage for WWL. Since our
design is an integral part of a microprocessor, we took the advantage of the supply voltages already accessible. Having solved the WSNM problem, the bitcell is only limited by HSNM.

Separate voltage sources for array and periphery are used. In order to prevent extra level converters between SRAM and logic, periphery works at 0.6V which is the lowest logic voltage. On the other hand, array is operational down to 0.4V. Having an extra voltage supply for array enables us to control array leakage independently.

**2.3.2 Word Length (WLen) Optimization**

Choosing the most appropriate WLen for the SRAM is an optimization problem. For four different WLen (32, 64, 128, and 256 bits), predicted total power consumptions are found in Figure 2-6. In this analysis, the memory size is fixed to 4 x 32kb and the number of rows is constant at 256. Furthermore, access frequency ranges from 0 to 150kHz since the estimated operation frequency at $V_{DDA} = 400\text{mV}$ is around 35kHz.

![Normalized Power vs Access Frequency](image)

Figure 2-6: WLen of the SRAM is chosen to be 128 bits considering the power at idle and active states.

In Figure 2-6, access frequency being equal to zero means that the total power
is only due to leakage. Conversely, when access frequency increases, dynamic power dominates the total power consumption. SRAM power should be optimized for both regions.

To achieve 4 x 32kb data storage capacity 16, 8, 4, and 2 macros are needed for word lengths of 32, 64, 128, and 256 bits respectively. This is clearly pictured in Figure 2-7 for the systems with 32 and 64 WLens. As a first approximation, each macro will use same size address decoder and row circuitry since the number of rows is fixed. Therefore, when WLen is decreased linearly, linear increase in address decoder and row circuitry leakage power is observed. The array size is fixed so array leakage does not change. This is why we have nearly linear change in leakage power from WLen=256 to 32 bits. On the other hand, dynamic power increases with increasing WLen since more bitline capacitances are charged and discharged. WLen=256 bits has the highest slope and largest dynamic power whereas WLen=32 bits has the smallest slope.

![Figure 2-7: The configurations for (a) WLen=32 and (b) WLen=64.](image)

Considering both idle and dynamic region operations, WLen=128 bit is the best choice with its both small leakage and dynamic power.
2.3.3 Peripheral Circuit Design Considerations

The important design considerations of row circuit, column circuit, address decoder and timing circuit will be briefly explained.

Row Circuit

The row circuit consists of two blocks: read word line (RWL) and write word line (WWL) drivers. These blocks provide RWL and WWL signals to enable read or write operation, respectively.

RWL driver is a simple circuit that is made up of a few standard cells and an output buffer. On the other hand, WWL driver includes similar cells and an extra cell: an asynchronous level converter based on DCVSL gate. The schematics for WWL driver is given in Figure 2-8. RWL driver is not given since it is very similar to WWL driver excluding the level converter.

The level converter used is a ratioed circuit so its functionality and delay is very sensitive to transistor sizing. For correct operation, $M_1$ needs to overpower $M_2$. In order to size the level converter, the currents of $M_1$ and $M_2$ are compared for $V_{DS} = V_{DD} - V_T$ and $V_T$ respectively. $M_1$ is sized large enough so that it has larger current than $M_2$ for worst case operation.

![Figure 2-8: Schematics for the WWL driver including an asynchronous level converter based on DCVSL gate.](image)

RWL signal at $V_{DDP}$ voltage is asserted when decAddr, internal timing control signal WLenable, and RWin are high (RW=1 is read). On the other hand, WWL is asserted when decAddr, WLenable are high and RWin is low. The level converter
steps up the WWL signal at $V_{DPP}$ voltage into a new WWL signal at $V_{DDW}$ voltage. This way, WSNM is improved drastically.

**Column Circuit**

The column circuit consists of a precharge PMOS, a sense amplifier followed by a latch, and a flip flop that holds the input data. This circuit is very important for read operation.

In SRAMs that use 8T bitcells a read operation initiates by charging the RBL capacitance to the supply voltage. Then, RBLs are allowed to float and RWL is asserted. The differential voltage between a reference supply voltage (REF) and RBL voltage are compared by the sense amplifier. Lastly, the output is latched to its new value.

BL and BLB are charged with the input data and its inverse, respectively. Bitlines are only used in a write operation to drive the new data.

![Schematics](image)

Figure 2-9: Schematics for the NMOS input strong-arm sense amplifier and the following NOR-based latch.

In SRAMs, sensing can either be differential or single ended depending on the array architecture and bitcell design. For the 8T bitcell, a single ended sense amplifier is necessary since RBL is the only port used for read. Our sense amplifier is a strong-arm type with NMOS inputs. The schematics of this sense amplifier followed by a NOR
type latch can be seen in Figure 2-9. This design has an NMOS differential pair with cross-coupled inverters as a load. One of the inputs is connected to REF whereas the other input is connected to RBL provided externally. Four PMOS transistors are used for precharging internal nodes to $V_{DDP}$ before enabling the sense amplifier. $M_1$ is used to disable the circuitry when sense signal is low. When sense is asserted, the circuit detects any small difference between REF and RBL inputs and output is latched. The sense amplifier’s input offset voltage should be small enough to sense small changes on RBL voltage. However, in advanced technologies, transistor mismatches are becoming higher causing larger offset voltages for sense amplifiers. In Figure 2-10, input offset distribution for the sense amplifier used in this design is shown for 1,000 occurrences. This sense amplifier is designed for 50mV offset voltage. The offset voltage is measured with a 10mV step size.

![Figure 2-10: Input offset distribution for our sense amplifier.](image)

**Address Decoder**

The address decoder used in the design is constructed by using static CMOS gates to ensure functionality down to low voltages. An 8 bit address word is decoded into 256 decoded addresses (decAddr), and this signal is used in the row circuit to enabling necessary row for either a read or a write operation.
Timing Circuit

Timing circuitry is used for generating the internal timing control signals of the SRAM using CLK input only. To generate delays, we used inverters with length larger than $L_{min}$. In Figure 2-11 we see the timing diagram for these signals. Some detailed description for these signals are given below:

1. **precharge**: It is used for charging RBL capacitance to $V_{DDP}$. When precharge is low, RBL is charged to $V_{DDP}$. The precharge becomes '1' with the rising edge of the clock to float the RBL signal.

2. **snsEn**: It is used for activating sense amplifiers for a read operation.

3. **WLenable**: It is used in enabling WWL and RWL signals. This signal becomes high after a certain delay which is equal to the worst case decoder delay.

![Timing diagram](image)

Figure 2-11: Internal timing control signals of the 8T SRAM design for $V_{DDP}$ at 600mV.
2.4 Simulation Results of the SRAM

Testing results are not ready to be included yet. We performed nanosim analysis and tested functionality. Important properties of this design is tabulated in Table 2.1.

Table 2.1: 65nm CMOS SRAM design properties and simulation results.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>8T SRAM @ 0.4V</th>
<th>8T SRAM @ 1.2V</th>
</tr>
</thead>
<tbody>
<tr>
<td>Macro Area</td>
<td>mm²</td>
<td>mm²</td>
</tr>
<tr>
<td>Access Time</td>
<td>s</td>
<td>35μs</td>
</tr>
<tr>
<td>Read Energy/b</td>
<td>19 fJ</td>
<td>76 fJ</td>
</tr>
<tr>
<td>Write Energy/b</td>
<td>22 fJ</td>
<td>86 fJ</td>
</tr>
<tr>
<td>Bitcell leakage</td>
<td>1.55 pW</td>
<td>16.5 pW</td>
</tr>
</tbody>
</table>

In Table 2.1 it is shown that the target operation frequency for this SRAM is around 35kHz at 0.4V and 500MHz at 1.2V providing more than 4 orders of magnitude performance scaling. The leakage per bitcell is decreased more than 10X from 1.2V to low-voltage operation at 400mV.

The Figure 2-12 shows the layout of a single 256X128 SRAM block.

Figure 2-12: Layout of the 32kb low-voltage SRAM designed in 65nm.
2.5 Test Chip Architecture

Table 2.2 shows the on chip memories included in the ReISC chip. 4 blocks of 32kb 256X128 SRAM designs are used in this design. They are used as data and instruction RAMs. ReISC design includes numerous blocks such as the CPU, a custom SRAM, couple of real time clocks, serial ports, an A/C, on-chip caches and off-chip memory interfaces. It is a very complex system with around 43k gates.

Table 2.2: On-chip memories used in low-voltage ReISC processor chip

<table>
<thead>
<tr>
<th>Cache #</th>
<th>Capacity</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>32kbits</td>
<td>Data RAM</td>
</tr>
<tr>
<td>2</td>
<td>32kbits</td>
<td>Data RAM</td>
</tr>
<tr>
<td>3</td>
<td>32kbits</td>
<td>Instruction RAM</td>
</tr>
<tr>
<td>4</td>
<td>32kbits</td>
<td>Instruction RAM</td>
</tr>
</tbody>
</table>

Figure 2-13 shows the layout of the ReISC microprocessor chip. The D-RAM and I-RAMs account for 23% of the total chip area.

![Figure 2-13: Layout of the low-voltage ReISC microprocessor chip. On-chip SRAM blocks are highlighted.](image)
Chapter 3

Error Correction Coding For SRAM Design

Figure 3-1: The block diagram for an SRAM protected by ECC.

Figure 3-1 pictures the block diagram for an SRAM which is protected by ECC. For a write operation, new check bits need to be created. Therefore, the encoder is on the critical path. For a read operation, received word needs to be decoded. Decoding consists of two phases: error detection and error correction. In the first phase, the error detection is performed which is also known as syndrome generation. Syndrome being zero indicates no error, so the second phase is bypassed and the received word is passed as the output data. Conversely, if an error is detected, the error correction block is activated and the output data is calculated from the check bits.
Due to reasons explained in Chapter 1, binary BCH codes are selected as the most suitable ECC scheme for SRAM design. In this chapter, we investigate this coding class in the standpoint of theory, hardware implementation and optimization for SRAMs.

A summary of algebra and coding theory that will aid in understanding binary BCH codes is provided in Appendix A. For more detailed information, [14] can be referred.

3.1 Definition of BCH Codes

Binary BCH coding class is a remarkable generalization of the Hamming codes for multiple error correction. It is also a very important subset of block codes. For any positive integers \( m \) (\( m \geq 3 \)) and \( t \) (\( t < 2^{m-1} \)), there exists a binary BCH code with the following parameters:

- Block length: \( n = 2^m - 1 \),
- Number of parity (or check) bits: \( n-k \leq mt \),

![Figure 3-2: Number of Parity Bits and Area Overhead vs. Data Word Length.](image-url)
This code is capable of correcting any combination of $t$ or fewer errors in a block of $n = 2^m - 1$ digits. We call this a $t$-error-correcting BCH code.

In hardware implementation of ECC for SRAM design, an important consideration was to choose a proper block length. This block length is the word length (WLen) of SRAM in the scope of this thesis.

The area overhead associated with parity bits has a dependency on WLen. The parity area overhead for a double error correcting BCH code is shown in Figure 3-2. For instance, for $WLen = 16$, 10 parity bits are required per word which means 62% area overhead due to parity bits only. The overhead becomes less than 12% for word lengths of 128 bits. Increasing the WLen not only results into diminishing returns in area overhead, but also increases decoding and encoding complexity. Hence, working with a $WLen = 128$ is a proper choice both SRAM power considerations and for the parity area overhead optimization.

### 3.2 Decoding of BCH Codes

The first decoding algorithm for binary BCH codes was devised by Peterson in 1960. Then Peterson’s algorithm was generalized and refined by Chien, Berlekamp, and others [30], [31]. Among all the decoding algorithms, Berlekamp’s iterative algorithm and Chien’s search algorithm are the most efficient ones. Here, decoding of BCH codes will be briefly summarized in order to provide a basis for our tree-based decoder. For more detailed information, [14] is referred.

Suppose that a codeword $\mathbf{v}(X) = v_0 + v_1 X + v_2 X^2 + \ldots + v_{n-1} X^{n-1}$ is stored in the SRAM. After a soft error, we receive the following word:

$$r(X) = r_0 + r_1 X + r_2 X^2 + \ldots + r_{n-1} X^{n-1}. \quad (3.1)$$

Let $e(X)$ be the error pattern. Then,

$$e(X) = u(X) + r(X) \quad (3.2)$$
As usual, the first step of decoding a code is to compute the syndrome from the received word, \( r \). For decoding a \( t \)-error-correcting BCH code, the syndrome is a 2\( t \)-tuple:

\[
S(X) = (S_1, S_2, \ldots, S_{2t}) = r \cdot H^T
\]

where \( H \) is given as:

\[
H = \begin{bmatrix}
1 & \alpha & \alpha^2 & \ldots & \alpha^{n-1} \\
1 & \alpha^2 & \alpha^4 & \ldots & \alpha^{2(n-1)} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \alpha^{2t} & \alpha^{4t} & \ldots & \alpha^{2t(n-1)}
\end{bmatrix}
\]  

From (3.3) and (3.4) we find that the \( i^{th} \) component of the syndrome is:

\[
S(X) = r(\alpha^i) = r_0 + r_1\alpha^i + r_2\alpha^{2i} + \ldots + r_{n-1}\alpha^{(n-1)i}
\]  

for \( 1 \leq i \leq 2t \). Note that the syndrome components are elements in the field \( \text{GF}(2^m) \).

From (A.1), it can be seen that the elements \( \alpha, \alpha^2, \ldots, \alpha^{2t} \) are roots of each code polynomial, \( v(\alpha^i) = 0 \). So (3.3) can be written as:

\[
S(X) = e(\alpha^i)
\]  

From (3.6), we see that there is one to one dependence between error and syndrome. Suppose that the error pattern \( e(X) \) has \( \nu \) errors at locations \( X^{j_1}, X^{j_2}, \ldots, X^{j_\nu} \). So error pattern can be written as:

\[
e(X) = X^{j_1} + X^{j_2} + \ldots + X^{j_\nu}
\]
So the syndrome can be written as:

\[
S_1 = \alpha^{j_1} + \alpha^{j_2} + \ldots + \alpha^{j_v} \\
S_2 = (\alpha^{j_1})^2 + (\alpha^{j_2})^2 + \ldots + (\alpha^{j_v})^2 \\
\vdots \\
S_{2t} = (\alpha^{j_1})^{2t} + (\alpha^{j_2})^{2t} + \ldots + (\alpha^{j_v})^{2t}
\]

where the \( \alpha^{j_1}, \alpha^{j_2}, \ldots, \alpha^{j_v} \) are the unknowns. Any method for solving these equations is a decoding algorithm for the BCH codes. Once we find \( \alpha^{j_1}, \alpha^{j_2}, \ldots, \alpha^{j_v} \), we can find the error locations from the powers. For convenience let:

\[
\beta_t = \alpha^{j_t}
\]

These elements are called \textit{error location numbers}. So we can express the equations of \((3.8)\) in the following form:

\[
S_1 = \beta_1 + \beta_2 + \ldots + \beta_v \\
S_2 = (\beta_1)^2 + (\beta_2)^2 + \ldots + (\beta_v)^2 \\
\vdots \\
S_{2t} = (\beta_1)^{2t} + (\beta_2)^{2t} + \ldots + (\beta_v)^{2t}
\]

Now let's define the following polynomial:

\[
\sigma(X) = (1 + \beta_1 X)(1 + \beta_2 X)\ldots(1 + \beta_v X) \\
= a_0 + a_1 X + a_2 X^2 + \ldots + a_v X^v
\]

The roots of \( \sigma(X) \) are \( \beta_1^{-1}, \beta_2^{-1}, \ldots \) which are the inverses of the error-location numbers. For this reason, \( \sigma(X) \) is called the \textit{error location polynomial}. 
3.3 Hardware Implementation

The architecture of a generic BCH codec implementation is given in Figure 3-3. It is clearly seen from this figure that the encoder and the syndrome generator (error detection block) are on the critical path of the codec. Yet, these blocks are much simpler in terms of hardware implementation. The most complex part of the codec is the decoder and designing effective decoders for BCH codes have been a very popular research topic for many years.

The hardware implementation for two different double error correcting BCH codecs are implemented in this thesis using Verilog. These decoders are synthesized using design compiler of Synopsys with cell libraries. The decoders of these two designs can be defined as a look-up table based decoder (LUT decoder) and the analyzed decision-tree based decoder. These two types are selected since they are combinational and enable much faster decoding than the conventional iterative decoding scheme proposed by Berlekamp. Berlekamp’s iterative decoder requires a small area, however it takes more than a hundred clock cycles for error correction with 128 bits of word length. It is commonly used for applications that can tolerate this latency.

In the following sections, each block of these two codes will be explained and compared in terms of area, latency and leakage power. Active power is not included since the design tool does not provide reliable numbers. The third decoder is not synthesized since it has a very long error correction latency.
3.3.1 Encoder and Syndrome Generator Implementation

The encoder and syndrome generator of binary BCH codes can be designed as very fast combinational circuits since they only implement matrix multiplication. Simple XOR trees can be used for these blocks.

The generator and parity check matrices \((k \times n)\) and \((n - k) \times n\) matrices respectively) are generated using \texttt{cyclgen} function of Matlab. Then, the XOR trees are implemented using synthesis tool.

In order to visualize the circuitry of encoder and syndrome generator blocks, a simpler \((15,7)\) encoder is implemented. The syndrome generator hardware implementation would be very similar to encoder circuit so only the encoder circuit is given in this thesis. The encoder and syndrome generators of the actual code used for the SRAM with \(WL = 128\) generates more complex circuit versions of the shown encoder but use the same idea.

The matlab code that generates generator matrix is as follows:

- \texttt{genpoly} = [1 0 0 0 1 0 1 1 1];
- \texttt{[H,G] = cyclgen(15, genpoly, 'sys');}

\[
G = \begin{bmatrix}
1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}.
\]

The encoding can be calculated as:

\[
v = u \cdot G
\]
\[ v_0 = u_0 \oplus u_1 \oplus u_3 \\
v_1 = u_1 \oplus u_2 \oplus u_4 \\
v_2 = u_2 \oplus u_3 \oplus u_5 \\
v_3 = u_3 \oplus u_4 \oplus u_6 \\
v_4 = u_0 \oplus u_1 \oplus u_3 \oplus u_4 \oplus u_5 \\
v_5 = u_1 \oplus u_2 \oplus u_4 \oplus u_5 \oplus u_6 \\
v_6 = u_0 \oplus u_1 \oplus u_2 \oplus u_5 \oplus u_6 \\
v_7 = u_0 \oplus u_2 \oplus u_6 \\
v_8 = u_0 \\
v_9 = u_1 \\
v_{10} = u_2 \\
v_{11} = u_3 \\
v_{12} = u_4 \\
v_{13} = u_5 \\
v_{14} = u_6 \]

Therefore, we get the codeword bits as follows:

\[ \begin{bmatrix} v_0, v_1, v_2, v_3, v_4, v_5, v_6 \\ v_7 \\ v_8 \\ v_9 \\ v_{10} \\ v_{11} \\ v_{12} \\ v_{13} \\ v_{14} \end{bmatrix} = G \cdot \begin{bmatrix} u_0, u_1, u_2, u_3, u_4, u_5, u_6 \end{bmatrix} \]

(3.10)

Therefore, we get the codeword bits as follows:

The encoder circuit for this code is given in Figure 3-4.

The encoder and syndrome generator blocks for the SRAM with WL = 128 of the two codecs are implemented the same way. Table 3.1 shows the synthesized results of the area, power and latency for the encoder. Table 3.2 shows the same results for the syndrome generator.
After the syndrome is generated, the error detection can be done. If the syndrome bits are all zero, there is no error in the received word therefore the rest of the decoder can be bypassed. If it is not equal to zero, error correction block is activated. Therefore, the latency associated with the encoder and decoder blocks are very important since encoding and syndrome generation are done regardless of an error detection or not.

Table 3.1: Synthesis results for the encoder.

<table>
<thead>
<tr>
<th>Area</th>
<th>2672 $\mu m^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timing</td>
<td>0.71 $ns$</td>
</tr>
<tr>
<td>Leakage Power</td>
<td>10.1 $nW$</td>
</tr>
</tbody>
</table>

Table 3.2: Synthesis results for the syndrome generator.

<table>
<thead>
<tr>
<th>Area</th>
<th>2123 $\mu m^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timing</td>
<td>1.48 $ns$</td>
</tr>
<tr>
<td>Leakage Power</td>
<td>7.9 $nW$</td>
</tr>
</tbody>
</table>
3.3.2 Decoder Implementation

The decoding of binary BCH codes has three steps. The first one is generating the syndrome. Second one is determining the error location polynomial from the syndrome. The last step is determining the error-location numbers by finding the roots of error location polynomial and correcting the errors in the received word.

![Diagram of decoding blocks](image)

Figure 3-5: The block diagram for the three decoding blocks: (a) LUT decoder, (b) Decision-tree based decoder, and (c) Iterative decoder.

In Figure 3-5 block diagrams for three different decoding algorithms that are considered in this thesis are shown. The first two of these decoders are implemented in hardware. The third one is not suitable for SRAM design so it is not implemented. Decision-tree based decoder is the one that is analyzed in this thesis.

1. LUT Decoder

This decoder takes the syndrome as an input and outputs the error associated with this syndrome. The syndrome for our design is 16 bits and the error is 144 bits. Hence every location of this table has 144 bits of data, in other words, the table needs 144 columns. The number of rows (NoR) for the table can be calculated as follows:

\[
\text{NoR} = \binom{144}{1} + \binom{144}{2} = 144 + 10296 = 10440
\]  
(3.12)
Table 3.3 shows the synthesized results of the area, power and latency for this decoder.

Table 3.3: Synthesis results for the LUT decoder.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Area</strong></td>
<td>59161 um²</td>
</tr>
<tr>
<td><strong>Timing (no error detected)</strong></td>
<td>2.1 ns</td>
</tr>
<tr>
<td><strong>Timing (error detected)</strong></td>
<td>2.3 ns</td>
</tr>
<tr>
<td><strong>Leakage Power</strong></td>
<td>176 nW</td>
</tr>
</tbody>
</table>

2. Decision-tree Based Decoder

In the decision tree-based decoder, the closed form of the iterative algorithm is used for computing the error location polynomial \[32\], \[32\]. Using Galois field arithmetic a combinational decoder is implemented. In the Chien-search block, we used a fast algorithm for computing the roots of error location polynomials up to degree 11 \[33\], \[34\].

Table 3.4 shows the synthesized results of the area, power and latency for this decoder.

Table 3.4: Synthesis results for the Tree decoder.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Area</strong></td>
<td>30092 um²</td>
</tr>
<tr>
<td><strong>Timing (no error detected)</strong></td>
<td>5.01 ns</td>
</tr>
<tr>
<td><strong>Timing (error detected)</strong></td>
<td>16.2 ns</td>
</tr>
<tr>
<td><strong>Leakage Power</strong></td>
<td>102 nW</td>
</tr>
</tbody>
</table>

Compared to LUT decoder, the tree-based decoder is almost 50% smaller in area and leakage power. However, we increase the critical path around 2.5ns for error detection and around 14ns for error correction. Compared to the iterative decoder, the tree-based decoder has larger area. On the other hand, with 16ns, error correction latency is much smaller than more than hundred clock cycles needed for the iterative decoder. Therefore the analyzed decision-tree based decoder does not have the extreme area of a LUT-decoder and extreme error-correction latency of an iterative-decoder.
### 3.4 Using ECC to Increase SRAM Yield versus Redundancy

Scaling down power supply voltage degrades bitcell stability due to device variation effects. Use of higher-order BCH codes capable of correcting multiple bits per word is explored in order to address both radiation and bitcell variation induced errors. In particular, we considered how error correcting capability might be efficiently scaled with the operating voltage. Increasing error correction capability increases the area overhead whereas it decreases the minimum supply voltage at which the SRAM can work. This tradeoff is shown in Figure 3-6(a).

![Figure 3-6: For a 65nm, 32kB SRAM with 95% yield and 128 bits of word length (a) The minimum supply voltage drops when error correction capability is increased (b) There is an optimum point for the energy (parity bit overhead is considered only)](image)

Assuming that the energy can be simplified as $C_{total} \times V_{DD}^2$, the capacitance increases and $V_{DD}$ decreases as the error correction capability increases. There is an optimum point for the normalized energy and this optimum point is at 6-7 ECC complexity for the 65nm bitcell. Yet, in this analysis we ignore the decoder and encoder area and complexity. Actually, implementing more than 2 error correction makes the decoder and encoder extremely complex and large. Therefore, error correction can
also be used for yield errors.

![Graphs showing Redundancy vs. ECC analysis results for word lengths of 256 bits (a) and 128 bits (b).](image)

Figure 3-7: For a 65nm, 32kB SRAM with 95% yield Redundancy vs. ECC analysis results. (a) Word Length = 256 bits (b) Word Length = 128 bits

On the other hand, using redundancy is a second way to cope with variation induced errors. In order to understand whether ECC or redundancy is more effective for solving variation induced errors, we compared these two in terms of how much voltage reduction can be achieved for a certain area overhead. We assume that one ECC is reserved for soft error reduction and the rest of the error-correction capability is used for yield errors. Area overhead with a single error correction is fixed to 1. We fixed the memory size, target yield and word length of the SRAM.

In Figure 3-7(a) the word length is 256 bits whereas in Figure 3-7(b) the word length is 128 bits. When the word length is increased, the area overhead of ECC becomes smaller therefore ECC curve shifts down. So for the larger word length of 256, using three or more bits of ECC is more area efficient than using redundancy. This trade-off is shown in Figure 3-7.
Chapter 4

Conclusions

This thesis presents a low-voltage SRAM design in 65nm CMOS. It also investigates ECC design considerations suitable for SRAM design. Comparison of ECC and redundancy is also presented in terms of their hard error correction capabilities. This chapter summarizes the important conclusions of this research and discusses opportunities for future work.

4.1 Summary of Contributions

1. Simulation results of a 65nm CMOS low-voltage 8T SRAM design.

2. Comparison of three different ECC decoders: A look-up table based decoder, an iterative decoder and the analyzed decision-tree based decoder. analyzed decoder trades off area and power vs. latency and provides an alternative design.

3. Comparison of SRAM variation induced error reduction using ECC vs. using redundant rows.

4.2 Low Voltage SRAM Design in 65nm CMOS

In Chapter 2, an 8T low-voltage SRAM design is presented. This design will be fabricated in 65nm low-power, high $V_T$ process. It is designed for the instruction
and data caches of the low-voltage ReISC processor. One block of this SRAM has 256 rows and 128 columns and the processor will contain 4 blocks of SRAM. Target operation voltage of the SRAM array is 400mV to 1.2V.

In order to enable the low-voltage operation, we used 8T bitcells. This way, read port is separated from the write port and read static noise margin problem is eliminated. In order to increase write static noise margin, a relatively higher write word line (WWL) voltage is used. This is possible since an extra voltage supply is available from the ReISC processor.

Designing a memory as a part of a larger system introduces more considerations. Performance and energy models are useful for the memory designer and the designer of the larger architecture to fully understand the dynamics and trade-offs of the SRAM.

SRAM design achieves more than four orders of magnitude performance scaling (from 30kHz to 500MHz) over the voltage range. Leakage per bitcell scales more than 10X from 1.2V to 0.4V and this result emphasizes low-voltage SRAM design for low-power applications.

### 4.3 Error Correction Coding for SRAM

In Chapter 3, binary BCH coding for double error correction is investigated. The most complex part of the error correction block is the decoder. Conventionally, for binary BCH codes, sequential decoding algorithm based decoders are used in the literature. However, this iterative decoder results into more than a hundred clock cycle latency for error correction which cannot be tolerated for SRAM designs. In order to address this latency problem, we focused on two combinational decoder designs. The first one is a look-up table (LUT) based BCH decoder which is commonly used in error correction of SRAM designs. A second design is an alternative decoder, which is based on a decision-tree algorithm. This decoder uses the closed solution of Berlekamp's iterative algorithm resulting into a combinational solution. It also uses a fast algorithm for computing roots of error location polynomial rather than the conventional Chien search. Compared to LUT decoder, this alternative decoder has
a 2X area and leakage reduction in expense of increasing the latency. 16ns is needed for error correction at 1.2V whereas more than 100 cycles are needed for the iterative algorithm.

In order to correct hard errors both redundancy and ECC can be used. Adding spare rows is compared to using ECC in terms of the area overhead vs. supply voltage reduction. For a 95% yield, 32kb memory size, and for a word length of 256 bits, using extra two or more error correction results into more supply voltage reduction than adding spare rows. This trade-off is also shown in Chapter 3.

4.4 Future Work

Scaling the transistor sizes as well as increase in on-chip SRAM sizes makes low-power and fault tolerant SRAM design more challenging. The effect of variation is exacerbated at low voltages and device operation is altered significantly. Intelligent design techniques for increasing noise margins are needed for low-power operation.

Reliability is also degraded with increased levels of SRAM sizes at advanced processes. Multiple ECC blocks require considerable area, power and latency. Achieving better algorithms for error correction blocks is an open question.
Appendix A

Overview of Galois Fields and Linear Block Codes

A.1 Galois Field Definition and Algebra

[14] The purpose of this section is to provide an elementary knowledge of algebra and coding theory that will aid in understanding binary BCH codes.

A.1.1 Definition of Galois Fields

A field is a set of elements in which addition, subtraction, multiplication, and division can be performed without leaving the set. Addition and multiplication must satisfy commutative, associative, and distributive laws. For instance, the set of real numbers is a field with infinite number of elements under real-number addition and multiplication.

Consider the set \{0, 1\} together with modulo-2 addition and multiplication, defined in Table A.1. The set \{0, 1\} is a field of two elements under modulo-2 addition and modulo-2 multiplication. This field is called a binary field and is denoted by GF(2). It plays an important role in coding theory. Similarly the set of integers \{0, 1, 2, ... , p-1\} is also a field of order \(p\) under modulo-\(p\) addition and multiplication. This field is constructed by a prime number, \(p\) and is called a prime field and is
denoted by $GF(p)$.

For any positive integer $m$, it is possible to extend the prime field $GF(p)$ to a field of $p^m$ elements, which is called an extension field of $GF(p)$ and is denoted by $GF(p^m)$. Finite fields are also called Galois field, in the honor of its discoverer. Codes with symbols from the binary field $GF(2)$ or its extension $GF(2^m)$ are most widely used in digital data transmission and storage systems as well as this thesis.

Consider that we have three elements: 0, 1 and $\alpha$. Now we have the following set of elements on which a multiplication operation is defined:

$$F = \{0, 1, \alpha, \alpha^2, \ldots, \alpha^j, \ldots\}$$

(A.1)

It can be shown that $F^* = \{0, 1, \alpha, \alpha^2, \ldots, \alpha^{2^{m-2}}\}$ is a Galois field of $2^m$ elements, $GF(2^m)$. There are three different representations of $GF(2^m)$: power, polynomial and n-tuple representations. In Table A.2 the three representations of the elements of field $GF(2^4)$ are given.

### A.1.2 Binary Field Arithmetic

In binary arithmetic we use modulo-2 addition and multiplication given in Table A.1. Briefly, this arithmetic is equivalent to ordinary arithmetic but considers 2 to be equal to 0 (i.e., $1 + 1 = 2 = 0$). Note that since $1 + 1 = 0$, $1 = -1$. Hence, in binary arithmetic, subtraction is equal to addition.

Next, let’s consider computations with polynomials whose coefficients are from the binary field $GF(2)$. A polynomial $f(X)$ with one variable $X$ and with coefficients
Table A.2: Three representations for the elements of GF(2^4) generated by primitive polynomial \( p(X) = 1 + X + X^4 \)

<table>
<thead>
<tr>
<th>Power representation</th>
<th>Polynomial representation</th>
<th>4-tuple representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>(0 0 0 0)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>(1 0 0 0)</td>
</tr>
<tr>
<td>( \alpha )</td>
<td>( \alpha )</td>
<td>(0 1 0 0)</td>
</tr>
<tr>
<td>( \alpha^2 )</td>
<td>( \alpha^2 )</td>
<td>(0 0 1 0)</td>
</tr>
<tr>
<td>( \alpha^3 )</td>
<td>( \alpha^3 )</td>
<td>(0 0 0 1)</td>
</tr>
<tr>
<td>( \alpha^4 )</td>
<td>( 1 + \alpha )</td>
<td>(1 1 0 0)</td>
</tr>
<tr>
<td>( \alpha^5 )</td>
<td>( \alpha + \alpha^2 )</td>
<td>(0 1 1 0)</td>
</tr>
<tr>
<td>( \alpha^6 )</td>
<td>( \alpha^2 + \alpha^3 )</td>
<td>(0 0 1 1)</td>
</tr>
<tr>
<td>( \alpha^7 )</td>
<td>( 1 + \alpha + \alpha^3 )</td>
<td>(1 1 0 1)</td>
</tr>
<tr>
<td>( \alpha^8 )</td>
<td>( 1 + \alpha^2 )</td>
<td>(1 0 1 0)</td>
</tr>
<tr>
<td>( \alpha^9 )</td>
<td>( \alpha + \alpha^3 )</td>
<td>(0 1 0 1)</td>
</tr>
<tr>
<td>( \alpha^{10} )</td>
<td>( 1 + \alpha + \alpha^2 )</td>
<td>(1 1 1 0)</td>
</tr>
<tr>
<td>( \alpha^{11} )</td>
<td>( \alpha + \alpha^2 + \alpha^3 )</td>
<td>(0 1 1 1)</td>
</tr>
<tr>
<td>( \alpha^{12} )</td>
<td>( 1 + \alpha + \alpha^2 + \alpha^3 )</td>
<td>(1 1 1 1)</td>
</tr>
<tr>
<td>( \alpha^{13} )</td>
<td>( 1 + \alpha^2 + \alpha^3 )</td>
<td>(1 0 1 1)</td>
</tr>
<tr>
<td>( \alpha^{14} )</td>
<td>( 1 + \alpha^3 )</td>
<td>(1 0 0 1)</td>
</tr>
</tbody>
</table>

from GF(2) is in the following form:

\[
f(X) = f_0 + f_1 X + f_2 X^2 + \ldots + f_n X^n,
\]

(A.2)

where \( f_i = 0 \) or 1 for \( 0 \leq i \leq n \). The degree of a polynomial is the largest power of \( X \) with a nonzero coefficient. For the preceding polynomial, if \( f_n = 1 \), \( f(X) \) is a polynomial of degree \( n \). A polynomial over GF(2) means a polynomial with coefficients from GF(2).

Polynomials over GF(2) are added (or subtracted), multiplied, and divided in the usual way. Let

\[
g(X) = g_0 + g_1 X + g_2 X^2 + \ldots + g_m X^m,
\]

(A.3)

be a second polynomial over GF(2). To add \( f(X) \) and \( g(X) \), we simply add the coefficients of the same power of \( X \) in \( f(X) \) and \( g(X) \) where \( f_i + g_i \) is carried out in modulo-2 addition. For example, adding \( a(X) = 1 + X + X^3 + X^5 \) and \( b(X) = 1 + X^2 + X^3 + X^4 + X^7 \),
we obtain the following sum:

\[
a(X) + b(X) = (1 + 1) + X + X^2 + (1 + 1)X^3 + X^4 + X^5 + X^7
= X + X^2 + X^4 + X^5 + X^7
\]  

(A.4)

Multiplication and division are also similar and will not be discussed here.

### A.2 Linear Block Codes

#### A.2.1 Generation of The Code

In this section, we restrict our attention to a subclass of the block codes: the linear block codes. The linear block codes are defined and described in terms of their generator and parity-check matrices. In block coding each message block, denoted by \( u \), consists of \( k \)-bits and it has fixed length. There are total of \( 2^k \) distinct messages. The encoder, according to certain rules, transforms each input message \( u \) into a binary \( n \)-tuple \( v \) with \( n > k \). This set of \( 2^k \) codewords is called a block code. A binary block code is linear if and only if the modulo-2 sum of two codewords is also a codeword. So \( k \) independent codewords can be found such that every codeword \( v \) is a linear combination of these \( k \) codewords. We arrange these independent codewords as the rows of a \( k \times n \) matrix as follows:

\[
G = \begin{bmatrix}
g_{00} \\
g_{10} \\
\vdots \\
g_{k-1,0}
\end{bmatrix}
= \begin{bmatrix}
g_{00} & g_{01} & g_{02} & \cdots & g_{0,n-1} \\
g_{10} & g_{11} & g_{12} & \cdots & g_{1,n-1} \\
\vdots & \vdots & \vdots & \cdots & \vdots \\
g_{k-1,0} & g_{k-1,1} & g_{k-1,2} & \cdots & g_{k-1,n-1}
\end{bmatrix}
\]  

(A.5)

If \( u=(u_0, u_1, \ldots, u_{k-1}) \) is the message to be encoded, the corresponding codeword can be given as follows:

\[
v = u.G
\]  

(A.6)
The rows of \( G \) generate the \((n,k)\) linear code. For this reason, the matrix \( G \) is called a generator matrix for the code. A desirable property of a linear block code to posses is the systematic structure. This is shown in Figure A-1 in which a code is divided into two parts, the message part and the redundant checking part. Message part has \( k \) bit unaltered message, whereas the redundant checking part has \( n-k \) parity-check digits, which are linear sums of the message bits. A linear code with this structure is referred as a linear systematic code. A linear systematic \((n,k)\) code is completely specified by a \( k \times n \) matrix \( G \).

There is a second important matrix associated with every linear block code. For any \( k \times n \) matrix \( G \) with \( k \) linearly independent rows, there exists an \((n-k) \times n\) matrix \( H \) with \( n-k \) linearly independent rows such that any vector in the row space of \( G \) is orthogonal to the rows of \( H \). An \( n \)-tuple \( v \) is a codeword in the code \( C \) generated by \( G \) if \( v.H^T = 0 \).

\[
H = \begin{bmatrix}
h_{00} \\
h_{10} \\
\vdots \\
h_{n-k-1,0}
\end{bmatrix}
= \begin{bmatrix}
h_{00} & h_{01} & h_{02} & \ldots & h_{0,n-1} \\
h_{10} & h_{11} & h_{12} & \ldots & h_{1,n-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
h_{n-k-1,0} & h_{n-k-1,1} & h_{n-k-1,2} & \ldots & h_{n-k-1,n-1}
\end{bmatrix}
\]
A.2.2 Syndrome and Error Detection

Consider an (n,k) linear code with a generator matrix $G$ and parity check matrix $H$. Let $v = (v_0, v_1, ..., v_{(n-1)})$ be a codeword that is stored in the memory. Due to a soft error, let the received vector, $r = (r_0, r_1, ..., r_{(n-1)})$ is different than the codeword, $v$.

\[
e = r + v \tag{A.9}
\]

\[
e = (e_0, e_1, ..., e_{n-1}) \tag{A.10}
\]

is an n-tuple. This n-tuple is called the error vector or pattern, which displays the positions where the received vector $r$ differs from $v$.

On receiving $r$, the decoder must first determine if $r$ contains any errors. If the presence of errors is detected, the decoder will take actions to locate the errors and correct them. When $r$ is received the decoder computes the following (n-k) tuple:

\[
s = rH^T \tag{A.11}
\]

\[
s = (s_0, s_1, ..., s_{n-k-1}) \tag{A.12}
\]

which is called the syndrome of $r$. $s=0$ if $r$ is a codeword, therefore if $s\neq0$, an error is detected. If $s=0$, $r$ is accepted as the codeword. It is possible that certain error vectors are not detectable (if there are more errors than detection ability of the code, etc). These are called undetectable errors. If an error is detected, decoding procedure is used to determine the error pattern.
Bibliography


62


