Reliable Computing at the Nanoscale

by Eric Rachlin
ScB, Brown University, 2003
ScM, Brown University, 2006

A Dissertation submitted in partial fulfillment of the
requirements for the Degree of Doctor of Philosophy
in the Department of Computer Science at Brown University

Providence, Rhode Island
May 2010

This dissertation by Eric Rachlin is accepted in its present form
by the Department of Computer Science as satisfying the
dissertation requirement for the degree of Doctor of Philosophy.

Date _____ _____

John E. Savage, Director

Recommended to the Graduate Council

Date _____ _____

John E. Savage, Reader

Date _____ _____

André DeHon, Reader

Date _____ _____

Franco P. Preparata, Reader

Approved by the Graduate Council

Date _____ _____

Sheila Bonde, Dean of the Graduate School

Vitæ

After graduating from Stuyvesant High School in New York City, Eric Rachlin began his multi-decade stint at Brown University. In 2003, Eric graduated magna cum laude with an Sc.B. in Applied Math and Computer Science. In the summer of 2003, he began working as a research assistant to Computer Science Professor John Savage. Collaborating closely with Professor Savage, he investigated the resource requirements of stochastically assembled nanowire-based memories. He also helped draft a successful NSF NIRT proposal that would later fund the bulk of his PhD research.

In the fall of 2004, Eric reenrolled at Brown to pursue his Ph.D. in nanoscale computing. As a graduate student, Eric's primary research interests included probabilistic analysis, stochastic nanoscale assembly, information theory and reliable computation. Eric's research has focused on demonstrating how emerging nanoscale architectures can be modeled probabilistically and made robust against random variations. He has analyzed a range of proposed nanoscale devices, and has also explored how error-correcting codes can be employed to perform reliable nanoscale computations.

In addition to his work on nanoscale computing, Eric has collaborated with researchers in computer vision, P2P networks and cryptography. He has also helped teach several courses on computational complexity. Outside of the world of computer science research, Eric is involved with the design and maintenance of several high-traffic blogs, including passiveaggressivenotes.com, winner of a 2008 SXSWi WebAward. He is also an extremely proficient juggler and avid drummer.

Dedicated to my grandfather
for investing in my Brown education

Acknowledgements

First and foremost I would like to thank John Savage. During my time at Brown, John has not only been an advisor and collaborator, but also a friend and mentor. His continual advice and support with regard to research, my career, and life in general, will serve me for years to come. I must also thank our primary collaborators, André DeHon, Ben Gojman and Charles Lieber. André, in particular, brought crucial expertise to John and my research. This thesis would not have been possible without his contribution on several publications, as well as on the grants that funded my graduate career. His service on my thesis committee is also greatly appreciated. Similarly, I thank Franco Preparata for serving on both my research comps and thesis committees, and for taking an ongoing interest in my work.

In addition to my other collaborators, Mira Belenkiy, Melissa Chase, Chris Erway, John Jannotti, Alp Kupcu, Anna Lysyanskaya and Yue Wu, I would also like to thank the many friends who made my time at Brown more than just a research endeavor. In particular, Chris Erway and Tibet Sprague continually got me out of the house and reminded me of the importance of life beyond the CS department (while simultaneously showing me a thing or two about how to write code). Harry Siple, David Segal and Daniel Bass not only made living in Providence significantly more enjoyable, but conveniently provided me with a place to stay during my many trips back to town while living in New York City.

Finally, I would like to thank my family, especially my parents for their perpetual support and encouragement, my grandmothers for their remarkable interest in any and all Eric-related activities (despite having never used a computer), and my grandfather for his substantial contribution to my early years at Brown. Most importantly, I would like to thank the lovely and talented Jessica Purmort for her continual reminder that, more than anything, I needed to hurry up and finish this thing already!

Contents

Dedication **v**

1 Introduction **1**
1.1 Reliable Nanoscale Computation . . . 2
1.1.1 A Brief History of Nanocomputing . . . 2
1.1.2 Device Reliability and General Purpose Computing . . . 7
1.2 Fundamental Characteristics of Nanoscale Computing . . . 9
1.2.1 Stochastic Assembly . . . 9
1.2.2 Post-Assembly Testing and Configuration . . . 10
1.2.3 Strict Assembly Constraints . . . 10
1.2.4 Imperfect Operation . . . 10
1.3 An Overview of this Thesis . . . 11

2 Overview of Nanoscale Computing **12**
2.1 Technology Overview . . . 12
2.1.1 Nanoscale Semiconductor-Based Architectures . . . 12
2.1.2 DNA-Based Assemblage . . . 13
2.1.3 Quantum-Dot Cellular Automata . . . 14
2.1.4 Biological Computing . . . 15
2.2 Nanowire Crossbars . . . 15
2.2.1 Crossbar Assembly . . . 16
2.2.2 Crossbar-based Memories . . . 17
2.2.3 Crossbar-based Logic . . . 19

3 Nanowire Decoders **22**
3.1 Decoder Requirements . . . 22
3.1.1 Nanowire Addressing . . . 22
3.1.2 Address Requirements . . . 24
3.1.3 Simple Versus Compound Decoders . . . 24
3.2 Decoding Technologies . . . 25
3.2.1 Encoded Nanowire Decoders . . . 25
3.2.2 Mask-Based Decoders . . . 27
3.2.3 The Randomized-Contact Decoder . . . 28

3.2.4 Additional Decoding Technologies . . . 29
3.3 Post-Assembly Configuration . . . 31
3.3.1 Address Discovery . . . 31
3.3.2 Address Translation Circuitry . . . 32
3.4 Modeling Nanowire Decoders . . . 33
3.4.1 The Binary Model of Nanowire Control with Errors . . . 33
3.4.2 Real-valued Physical Models . . . 36
3.5 Decoder Analysis Framework . . . 38
3.5.1 Memory Area Estimate . . . 39
3.5.2 Memory Addressing Strategies . . . 39
3.5.3 Expectation versus with High Probability . . . 41

4 The Randomized-Contact Decoder 42
4.1 Bounds Using Inclusion-Exclusion . . . 42
4.1.1 A Single Contact Group . . . 43
4.1.2 Multiple Contact Groups . . . 46
4.2 Bounds Using Expectation . . . 47
4.2.1 A Single Contact Group . . . 47
4.2.2 Multiple Contact Groups . . . 48
4.2.3 Additional Addressing Strategies Using Expectation . . . 50
4.3 Comparison of Addressing Strategies . . . 50
4.4 Summary of Results . . . 51

5 Encoded Nanowire Decoders 53
5.1 NW Encodings . . . 54
5.1.1 Code Requirements . . . 55
5.1.2 (h, M)-Hot Codes . . . 56
5.1.3 Binary Reflected Code . . . 57
5.1.4 Generating Random Ensembles of Axial Codewords . . . 57
5.2 Analysis of Encoded NW Decoders . . . 58
5.2.1 Bounds Using Expectation . . . 58
5.2.2 Bounds Using Inclusion-Exclusion . . . 62
5.2.3 Area Estimates . . . 64
5.3 Misalignment of Axial Codes . . . 66
5.4 Radially Encoded Nanowire Decoders . . . 67
5.4.1 The Linear Radially Encoded NW Decoder . . . 67
5.4.2 The Linear-Logarithmic Radially Encoded NW Decoder . . . 69
5.4.3 The Fully Logarithmic Radially Encoded NW Decoder . . . 70
5.4.4 Fault-tolerant Etching Error Correction . . . 72
5.4.5 Hybrid Nanowire Codes and Decoders . . . 73
5.5 Summary of Results . . . 74

6 Masked-Based Decoders **75**
6.1 Modeling Decoder Manufacture . . . 77
6.1.1 LR Manufacture . . . 77
6.1.2 Modeling Variation in Mask Placement . . . 77
6.1.3 Modeling Variation in LR Boundary Placement . . . 79
6.1.4 InterNW Regions . . . 79
6.1.5 Additional Sources of LR Boundary Variation . . . 79
6.2 Analyzing the n-Cycle Mask-Based Decoder . . . 80
6.2.1 Models for Decoder Analysis . . . 81
6.3 Coupon Collection . . . 83
6.3.1 The Coupon Collector Problem with Failures . . . 83
6.3.2 The Targeted Coupon Collector Problem . . . 83
6.3.3 The Multi-Stage Targeted Coupon Collector Problem . . . 85
6.4 Performance of the n-Cycle Mask-Based Decoder . . . 88
6.4.1 The Coarse-Grained Model . . . 88
6.4.2 The Fine-Grained Model . . . 89
6.5 Additional Considerations . . . 91
6.5.1 Address Translation Circuitry . . . 91
6.5.2 Alternative Addressing Strategies . . . 91
6.6 Summary of Decoder Analysis . . . 93

7 Nanowire Addressing for Crossbar-based Logic **95**
7.1 NW Decoders for Logic . . . 95
7.2 Stochastic Assembly of NW Logic Decoders . . . 96
7.2.1 Unique Couplings . . . 97
7.2.2 Area Bounds . . . 98
7.3 Lower Bounding β . . . 101
7.3.1 A Lower Bound for RCDs . . . 102
7.3.2 A Lower Bound for Encoded NW Decoders . . . 104
7.4 Stochastic Crossbar Interconnect . . . 104

8 Nanowire Address Discovery **106**
8.1 Address Discovery via Read/Write Operations . . . 107
8.1.1 Coping with Errors . . . 108
8.2 Exhaustive Search . . . 109
8.2.1 Parallel Exhaustive Search . . . 109
8.2.2 Coping With Codeword Errors . . . 110
8.3 Encoded NW decoders . . . 111
8.3.1 Binary Search . . . 111
8.3.2 Searching Across Contact Groups . . . 111
8.3.3 A Lower Bound . . . 112
8.3.4 Coping With Misalignment Errors . . . 112

8.4 Arbitrary Codes . . . 113
8.4.1 Asymptotic analysis . . . 114
8.4.2 Experimental Results . . . 116

9 Coded Computation **118**
9.1 Approaches to Reliable Computation . . . 119
9.1.1 Modular Redundancy . . . 119
9.1.2 Two-Tiered Reliability and Coded Computation . . . 120
9.1.3 Previous Work on Coded Computation . . . 121
9.2 A Model of Computation . . . 122
9.2.1 Formalizing the Model . . . 123
9.2.2 Examples . . . 124
9.3 The Coded Computation Framework . . . 126
9.3.1 One Step of Coded Computation . . . 127
9.3.2 Transcoding the Output . . . 129
9.3.3 Conditions on Permutations . . . 129
9.3.4 Spielman's Model . . . 130
9.4 Interpolation Polynomials . . . 130
9.4.1 Examples of interpolation polynomials . . . 131
9.4.2 Applying Polynomials to Linear Codes . . . 132
9.5 Transcoding . . . 134
9.5.1 Transcoding Using 2D Codes . . . 134
9.5.2 Transcoding Using Checksums . . . 135
9.5.3 Transcoding in Parallel Architectures . . . 136
9.6 Codeword Permutations . . . 137
9.6.1 Direct Application of Extension Permutations . . . 137
9.6.2 Composing Permutations to Realize Extensions of Π . . . 138
9.6.3 Data-Movement Overhead . . . 140
9.7 Families of Codes . . . 140
9.7.1 Reed-Solomon Codes . . . 140
9.7.2 Reed-Muller Codes . . . 141
9.7.3 Other Polynomial Codes . . . 142
9.7.4 Multidimensional Codes . . . 142
9.8 Overhead . . . 143
9.8.1 Reliability via Repetition . . . 143
9.8.2 Basic Analysis Framework . . . 145
9.8.3 Coded Computation Using 2D Codes . . . 148
9.8.4 Coded Computation Using 1D Codes . . . 150
9.8.5 Summary of Results . . . 153

10 Exploring the Power of Coded Computation **155**
10.1 Lower Bounds 155
10.2 Efficiently Encoding Most-Boolean Function 158
10.2.1 Pippenger's Construction 159
10.3 Coded Prefix Computations 160
10.3.1 Encoding a Parallel Prefix Computation 161

11 Conclusion **163**

List of Tables

10.1 A 3-step parallel prefix computation on a hypercube 161

List of Figures

1.1 Richard Feynman . . . 2
1.2 Early Integrated Circuits . . . 3
1.3 The etching of a chip via photolithography . . . 4
1.4 Atomic scale microscopy from an STM and AFM . . . 5
1.5 Wireframe models of fullerene and a carbon nanotube . . . 5
1.6 A theoretical molecular machine . . . 6

2.1 A range of semiconductor-based nanoscale computing technologies . . . 13
2.2 Stochastically assembled DNA structures . . . 14
2.3 Circuits based on quantum-dot cellular automata . . . 15
2.4 A programmable nanowire crossbar . . . 16
2.5 A nanowire crossbar-based memory performing read and write operations . . . 18
2.6 A level of reconfigurable crossbar-based logic . . . 20

3.1 The two extreme cases when reading data from a nanowire crosspoint . . . 23
3.2 A compound nanowire decoder . . . 25
3.3 An axially encoded nanowire decoder . . . 26
3.4 A radially encoded nanowire decoder . . . 27
3.5 A masked-based nanowire decoder . . . 28
3.6 A randomized-contact nanowire decoder . . . 29
3.7 Rotational offset decoders . . . 30
3.8 MNAB . . . 31
3.9 Address translation circuity used to interface a nanowire decoder . . . 32

4.1 Codewords in a randomized contact nanowire decoder . . . 43

5.1 Axially encoded nanowire codewords . . . 54
5.2 Radially encoded nanowire codewords . . . 55
5.3 Schema for calculating the probability of axial misalignment . . . 66
5.4 A linear radially encoded nanowire decoder . . . 68
5.5 A linear radially encoded nanowire decoder . . . 71
5.6 A logarithmic radially encoded nanowire decoder . . . 72

6.1 A masked-based nanowire decoder . . . 76
6.2 A cycle of high-K dielectric lithographic regions . . . 78

8.1 Codeword Discovery via Randomized Testing 117

9.1 A T-step regular computing network 122
9.2 A T-step coded computation 128

Chapter 1

Introduction

Emerging nanoscale computing technologies necessitate fundamental changes in the way computer architectures are designed and analyzed. The significant uncertainty associated with the assembly and operation of nanoscale devices must not only be modeled and accounted for, but actively embraced as part of the design process. In stark contrast with today's VLSI, probabilistic modeling and analysis are primary requirements for the successful realization of nanoscale computer architectures.

In this thesis we examine how faults, defects and unavoidable nanoscale variation can be tolerated in emerging nanoscale computer architectures. The first portion of this thesis focuses on the nanowire crossbar, a particularly promising building block for near-term nanoscale hardware. In the context of the crossbar, we describe how a range of stochastic nanoscale fabrication technologies can be used to reliably address (i.e. control) nanoscale wires. We provide tight analytic bounds on the resources required to cope with stochastic nanoscale assembly. We also demonstrate a range of promising design strategies.

A technical overview of the nanowire crossbar and related nanowire addressing technologies is provided in Chapters 2 and 3. The chapters provide the background and models utilized in the analysis of Chapters 4 through 8. These chapters collectively explore a number of related approaches for integrating stochastically assembled crossbar-based nanoscale hardware with traditional lithographically produced mesoscale hardware. More importantly, they explicitly demonstrate the type of probabilistic modeling and analysis that is vital to the success of nanoscale computing.

The second portion of this thesis investigates the more general problem of performing reliable computations using unreliable devices. Designing architectures that function properly in the presence of transient faults (for example, building circuits out of gates that occasionally produce incorrect outputs) has been a longstanding challenge in the fields of computer science and computer engineering. In the context of nanoscale technologies the problem is particularly pressing, as it is anticipated that nanoscale devices will be significantly less reliable than their current mesoscale counterparts.

To attack this problem, Chapter 9 demonstrates the promise of "two-tiered" reliability, meaning the structuring of computations such that some operations are much more reliable than others. Two-tiered reliability reflects the ability of highly reliable mesoscale devices to supervise less reliable nanoscale devices. As such, it represents an important design paradigm that is likely to become

Figure 1.1: The potential for nanoscale computing was first described by Richard Feynman in his 1959 speech to the American Physical Society. In recognition of his early vision of where computers were headed, the Foresight Nanotech Institute awards an annual Feynman Prize for significant advancements in nanotechnology.

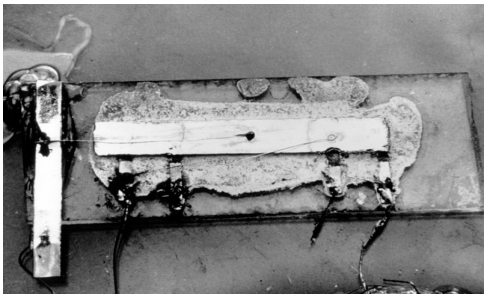
increasingly prominent as computer architectures continue to shrink. What's more, the utility of circuits and algorithms that employ tiered reliability is not limited to nanoscale architectures. For example, up-and-coming multicore chips may contain cores that operate at varying levels of reliability (since highly reliable cores likely require more area, power or time per operation). Such cores could be used to implement algorithms that have been designed with tiered reliability in mind.

1.1 Reliable Nanoscale Computation

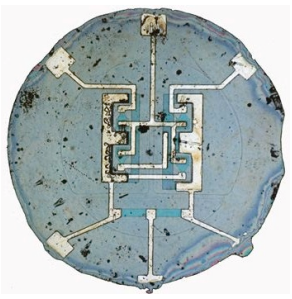
This section presents the broader context for the research on reliable nanoscale computation contained in this thesis. It begins with a brief history of nanoscale computing. This is followed by a discussion of the historic importance of device reliability in digital hardware. Both sections highlight the significant challenges currently facing the design and implementation of reliable nanoscale architectures.

1.1.1 A Brief History of Nanocomputing

One nanometer, or 10^{-9} meters, is the length of a single sugar molecule, and a cubic nanometer provides only enough room for a few hundred carbon atoms. It may never be possible to create novel arrangements of subatomic particles, and as such, a nanometer represents the approximate lower limit on the size of technology. While nanometer-scale technology, or "nanotechnology", has a wealth of applications, nanoscale computing is among the most prominent. What's more, the successful realization of reliable nanoscale computation would serve as a crucial step towards the realization of other nanotechnologies that require the integration of a large number of functional nanoscale devices.



(a) The first integrated circuit



(b) An early integrated circuit produced using photolithography

Figure 1.2: In the late 1950s, two companies independently developed semiconductor-based integrated circuits. In 1958, Jack Kilby developed the first integrated circuit (a) at Texas Instruments as a summer research project. In 1959 Fairchild Semiconductor patented a process for producing planar semiconductor devices. Soon after, Robert Noyce demonstrated that this process could be adapted to produce general purpose integrated circuits within silicon chips (b). In 1968, Noyce went on to cofound Intel with his colleague Gordon Moore.

The dream of nanoscale computing was first articulated by Richard Feynman (see Figure 1.1) in his 1959 speech given to the American Physical Society. He argued that no known physical law would prevent the room-sized computers of the 1950s from being replaced with far more powerful, pin-sized computers built from nanoscale components. As many at the time realized, general purpose computers were poised to become vastly more useful once their computing power increased by several orders of magnitude. This has long since come to pass, but we are only now approaching the nanoscale devices that Feynman asserted we could one day produce.

At the time of Feynman's speech, computer hardware was undergoing a major transition. The bulky vacuum tube-based logic of the 1950s was being replaced with solid-state transistors (see Figure 1.2). Crucially, solid-state transistors can be manufactured using light-based etching, known as photolithography. This allows for the efficient production of microscopic circuits compactly embedded into the surface of a silicon chip. Modern computer chips, which are often referred to as VLSI (very large scale integration) technology, are manufactured using photolithography.

Since the 1960s, photolithography has allowed for ever-smaller, ever-faster VLSI processors comprised of semiconductor-based logic that operates at high speeds and high levels of reliability. What's more, it allows for millions of identical copies of a computer chip to be produced from a single set of masks. These masks act like blueprints for a top-down assembly process. By shining light through the masks, complex two-dimensional patterns can be repeatedly etched into successive layers of a silicon wafers (see Figure 1.3). Selective application of dopants to different portions of the pattern produces functional devices, like logic gates. Metallic wires, which connect these devices, can also be produced by depositing a conductor into etched regions.

Starting in the 1960s, it became apparent that photolithographically produced computer hardware obeyed an empirical trend. With remarkable consistency, the area required for digital logic

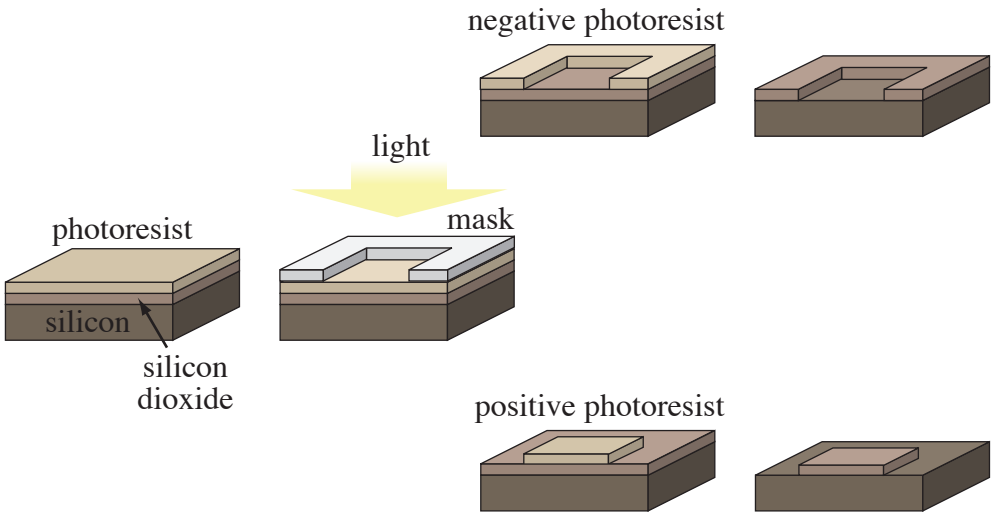
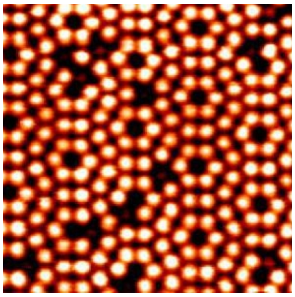


Figure 1.3: In photolithography, light is shown through a mask to define which regions of either positive or negative photoresist are removed. The remaining resist is then used to etch patterns into layers of a silicon and silicon dioxide. Functional devices can then be created by selectively depositing dopant, conductors, and other materials into the etched away regions. By repeating this entire process, multilayer chip-based architectures are built up one level at a time.

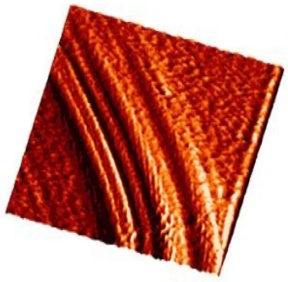
was cut in half approximately every 18 months. This in turn meant that the number of transistors on a chip, and hence the number of computations it could perform, approximately doubled every 18 months. This general trend, which is typically referred to as "Moore's Law", has become increasingly difficult and costly to maintain. The feature size of today's chips, meaning the minimum spacing between wires and gates, is 45 nanometers (nm). While computer manufactures would like nothing more than to shrink features further, doing so necessarily means confronting the challenges of nanoscale manufacturing.

Feynman's prescient, but highly speculative vision of nanoscale engineering gained significant focus in the 1980s. The advent of the scanning tunneling microscope (STM), and later the atomic force microscope (AFM) led to an increasingly precise understanding of how atoms are configured within molecules. An STM scans across the surface of a material with an atomic-scale tip. Electron tunneling between the tip and the surface produces a current. By measuring this current (or alternatively, adjusting the height of tip to maintain a constant current) STMs are able to obtain sub-nanometer resolution images of a sufficiently clean surface (see Figure 1.4a). Prior to the STM electron-based microscopy (which has existed since the 1930s) had great difficulty imaging molecular-scale structures. Soon after the STM, AFMs provided scientists with additional molecular-scale imaging capabilities. By precisely measuring the deflection an atomic-scale tip, AFMs are able to reveal the three-dimensional profile of a surface at the atomic scale (see Figure 1.4b).

An improved understanding of molecular structures facilitated, among other things, the discovery of fullerene, or "buckyballs", in 1985 and carbon nanotubes in 1992. Both are large carbon

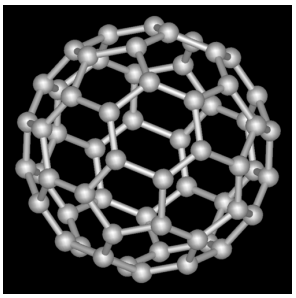


(a) STM image of Si(111)

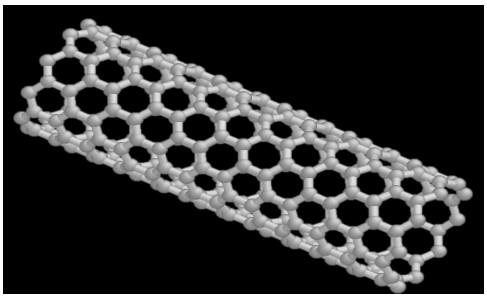


(b) AFM profile of nanotubes

Figure 1.4: Starting in the 1980s, scientists gained the ability to directly observe the structure of molecules. In a) a scanning tunneling microscope (STM) reveals the structure of a 5.4nm x 5.4nm segment of silicon crystal. In b) an atomic force microscope (AFM) provides the three-dimensional profile of single-walled carbon nanotubes within a 15.8 nm x 15.8 nm patch. Both images were obtained via Omicron Nanotechonlogy GmbH and are available on their website at `http://omicron.de/index2.html?/results/~Omicron`.



(a) Fullerene



(b) A carbon nanotube

Figure 1.5: Both fullerene molecules and carbon nanotubes are comprised of carbon atoms arranged in a highly regular geometric structure. Fullerene's, for example, are so named because their structure is identical to that of a geodesic dome, a building design first patented and popularized by Buckminster Fuller.

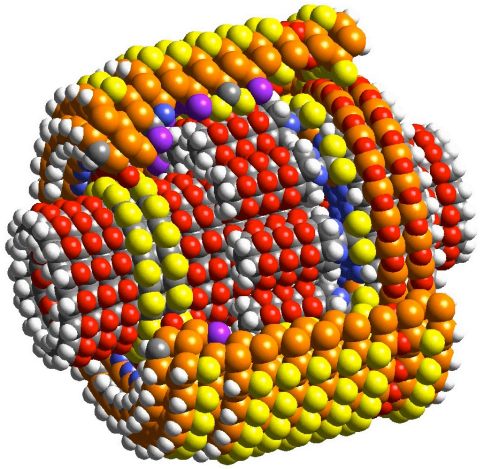


Figure 1.6: A differential gear of the type put forth in Drexler's 1992 book, *Nanosystems: Molecular Machinery, Manufacturing and Computation*. Here a molecular shaft, surrounded by "bearings", would be able to rotate independently of the molecular casing. This image was produced by Institute for Molecular Manufacturing and is available a `http://www.imm.org/research/parts/gear/` (Copyright 1997 IMM, all rights reserved).

molecules in which individual carbon atoms serve as the building blocks of highly regular geometric structures (see Figure 1.5). These structures, which closely resemble manmade geodesic domes, sparked a renewed enthusiasm for nanoscale engineering. Suddenly it appeared feasible for macroscale designs to be recreated as nanoscale structures.

The capabilities of the AFM furthered this excitement, as its molecular tip could be used not just to image surfaces, but to nudge and manipulate individual molecules. This suggested to some the possibility of building, atom-by-atom, molecular scale machines that could in turn facilitate the production of additional molecular scale components. An early and highly prominent advocate of this research agenda was Eric Drexler. In his 1986 book, *Engines of Creation: The Coming Era of Nanotechnology* [1], Drexler suggested that general purpose molecular scale assemblers could one day be constructed and used to bootstrap a wide array of modular nanoscale devices.

Nanosystems: Molecular Machinery, Manufacturing and Computation [2], Drexler's 1992 book based on his PhD Thesis, further popularized his vision of nanoscale structures that act like thermodynamically powered, substantially scaled-down versions of traditional macroscale machinery (see Figure 1.6). Drexler's writings also sparked concern that nanoscale machinery had the potential to one day run amok as unstoppable, self-replicating "grey goo". Some scientists, however, became highly critical of Drexler's view. Nanoscale engineering, they argued, could only be expected to produce structures that are compatible with highly stochastic assembly process. As a result, nanotechonlogy could not simply be envisioned as miniscule recreations of traditional technology.

Chemist and Nobel Laureate Richard Smalley, was among the most vocal of Drexler's critics. In 2001, he asserted that the so-called "fat fingers" and "sticky fingers" of even microscale manufacturing technology pose a huge challenge when trying to deterministically arrange the atomic

building blocks of hypothetical nanoscale machinery [3]. In other words, randomness is inherent to molecular assembly, and theoretical nanoscale designs that rely on arbitrary arrangements of hundreds of molecules may well be impossible to realize in practice. As such, nanotechnology must be designed and assembled in ways that are fundamentally different from how current technology is engineered. Probabilistic analysis is crucial.

Smalley's perspective is much more in line with the current state of nanotechnology research. In the past decade, billions of dollars has gone into funding the work of chemists, physicist and engineers. Although many novel nanoscale structures have been discovered and analyzed, techniques for general purpose molecular assembly remain elusive. Instead, slow and steady progress has resulted from the careful refinement of more traditional chemical processes. Even if molecular assemblers are an eventual possibility, they are unlikely to play a role in the manufacturing of near-term nanoscale computer architectures.

Since the year 2000, a number of individual nanoscale computing devices (e.g. wires, logic gates and memory cells) have been demonstrated. Producing architectures from these devices, however, is not simply a matter of substituting tiny wires and gates into today's architectures. Designers of nanoscale architectures must find ways to interconnect millions, or billions of devices, even while our ability to place individual devices remains poor. Furthermore, as we continue to push the limits of what can be reliably manufactured, we must find new ways to mitigate device variation. If nanoscale architectures are to be realized any time soon, they will have to function correctly even when individual devices fail. Although no nanoscale architectures have yet been produced, most researchers believe that they will incorporate stochastic assembly, reconfigurability, fault-tolerance and strict design constraints [4, 5].

Below, Section 1.2 elaborates on these four requirements. As explained in the first half of Chapter 2, they are relevant to a range of proposed nanoscale computing technologies. Nonetheless, much of the analysis in this thesis focuses on today's most viable nanotechnology, the nanowire crossbar. Nanowire crossbars provide a relatively concrete model of what nanoscale computing may look like (see Section 2.2), while at the same time highlighting many of the more general issues facing alternative nanoscale computing technologies. To date, the crossbar is the only nanoscale architectural component to have been demonstrated [6, 7, 8]. Furthermore, a number of crossbar-based designs have been proposed, demonstrating how crossbars can serve as a basis for both memories and circuits [9, 10, 11, 12, 13] (see Section 2.2).

1.1.2 Device Reliability and General Purpose Computing

Many of the fundamental ideas behind general purpose digital computing can be traced back not just to Alan Turing in 1930s, but to Charles Babbage in the 19th century. Despite his insights, however, Babbage was unable to construct his "Analytical Engine" using the fabrication technology of his time. Without reliable components, general purpose computing machines proved prohibitively difficult to implement. Instead, the successful realization of digital computers would have to wait until 1940s, when electronic components with high on/off ratios permitted the reliable implementation of digital logic.

Since then, the overwhelming success of general purpose computing has been continually fueled

by a stream of steadily shrinking, highly reliable hardware. Today, increasingly complex architectures are designed using ever-smaller, ever-faster processors comprised of logic gates that operate at astronomically high levels of reliability. Assume, for example, a hypothetical representative architecture consisting of 1,024 3GHz processors with 10^9 gates/processor. During a full year of operation, this collection of processors might perform as many as 10^{29} gate operations.

Now suppose that any given output of any given gate has probability p_f of being incorrect. If we assume that neither the processors, nor the programs run on them, are designed to accommodate errors, a 99% chance of fault free operations potentially requires that $p_f < 10^{-31}$! Even if we acknowledge that, at any given time, some fraction of the gates do not effect the output of the computation, our required value of p_f remains staggeringly minute.

As the size of individual gates shrinks and the number of gates per chip increases, it becomes increasingly burdensome to maintain such an astronomically high level of gate reliability. Up-and-coming nanoscale devices and multicore architectures both point to an impending need for fault-tolerant circuits and software. The ability to tolerate gate level faults would not only pave the way for larger architectures with smaller features sizes, it could allow current CMOS-based chips to operate at higher speeds or lower power. ("CMOS", or complementary metaloxidesemiconductor, refers to the design and manufacturing techniques used to implement the digital logic on today's chips.)

Early digital computers were built using vacuum tubes, which were unreliable. This presented the same type of scaling challenge we face today. As computers became increasingly complex (i.e. used more logic gates), the probability that some component would fail during a given computation approached 1. This motivated von Neumann, in his well-known 1956 paper [14], to propose a systematic approach to building logic from unreliable gates. He described how an arbitrary circuit, C, built from perfectly reliable gates, could be converted to a fault-tolerant circuit C', constructed from potentially faulty gates. This construction involved repeating each gate r times, and then suppressing errors with constant-sized, but potentially faulty, majority gates (see Chapter 9).

Although von Neumann's work was a theoretical success (his conclusions were later made more rigorous by Pippenger [15]) the overhead of repeating each individual computing elements is high. In subsequent decades, solid-state technologies allowed gates and wires to not only shrink, but become orders of magnitude more reliable. Indeed, modern transistors produced using photolithography are expected to operate many trillions of times before failing. As such, von Neumann style repetition is unnecessary.

Only now, as digital circuits continue to shrink, is the reliability of logic gates once again becoming a major cause for concern. The near-term viability of nanoscale architectures is closely tied to whether or not transient faults can be tolerated more efficiently than through simple repetition. After all, if many redundant copies of each nanoscale gate are called for, these copies could simply be replaced with a single lithographically produced CMOS gate.

Fortunately there is reason for optimism. Biological systems, for example the brain, demonstrate a significant level of fault-tolerance but do not appear to embody von Neumann style repetition. More concretely, von Nuemann's approach to reliable computation contrasts sharply with results from digital communication theory. Since the time of Claude Shannon, it has been known that repetition is a highly inefficient error control mechanism when encoding data. To achieve fault-

tolerant communication, a reliable encoder and decoder are used to send information across a noisy channel. When data is encoded in an error-correcting code, information about each input is effectively "spread" across check symbols. By allowing each transmitted check symbol to be a function of many information symbols, only a constant factor overhead is required to protect against random bit flips. Error-correcting codes have been embraced not only to transmit data, but to store data. It is only natural to ask whether similar ideas can be applied to reliable computation. This question is explored in Chapter 9, which presents a framework for computing using error-correcting codes.

1.2 Fundamental Characteristics of Nanoscale Computing

This section highlights the major challenges novel modeling and analysis must address in order for nanoscale computing to succeed. As was already noted in the opening paragraph of this thesis, emerging nanoscale computing technologies necessitate fundamental changes in the way computer architectures are designed and analyzed. Significant uncertainty is associated with the assembly and operation of nanoscale devices. This uncertainty must not only be modeled and accounted for, but actively embraced as part of the design process. This is in stark contrast with today's VLSI, where complex, meticulously optimized designs are realized through a deterministic, top-down etching process. For emerging nanoscale architectures, probabilistic modeling and analysis are primary requirements for the successful realization of nanoscale computer architectures.

Listed below are four fundamental ways in which emerging nanoscale computing technology differs from today's VLSI. In each case, we note the consequences with regard to modeling, analysis and design.

1.2.1 Stochastic Assembly

Assembly of VSLI technology has traditionally been viewed as deterministic. A circuit is designed, it is realized via a series of masks, and, in principle, each copy of the circuit produced by those mask is identical. As feature sizes have shrunk, however, more and more device-to-device variation has begun to appear. Consequently, circuit designers now see a need to take this variation into account in order to avoid defects.

Emerging nanoscale computing technologies are expected to produce large-scale architectures through a bottom-up assembly process. Such a process, in which nanoscale devices are deposited onto a chip in a directed fashion, potentially increases device-to-device variation by many orders of magnitude more. Designers will need to expect variation not only in the placement and functionality of devices, but also in how they interconnect.

In order to reliably manufacture nanoscale architectures, randomness can no longer be viewed as merely the potential for defects. Instead, it must be accepted as an expected occurrence. It must be anticipated, modeled, and embraced by design. Consequently, probabilistic analysis must be brought to the forefront of the design process.

1.2.2 Post-Assembly Testing and Configuration

A stochastic nanoscale assembly process will yield large amounts of device-to-device variation, and a significant number of permanent defects. To cope with this variation, nanoscale architectures will need to be configured post-assembly. This introduces a new architectural requirement, namely, chips must be configurable so as to provide consistent functionality despite faults and variations in layout.

Today's VLSI already requires testing to verify that chips are working properly. Nanoscale architectures, however, will require testing simply to allow chips to work properly. Architectures will need to be designed such that information about their internal interconnect and the locations of internal faults can be discovered efficiently. Once determined, relevant information will need to be supplied to the many configurable components on each chip. Ideally, this entire process will remain simple enough so that chips can test and configure themselves.

1.2.3 Strict Assembly Constraints

VLSI is manufactured using photolithography, allowing complex circuit designs to be repeatedly reproduced by shining light through a mask. Although these masks are very expensive to produce, they act as reusable templates in VLSI's top-down assembly. In contrast, emergent nanoscale architectures are expected to be assembled from the bottom-up, which implies that no reusable blueprint will be available. Instead millions of nanoscale devices will be grown en masse, then deposited onto a chip in a directed fashion.

At least in the near term, it is not anticipated that the deposited devices will be able to be arranged into arbitrary patterns. Instead they will be deposited to form relatively simple structures that are locally regular. For example, photolithography can still be used to define different regions on a chip, but within each region, nanoscale devices may be organized into nanowire crossbars. The highly regular crossbars would then be configured, or programmed, after assembly, not unlike today's programmable logic arrays. In this way, post-assembly configuration provides otherwise homogenous hardware with the additional nanoscale structure required to perform useful computations.

1.2.4 Imperfect Operation

The success of VLSI has relied heavily on the extremely high reliability of transistors, and hence logic gates. As feature sizes shrink, maintaining this level of reliability has become quite challenging. The problem is only expected to get worse as features shrink to the nanoscale. First, nanoscale devices may be much more likely to break over time. Second, even when operating correctly, they will likely be susceptible to transient errors.

A logic gate that outputs an incorrect value one time in a billion would be completely unacceptable in the context of today's circuit designs. This places an enormous burden on developing nanoscale technology. While it would be ideal if these technologies operated with perfect reliability, near perfect reliability may be much easier to achieve. If we can design circuits that are capable of tolerating gate errors, and algorithms that are capable of tolerating circuit errors, nanoscale

computing will be significantly easier to implement.

1.3 An Overview of this Thesis

Chapter 2 describes four broad categories of emergent nanoscale computing technology, followed by a more detailed description of the nanowire crossbar. Chapter 3 provides a detailed look at how nanowire crossbars can be interfaced with existing technology using a device called a "nanowire decoder". In Chapter 3 a range of potential decoder technologies is described and modeled. Using this model, Chapters 4, 5 and 6 analyze the resources required to construct three different types of nanowire decoders; randomized-contact decoders, encoded nanowire decoders, and masked-based decoders. A summarizing comparison of all three decoder types is given at the end of Chapter 6. Chapter 7 considers how decoder requirements change when they are used to control crossbar-based logic as opposed to a memories. Chapter 8 looks at the post-assembly testing that stochastically assembled nanowire decoders require. Chapter 9 investigates how nanoscale architectures can cope with transient faults using error-correcting codes. Chapter 10 provides additional examples of the promise and challenges of code-based fault-tolerance. Finally, a concluding summary of the entire thesis is provided in Chapter 11

Chapter 2

Overview of Nanoscale Computing

The first half of this chapter, Section 2.1, presents four broad categories of up-and-coming nanoscale computing technology; nanoscale semiconductor-based architectures (Subsection 2.1.1), DNA-based assemblage (Subsection 2.1.2), Quantum Dot Cellular Automata (Subsection 2.1.3), and Biological Computing (Subsection 2.1.4). In each case, we highlight how the technology embodies the fundamental characteristics of nanoscale computing outlined at the end of the previous chapter.

The second half of this chapter, Section 2.2, provides a more detailed description of semiconductor-base nanowire crossbars. Nanowire crossbars are seen as the current frontrunner for near-term nanoscale architectures and are the focus of much of the research presented in subsequent chapters. Also, since a range of crossbar-related technology has already been demonstrated, they provide the opportunity for practical and realistic models that simultaneously illustrate the fundamental challenges faced by nanoscale computing.

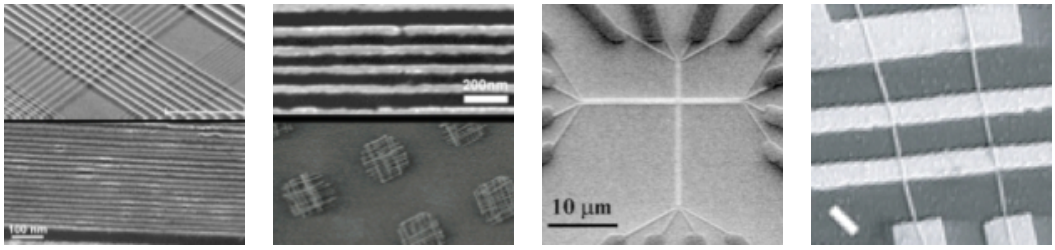
2.1 Technology Overview

Broadly speaking, emergent nanoscale computing technology can be placed into four categories: nanoscale semiconductor-based architectures, DNA-based assemblage, quantum-dot cellular automata (QCAs), and biological computing. Each of these categories is described below.

2.1.1 Nanoscale Semiconductor-Based Architectures

Today's architectures are based on CMOS technology, which relies on doped silicon (a semiconductor) to implement transistor-based logic. A seemingly natural path toward nanoscale computing would be to continue the downward scaling of today's CMOS devices. Unfortunately, the physical constraints of photolithography makes this extremely challenging, if not impossible. As a result, a number of alternative methods for manufacturing semiconductor-based architectures are being pursued.

Nanoscale silicon and germanium wires have been produced, and methods for doping these nanowires (NWs) have been demonstrated, as have individual nanoscale devices (e.g. diodes, field-effect transistors and memory cells) [19, 4] (see Figure 2.1). As explained in Section 2.2 below, these devices can then be organized into large-scale architectures via crossbars (i.e. grids) of NWs. Such



(a) SNAP NWs [16] (b) CVD NWs [6] (c) Nanoscale crossbar-based storage [17] (d) Axially-doped CVD NWs [18]

Figure 2.1: A range of semiconductor-based nanoscale computing technologies have already been demonstrated. (a) Uniform semiconducting nanowires have been produced by Heath *et al* at Caltech using a process known as SNAP [16]. (b) Lieber *et al* at Harvard have used chemical vapor deposition (CVD) to grow nanowires off chip, then deposit them fluidically in parallel [6]. (c) Molecular storage devices have been demonstrated by Williams *et al* at Hewlett-Packard [17] capable of storing bits at nanowire crosspoints (d) CVD nanowires can be grown with heavily and lightly doped regions along their axis, allowing them to form field-effect transistors with orthogonally placed lithographically produced mesoscale wires [18].

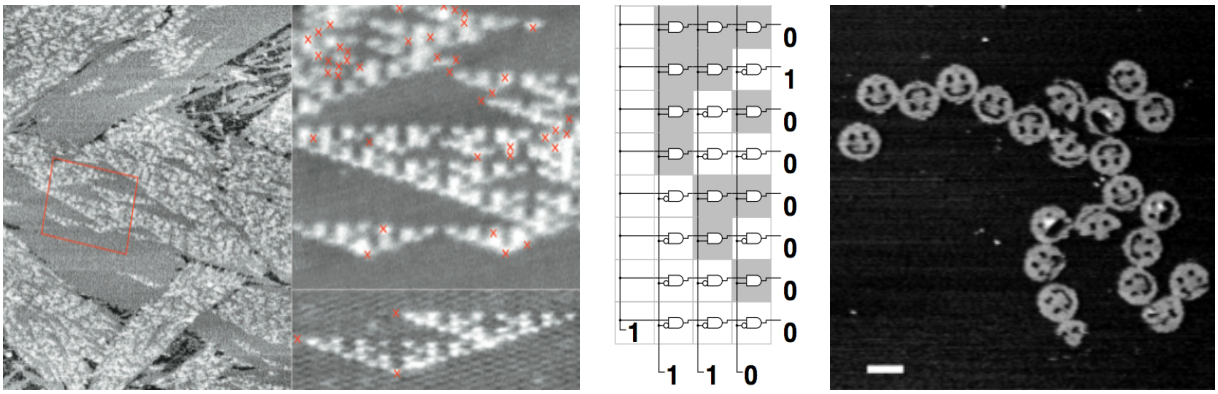
crossbars offer a promising basis for both nanoscale memories (see Section 2.2.2) and programmable logic (see Section 2.2.3).

Even with NW crossbars as a building block, it remains a significant challenge to understand how millions, or billions of nanoscale devices can be organized into general purpose nanoscale architectures. Nanoscale manufacturing constraints, along with significant amounts of device variation make the direct realization of existing chip designs infeasible. Instead new designs and accompanying analysis are required to accommodate the assembly constraints of emerging nanoscale semiconductor-based technologies. The design and analysis of semiconductor-based NW crossbars is the primary focus of this thesis.

2.1.2 DNA-Based Assemblage

In 1994 Adleman observed that carefully selected strands of DNA, assembled in vitro, could be used to carry out arbitrary computations [20]. His key insight was that the sequence of nucleotides on each strand of DNA, which determines which other strands it will bind with, can be viewed as a logical constraint. There are only certain allowed ways in which large, multi-strand sequences can form when many copies of each stand are placed in a fluid. By checking for the presence of a particular multi-strand sequence, one is effectively checking whether a certain set of constraints has been satisfied. This allowed Adleman to use DNA to solve a small instance of the Hamiltonian Path problem, albeit with questionable efficiency (DNA-based computing is relatively slow, and requires a large number of strands).

DNA-based computation does not provide a viable alternative to today's computers. What is noteworthy, however, is the ability of DNA to be viewed as a means of programmatically assembling complex structures. Rather than use single strands of DNA, Erik Winfree has demonstrated



(a) DNA Tiles [21]

(b) Tile-Based Decoder Layout [22]

(c) DNA Smiley Faces [23]

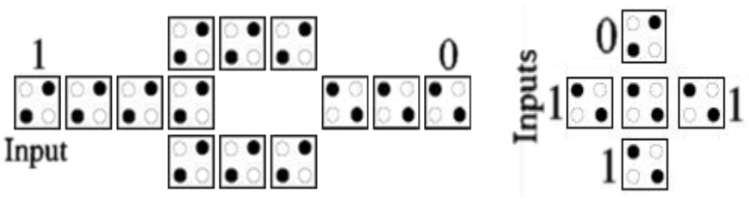
Figure 2.2: Strands of DNA with appropriately chosen sequences of amino acids can come together to form complex structures. In (a), nanoscale DNA tiles have assembled to form Sierpinski Triangles [21] (erroneous connections are indicated with red X's). In (b), a hypothetical tile-based decoder circuit is illustrated [22]. In (c) long strands of DNA have been guided by smaller "staple strands" to form multiple copies of an arbitrarily chosen 2D shape [23].

that DNA-based tiles can be manufactured such that they combine to form complex 2D patterns [24] (See Figure 2.2a). Square-shaped abstractions of these tiles have also been the focus of significant theoretical analysis [25, 26, 22, 27](See Figure 2.2b) as they represent a valuable stochastic generalization of traditional "Wang Tiles" [28]. Along similar lines, Rothemund has studied, and demonstrated, how a long strand of DNA can be directed to fold into arbitrary 2D shapes using a set of carefully chosen "staple strands" (see Figure 2.2c) [23].

The appeal of DNA-based assembly is that a wide range of complex 2D, and even 3D structures can be assembled systematically, thus allowing for a wide range of nanoscale designs. The shortcoming, however, is that the structures themselves cannot be used for computation (they are just static structures). Ideally, computationally functional components could be attached to the DNA, allowing the DNA to act as a scaffolding on which, say, a nanoscale semiconductor-based architecture could be assembled. Unfortunately, the ability to attach DNA to semiconductor-based logic remains undemonstrated. As such, DNA's immediate role in nanoscale computing appears limited.

2.1.3 Quantum-Dot Cellular Automata

The term "Quantum-dot cellular automata", or QCA, refers to a system of cells in which each cell influences its neighbors via quantum effects. If the cells are properly arranged, they can emulate traditional logic circuits (i.e. implement wires and logic gates, see Figure 2.3) [29, 30]. Although quantum effects are used to compute, the computation being performed is classical (although QCAs capable of performing quantum computation have also been proposed [31]). Since QCAs do not rely on currents or voltages to compute, they have the potential to operate at significantly lower



(a) A QCA-based inverter

(b) A QCA-based majority gate

Figure 2.3: In a QCA, the state of each cell influences the states of its neighbors. To compute, the state of one or more input cells is held fixed while their influence propagates. In the above figures, the black dots contained within each cell exert a repulsive force on each other, and on the dots of neighboring cells. In (a) a QCA-based inverter is illustrated in which a row of cells acts as wires and a forking path along that wire serves to invert the state being transmitted. In (b) whichever three input cells are designated as inputs exert their influence over the center cell. The state of that cell is the propagated to the output. (The above figures are modified from `http://www.ece.neu.edu/~mottavi/research.htm`)

power and higher densities than today's CMOS.

Although QCAs offer a nanoscale alternative to semiconductor-based logic, there is currently no means of actually assembling and arranging large scale ensembles of nanoscale quantum-dot cells. As such, much quantum-dot research has focused on the simulation and layout of nanoscale quantum-dot circuits. These hypothetical QCA circuits typically rely on complex and irregular arrangements of cells. Unless a means of fabricating QCA-based circuits is developed that allows for a wide range of nanoscale features, nanoscale QCA architectures cannot be realized.

2.1.4 Biological Computing

A very different approach to nanoscale computing involves the use of biological systems, for example, in vivo protein production. This is not viewed as an alternative to traditional computing, but rather an additional domain in which nanoscale computing is anticipated. Within a cell, one protein can promote another protein, which can in turn suppress other proteins. These dependencies can be modeled as a "biological circuit" capable of implementing general logic [32, 33]. The inherent volatility of these systems, however, poses a substantial design challenge, as does the fact that they are being carried out within a living cell.

2.2 Nanowire Crossbars

All four of the above categories exhibit fundamental characteristics of nanoscale computing, but the modeling and analysis of future chapters focuses on semiconductor-based nanowire (NW) crossbars (see Figure 2.4). A significant range of crossbar-related technology has already been demonstrated. As a result, NW crossbars are viewed as offering the greatest current promise for nanoscale computing. This section provides an overview of NW crossbar technology and explains how crossbars

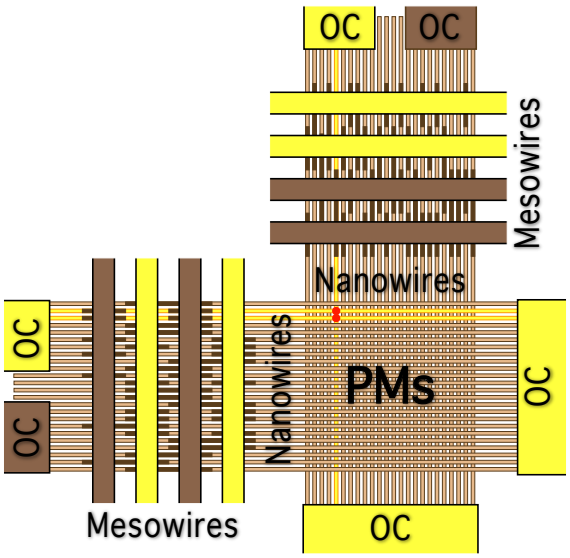


Figure 2.4: A crossbar formed from two orthogonal sets of NWs with programmable molecules (PMs) at the crosspoints defined by intersecting NWs. NWs are divided into groups by connecting them to ohmic contacts (OCs). To address a NW in one dimension, an OC is activated and mesoscale wires are used to turn off all but one NW in that group (see Figure 2.5). Data is stored at a crosspoint by applying a large electric field across it. Data is sensed with a smaller field.

can serve as both memories and programmable logic.

2.2.1 Crossbar Assembly

Multiple approaches have been demonstrated for manufacturing NWs for use in crossbars. One method, known as nanoimprint lithography, effectively stamps a set of parallel undifferentiated (i.e. identical) NWs onto a chip [34]. A related method, termed SNAP, also transfers a pattern of undifferentiated NWs onto a chip [16]. A third, more distinct approach, grows many types of differentiated NWs (i.e. NWs comprised of different sequences of materials) off chip, collects the NWs in a large ensemble, then deposits them onto the chip fluidically [35, 36]. Once deposited, both undifferentiated and differentiated NWs can interface with today's photolithographically produced technology (see Chapter 3).

A NW crossbar is formed by depositing a layer of molecular devices between two orthogonal sets of parallel NWs. At each NW crosspoint, this device layer prevents the two orthogonal NWs from coming into direct contact. Instead the NWs are each connected to opposite ends of what is effectively a two-terminal device across which they may apply an electric potential. One promising two-terminal device is a molecular diode that switches between states of low and high resistance in the presence of a sufficiently large positive or negative electric field [37, 38]. A layer of amorphous silicon has also been proposed as a nanoscale programmable medium [39]. Molecular devices that do not behave like diodes (e.g. programmable resistors and transistors) have been considered as well [19, 40]. Comparisons between these alternatives with regard to their information storage

capacity and ability to provide control over NWs can be found in [41, 42, 43].

Once a NW crossbar is assembled, g photolithographically produced ohmic contacts (OCs) and M photolithographically produced MWs are placed along each dimension of the crossbar (see Figure 2.4). Each OC is in electrical contact with a group of N consecutive NWs. As such, the OCs allow voltages to be applied to blocks of NWs, while each MW provides control over (i.e. makes nonconducting) subsets of NWs within each block. These subsets, however, cannot be chosen deterministically. Instead, the subsets are determined by a stochastic assembly process. As covered in Section 3.2, a number of methods have been proposed for stochastically coupling MWs to NWs. When a MW is turned on during crossbar operation all of the NWs it controls are turned off. This effectively disconnects any molecular devices to which they are connected.

The interface between NWs and MWs is called a **nanowire decoder**, A **simple nanowire decoder** is one in which all NWs are connected to a single OC. A **compound NW decoder** consists of g simple decoders, arranged in parallel (i.e. side by side), which share a common set of MWs (see Figure 2.4). In a compound NW decoder, we sometimes refer to the N NWs within one of the g simple decoders (i.e. the N NWs connected to one of the g OCs) as a **contact group**. A more detailed description of compound versus simple decoders is given in Section 3.1.3.

A NW decoder is said to **address** a particular set of NWs if all NWs in the set remain conducting, or on, while all other NWs are nonconducting, or off. A NW is said to be **individually addressed** if it remains on while all other NWs are off. A more precise definition of what it means to address one or more NWs is given in Section 3.1. If a NW is individually addressed, the OC it is connected to is turned on along with all MWs that *do not* control that NW. As explained in Chapter 3, MWs are coupled to NWs stochastically during NW assembly, and thus not all NWs will be individually addressable.

When NWs along each dimension of a NW crossbar are addressed, the molecular devices at their crosspoints can be accessed and controlled. To accomplish this, the OCs connected to the addressed NWs are used to place either a small or large potential across just the devices being accessed. Since all other NWs are off, the devices located at other NW crosspoints are unaffected. The current flowing across the devices being accessed can also be measured. As we now explain in Sections 2.2.2 and 2.2.3, this combination of addressing NWs via MWs and applying electrical potentials via OCs allows both storage and logic operations to be performed.

2.2.2 Crossbar-based Memories

When many NWs along each dimension of the NW crossbar are addressable, the crossbar can function as a memory (see Figure 2.5). In the crossbar-based memory, read and write operations are executed as follows:

- In a **write operation**, the molecular diodes at NW crosspoints are addressed by creating a sufficiently large potential between one or more addressed pairs of orthogonal NWs. To apply this potential, the OCs at ends of the NWs in each dimension apply the same voltage. This in turn applies a voltage across the crosspoints being addressed, the polarity of which determines their resistive state. By setting the resistive state of a molecular diode to either high or low, a bit of data is written.

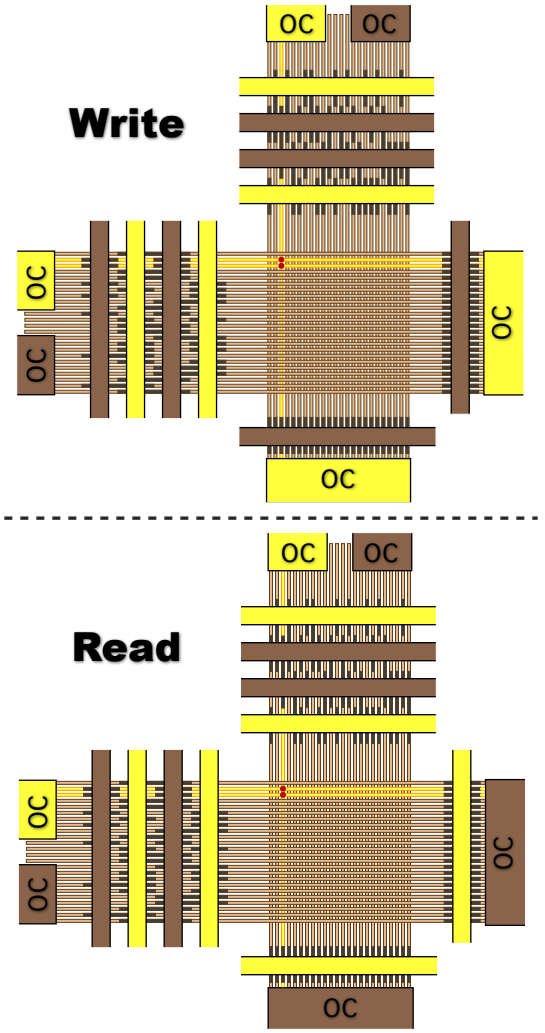


Figure 2.5: A crossbar-based memory in which OCs and MWs read and write data to programmable molecules at crosspoints. The darkened segments along each NW indicate lightly doped regions. These regions become nonconducting when the adjacent MW is turned on. In a read operation an OC at each end of a NW is disconnected from ground. Current flows through any conducting NW crosspoints that are addressed by MWs. The amount of current reveals the value stored at the crosspoints. In a write operation, NWs along each dimension apply a larger electric field across their crosspoints. The direction of the field determines the value stored at the crosspoints. In this figure, the same bit of information is stored at two crosspoints.

- In a **read operation**, crosspoints are again addressed using pairs of NWs. A smaller potential is placed across the crosspoints, preserving their state, and allowing their conductivity to be measured. In the read operation, the addressed NWs in each dimension are disconnected from one of their OCs. As a result, current flows through the crosspoint. The amount of current reveals the resistance of the crosspoint, and hence the value being stored.

2.2.3 Crossbar-based Logic

A read operation can potentially be used to read from multiple crosspoints at once, which, as described below, provides a basis for programmable logic by performing WIRED-ORs. In the context of crossbar-based logic, the NWs along one dimension of a crossbar are called "input NWs", and the NWs along the other dimension are called "output NWs". The output NWs from one crossbar can be supplied as input NWs to a second crossbar (see Figure 2.6).

Consider a NW crossbar memory in which a set S of input NWs are addressed. Notice that any output NW that is connected to one of these inputs via a conducting crosspoint (i.e. a crosspoint that stores a "1") will carry a current. In this sense, each output NW performs a WIRED-OR on whichever input NWs are connected to it. In other words, each output NW carries a 1 if and only if at least one of the input NWs connected to it carries a 1. We formalize this below.

- In a **wired-or operation**, NWs along one dimension of the crossbar serve as inputs and NWs along the other dimension serve as outputs. Molecular diodes connect each input NW to a subset of the output NWs. Multiple input NWs can be addressed, as in a read operation (see above), or alternatively, these NWs can be the current-carrying outputs of another crossbar.

  In the absence of MW control over the output NWs, any output NW connected to an input NW that carries a current will also carry a current (see Figure 2.6). This allows each output NW to perform a WIRED-OR over the NWs to which it is connected. If MWs are used to address a subset of the output NWs, only those NWs will perform a WIRED-OR. In either case, the current-carrying outputs can serve as inputs to a second crossbar [4].

By supplying the outputs of one crossbar as inputs to another crossbar, multiple logical operations can be carried out. If all of these operations are WIRED-ORs, however, the interconnected crossbars will not be able to implement general logic. Also, signal strength will degenerate because an input NW may drive many outputs. DeHon's proposed solution achieves signal restoration via NW-based field effect transistors (FETs) [4] (see Figure 2.6). Here pairs of orthogonal NWs form FETs, allowing each output NW to be coupled to an input NW such that either an inverter or buffer is formed.

- In a **signal restoration operation**, NWs along one dimension of the crossbar serve as inputs and NWs along the other dimension serve as outputs. Each output NW forms an FET with at most one input NW. OCs are used to apply a voltage across the output NWs. The configuration of the OCs and output NWs determines whether each output computes the NOT of the input NW it forms an FET with, or instead acts as a buffer (i.e. computes the identify function).

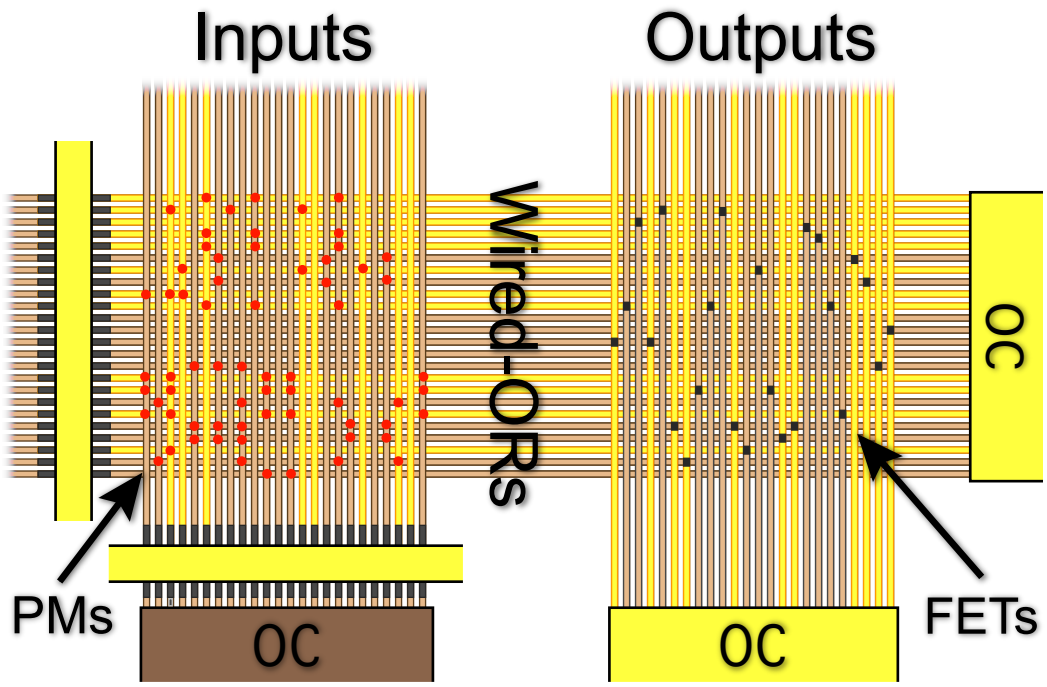


Figure 2.6: A level of reconfigurable crossbar-based logic in which a WIRED-OR operation is followed by a signal restoration operation that also implements negation. Light NWs indicate that a boolean value of "1" is being applied, dark NWs indicate a "0". The two operations collectively implement a WIRED-NOR, and thus form a complete basis for boolean logic. The WIRED-OR operation is implemented like a read operation (See Figure 2.5), except that multiple vertical NWs, and all horizontal NWs, are addressed. Any horizontal NW which is connected to an addressed input NW carries a current. The current carrying horizontal NWs then gate (i.e. make nonconducting) a subset of the output NWs using field-effect transistors (FETs). The diode connections used to perform the WIRED-OR operation can be configured via write operations using a NW decoder (not shown), which is disconnected during normal operation. The FETs used to implement the restoration operation may be placed stochastically.

Since nanoscale programmable FET may not be technologically feasible, DeHon has observed that randomly placed FETs can be used to implement this restoration operation [4]. Though promising, the additional overhead associated with stochastic FET placement has not been rigorously analyzed such that manufacturing errors are taken into account.

Chapter 3

Nanowire Decoders

The previous chapter summarized how NW crossbars can provide a promising basis for nanoscale memories and programmable logic. This chapter focuses on the specific problem of gaining control over individual NWs. One of the primary requirements for realizing crossbar-based architectures is a method for addressing individual NWs with much larger, lithographically-produced MWs. As explained in Section 2.2.1, this interface between MWs and NWs is referred to as a nanowire decoder.

In this chapter, Section 3.1 provides explicit requirements that NW decoders must meet in order to provide reliable control over memories and logic circuits. Section 3.2 reviews a range of proposed decoder manufacturing technologies, highlighting in each case the stochastic aspects of the decoder's assembly process. Stochastic decoder assembly necessitates post-assembly testing and configuration, which is the focus of Section 3.3. Section 3.4 then describes how the behavior of stochastically-assembled NW decoders can be modeled. The "binary model of nanowire control with errors", defined in Section 3.4.1, is central to the probabilistic analysis presented in subsequent chapters. Section 3.5 provides a framework for this analysis.

3.1 Decoder Requirements

In subsequent chapters, we seek to bound the area required to gain control over N NWs using different types of stochastically assembled NW decoders. In this section, we establish the requirements that these decoders must meet in order to provide a sufficient level of control over the NWs. As explained below, the requirements we impose depend on whether the NWs are part of a memory or a circuit.

3.1.1 Nanowire Addressing

First we revisit the definition of NW addressing given at the end of Section 2.2.1. Recall that a NW decoder is said to address a set of NWs if all NWs in the set remain conducting, or on, while all other NWs are nonconducting, or off. A NW is said to be individually addressed if it remains on while all other NWs are off. In order for nanoscale architectures to function fast and reliably, the on/off ratio of addressed versus non-addressed NWs should be large. Sometimes we refer to the set of MWs used to individually address a particular NW as that NW's address.

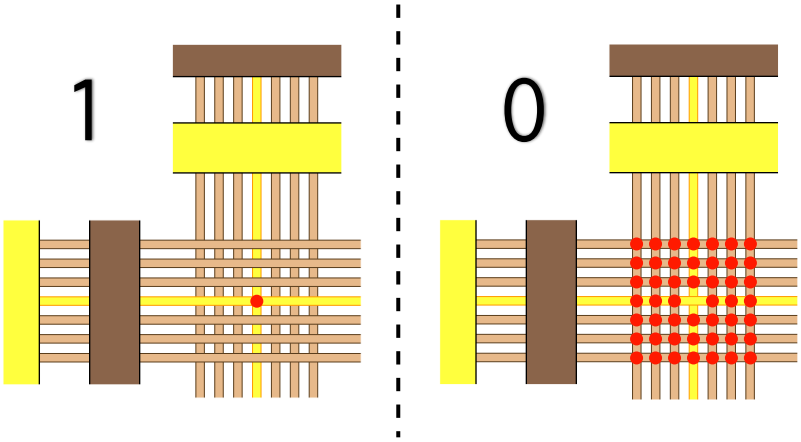


Figure 3.1: On the left, the crosspoint being read has a low resistance, indicating that a "1" is being stored, but all other crosspoints have a high resistance. On the right, however, the crosspoint being read has a high resistance, indicating that a "0" is being stored, but all other crosspoints have a low resistance. To quickly and correctly determine the state of the crosspoint in each case, the amount of current flowing from one dimension of the crossbar to the other must be significantly greater on the left than on the right. This is accomplished only if the on/off ratio of the addressed versus non-addressed NWs is sufficiently large. The same requirement applies to the WIRED-OR portion of crossbar-based logic.

More formally, consider a NW decoder in which each MW, when turned on, increases each NW's resistance by some amount. In this case, we can more formally define what it means for a set of NWs to be addressed as follows [44].

- A set, $\mathcal{S}$, of NWs is **addressed** if and only if a) every NW not in $\mathcal{S}$ has a resistance that is at least α times that of every NW in $\mathcal{S}$ and b) the combined resistance of all NWs not in $\mathcal{S}$ is at least α times that of the combined resistance of all NWs in $\mathcal{S}$, where $\alpha \gg 1$.

In this definition, condition a) ensures that crossbar write operations function correctly (i.e. it ensures that only the addressed crosspoints get written to), and condition b) ensures that the read operations function correctly (see Figure 3.1). The choice of an actual value of α is application specific. For example, a larger value would be required to read data from molecular devices with poor on/off ratios. A larger value would also facilitate reading data more quickly and reliably.

In the context of the above definition, a MW is said to **control** a NW if that MW, when turned on, increases the NW's resistance by a factor larger than α. A MW is said to **not control** a NW if it increases the NWs resistance by a factor very close to 1 (The phrase "very close" is quantified in Section 3.4.2). If the MW increases the NWs resistance by an intermediate factor, the MW is said to **partially control** the NW.

It is also possible for MWs to control NWs in ways other than increasing their resistance. For example, in a diode-based NW decoder (see Section 3.4.2) each MW is connected to a subset of the NWs via diodes. When a MW is grounded, it provides control over NWs by siphoning current from

the NWs to which it is connected. For this decoder, the word "resistance" in the above definitions is no longer appropriate, but the overall sentiment of the definitions remains applicable. Namely, the combined current carrying capacity of addressed NWs must still be significantly greater than that of the non-addressed NWs. Part of the appeal of the model of decoder behavior introduced in Section 3.4.1 is that it avoids the need to distinguish between different ways in which MWs control NWs.

3.1.2 Address Requirements

Having defined what it means for a NW decoder to address a set of NWs, we now establish requirements for the sets themselves. The analysis of subsequent chapters relies on calculating the probability that stochastically assembled decoders fulfill these requirements.

Nanowire Decoders for Memories

As explained in Section 2.2.2, read/write memory operations are performed in a NW crossbar-based memory by using a NW decoder to address NWs along both dimensions of the crossbar. If the decoders along each dimension are each capable of addressing at least N_A disjoint sets of NWs, they can collectively control $(N_A)^2$ disjoint sets of NW crosspoints, each of which can store a bit of information. Since these decoders are comprised of g ohmic contacts (OCs), each of which can be turned on independently, $N_A = \sum_{i=1}^{g} N_A^i$, where N_A^i denotes the number of disjoint sets of NWs that can be addressed when only the i^{th} OC is turned on. NW decoders for memories are analyzed in Chapters 4, 5 and 6.

Nanowire Decoders for Circuits

A NW decoder may also be used to supply patterns of N_A inputs to crossbar-based logic circuits. In order to provide a circuit with arbitrary patterns of inputs, it is not sufficient for the decoder to be able to address N_A disjoint sets of NWs. Instead there must exist a set of N_A NWs such that all subsets of the NWs are addressable. Such a set is said to be **fully addressable**. NW decoders for logic are analyzed in Chapter 7.

3.1.3 Simple Versus Compound Decoders

From Section 2.2.1, recall that a simple NW decoder is one in which all NWs are connected to a single OC. A compound NW decoder is g simple decoders, arranged in parallel, that share a common set of MWs (see Figure 3.2). The NWs within a given simple decoder are sometimes referred to as a contact group. Compound NW decoders are well-suited to meeting the addressing requirements of crossbar-based memories, but are less useful in the context of crossbar-based logic.

When a compound NW decoder is used in a crossbar-based memory, dividing the NWs into many contact groups greatly reduces the number of MWs required to address many NWs. The simple NW decoders associated with the g OCs all share the same MWs, but when NWs are addressed only a single OC is turned on at a time. As a result, whether or not a given NW is individually addressable depends only on the NWs within its contact group. This allows us to

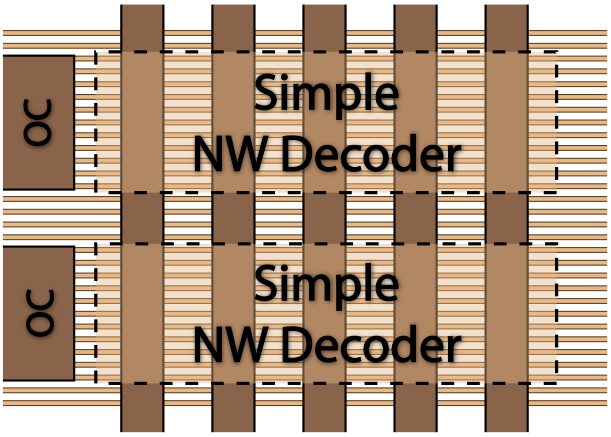


Figure 3.2: A compound NW decoder in which two simple decoders are each connected to a different OC, but share a common set of MWs. Since the OCs are controlled by mesoscale circuitry, they can each be turned on or off independently. The NWs within a particular simple decoder are referred to as a contact group.

maximize the probability that many NWs are individually addressable by making the number of NWs per OC, N, as small as manufacturing constraints allow. The limits of photolithography suggest that $N \approx 10$.

Interestingly, compound NW decoders are not as valuable for controlling crossbar-based logic. In the absence of any MWs, providing N_A fully addressable inputs to a circuit would require N_A OCs, and $N' = N(N_A)$ total NWs. To do better, we would hope that each OC is connected to several of the N_A fully addressable NWs. This implies, however, that most input patterns to the circuit require most or all OCs be on. As a result, the subsets of MWs that are used to supply most input patterns would still work if all NWs were connected to a single, larger OC. The gain of using the compound decoder over a simple decoder is modest. (This line of reasoning is revisited in Chapter 7.)

3.2 Decoding Technologies

There are a number of decoder technologies which appear capable of fulfilling the addressing requirements described in the previous section. In this section we review these technologies. Even though a wide range of NW decoders can plausibly be considered, Section 3.4 and 3.5 describe how most of these decoders can be modeled and analyzed in a unified fashion.

3.2.1 Encoded Nanowire Decoders

In an **encoded NW decoder**, many differently encoded NWs are grown separately, collected in a large ensemble, then deposited onto a chip via fluidic methods. Two technologies for encoding NWs have been considered. Both are analyzed in detail in Chapter 5.

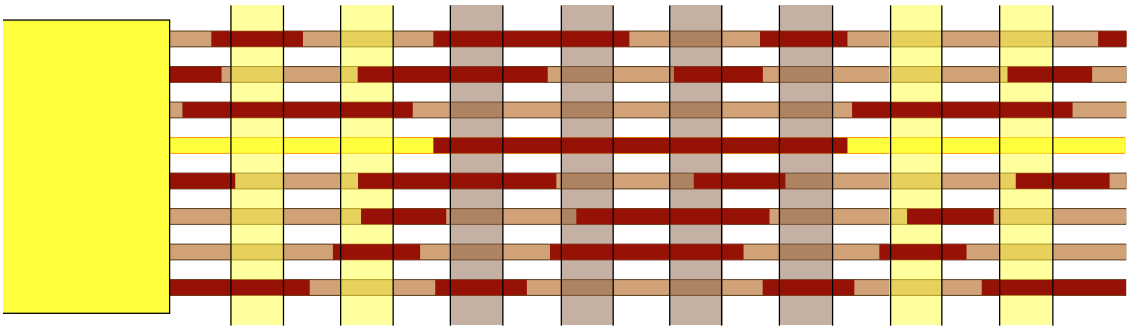


Figure 3.3: An encoded NW decoder in which each NW has a sequence of lightly and heavily doped regions along its axis. When a subset of the MWs is activated, all NWs with lightly doped regions under those MWs become nonconducting. This allows MWs to address NWs with a particular encoding. Since encodings are assigned randomly to NWs, the probability that many NWs are individually addressable is determined by the total number of possible encodings. Fluidic assembly cannot guarantee that lightly doped regions align with MWs.

Axially Encoded NW Decoders

The axially encoded NW decoder is produced using modulation-doped NWs [45, 46]. These NWs are grown with sequences of lightly and heavily doped regions along their axis [18]. Once grown, many NWs with each encoding are collected in a large ensemble, and a random subset of the NWs is deposited in parallel to form each dimension of a crossbar [6]. MWs are then laid down along the periphery of the crossbar to form decoders. To prevent adjacent NWs from coming into electrical contact, NWs can potentially be grown with an insulating shell that would then be etched away before the MWs are placed.

When a MW is deposited on top of a NW's lightly doped region, that MW provides control over that NW by forming a field-effect transistor (FET). When the MW is turned on, it applies an immobilizing electric field that greatly increases the resistance within the adjacent lightly doped region. If NWs are properly encoded, NWs with each encoding can addressed separately (see Chapter 5 Section 5.1). NWs are addressed by turning on all MWs that do not control them (see Figure 3.3). As explained in Chapter 5, using more encodings, and hence more MWs, increases the probability that many NWs are individually addressable.

A potential challenge facing axially encoded NW decoders is that fluidic NW deposition does not guarantee end-to-end alignment of NWs. As a result, a MW may only partially lie on top of a particular lightly doped region, and thus only partially control a particular MW. Radially encoded NWs offer a potential solution to this problem.

Radially Encoded Nanowire Decoders

In a radially encoded NW decoder, core-shell NWs are used in place of modulation-doped NWs [47]. Core-shell NWs are produced by growing shells composed of separately etchable materials around a lightly doped core. In this way, NWs are encoded using shell sequences in place of doping

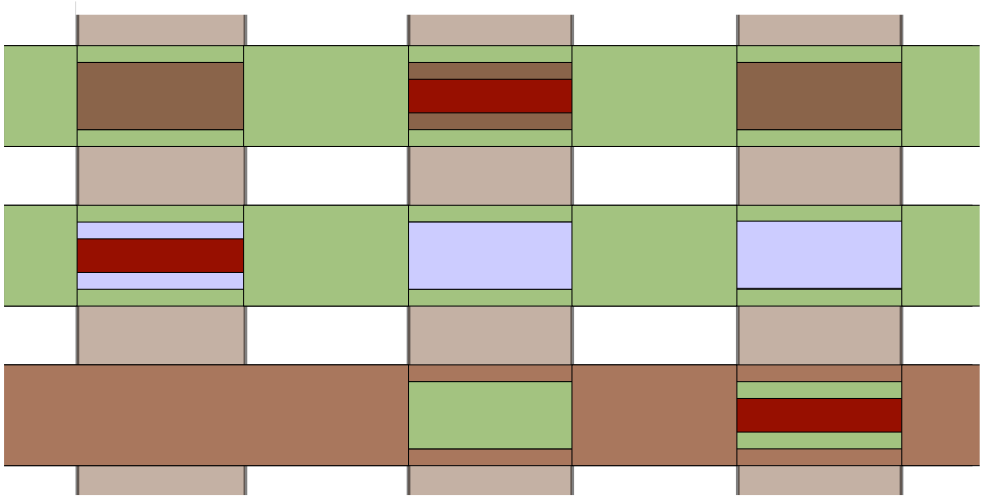


Figure 3.4: A radially encoded NW decoder in which a different shell sequence has been etched away under each MW. When turned on, each MW only controls (i.e. makes nonconducting) the NWs with an exposed lightly doped core under that MW. In this decoder, each NW encoding is controlled by a different MW, but other decoder designs are possible (see Chapter 5 Section 5.4).

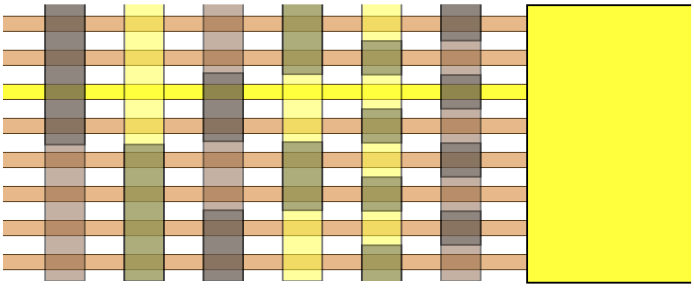
sequences. As explained in Chapter 5 Section 5.4, there are multiple approaches for gaining control over differently encoded core-shell NWs.

The simplest approach uses one MW to control NWs with each shell sequence (see Figure 3.4). In the space reserved for each MW, a particular sequence of k shell materials is etched away. If each NW has k shells initially, this process exposes only the lightly doped core of NWs with one particular shell sequence. On all other NWs, at least one shell remains. As a result, each MW controls only NWs with one specific shell sequence. Those NWs are addressed by turning on all other MWs.

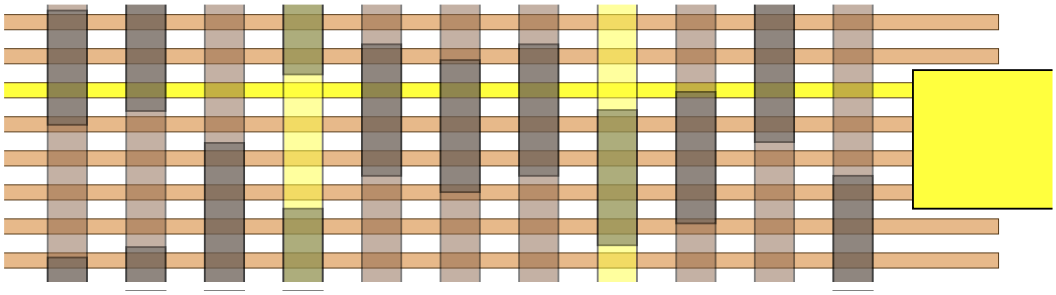
3.2.2 Mask-Based Decoders

Encoded NW decoders require that NWs with different encodings be grown off chip, then deposited fluidically. In contrast, mask-based decoders can be used to control any type of straight uniformly-spaced lightly doped NW. This decoder was first proposed for use with NWs produced by the superlattice nanowire pattern transfer method (SNAP) [16]. It can also be used with NWs grown by nanoimprinting [34, 48].

In a **mask-based decoder**, lithograpically-defined high-K dielectric rectangles are deposited between NWs and MWs. These regions of high-K dielectric focus the field strength of adjacent MWs, thereby causing the lightly doped NWs sitting under each region to turn off when the adjacent MW is turned on. If the lithographically-defined rectangles could be as small as the pitch of NWs and placed with nanometer accuracy, a mask-based decoder with $M = 2\log_2 N$ MWs could be used to individually address N NWs (see Figure 3.5(a)). Unfortunately, the rectangles cannot be manufactured with this level of precision. Thus, it has been proposed that many copies of



(a) A logarithmic-sized mask-based decoder



(b) A randomized mask-based decoder

Figure 3.5: A masked-based NW decoder in which regions of high-K dielectric allow each MW to control a different subset of NWs. (a) If arbitrarily small high-K dielectric regions could be manufactured and placed with nanoscale precision, $2\log(N)$ MWs could be used to address each of N NWs. (b) Since this is not possible, many randomly shifted copies of the smallest manufacturable region can be used to gain control over individual NWs.

the smallest manufacturable lithographically-defined rectangles be deposited and that the natural randomness in their placement be used to gain control over individual NWs with high probability [7, 49, 50] (see Figure 3.5(b)). This approach is analyzed in Chapter 6.

3.2.3 The Randomized-Contact Decoder

The **randomized-contact decoder** is another proposed method for addressing undifferentiated NWs. In a randomized-contact decoder, MWs are randomly coupled to NWs by a manufacturing process that makes each NW/MW junction controlling, noncontrolling, or partially controlling independently at random (see Figure 3.6). There are a number of ways such decoders might be produced. One proposed approach is to randomly deposit impurities, such as gold particles, onto undifferentiated NWs [51]. Another approach is to randomly deposit small regions of high-K dielectric, or alternatively, randomly etch or fill holes in a low-K dielectric [52]. A randomized-contact decoder can also be constructed from axially encoded NWs. If many sets of axially encoded NWs are produced with randomly placed lightly doped regions, each NW/MW junction can be treated as an independent random variable. As a result, analysis of randomized-contact decoders

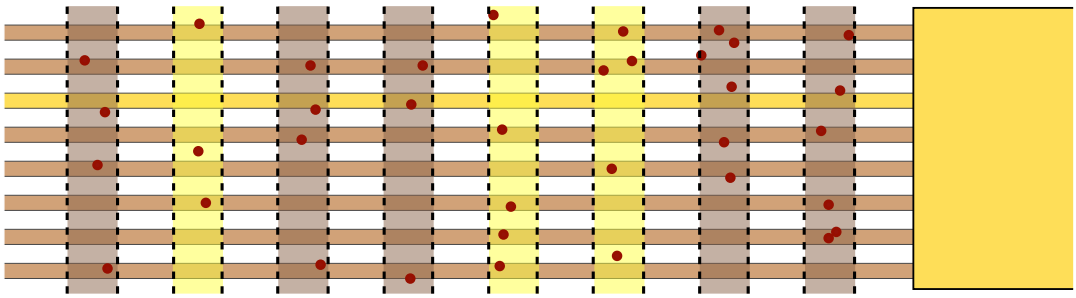


Figure 3.6: A randomized-contact decoder in which random particle deposition causes each MW to control each NW independently at random

provides bounds that apply to encoded NW decoders as well. A detailed analysis of randomized-contact decoders is presented in Chapter 4

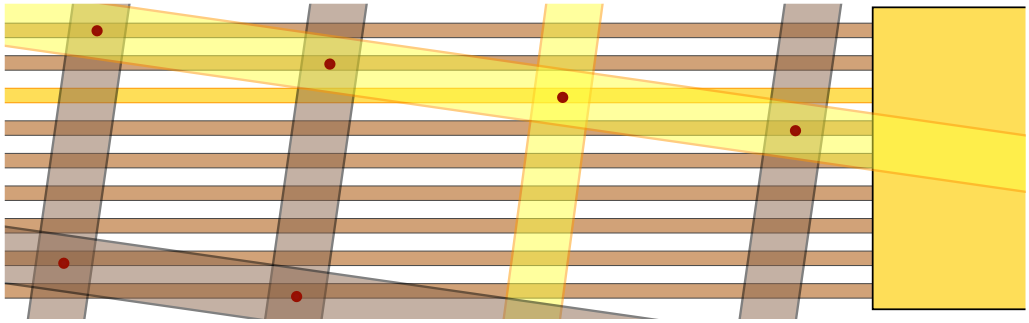
3.2.4 Additional Decoding Technologies

Other methods for controlling NWs with MWs have been proposed. Each has uncertainties in their construction that have not been fully analyzed. The decoders described in this subsection are not the focus of later chapters.

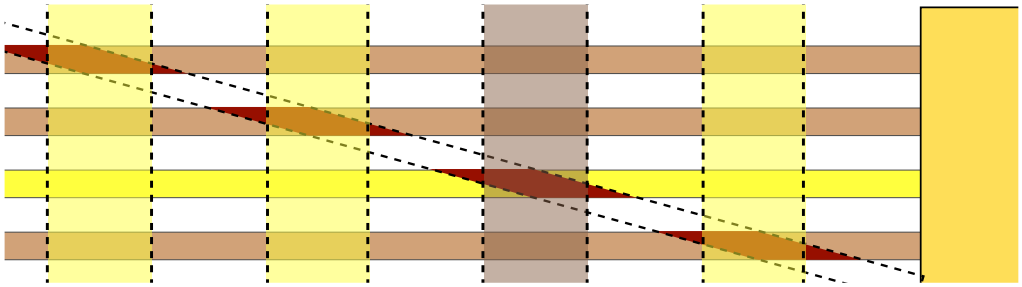
Rotational Offset Decoders

Likharev *et al* have proposed a hybrid CMOS/nanoscale architecture know as "CMOL" in which the NWs in a NW crossbar would be controlled by MWs in a MW crossbar [53]. To interface MWs with NWs, pins with nanoscale diameter tips would be formed at the MW crosspoints. If the mesoscale grid is appropriately rotated relative to the nanoscale grid, MW crosspoints can be coupled to NWs in a one-to-one fashion (see Figure 3.7(a)). Currently this pin-based interface has not been demonstrated. Also, it appears sensitive to small changes in the angle of rotation between the two crossbars, as well as nanoscale variation in pin placement and orientation.

Franzon *et al* have proposed a NW decoder that is also based on the rotational offset between two sets of wires [54]. Here a parallel set of insulated NWs would be interfaced with MWs by first exposing NW-width regions that cut diagonally across multiple NWs (see Figure 3.7(b)). These regions would be defined by a temporarily placed set of NWs acting as a mask. Each MW, once laid down, would only control a NW if that NW's adjacent insulation had been removed. The angle of the exposed regions should be chosen so that each NW has an exposed portion under a different MW. As with the CMOL decoder, this manufacturing technique appears sensitive to small changes in the angle of the nanoscale exposed region. Also, translational misalignment of the region relative to the MWs could result in MWs that provide only partial control over NWs.



(a) A CMOL decoder



(b) A diagonal cut decoder

Figure 3.7: Several decoders have been proposed based on the rotational offset of two parallel sets of NWs. Likharev *et al* have proposed (a), a NW decoder in which NWs are interfaced with MWs via nanoscale pins placed at the crosspoints of a mesoscale grid [53]. Franzon *et al* have proposed (b), a NW decoder in which a second set of temporarily placed NWs are used to define diagonal cuts. If angled appropriately, these nanoscale cuts would expose a portion of each NW under a different MW [54].

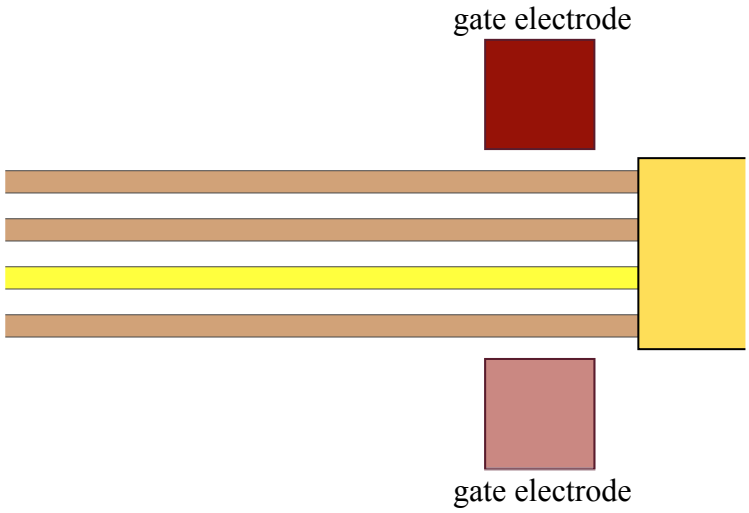


Figure 3.8: A "micro to nano addressing block", or MNAB, in which two opposing gate electrodes produce a variable strength electric field to turn off all but one of the four semiconducting NWs [39].

MNAB

Wickramasinghe *et al* have demonstrated control over NWs using a "micro to nano addressing block", or MNAB [39]. Here a small number of parallel NWs are attached to each OC and gate electrodes are positioned on either side of those NWs (see Figure 3.8). To address individual NWs, the two gate electrodes must simultaneously apply different voltages chosen so as to deplete (i.e. turn off) all but one of the lightly doped NWs. For this decoder to function reliably, both the gate electrode voltages and the NW doping concentrations must be chosen properly. As yet, no modeling of the variability in the assembly of MNAB has been presented. It is also unclear if this method can control blocks of more than three or four NWs with sub-10nm diameters.

3.3 Post-Assembly Configuration

Since NW decoders are assembled stochastically, post-assembly testing and configuration are required before they can be used to control memories or logic. A testing algorithm is needed to discover which sets of MWs can be used to address NWs. Configurable addressing circuitry is needed to store these addresses and provide a consistent external interface to the decoders.

3.3.1 Address Discovery

In a stochastically assembled NW decoder each NW has an address, or "codeword" (see Section 3.4 below), determined by which MWs do and do not control it. Since NW addresses are randomly generated during decoder assembly, they must be discovered through testing. This is a difficult problem, as some addresses may mask others, and faults may make test outputs unreliable. Several

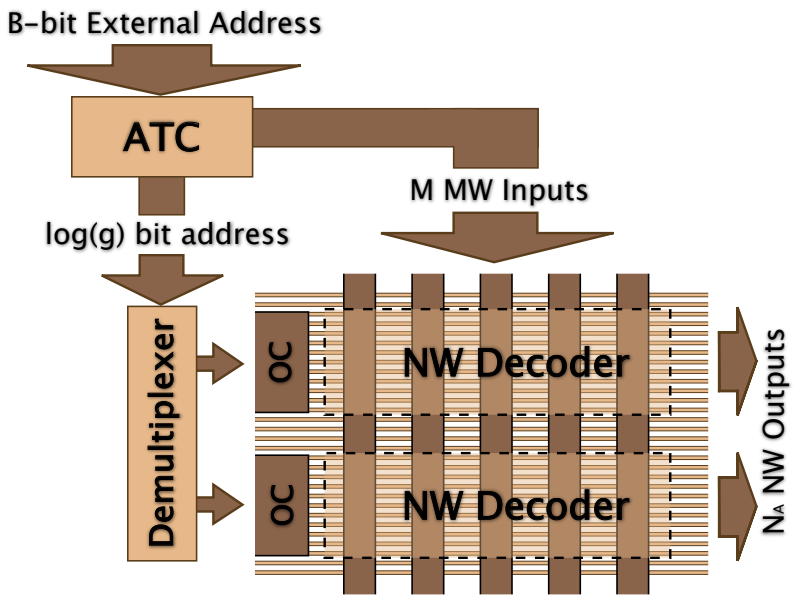


Figure 3.9: In order to control a NW decoder and provide a consistent external interface, programmable address translation circuitry (ATC) is required to map B-bit inputs to OCs and MWs.

approaches to address discovery have been considered. Chapter 8 analyzes the number of test operations these approaches require.

In [12] an efficient testing procedure involving read/write operations was given for encoded NW decoders. Although the procedure could be adapted for other decoders, its reliance on nanoscale storage devices is a drawback. Tests using read/write operations are relatively time consuming and possibly faulty. Also, in crossbar-based circuits they may not even be possible, as not all NWs are used to control nanoscale storage devices.

As an alternative, one can consider testing algorithms that apply a voltage across the N NWs within a single contact group, turn on a subset of the MWs, then measure if any NW remains conducting [55, 56] (i.e. if the N NWs collectively carry any current). This conductance test does not reveal which NW is on, nor does it reveal if multiple NWs are on. Nonetheless, it is sufficiently powerful to determine which subsets of MWs address individual NWs [56].

3.3.2 Address Translation Circuitry

When a memory is supplied with a particular external binary address, **address translation circuitry** (ATC) along each dimension of the crossbar maps that address to an OC and set of MWs to activate. This mapping depends on how the decoder was stochastically assembled. To ensure each external address does in fact address some NW, the ATC must store information about which MWs control some or all of the NWs (see Figure 3.9)

In order to make this ATC fast, reliable and easy to manufacture, it may be implemented at the mesoscale. Any approximation of the area required to control a NW-crossbar must take into

account not just the area of MWs and OCs, but also the area used to store NW addresses using a mesoscale ATC (the prospect of implementing the ATC using nanoscale storage is considered in [40]). The size of the ATC is explicitly modeled in [12] and [56] (see Section 3.5.1), but it has received less attention elsewhere. The appendix of [11] also estimates the area required for the ATC, but does so without exploring how different address mapping strategies affect area requirements.

The ATC must associate an OC and subset of MWs with each B-bit external address. In the worst case, this requires $\log_2 g + M$ bits of storage for each of the 2^B addresses. In some cases fewer bits are required. For example, if every NW is individually addressable, and the number of NWs per contact group is a power of 2, the high order bits of each external address can be used to index an OC. This fixed mapping between high order bits and OCs allows the ATC to store only M bits per address. The way in which B-bit addresses are mapped to OCs and MWs is called an **addressing strategy**. In Section 3.5.2 multiple addressing strategies are discussed in detail. The analysis of subsequent chapters reveals that some addressing strategies require significantly more overall area than others.

3.4 Modeling Nanowire Decoders

The goal of subsequent chapters is to bound the resources required to gain control over NWs. More concretely, we wish to know how many MWs, M, are required to address N NWs, and how much ATC area is required to control those MWs. To accomplish this, we need an explicit model for how MWs address NWs. Rather than define a technology-specific physical model of decoder behavior, which would be difficult to analyze probabilistically, Section 3.4.1 defines a simpler, more abstract model termed the **binary model of nanowire control with errors**. Section 3.4.2 contrasts this model with a real-valued physical model, highlighting its simplicity and generality.

3.4.1 The Binary Model of Nanowire Control with Errors

For each NW, $\boldsymbol{n_i}$, we can describe the subset of MWs that control it using an M-tuple, $\boldsymbol{c^i} \in \{0, 1, e\}^M$, called its **codeword**. Let c^i_j indicate the j^{th} position of $\boldsymbol{c^i}$. Then

- $c^i_j = 1$ if MW $\boldsymbol{m_j}$ controls $\boldsymbol{n_i}$. In this case the MW/NW junction is said to be controlling.
- $c^i_j = 0$ if MW $\boldsymbol{m_j}$ does not control $\boldsymbol{n_i}$. In this case the MW/NW junction is said to be noncontrolling.
- $c^i_j = e$ if MW $\boldsymbol{m_j}$ partially controls $\boldsymbol{n_i}$. In this case the MW/NW junction is said to be partially controlling, or "in error".

Since proposed NW decoders employ stochastic assembly processes, codewords are assigned to NWs according to some probability distribution. This distribution is determined by how the decoder is manufactured. For example, in a randomized-contact decoder each bit of each codeword is an i.i.d random variable.

A NW decoder addresses a set of NWs by applying an electric field to a subset of the MWs. These MWs are said to be **activated** or on. The set of activated MWs is called an **activation**

pattern. A particular activation pattern, $\boldsymbol{a} \in \{0,1\}^M$, is represented as a binary M-tuple where $a_j = 1$ if and only if the j^{th} MW is activated. Given a MW activation pattern and a codeword associated with each NW, we model the decoder's behavior as follows.

- When a NW decoder is used to address NWs, NW $\boldsymbol{n_i}$ is said to be reliably off if some MW $\boldsymbol{m_j}$, for which $c_j^i = 1$, is activated.
- $\boldsymbol{n_i}$ is said to be reliably on if no MW for which $c_j^i = 1$ or $c_j^i = e$ is activated.
- If a NW is neither reliably off nor reliably on, it is said to be in error. In other words, a NW is in error if there is an activated MW, $\boldsymbol{m_j}$, such that $c_j^i = e$ and there is no activated MW, $\boldsymbol{m_k}$, such that $c_k^i = 1$.
- A decoder with junctions that are in error behaves reliably when activation patterns are supplied such that no NW is in error.

Nanowire Addressability

From Section 3.1.1, recall that a NW is individually addressed if it remains on while all other NWs are turned off. In an error-free decoder (i.e. a decoder where no MW/NW junctions are in error), NW $\boldsymbol{n_i}$ is individually addressable if and only if it is individually addressed by the activation pattern $\boldsymbol{a} = \overline{\boldsymbol{c^i}}$. In other words, if $\boldsymbol{n_i}$ is individually addressed by any activation pattern, $\boldsymbol{a'}$, it must also be individually addressed when any additional noncontrolling MWs (for which $c_j^i = 0$) are turned on.

This observation implies that if a $\boldsymbol{n_i}$ is not individually addressable, it is not addressed by $\boldsymbol{a} = \overline{\boldsymbol{c^i}}$, and thus there is some other codeword $\boldsymbol{c^k}$ such that for each j it is not true that $c_j^i = 0$ and $c_j^k = 1$. This is the mathematical definition of implication; that is, c_j^k implies c_j^i. When this condition holds for all values of j, we say that $\boldsymbol{c^k}$ **implies** $\boldsymbol{c^i}$, and write $\boldsymbol{c^k} \Rightarrow \boldsymbol{c^i}$. From this, we can succinctly state that in an error-free decoder, $\boldsymbol{n_i}$ is individually addressable if and only if no NW's codeword implies $\boldsymbol{c^i}$.

In the case of errors, a reasonable generalization of the above definition of codeword implication is to say that c_j^k "possibly implies" c_j^i if it is not true that $c_j^i = 0$ and $c_j^k = 1$. When this condition holds for all values of j, we say that $\boldsymbol{c^k}$ **possibly implies** $\boldsymbol{c^i}$. This definition still allows us to assert that $\boldsymbol{n_i}$ is individually addressable if no NW's codeword "possibly implies" $\boldsymbol{c^i}$. In future chapters, when it is clear that we are using the binary model with errors, we still write $\boldsymbol{c^k} \Rightarrow \boldsymbol{c^i}$ and use the term "implies" when we technically mean "possibly implies".

Example 3.4.1 *To better understand the above definitions, consider a simple NW decoder with* 5 *MWs and* 4 *NWs. Suppose codewords* $\boldsymbol{c^1} = 10100$, $\boldsymbol{c^2} = 10101$, $\boldsymbol{c^3} = 1e010$ $\boldsymbol{c^4} = ee011$ *are present on the four NWs.*

In this case NW $\boldsymbol{n_1}$ *is individually addressed by the MW activation pattern* $\boldsymbol{a} = 01011$. *When this pattern is applied,* $\boldsymbol{n_2}$, $\boldsymbol{n_3}$ *and* $\boldsymbol{n_4}$ *are all reliably turned off. Since* $\boldsymbol{c^1}$ *implies* $\boldsymbol{c^2}$, *however,* $\boldsymbol{n_2}$ *cannot be individually addressed.*

Even though $\boldsymbol{c^3}$ *contains an error,* $\boldsymbol{n_3}$ *can still be individually addressed with the activation pattern* $\boldsymbol{a} = 00101$*. The same guarantee cannot be made for* $\boldsymbol{n_4}$*, however, since* $\boldsymbol{c^3}$ *possibly implies* $\boldsymbol{c^4}$*.*

As the above example illustrates, the binary model with errors provides a way of accommodating errors. Even if NWs contain junctions which are in error, it is still possible for them to be addressed reliably. An additional level of fault-tolerance can be added if codewords are sufficiently far apart [57, 58, 44].

- Let the **directed distance** between two codewords, denoted $d_{dir}(\boldsymbol{c^i}, \boldsymbol{c^j})$, be the number of positions in which $\boldsymbol{c^i}$ has a 0 and $\boldsymbol{c^j}$ has a 1. Let the **symmetric distance** denoted $d_{sym}(\boldsymbol{c^i}, \boldsymbol{c^j})$ be the minimum of $d_{dir}(\boldsymbol{c^i}, \boldsymbol{c^j})$ and $d_{dir}(\boldsymbol{c^j}, \boldsymbol{c^i})$.

If the symmetric distance between all codewords is at least 1, then all NWs are individually addressable (since no codeword possibly implies another). If the symmetric distances are all greater than 1, then not only will all NWs be individually addressable, but the guarantee on their on/off ratios will have increased (see the end of Section 3.4.2). This can allow for faster and more reliably decoders. It can also allow the decoder to cope with transient faults in which a MW fails to adequately control a NW.

Monotone DNFs with Errors

Rather than describe the binary model with errors using codewords, we can also describe the model using monotone disjunctive normal forms (DNFs) in which some variables are designated as "in error". The main advantage of this approach is that during testing (see Section 3.3.1 above) selective queries are used to learn which NW addresses are present. This can be recast as learning which clauses are present in the monotone DNF. As discussed at the end of Chapter 8, monotone DNF learning algorithms have already been extensively studied among computer scientists (although most of this work does not accommodate variables which are in error).

In the case of a simple NW decoder consisting of N NWs and M MWs, the corresponding DNF consists of N clauses, each with up to M boolean variables. In each clause, all variables are negated. Each clause corresponds to a codeword as follows.

- In the N-term monotone DNF, $\phi = \phi_1 \vee \phi_2, \ldots \vee \phi_N$, that models a simple NW decoder, each clause, ϕ_i, corresponds to NW $\boldsymbol{n_i}$ with codeword $\boldsymbol{c^i}$. Each boolean variable, x_j, corresponds to a MW, $\boldsymbol{m_j}$. If $\boldsymbol{m_j}$ is activated, $x_j = 1$.

- In each clause, ϕ_i, the literal $\overline{x_j}$ (meaning the negation of x_j) appears if $c^i_j = 1$. In other words, when ϕ_i contains $\overline{x_j}$, $\boldsymbol{m_j}$ controls $\boldsymbol{n_i}$. The clause evaluates to 0 when $\boldsymbol{m_j}$ is activated, since in this case $x_j = 1$ and the literal $\overline{x_j}$ evaluates to 0.

- In each clause, ϕ_i, the literal $\overline{x_j}$ appears *in error* if $c^i_j = e$. When ϕ_i contains $\overline{x_j}$ in error, $\boldsymbol{m_j}$ provides only partial control of $\boldsymbol{n_i}$. In this case, $\overline{x_j}$ evaluates to 1 if $x_j = 0$, but evaluates to e if $x_j = 1$. Literals that are in error *cannot* evaluate to 0.

- On a given binary input, ϕ_i evaluates to 1 if all literals, $\overline{x_j}$, that appear in the clause are 1. It evaluates to 0 if at least one literal in the clause that is not in error is 0. Otherwise, the clause evaluates to e and is said to be *in error*. In other words, when some literals evaluate to e, $e \wedge 1 = 1 \wedge e = e$, $e \wedge 0 = 0 \wedge e = 0$ and $e \wedge e = e$.

- On a given binary input, ϕ evaluates to 0 if all clauses evaluate to 0, and evaluates to 1 if at least one clause evaluates to 1. Otherwise the entire DNF evaluates to e. In other words, when some clauses evaluate to e, $e \vee 1 = 1 \vee e = 1$, $e \vee 0 = 0 \vee e = e$ and $e \wedge e = e$. When a DNF evaluates to e, it is said to be *in error*.

This DNF-based model is well-suited toward study of decoder testing, since testing a given MW activation pattern corresponds to querying the DNF on a particular input. When the underlying DNF evaluates to e, this reflects the fact that each NW is either nonconducting or partially conducting. As a result, the observed value of the DNF (i.e. the current measured between OCs) is unreliable. As explained in Chapter 8, the goal of a NW testing algorithm is to learn as many clauses of the underlying DNF as possible, even though certain tests may be unreliable.

Example 3.4.2 *The example given at the end of the previous subsection can be restated in terms of a monotone DNF. As before, consider a simple NW decoder in which $M = 5$, $N = 4$ and the codewords $\boldsymbol{c^1} = 10100$, $\boldsymbol{c^2} = 10101$, $\boldsymbol{c^3} = 1e010$ $\boldsymbol{c^4} = ee011$ are present on the four NWs.*

These codewords correspond to the clauses $\phi_1 = \overline{x_1} \wedge \overline{x_3}$, $\phi_2 = \overline{x_1} \wedge \overline{x_3} \wedge \overline{x_5}$, $\phi_3 = \overline{x_1} \wedge \overline{x_2}^e \wedge \overline{x_4}$ and $\phi_4 = \overline{x_1}^e \wedge \overline{x_2}^e \wedge \overline{x_4} \wedge \overline{x_5}$ respectively. Here the superscript e indicates that a particular variable is in error.

The entire decoder is then represented by the monotone DNF $\phi = \phi_1 \vee \phi_2 \vee \phi_3 \vee \phi_4$. On any given input, each clause either evaluates to 0, 1 or e. A clause evaluates to e if at least one variable which is in error is 1, but no variable not in error is 1. The entire DNF, ϕ evaluates to e if at least one clause evaluates to e, and no clause evaluates to 1.

3.4.2 Real-valued Physical Models

Instead of using the binary model of nanowire control with errors, a NW decoder can be modeled as a simple physical system. The problem with this approach is that such a model would be both difficult to analyze, and highly technology-specific. In this subsection we give an example of a real-valued physical decoder model. We then highlight the difficulties of working with such a model and explain why the binary model with errors provides an appealing alternative.

Resistive Nanowire Decoders

Many of the proposed methods for constructing NW decoders discussed in Section 3.2 utilize stochastically placed FETs at MW/NW junctions. In such decoders each MW, when activated, increases each NW's resistance by some amount. Such decoders can be modeled as follows [44].

- In the **resistive model of NW control**, each NW $\boldsymbol{n_i}$ has initial resistance η_i when no MWs are activated.

- Associated with each NW is a length-M vector of reals, or a **real-valued nanowire codeword**, $\boldsymbol{r^i}$. The jth entry of $\boldsymbol{r^i}$, r^i_j, is the amount by which the jth MW increases the resistance of $\boldsymbol{n_i}$ when activated.

- When the decoder is supplied with an activation pattern, $\boldsymbol{a}$, the resistance of NW $\boldsymbol{n_i}$ is $\eta_i + \boldsymbol{a} \cdot \boldsymbol{r^i}$ where $\boldsymbol{a} \cdot \boldsymbol{r^i}$ is the inner product of $\boldsymbol{a}$ and $\boldsymbol{r^i}$.

The definition of NW addressing given in Section 3.1.1 directly applies to this model of NW control, as do the decoder address requirements described in Section 3.1.2. Even so, to analyze a stochastically assembled decoder using this real-valued model, one would need to first specify a continuous probability distribution with which real-valued nanowire codewords are assigned. One would also need to quantify the probability that these codewords satisfy various addressability requirements. Both tasks are challenging and mathematically cumbersome. It is much simpler to map the resistive model on to the binary model with errors.

Consider the following approach for mapping each $\boldsymbol{r^i}$ to a binary codeword with errors, $\boldsymbol{c^i}$.

- $c^i_j = 0$ if $r^i_j \leq r_{low}$
- $c^i_j = 1$ if $r_{high} \leq r^i_j$
- $c^i_j = e$ if $r_{low} \leq r^i_j \leq r_{high}$

Here r_{low} and r_{high} must be chosen so that a set $\mathcal{S}$ of NWs can correctly be considered addressed (as defined in Section 3.1.1) by any activation pattern, $\boldsymbol{a}$, for which the following two conditions hold.

- for each $\boldsymbol{n_i} \in \mathcal{S}$, $c^i_j = 0$ when $a_j = 1$,
- for each $\boldsymbol{n_k} \notin \mathcal{S}$, there exists a j such that $c^k_j = 1$ and $a_j = 1$.

To bound r_{high} and r_{low}, suppose $\boldsymbol{a}$ meets these two conditions. Let $r_{base} = \max_i \eta_i$. Observe that every NW in $\mathcal{S}$ has resistance at most $R_L = r_{base} + (M-1)r_{low}$ because at most $M-1$ MWs are activated. Also, note that every NW not in $\mathcal{S}$ has resistance at least $R_H = r_{high}$. From definition of "addressed" in Section 3.1.1, it is clear that $\mathcal{S}$ is addressed if $R_H \geq \alpha(N-1)R_L$ or $r_{high} \geq \alpha(N-1)(r_{base} + (M-1)r_{low})$. To simplify, let $r_{low} = cr_{base}$ for some constant $c > 0$. Then, the condition becomes $r_{high} = \alpha(N-1)(cM - c + 1)r_{base}$.

Using the above values of r_{high} and r_{low}, the resistive model can be mapped to the binary model with errors such that a NW $\boldsymbol{n_i}$ is addressable in the binary model it is addressable in the resistive model. Once again, we emphasize that the value of this approach is that the binary model is significantly easier to work with in the context of probabilistic analysis.

As a generalization of the above mapping strategy, we could also lower the threshold r_{high} (or raise r_{low}), but require that the symmetric distance (as defined in Section 3.4.1) between all NWs be at least d. In this case we would again set $r_{low} = cr_{base}$ and $r_{high} = (k/d)(cM - c + 1)(N-1)r_{base}$.

Diode-Based Nanowire Decoders

Not all proposed NW decoding technologies are based on FETs that allow each to increase the resistance of certain NWs. Another potential type of NW decoder is one in which diode connections are made between MWs and NWs [57]. In a diode current only flows in one direction. If each MW is connected to a subset of NWs via diodes, then that MW, when activated (i.e. grounded), will siphon off current from just those NWs. As with FET-based control over NWs, diode-based control can also be modeled using the binary model with errors.

In a diode-based NW decoder, when MW $\boldsymbol{m_j}$ is not connected to NW $\boldsymbol{n_i}$, it provides no control over that NW, and hence $c_j^i = 0$. When $\boldsymbol{m_j}$ is connected to $\boldsymbol{n_i}$ by a sufficiently conductive diode it provides reliable control over $\boldsymbol{n_i}$, and thus $c_j^i = 1$. Finally, if $\boldsymbol{m_j}$ is connected to $\boldsymbol{n_i}$ by a faulty diode, and provides only partial control over $\boldsymbol{n_i}$, $c_j^i = e$. Once again, use of the binary model with errors avoids the need to work with a more complicated real-valued physical model.

More Complex Behavior

Proposed NW/MW interfaces are not limited to diodes and FETs. For example, negative-differential resistors have also been proposed. These are connections that actually demonstrate a decrease in current when the voltage across them increases. Fortunately the binary model with errors is accommodating to a wide range of decoding technologies.

The Difficulty with Real-Valued Models

As this subsection has illustrated, a key difficulty in using real-valued models to describe NW decoder behavior is that very different models are required for different decoder technologies. This fails to exploit the fact that the different technologies are being used to produce devices that are, in essence, functionally equivalent (i.e. each MW provides control over a subset of NWs). Also, since decoders are assembled stochastically, real-valued models require that continuous probability distributions be associated with their various parameters. Not only does this make probabilistic analysis more challenging, it requires fairly detailed assumptions about the nature of a decoders assembly process itself. It is not currently known, for example, how the resistances, r_j^i, may be distributed for specific types of decoder. To cope with all of the above complications, the analysis carried out in subsequent chapters utilizes the binary model of NW control with errors.

3.5 Decoder Analysis Framework

Having established a general-purpose model of how MWs control NWs, we now wish to determine the area required to implement different NW decoders. To accomplish this, subsequent chapters derive bounds on M, the number of MWs required for various stochastically assembled simple and compound NW decoders to be able to address N_A out of N NWs with some desired probability, $1 - \epsilon$. Once a bound on M has been obtained in terms of N, N_A, g (the number of OCs) and ϵ, the bound can be translated into a bound on area using the formula presented in Section 3.5.1.

The area required for a NW crossbar-based memory depends not only on N, M and g, but on the amount of programmable storage required by the ATC. As was explained in Section 3.3.2, this in turn depends on the addressing strategy being employed. Section 3.5.2 describes a number of possible addressing strategies. Chapters 4, 5 and 6 bound the number of MWs required to implement these addressing strategies using randomized-contact, encoded NW and mask-based decoders, respectively. Section 3.5.3 briefly outlines the type of probabilistic analysis these bounds require.

3.5.1 Memory Area Estimate

In the case of a crossbar-based memory, the total area, A_T, required depends on the number of NWs per OC, N, the number of MWs, M, the number of OCs, g, and the size of the ATC. We use the approach of [12] and write:

$$A_T \approx 2\chi\beta + 2\lambda_{meso}^2 g\lceil \log_2 g \rceil + (\lambda_{meso}M + \lambda_{nano}N')^2$$

Here λ_{meso} and λ_{nano} denote the pitch of MWs and NWs respectively, that is, the center-to-center distance between wires. Also, χ denotes the area of a mesoscale memory cell, and β denotes the number bits stored in each dimension of the memory's ATC. In the next subsection, β is given for a variety of addressing strategies.

In above formula, $\chi\beta$ approximates the area required for the ATC's programmable storage along each dimension of the crossbar. $\lambda_{meso}^2 g\lceil \log_2 g \rceil$ approximates the area required for the standard demultiplexer used to individually turn on OCs along each dimension of the crossbar. $(\lambda_{meso}M + \lambda_{nano}N')^2$ approximates the area occupied by the N'-by-N' NW crossbar with M MWs along each dimension, where $N' = gN$. The storage capacity of the memory, N_A^2, depends on the addressing strategy being employed.

3.5.2 Memory Addressing Strategies

In this subsection we define a number of addressing strategies, that is, ways of using the ATC to map a B-bit external binary address, $\boldsymbol{E}$, to an activation pattern, $\boldsymbol{a}$ and a particular OC, denoted σ. For each addressing strategy, we note the amount of programmable storage it requires. The strategies we define are by no means exhaustive. It is interesting that such a wide range of possibilities exists.

All Wires Addressable

Here we choose M so that, with probability at least $1 - \epsilon$, all NWs in every contact group are individually addressable. If we assume that the number of NWs in each contact group is 2^k, we can simply use the first $b - k$ bits of $\boldsymbol{E}$ to select σ. This fixed mapping does not depend on the particular NW codewords that are present, although the mapping of $\boldsymbol{E}$ to $\boldsymbol{a}$ does. To execute the second mapping, the ATC stores each NW codeword that is present in a lookup table. This requires $N_a'M$ bits of storage where N_a' is the number of addressable NWs in the decoder.

All Wires Almost Always Addressable

Here we choose M so that with probability at least $1-\epsilon$, all NWs in nearly all contact groups are addressable. Contact groups in which not all NWs are addressable are not used. Since the particular contact groups that are not used will vary from decoder to decoder, the ATC cannot use a fixed mapping from $\boldsymbol{E}$ to contact groups σ. Instead, a lookup table is used to obtain an integer to be added to the first $b-k$ bits of $\boldsymbol{E}$ so that it corresponds to the proper contact group. Let g be the number of contact groups and g' be the number for which all NWs are addressable. Then $g-g'$ is an upper bound on the values in the table. We also use a lookup table to map $\boldsymbol{E}$ to $\boldsymbol{a}$. The two tables combined require approximately $g'\lceil\log_2(g-g')\rceil + N'_a M$ bits.

Half of Wires Addressable

The previous two addressing strategies require that all N NWs be individually addressable in all or almost all, contact groups, with probability at least $1-\epsilon$. The ATC required to implement these two addressing strategies requires $N'_a M$ and $g'\lceil\log_2(g-g')\rceil + N'_a M$ bits per address, respectively. As an extension of the two strategies, we can require that at least αN NWs be individually addressable, for some $\alpha < 1$. If more than αN NWs are addressable within a particular contact group, only αN are used. Since the number of addresses per contact group is still fixed, the amount of storage required by the ATC is unchanged.

Take What You Get (TWYG)

In this addressing strategy, the number of addresses per OC is no longer held fixed. Instead we simply choose M so that a fixed fraction of all NWs are individually addressable with probability at least $1-\epsilon$. In this case, some contact groups may have all NWs addressable, but some may not. Since the number of addressable NWs per contact group varies, we can no longer map fixed blocks of binary memory addresses to a particular contact group. Instead, we store a value of σ and $\boldsymbol{a}$ for each addressable NW. This requires $N'_a(\lceil\log_2 g\rceil + M)$ bits.

All Present

Here we choose M to be small enough relative N such that all codewords are present at each OC with probability at least $1-\epsilon$. This has the advantage of eliminating the need for ATC, but since each address will likely be present multiple times, most NWs will not be individually addressable. Also, this strategy only applies to decoders (such as the encoded NW decoder) in which each codeword can be guaranteed to be individually addressable.

Address Sets Across Groups (ASAG)

Here a fixed set of individually addressable codewords, C, is preselected, then M is chosen so that each codeword in C appears, and is individually addressable, at least t times across all g ohmic contacts. Here t should be chosen such that $t|C| \geq N'_a$. If we assume $|C|$ is a power of 2, this strategy allows for a fixed mapping between the lower order bits of $\boldsymbol{E}$ and codewords in C (and thus $\boldsymbol{a}$). The high order of bits of $\boldsymbol{E}$ are then mapped to values of σ using $N'_a(\lceil\log_2 g\rceil)$ bits.

3.5.3 Expectation versus with High Probability

A simple NW decoder consists of N NWs connected to a single OC, controlled by M MWs. When analyzing a simple decoder, in order to determine the number of MWs required for the addressing strategies defined in the previous subsection, there are several questions it may make sense to ask.

- How large must M be so that all NWs are individually addressable?
- How large must M be so that all NWs are fully addressable?
- How large must M be so that at least d NWs are individually addressable? fully addressable?
- How large must M be so that at least d disjoint groups of NWs can be addressed.
- How large must M be so that there exists a group of d NWs such that all subsets of the d NWs can be addressed (this is of interest if the decoder is being used to supply inputs to a circuit).

Since NW decoders are assembled stochastically, each of the above conditions can only be satisfied with some (ideally high) probability. Hence any bounds on M must be given in terms of their probability of failure. In order to compute these bounds on M, we must identify, for a particular type of NW decoder, the probability that a given NW will be individually addressable. This, in turn, can be bounded by the probability that any two NWs are **independently controllable**, meaning each can be turned off while the other remains on. As we show in the subsequent chapters, once this probability has been identified, various bounds on M can be determined.

Now consider a compound NW decoder consisting of g ohmic contacts arranged in parallel. This can be viewed as g independently assembled simple decoders that share the same set of M MWs. This gives rise to some potentially more complex questions. To suggest a few:

1. How large must M and g be to ensure that some total number of NWs are individually addressable
2. How large must M be so that at least d NWs are individually addressable in all g OCs.
3. How large must M be so that in most OCs, at least d NWs are individually addressable.
4. How large must M and g be so that each codeword appears at least d times across all ohmic contacts.

Which of these questions should be asked depends on the addressing strategy being considered. One general observation, however, is that when many contact groups are considered, it makes sense not only to give bounds in high probability, but also bounds in expectation. In other words, if we can bound M such that the average number of individually addressable NWs per OC is large, then when g is large, we translate this into a bound on the total number of addressable NWs across all OCs.

Chapter 4

The Randomized-Contact Decoder

In this chapter we analyze the randomized-contact decoder. The term "randomized-contact decoder" (RCD) refers to any stochastically-assembled NW decoder in which NW/MW junctions can be modeled as identically distributed independent random variables (see Figure 4.1 as well as Section 3.2.3). More formally, when an RCD is assembled, each NW/MW junction becomes controlling with probability p, noncontrolling with probability q and in error with probability $r = 1-p+q$ (see Section 3.4.1 for a precise definition of "controlling", "noncontrolling" and "in error"). We note that this model of an RCD, first analyzed in [59], is a very practical generalization of the earlier error-free model presented in [55].

In this chapter, we use this binary model with errors to analyze the relationship between M, the number of MWs, and N_A, the number of individually addressable NWs in a simple RCD (an RCD in which all NWs are connected to a single OC). We then build on these results to determine the area required to implement a number of the addressing strategies defined in Section 3.5.2 using a compound RCD comprised of g OCs.

In [55], Hogg *et al* first explored the conditions under which all of the N NWs in an error-free simple RCD can be addressed by M MWs. They observe through simulation and asymptotic analysis that when M passes a threshold of approximately $4.8 \log_2 N$, the probability that all N NWs are individually addressable grows rapidly as N increases. Their asymptotic analysis is in agreement with Corollary 4.1.1 below, but it does not make explicit the dependence of M on the probability, ϵ, of failing to have all NWs be individually addressable. It also fails to capture the impact of manufacturing errors (i.e. it assumes $r = 0$).

In contrast, [56] (as well as [59] and [60]) bounds M in terms of N, ϵ, p, q and r. These bounds are presented in this chapter. In Section 4.1 M is bounded such that all NWs are individually addressable with probability $1 - \epsilon$. In Section 4.2 M is bounded such that some fixed fraction of all NWs are individually addressable. Both bounds are used in the decoder area analysis presented in Section 4.3. A concluding summary of the chapter is given in Section 4.4.

4.1 Bounds Using Inclusion-Exclusion

In this section we derive upper and lower bounds on the number of MWs, M, required for all N NWs in an RCD to be individually addressable. These bounds are based on the principle of

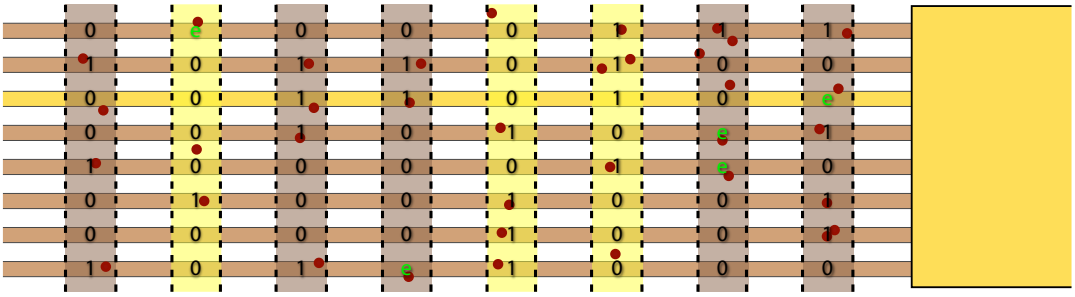


Figure 4.1: A randomized contact decoder in which random particle deposition causes each MW to control certain NWs. In this way, each junction behaves like an independent random variable. Each symbol of each nanowire codeword is thus generated independently at random. Errors (indicated in green) can occur when a particle is misaligned with a particular junction. Fortunately these errors can be masked when junctions that are not in error are used to reliably turn off each NW that is not being addressed.

inclusion-exclusion (stated below).

4.1.1 A Single Contact Group

Given n events, $E_1, E_2, \ldots, E_n$, the **principle of inclusion-exclusion** states that

$$\sum_{i=1}^{n} P(E_i) - \frac{1}{2}\sum_{i \neq j} P(E_i \cap E_j) \leq P(E_1 \cup E_2 \cup \ldots \cup E_n) \leq \sum_{i=1}^{n} P(E_i)$$

In our case each E_i denotes the event that a particular pair of NWs is independently controllable, meaning neither NW's codeword possibly implies the other's.

Theorem 4.1.1 *In a simple RCD, consisting of a single OC and M NWs, let ϵ be the probability that all N NWs are not individually addressable. Then ϵ satisfies the following bounds,*

$$Q(1 - Q/2) - \Delta \leq \epsilon \leq Q$$

where $Q = N(N-1)(1-pq)^M$, $\Delta = 2N(N-1)(N-2)(\mu_3^M + \mu_5^M - 2\mu_1^{2M})$ and $\mu_1 = (1-pq)$, $\mu_3 = (1 - pq(p+2q))$, and $\mu_5 = (1 - pq(2p+q))$.

Proof Let $E_{a,b}$ (where $a \neq b$) be the event that $\boldsymbol{c^a}$ possibly implies $\boldsymbol{c^b}$. Recall from Section 3.4.1 that all NWs are individually addressable if no event $E_{a,b}$ occurs. Thus probability that not all NWs are individually addressable is

$$\epsilon = P(\bigcup_{(a,b)} E_{a,b})$$

where the union is over all pairs of NWs. By expressing ϵ as a union, we can now use inclusion-exclusion to obtain a bound.

Codeword $\boldsymbol{c^i}$ possibly implies $\boldsymbol{c^k}$ if there is no j such that $c^i_j = 1$ and $c^k_j = 0$. $P(E_{a,b}) = (1-pq)^M$. Let $Q = \sum_{a \neq b} P(E_{a,b})$. Since a and b can both take values from 1 to N, we have

$$\epsilon = P(\bigcup_{(a,b)} E_{a,b}) \leq Q = N(N-1)(1-pq)^M$$

This serves as the inclusion portion of the inclusion-exclusion bound on ϵ. For the exclusion portion of the bound, We must now bound $\sum_{(a,b)\neq(c,d)} P(E_{a,b} \cap E_{c,d})$. Here $1 \leq a, b, c, d \leq N$ provided that $(a,b) \neq (c,d)$, i.e. either $a \neq b$ or $c \neq d$ or both.

To compute $P(E_{a,b} \cap E_{c,d})$, we consider 3 cases:

In case (1), a, b, c and d are all different. There are $N(N-1)(N-2)(N-3)$ ways of selecting them. Since $E_{a,b}$ and $E_{c,d}$ are independent, $P(E_{a,b} \cap E_{c,d}) = P(E_{a,b})P(E_{c,d}) = \mu_1^{2M}$, where $\mu_1 = (1-pq)$.

In case (2), two of the four variables are equal. Here either $a = c$, $a = d$, $b = c$ or $b = d$. As stated earlier, we do not allow $a = b$ or $c = d$. There are $N(N-1)(N-2)$ ways to choose indices in each case. These cases are considered below.

In case (3), there are only two different values for a, b, c, and d. Since $(a,b) \neq (c,d)$, $a = d$ and $b = c$, which can occur in $N(N-1)$ ways. Here $P(E_{a,b} \cap E_{c,d}) = P(E_{a,b} \cap E_{b,a})$, which is the probability that, for no j is $c^a_j = 0$ and $c^b_j = 1$, or $c^a_j = 1$ and $c^b_j = 0$. So $P(E_{a,b} \cap E_{b,a}) = \mu_2^M$ where $\mu_2 = (1-2pq)$.

Returning to case 2, we have four subcases to consider.

Let $F_{a,b}(m)$ be the event that $c^a_m = 0$ and $c^b_m = 1$. Let $E_{a,b}(m)$ be the complement of $F_{a,b}(m)$. Since the probability of $F_{a,b}(m)$ is pq, it follows that the probability of event $E_{a,b}(m)$ is $P(E_{a,b}(m)) = 1 - pq$. Since the event $E_{a,b}$ is $\prod_m E_{a,b}(m)$, $P(E_{a,b}) = \mu_1^M$.

1. $\boldsymbol{n_a = n_c}$. $F_{a,b}(m) \cup F_{a,d}(m)$ occurs only if $(c_{a,m}, c_{b,m}, c_{d,m})$ assumes the value $(0,1,0)$, $(0,1,1)$, or $(0,0,1)$. Thus, $P(F_{a,b}(m) \cup F_{a,d}(m)) = pq(p+2q)$ and $P(E_{a,b} \cap E_{c,d}) = \mu_3^M$ where $\mu_3 = (1 - pq(p+2q))$.

2. $\boldsymbol{n_a = n_d}$. Thus, $F_{a,b}(m) \cup F_{c,a}(m)$ occurs if $(c_{a,m}, c_{b,m}, c_{c,m})$ assumes the value $(0,1,0)$, $(0,1,1)$, $(1,1,0)$, or $(1,0,0)$. Thus, $P(F_{a,b}(m) \cup F_{c,a}(m)) = 2pq(p+q)$ and $P(E_{a,b}) \cap E_{c,a}) = \mu_4^M$ where $\mu_4 = (1 - 2pq(p+q))$.

3. $\boldsymbol{n_b = n_c}$. Thus, $F_{a,b}(m) \cup F_{b,d}(m)$ occurs if $(c_{a,m}, c_{b,m}, c_{d,m})$ assumes the value $(0,1,0)$, $(0,1,1)$, $(0,0,1)$, or $(1,0,1)$. Thus, $P(F_{a,b}(m) \cup F_{c,b}(m)) = 2pq(p+q)$ and $P(E_{a,b}) \cap E_{b,d}) = \mu_4^M$.

4. $\boldsymbol{n_b = n_d}$. Thus, $F_{a,b}(m) \cup F_{c,b}(m)$ occurs if $(c_{a,m}, c_{b,m}, c_{c,m})$ assumes the value $(0,1,0)$, $(0,1,1)$, or $(1,1,0)$. Thus, $P(F_{a,b}(m) \cup F_{c,b}(m)) = pq(2p+q)$ and $P(E_{a,b}) \cap E_{c,a}) = \mu_5^M$ where $\mu_5 = (1 - pq(2p+q))$.

Let $D = \sum_{(a,b)\neq(c,d)} P(E_{a,b} \cap E_{c,d})$. Then,

$$D/(N(N-1)) = (N-2)(N-3)\mu_1^{2M} + \mu_2^M + (N-2)\left(\mu_3^M + 2\mu_4^M + \mu_5^M\right)$$

where $\mu_1 = (1-pq)$, $\mu_2 = (1-2pq)$, $\mu_3 = (1-pq(p+2q))$, $\mu_4 = (1-2pq(p+q))$, and $\mu_5 = (1-pq(2p+q))$.

The behavior of D is dominated by the largest term μ_i^M. Note that $\mu_2 \leq \mu_1^2$ and $\mu_4 \leq \min(\mu_3, \mu_5) \leq (\mu_3+\mu_5)/2$. It follows that $(N-2)(N-3)\mu_1^{2M} + \mu_2^M \leq ((N-2)(N-3)+1)\mu_1^{2M} \leq N(N-1)\mu_1^{2M} - 4(N-2)\mu_1^{2M}$ and $(\mu_3^M + 2\mu_4^M + \mu_5^M) \leq 2(\mu_3^M + \mu_5^M)$. Thus, D satisfies the following bound.

$$D \leq Q^2 + 2N(N-1)(N-2)\left(\mu_3^M + \mu_5^M - 2\mu_1^{2M}\right)$$

The lower bound to ϵ follows directly from the above. ■

Theorem 4.1.1 implies upper and lower bounds on M in terms of N and ϵ. For the cases examined below, when $p = q$ and ϵ is small, these bounds are tight, meaning the upper and lower bounds they imply on M agree. Slightly weaker, but simpler bounds, are given in the following corollary, in which upper and lower bounds on M differ by $\ln(2)/\ln(1-pq)$.

Corollary 4.1.1 *In a simple RCD, consisting of a single OC and N NWs, let M be the minimum number of MWs such that all NWs are individually addressable with probability* $1-\epsilon$. *Then M satisfies the following bounds,*

$$\frac{\ln(N(N-1)/2\epsilon)}{-\ln(1-pq)} \leq M \leq \frac{\ln(N(N-1)/\epsilon)}{-\ln(1-pq)}$$

where the lower bound holds when $q > r$, $p > r$, $\epsilon \leq 1/6$, *and the true value of M is itself at least* $M \geq \ln(\epsilon/4N)/\ln[1 - pq\min(q-r, p-r)/(1-pq)]$.

Proof The upper bound on M,

$$M \leq \frac{\ln(N(N-1)/\epsilon)}{-\ln(1-pq)}$$

follows directly from the righthand side of Theorem 4.1.1, $\epsilon \leq Q$, where $Q = N(N-1)(1-pq)^M$.

For the lower bound, consider the lefthand side of Theorem 4.1.1, $Q(1-Q/2) - \Delta \leq \epsilon$, where $\Delta = 2N(N-1)(N-2)(\mu_3^M + \mu_5^M - 2\mu_1^{2M})$, $\mu_1 = (1-pq)$, $\mu_3 = (1-pq(p+2q))$, and $\mu_5 = (1-pq(2p+q))$. In Δ, $(\mu_3^M + \mu_5^M - 2\mu_1^{2M})^M$ can be replaced with a larger quantity, $2\max(\mu_3, \mu_5)^M$.

Since $\mu_3 = (1-pq(1-r+p)) = \mu_1 - pq(p-r)$ and $\mu_5 = (1-pq(1-r+q)) = \mu_1 - pq(q-r)$, this gives $Q(1-Q/2) - 4N(N-1)(N-2)(\mu_1 - \min(pq(q-r), pq(p-r)))^M \leq \epsilon$, or

$$Q(1 - Q/2 - 4N(1 - pq\min(q-r, p-r)/\mu_1)^M) \leq \epsilon$$

To continue, we require $q > r$ and $p > r$, which allows us to further require that $M \geq \ln(\epsilon/4N)/\ln[1 - pq\min(q-r, p-r)/\mu_1]$. Since $\epsilon \leq Q$, this implies that

$$Q(1 - (3/2)Q) \leq Q(1 - Q/2 - \epsilon) \leq \epsilon$$

Finally, notice that if $\epsilon \leq 1/6$, the above inequality implies that $Q \leq 1/3$, and so

$$Q \leq 2\epsilon$$

Thus, since $M = \ln(N(N-1)/Q)/-\ln(1-pq)$, we have $M \geq \ln(N(N-1)/2\epsilon)/-\ln(1-pq)$, the desired lower bound. ■

4.1.2 Multiple Contact Groups

We now use Corollary 4.1.1 to obtain upper bounds on the number of MWs in compound RCDs with g OCs.

Corollary 4.1.2 *In a compound RCD with g OCs, N NWs per contact group, and $N' = gN$ NWs total, all NWs are individually addressable with probability $(1-\epsilon)$ if*

$$M \geq \frac{\ln(N'(N-1)/\epsilon)}{-\ln(1-pq)}$$

Proof Let δ be the probability of failure of all NWs in a contact group to be individually addressable. Then, the probability that one or more contact groups fails to have all its NWs be individually addressable is at most $g\delta$. If $g\delta \leq \epsilon$, the probability that all N' NWs are addressable is at least $1-\epsilon$. We use the upper bound on M given in Corollary 4.1.1 when N is replaced by N'/g and ϵ by ϵ/g. ■

The utility of Corollary 4.1.2 and Corollary 4.1.1 are illustrated by the following two examples.

Example 4.1.1 *To bound the area required to control a crossbar-based memory using a compound RCD, we wish to bound the number of MWs required to implement the "All Wires Addressable" addressing strategy (see Section 3.5.2).*

In a compound RCD with $g = 128$ OCs, $N = 8$ NWs per OC and $N' = 8 * 128 = 1,024$ total NWs, Corollary 4.1.2 asserts that all N' NWs are individually addressable with probability $1-\epsilon = .99$ or better when $M \geq 47$.

What's more, evaluating Theorem 4.1.1 numerically for these values of N and M shows this threshold value of M is exact.

Example 4.1.2 *In the previous example, the number of MWs can be reduced if we don't require that all NWs in each contact group be individually addressable. As an alternative, we can implement the "Almost All Wires Addressable" addressing strategy (see Section 3.5.2) in which all NWs are individually addressable in almost all (as opposed to all) contact groups.*

Corollary 4.1.1 says that a failure rate of at most $\epsilon = .01$ can be achieved with a simple RCD when $p = q = .5$ and $N = 8$ if $M \geq 30$ (as noted in the above example, this threshold value of M is in fact exact). The number of individually addressable NWs in each OC is statistically independent. If all N NWs in a particular contact group are individually addressable with probability $1-\epsilon$, the probability that f or fewer contact groups fail to have all NWs addressable is $\phi(\epsilon, f, g) = \sum_{i=0}^{f} \binom{g}{i} \epsilon^i (1-\epsilon)^{g-i}$.

Let $\epsilon = .01$, $g = 133$ and $f = 5$. Because $\phi(.01, 5, 133) \geq .99$, at least 128 of $g = 133$ OCs have all NWs addressable with probability 0.99. Thus when $M = 30$, $g = 133$, and $N = 8 * 133 = 1064$, $N'_a = 8 * 128 = 1,024$ NWs are individually addressable with probability 0.99.

The Impact of Errors

The bounds on M given in Corollary 4.1.1 and Corollary 4.1.2 are proportional to $\alpha = -1/\ln(1-pq)$. This quantity is maximized when $p = q = 1/2$, in which case $\alpha = 1/\ln(4/3)$. In an error-free RCD,

$r = 1 - p - q = 0$, but if errors occur $p + q < 1$, in which case $pq < .25$. Thus as r increases α must also increase. For example, if $pq = .2$ then $\alpha = 1/\ln(5/4)$, in which case the bound on M grows by a factor of $\ln(4/3)/\ln(5/4) = 1.29$. If $pq = .1$, the factor is $\ln(4/3)/\ln(10/9) = 2.73$. Even for relatively high error rates, M is not prohibitively large.

Additional Fault-Tolerance

Corollary 4.1.1 and Corollary 4.1.2 (as well as Corollary 4.2.2 below) show that logarithmic number of MWs can reliably address many NWs in an RCD even when some fraction of MW/NW junctions are in error. As noted at the end of Section 3.4.1, an additional level of fault-tolerance is provided if the symmetric distances between NWs is greater than 1. In other words, if M is increased beyond the bounds given in Corollary 4.1.1 or Corollary 4.1.2, not only can all NWs be guaranteed to be individually addressable, but the unaddressed NWs can be guaranteed to be turned off by multiple activated MWs.

If each unaddressed NW is turned off by multiple controlling MWs, the on/off ratio of addressed to non-addressed NWs will be larger (see the end of Section 3.4.2). This in turn allows for faster and more reliably decoders. It also allows the decoder to cope with transient faults in which a MW fails to adequately control a NW [57, 58, 44]. In the case of an RCD, where each bit of each codeword is generated independently at random, as M increases, the average symmetric distance between codewords approaches Mpq.

4.2 Bounds Using Expectation

In this section we derive bounds on M in terms of N and N_A based on the expected number of individually addressable NWs in simple and compound RCDs.

4.2.1 A Single Contact Group

First we bound the expected number of individually addressable NWs, $E[N_A]$, in a single contact group.

Theorem 4.2.1 *In a simple RCD, consisting of a single OC and M NWs, let N_a be the number of individually addressable NWs. The expected number of individually addressable NWs, $E[N_a]$, is bounded as follows.*

$$N(1 - (N-1)(1-pq)^M) \leq E[N_a] \leq N(1 - (N-1)(1-pq)^M + \Delta)$$

where $\Delta = (1/2)N(N-1)(1 - 2pq + qp^2)^M$.

Proof Let $x_i = 1$ if NW $\boldsymbol{n_i}$ is individually addressable and 0 otherwise. Since $N_a = \sum_{i=1}^{N} x_i$, we have

$$E[N_a] = \sum_{i=1}^{N} E[x_i] = NE[x_1] = NP(x_1 = 1)$$

since the x_i are all identically distributed 0-1 random variables.

Let $E_{k,i}$ be the event that $\boldsymbol{c^k}$ "possibly implies" $\boldsymbol{c^i}$. (See Chapter 3 Section 3.4.1) $P(x_1 = 1) = 1 - P(x_1 = 0) = 1 - P(E_{2,1} \cup E_{3,1} \cup \ldots \cup E_{N,1})$.

By inclusion-exclusion we have

$$\sum_{k=2}^{N} P(E_{k,1}) - (1/2) \sum_{k \neq l}^{N} P(E_{k,1} \cap E_{l,1}) \leq P(E_{2,1} \cup E_{3,1} \cup \ldots \cup E_{N,1}) \leq \sum_{k=2}^{N} P(E_{k,1})$$

Since $P(E_{2,1}) = P(E_{3,1}) = \ldots = P(E_{N,1})$, and similarly $P(E_{k,1} \cap E_{l,1}) = P(E_{2,1} \cap E_{3,1})$ for all $k \neq l$,

$$1 - (N-1)P(E_{2,1}) \leq P(x_1 = 1) \leq 1 - (N-1)P(E_{2,1}) + (1/2)N(N-1)P(E_{2,1} \cap E_{3,1})$$

$\boldsymbol{c^2}$ possibly implies $\boldsymbol{c^1}$ if for all $1 \leq j \leq M$ it is not the case that both $c_j^1 = 0$ and $c_j^2 = 1$, thus $P(E_{2,1}) = (1 - pq)^M$. Similarly $\boldsymbol{c^2}$ and $\boldsymbol{c^3}$ possibly implies $\boldsymbol{c^1}$ if for all $1 \leq j \leq M$ it is not the case that (c_j^1, c_j^2, c_j^3) take values $(0,1,0), (0,1,e), (0,1,1), (0,e,1)$ or $(0,0,1)$, thus $P(E_{2,1} \cap E_{3,1}) = (1 - 2qp + qp^2)^M$. This gives

$$1 - (N-1)(1-pq)^M \leq P(x_1 = 1) \leq 1 - (N-1)(1-pq)^M + (1/2)N(N-1)(1 - 2pq + qp^2)^M$$

■

We note that in [56] (as well as [59] and [60]), Theorem 4.2.1 is given with the weaker upper bound $E[N_a] \leq N(1 - (1-pq)^M)$.

4.2.2 Multiple Contact Groups

Theorem 4.2.1 can be extended to compound RCDs with g OCs using the following corollary.

Corollary 4.2.1 *In a compound RCD with g OCs, N NWs per contact group, and $N' = gN$ NWs total, let N'_a be the total number of individually addressable NWs across all g contact groups. The expected number of individually addressable NWs, $E[N'_a]$, obeys the following bounds.*

$$N'(1 - (N-1)(1-pq)^M) \leq E[N'_a] \leq N'(1 - (N-1)(1-pq)^M + \Delta)$$

where $\Delta = (1/2)N(N-1)(12pq + qp^2)^M$.

Proof N'_a is the sum of the number of individually addressable NWs, N_A, in each contact group. Since each contact group has N NWs, $E[N'_a] = gE[N_a]$. Substituting the bounds from Theorem 4.2.1 yields the desired result. ■

Of course Corollary 4.2.1 only gives the expected number of addressable NWs across g contact groups. Really we are interested in the number that can be guaranteed with high probability. To obtain such a bound, we use Hoeffding's inequality.

Let $S = n_1 + n_2 + ... + n_t$ be the sum of t independent random variables, where each n_i ranges from a_i to b_i. **Hoeffding's Inequality** [61, p. 303] states that

$$P\left(E[S] - S \geq d\right) \leq e^{-2d^2/\sum c_i^2}$$

where $c_i = b_i - a_i$, and $d \geq 0$. We use this to bound the total number of individually addressable NWs with high probability.

Theorem 4.2.2 *Let N'_a be the total number of addressable NWs in a NW decoder with g contact groups, N NWs per contact group, and $N' = gN$ NWs in total.*

$$P(N'_a \leq E[N'_a] - N'k) \leq e^{-2k^2N'N/(N-1)^2} = e^{-2k^2g^*}$$

for any $k \geq 0$ where $g^* = g(N/(N-1))^2$.

Proof In Hoeffding's Inequality, let $t = g$, $d = N'k$, $S = N'_a$ and $c_i = (N-1)$. This gives $P(E[N'_a] - N'_a \geq N'k) \leq e^{-2(N'k)^2/g(N-1)^2} = e^{-2k^2N'N/(N-1)^2}$. We can then rewrite $P(E[N'_a] - N'_a \geq N'k)$ as $P(N'_a \leq E[N'_a] - N'k)$. ■

Theorem 4.2.2 is not specific to RCDs. It is applied to RCDs in the following corollary.

Corollary 4.2.2 *Let N'_a be the total number of addressable NWs in an RCD with g contact groups, N NWs per contact group, $N' = gN$ NWs in total and M MWs.*

$$P(N'_a > \kappa N') \geq 1 - \epsilon$$

if $\kappa \leq 1 - \sqrt{-\ln \epsilon/(2g^*)} - (N-1)(1-pq)^M$ where $g^* = g(N/(N-1))^2$.

Proof From Corollary 4.2.1 we have $E[N'_a] \geq N'(1-(N-1)(1-pq)^M)$ and by the above theorem,

$$P(N'_a \leq N'(1-(N-1)(1-pq)^M) - N'k) \leq e^{-2k^2g^*}$$

Thus, if $k = (1-(N-1)(1-pq)^M) - \kappa$, then

$$P(N'_a \leq \kappa N') \leq e^{-2g^*(1-(N-1)(1-pq)^M-\kappa)^2}$$

Thus, when $e^{-2g^*(1-(N-1)(1-pq)^M-\kappa)^2} \leq \epsilon$ the desired conclusion follows. This occurs when $\ln \epsilon \geq -2g^*(1-(N-1)(1-pq)^M - \kappa)^2$ or $\sqrt{-\ln \epsilon/(2g^*)} \leq (1-\kappa) - (N-1)(1-pq)^M$. ■

Example 4.2.1 *Corollary 4.2.2 is useful in bounding the number of MWs required to implement the "Take What You Get" (TWYG) addressing strategy (see Section 3.5.2) in which a significant fraction of all NWs are individually addressable. As an example, suppose $p = q = 1/2$, $g = 175$, $N = 8$, $N' = 1400$, $\epsilon = .01$, and $\kappa = .733$. When $M = 13$, $\kappa = .733 \leq 1 - \sqrt{-\ln .01/(2 * 175 * (8/7)^2)} - 7 * (3/4)^{13}$. Thus at least $\lceil .733 * 1400 \rceil = 1027$ NWs are addressable with probability .99.*

The Impact of Errors

As in Corollary 4.1.1 and Corollary 4.1.2, the bound on M implied by Corollary 4.2.2 is proportional to $\alpha = -1/\ln(1-pq)$. If g and N are held constant, but the decoders error rate, r, increases, M must also increase in order to hold the term $(N-1)(1-pq)^M$, and hence κ, constant. In an error-free decoder, $(1-pq)$ is maximized when $p = q = 1/2$, in which case $pq = .25$. If r is increased such that $pq = .2$, M must grow by a factor of $\ln(4/3)/\ln(5/4) = 1.29$. If $pq = .1$, the factor is $\ln(4/3)/\ln(10/9) = 2.73$. Manufacturing errors increase M by only a small constant factor.

4.2.3 Additional Addressing Strategies Using Expectation

Corollary 4.2.2 bounds the total number of individually addressable NWs across g OCs. This is directly applicable to bounding the number of MWs required by the TWYG addressing strategy. Hoeffding's Ineqaulty, can potentially be used to analyze additional addressing strategies.

In the ASAG addressing strategy (see Section 3.5.2), M is chosen so that all of the codewords in some preselected set of codewords, $\mathcal{C}$, are each present and addressable in at least t contact groups. As discussed in Chapter 5, this strategy is well-suited to encoded NW decoders, where $\mathcal{C}$ can simply be the set of all codewords. This is not viable for an RCD, since some codewords are very unlikely to be individually addressable (those with almost all 1's). As a work around, one could limit $\mathcal{C}$ to contain only codewords with close to the average number of 1's, Mp. More precisely, $\mathcal{C}$ would contain all codewords with $Mp \pm \delta$ 1s, where δ is several standard deviations ($\sqrt{Mpq}$), about the mean. This would ensure that most NWs have codewords in $\mathcal{C}$.

Given $\mathcal{C}$ and M, Hoeffding's inequality can be used to to bound the probability that each codeword in C is individually addressable in at least t OCs. As evident from the discussion in the proof of Theorem 4.2.1, a NW with at most $Mp+\delta$ 1's is individually addressable with probability at least $1-(N-1)(1-q)^{M-Mp-\delta}$. This in turn provides a bound on the probability that any given codeword in C appears and is individually addressable in a given OC. Hoeffding's inequality, or alternatively a Chernov bound (see Chapter 5), can then be used to bound the number of contact groups, t, in which each NW in $\mathcal{C}$ can be guaranteed to appears with probability $\epsilon/|C|$ (for additional details, see the corresponding analysis of the ASAG addressing strategy in Chapter 5, Section 5.2).

4.3 Comparison of Addressing Strategies

To compare addressing strategies, we estimate their area when used to produce a memory with a given storage capacity. In our comparison, we fix ϵ, the probability of failure, and N, the number of NWs per contact group. Given these values, we would ideally like to also fix N'_a, the number of addresses along each dimension of the crossbar, then estimate A_T for all three strategies (see Section 3.5.1). Unfortunately, for a given strategy, it is difficult to choose M and N' to yield an exact value for N'_a. In all three cases below we choose M and N' so that close to $1,024$ NWs are individually addressable along each dimension.

To compare addressing strategies, we draw on the examples of the previous two sections and consider the case when $p = q = 1/2$.

- **All Wires Addressable**:

  Here $M = 47$, $g = 128$, and $N' = N'_a = 1024$ with probability at least .99. The ATC requires $\beta = N'_a M = 47,990$ bits. This gives

$$A_T \approx 95,982\chi + \lambda_{meso}^2 1,792 + (\lambda_{meso} 49 + \lambda_{nano} 1,600)^2$$

- **All Wires Almost Always Addressable**:

  Here $M = 30$, $g = 133$, and $N' = 1,064$ yields $N'_a = 1,024$ and $g' = 128$ with probability at least .99. The ATC requires $\beta = g' \lceil \log g - g' \rceil + N'_A M = 31,104$ bits. This gives

$$A_T \approx 62,208\chi + 1,877\lambda_{meso}^2 + (\lambda_{meso} 30 + \lambda_{nano} 1,064)^2$$

- **Take What You Get**:

  Here $M = 13$, $g = 175$ and $N' = 1400$, yields N'_a of 1,027 with probability at least .99. The ATC requires $\beta = N'_a(\lceil \log g \rceil + M) = 21,567$ bits. This gives

$$A_T \approx 43,134\chi + 2,800\lambda_{meso}^2 + (\lambda_{meso} 13 + \lambda_{nano} 1,400)^2$$

Since the parameter χ, the area of a mesoscale memory unit, will be many times λ_{meso}^2, and it is expected that $\lambda_{meso} \geq 10\lambda_{nano}$, the **Take What You Get** addressing strategy is clearly best.

4.4 Summary of Results

In this chapter we have analyzed the area required to control N_A out of N NWs using an RCD. As explained at the beginning of this chapter, the term RCD refers to any stochastically-assembled NW decoder in which NW/MW junctions can be modeled as identically distributed independent random variables. As such, RCDs can potentially be realized through a wide range of assembly techniques (see Section 3.2.3 for several examples). Importantly, our model of RCDs accounts for the possibility of manufacturing errors, meaning some MWs may provide only partial control over some NWs. In our model each NW/MW junction is controlling with probability p, noncontrolling with probability q and in error with probability $r = 1 - p - q$.

The analysis of Sections 4.1 and 4.2 demonstrates that RCDs remain efficient in their use of MWs even when manufacturing errors occur. In Section 4.1, Corollary 4.1.1 reveals that in a simple RCD with M MWs and a single OC connected to N NWs, all N NWs are individually addressable with probability at least $1 - \epsilon$ when $M = -\ln(N(N-1)/\epsilon)/\ln(1-pq)$. If we set $p = q = 0.5$, this gives $M \approx 3.5 \ln(N(N-1)/\epsilon)$. If we assume $p = q = 0.45$, and $r = 0.1$, the bound becomes $M \approx 4.4 \ln(N(N-1)/\epsilon)$. This is a very modest increase in area given a 10% error rate.

Section 4.2 shows that the number of required MWs can be reduced further if we eliminate the requirement that all NWs connected to all (or almost all) OCs be individually addressable. In a compound RCD comprised of M MWs, g OCs, and N NWs per OC, Corollary 4.2.2 bounds the number of MWs required for $\kappa N'$ of the $N' = gN$ total NWs to be individually addressable with probability at least $1-\epsilon$. As g increases, our bound on M approaches $\ln((1-\kappa)/(N-1))/\ln(1-pq)$.

As demonstrated in Section 4.3, this reduced value of M leads to a reduction in the area required to construct a NW crossbar-based memory with 1Mb of storage. Furthermore, this more lenient condition on M once again increases by only a small constant factor when manufacturing errors occur. As such, it is perfectly reasonable to consider constructing RCDs with fewer than 15 MWs, even in the presence of errors, capable of individually addressing most NWs. A comparison between RCDs, encoded NW decoders and mask-based decoders is given at the end of Chapter 6.

Chapter 5

Encoded Nanowire Decoders

In this chapter we analyze the encoded nanowire decoder. The term "encoded nanowire decoder" refers to stochastically-assembled NW decoders constructed from differentiated NWs (see Section 3.2.1). To produce such a decoder, many differently encoded NW types are grown off chip, then many copies of each NW type are placed in a large ensemble. From the ensemble, random subsets of N NWs are collected, deposited fluidically, and connected to OCs. Fluidic assembly can ensure that the NWs are deposited in parallel, but it cannot guarantee that their endpoints are aligned, nor can it control how NWs are ordered [6]. Once NWs are deposited, the MWs that control a particular NW are determined by that NW's encoding. This is in contrast with randomized-contact and mask-based decoders, in which N identical NWs are deposited, and then M MWs are coupled to NWs via a stochastic process.

As with randomized-contact decoders in Chapter 4, we wish to bound the area required to construct encoded NW decoders using a variety of addressing strategies. This is achieved by bounding both the expected number of individually addressable NWs per contact group, and the probability that all N NWs in a contact group are individually addressable.

In order to calculate these bounds, we must first compute the probability that any two NWs are independently controllable (i.e. that each of the two NWs can be turned off independently of the other). Two NWs are independently controllable if neither NW's codeword implies the other (see Section 3.4.1). The probability distribution from which codewords are assigned, however, depends on how the NWs have been encoded during growth.

Section 5.1 explores how NWs can be encoded, and how the encodings of axially encoded NWs correspond to codewords once these NWs are deposited on chip. After a method of encoding NWs has been selected, it is straightforward to compute the probability that two NWs are independently controllable. This probability is used in Section 5.2 to bound the number of MWs, and thus the area, that encoded NW decoders require to implement various addressing strategies. Encoded NW decoders are quite efficient, but as Section 5.3 explains, misalignment of axially encoded NWs may make the decoders difficult to realize in practice. Section 5.4 demonstrates how radially encoded NWs offer a way of constructing efficient encoded NW decoders while avoiding errors associated with axial misalignment. Section 5.5 provides a concluding summary of this chapter.

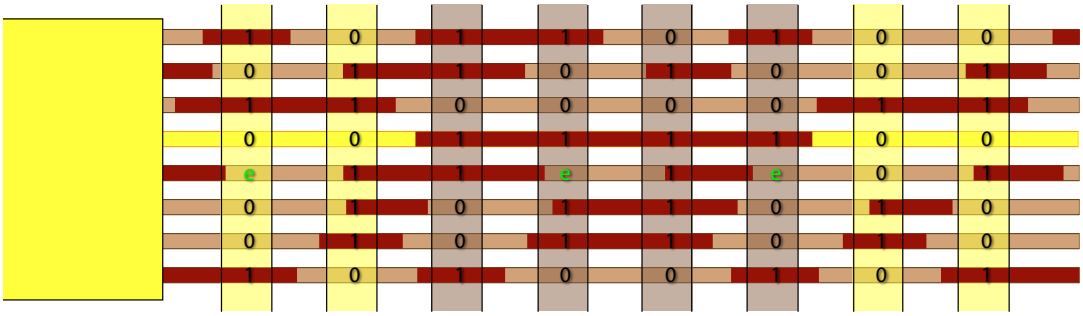


Figure 5.1: An axially encoded NW decoder in which each NW has a sequence of lightly and heavily doped regions along its axis. When a subset of the MWs is activated, all NWs with lightly doped regions under those MWs become nonconducting. To accommodate large amounts of axial shift, the sequences are repeated multiple times along the lengths of NWs. Notice that the lightly doped regions (indicated in dark red) are not guaranteed to align with MWs. Such axial misalignment can cause certain MW/NW junctions to be in error, denoted by e's.

5.1 NW Encodings

As noted in Section 3.2.1, two different methods have been considered for differentiating NWs during manufacture. An axially encoded NW decoder is formed from NWs with patterns of lightly and heavily doped regions along their lengths. A radially encoded NW decoder is formed from NWs with a sequence of shells surrounding their lightly doped cores. Both types of NW can be used to generate a range of possible codewords. The relation of each type of encoding to the codewords it generates is discussed below.

Axially Encoded NW Decoders

Axially encoded NW decoders are formed from modulation-doped NWs. These NWs are grown off chip with a sequence of lightly and heavily doped regions along their length [45, 18], then deposited. When M MWs are placed on top of the NWs, only the MWs that are adjacent to a NW's lightly doped regions control the NW (see Figure 5.1 in which controlling, noncontrolling and partially controlling MW/NW junctions are labelled 1, 0 and e, respectively). When deposited fluidically, it may not be possible to guarantee the alignment of a particular lightly doped region with a particular MW. To accommodate large amounts of random axial displacement of NWs relative to MWs, the same pattern of M lightly and heavily doped regions can be repeated several times along the NW. As a result, the set of codewords axially encoded NWs can generate as they shift is closed under cyclic shift. Sections 5.1.3 and 5.1.2 describe two ways in which axially encoded NWs can be encoded: (h, b)-hot codes and binary reflected codes.

Radially Encoded NW Decoders

Radially encoded NW decoders are formed from core-shell NWs [47]. These NWs consist of a lightly doped core surrounded by thin layers of insulating material, called "shells". As with modulation-

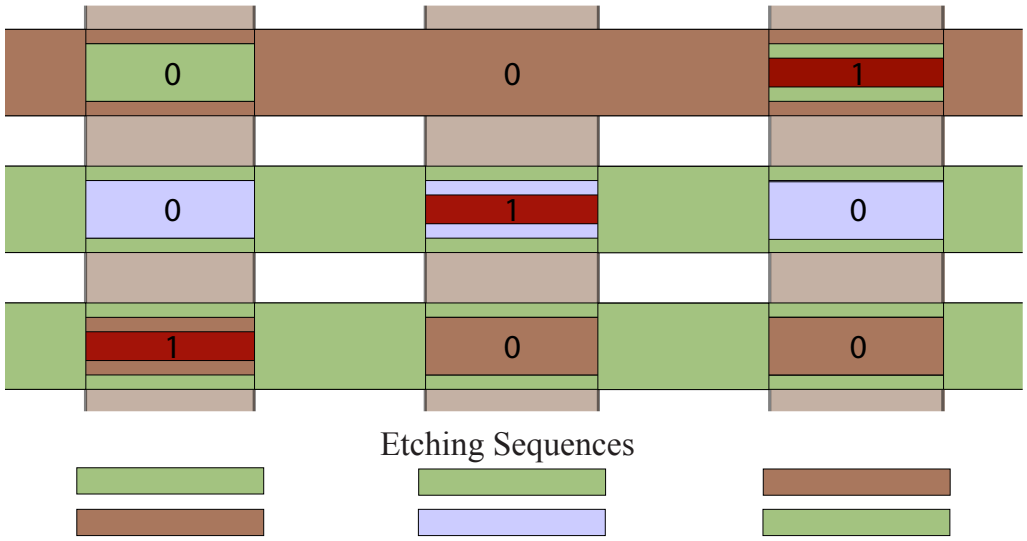


Figure 5.2: Portions of three radially encoded NWs constructed from three different shell materials. Each NW has two shells surrounding a lightly doped core. Under the three MWs, three different etching sequences have been applied. As a result, each of the MWs controls a different NW type.

doped NWs, they are grown off-chip then deposited fluidically. Instead of being grown with different sequences of lightly and heavily doped regions, different NW types are grown with different sequences of shell materials. The ability to grow a single shell around a lightly doped core has already been demonstrated [62]. As explained in Section 5.4, radially encoded NWs can be used to generate the same type of codewords as axially encoded NWs. After radially encoded NWs are deposited on chip, codewords are assigned by etching away different shell materials under each MW (the etching takes place before MWs are laid down). This exposes the lightly doped cores of only certain NWs under each MW, and hence each MW provides control over a different subset of NWs (see Figure 5.2)

5.1.1 Code Requirements

When a differentiated NW is deposited on a chip, its encoding, along with its axial displacement in the case of an axially encoded NW, determines which MWs control it and thus its codeword. In a NW crossbar-based memory, the objective is to have many individually addressable NWs, and hence encodings should be selected such that each potential codeword is individually addressable. In other words, NW encodings should generate codewords that do not imply each other. Also, since the axial displacement of NWs cannot be precisely controlled, axial encodings will be repeated along the length of a NW. This means that the set of codewords they generate must be closed under cyclic shift [45]. In this section we present two families of codes that meet these requirements. As explained, they are natural choices even if the cyclic shift requirement is removed. At the end of this chapter, we demonstrate how the codes apply to radially encoded NWs as well, even though these NWs are not sensitive to shifting.

5.1.2 (h, M)-Hot Codes

Suppose each NW is divided into M regions along its axis with the same center-to-center distance, λ_{meso}, as MWs. An **(h, M)-hot code** [9] is one in which h of M regions are lightly doped (i.e. controllable) and the $(M - h)$ remaining regions are heavily doped (i.e. not controllable). In this code, assuming no misalignment errors occur (see Section 5.3 below) there are $C = \binom{M}{h}$ M-bit codewords, each with h 1's and $(M - h)$ 0's. Since no codeword implies any other codeword, all codewords are individually addressable. Furthermore, the code is closed under cyclic shifts. As a result, the sequence of M regions can be repeated multiple times along the NW. In this way, an axial shift of a NW (relative to the MWs) by λ_{meso} has the effect of cyclically shifting that NW's codeword.

To bound M in terms of C, Consider an (h, M)-hot code where M is even and $h = M/2$. In this case $C = \binom{M}{M/2} = M!/(M/2)!^2$. By Stirling's approximation, which is quite tight, $\ln M! \approx M \ln M - M + \frac{1}{2}\ln(2\pi M)$ and thus $2\ln(M/2)! \approx M \ln M/2 - M + \ln(\pi M)$. This implies that

$$\log_2 C \approx M - \frac{1}{2}\log_2 M - \frac{1}{2}\log_2(\pi/2)$$

To simplify this bound, we observe that $M < 2\log_2 C$ (this is implied by the size of BRC codes, as defined in Section 5.1.3 below), and so $\log_2 M < \log_2 2\log_2 C = \log_2\log_2 C + 1$. This gives

$$M < \log_2 C + \frac{1}{2}\log_2\log_2 C + \frac{1}{2}\log_2 \pi$$

Thus the number of MWs required to generate C individually addressable codewords is very close to $\log_2 C$. We now show that, $(\lceil M/2 \rceil, M)$-hot codes are optimal in their use of MWs.

Lemma 5.1.1 *Let $\mathcal{C} \subseteq \{0,1\}^M$ be a set of codewords which are all individually addressable. If the minimum weight codeword has weight $w < \lfloor M/2 \rfloor$, there exists a code $\mathcal{C}'$, such that $|\mathcal{C}'| > |\mathcal{C}|$, all codewords in $\mathcal{C}'$ have weight at least $w + 1$, and all codewords are individually addressable.*

Proof No codeword in $\mathcal{C}$ implies any other codeword in $\mathcal{C}$. Let $\mathcal{C}_w$ be the set of w-weight codewords in $\mathcal{C}$. Let $\mathcal{C}_{w+1}$ be the set of $(w+1)$-weight codewords implied by at least one codeword in $\mathcal{C}_w$. Consider the code $\mathcal{C}' = (\mathcal{C} - \mathcal{C}_w) \cup \mathcal{C}_{w+1}$.

Each codeword in $\mathcal{C}_w$ implies $M - w$ codewords in $\mathcal{C}_{w+1}$, but each codeword in $\mathcal{C}_{w+1}$ is implied by at most $w+1$ codewords in $\mathcal{C}_w$. Thus $|\mathcal{C}_{w+1}| \geq |\mathcal{C}_w|(M-w)/(w+1)$. Since $(M-w)/(w+1) > 1$, $|\mathcal{C}'| = |\mathcal{C}| - |\mathcal{C}_w| + |\mathcal{C}_{w+1}| \geq |\mathcal{C}|$.

Let $\mathcal{A} = \mathcal{C}' \cap \mathcal{C}$. Since no codeword in $\mathcal{C}$ implies any other codeword in $\mathcal{C}$, no codeword in $\mathcal{A}$ implies any other codeword in $\mathcal{A}$. The remaining codewords in $\mathcal{C}'$, $\mathcal{C}' - \mathcal{C} = \mathcal{C}_{w+1}$, are all implied by some codeword in $\mathcal{C}_w$. Since no codeword in $\mathcal{C}_w$ implies any codeword in $\mathcal{A}$, no codeword in $\mathcal{C}_{w+1}$ implies any codeword in $\mathcal{A}$. Finally, since codewords can only imply codewords of greater weight, no codeword in $\mathcal{C}'$ implies a codeword in $\mathcal{C}_{w+1}$. Thus no codeword in $\mathcal{C}_{w+1}$ implies any codewords in $\mathcal{C}'$, and all codewords are individually addressable. ■

Lemma 5.1.2 *Let $\mathcal{C} \subseteq \{0,1\}^M$ be a set of codewords which are all individually addressable. If all codewords have weight $\lfloor M/2 \rfloor$ or $\lceil M/2 \rceil$, there exists a code $\mathcal{C}'$ such that $|\mathcal{C}'| \geq |\mathcal{C}|$ and all codewords have weight $\lceil M/2 \rceil$.*

Proof If M is odd, consider the same replacement operation described in the proof of Lemma 5.1.1 for $w = \lfloor M/2 \rfloor$. Now $(M - w)/(w + 1) = 1$, thus the code will not decrease in size. ■

Theorem 5.1.1 *Given M MWs, there exist at most $\binom{M}{\lceil M/2 \rceil}$ addressable codewords.*

Proof Consider any length M code, $\mathcal{C}$, that maximizes the number of individually addressable codewords. Let $\overline{\mathcal{C}}$ denote the code consisting of the complement of all codewords in $\mathcal{C}$. For any two codewords, $\boldsymbol{c^a}, \boldsymbol{c^b} \in \mathcal{C}$, neither implies the other, thus the same holds true for any $\overline{\boldsymbol{c^a}}, \overline{\boldsymbol{c^b}} \in \overline{\mathcal{C}}$. Since $|\overline{\mathcal{C}}| = |\mathcal{C}|$, $\overline{\mathcal{C}}$ also maximizes the number of individually addressable codewords.

Lemma 5.1.1 implies that both C and $\overline{\mathcal{C}}$ have minimum weight codewords of weight at least $\lfloor M/2 \rfloor$. This means that all codewords in either code have weight $\lfloor M/2 \rfloor$ or $\lceil M/2 \rceil$.

Lemma 5.1.2 states that an equal size code exists where all codewords have weight $\lceil M/2 \rceil$. There are at most $\binom{M}{\lceil M/2 \rceil}$ such codewords. ■

5.1.3 Binary Reflected Code

An M-bit **binary reflected code (BRC)** consists of codewords of the form $\boldsymbol{c^i} = \boldsymbol{x}\overline{\boldsymbol{x}}$ where $\boldsymbol{x} \in \{0,1\}^{M/2}$, M is even, and $\overline{\boldsymbol{x}}$ denotes the complement of $\boldsymbol{x}$. As in (h, M)-hot codes, 1 and 0 denote lightly and heavily doped regions, respectively. The code contains $C = 2^{M/2}$ codewords, none of which imply each other. When using a BRC

$$M = 2 \log_2 C$$

From the previous subsection, we know that this is within a factor of 2 of optimal.

Like (h, M)-hot codes, BRCs are closed under cyclic shifting. To see why, observe that codewords in a BRC have the property that bits are separated by $M/2$ positions from their complement. Cyclically shifting a codeword one position to the left or right leaves this property unchanged.

The main advantage of BRC codes is that each codeword directly corresponds to an M-bit tuple. This makes address translation circuitry particularly simple to construct, and also facilitates efficient testing (see Chapter 8). In addition, BRCs are well-suited to radially encoded NWs, as discussed in Section 5.4. Also in some cases, BRCs can potentially allow for multiwrite operations based on "wildcarding" (see [12] as well as Section 8.3).

5.1.4 Generating Random Ensembles of Axial Codewords

We have just described two classes of NW encodings such that all NW codewords, in the absence of misalignment, are individually addressable. In an axially encoded NW decoder, each NW's codeword depends on its encoding and its axial displacement once deposited on chip. In order to maximize the probability that many different codewords appear within each contact group, our goal is to create an ensemble of axially encoded NWs such that each NW codeword is equally likely.

As explained, axial displacement causes a NW's codeword to shift cyclically. This allows a single axial encoding to produce multiple codewords, which suggests that not all NW encodings need be manufactured. In order to correctly determine how many NWs to produce with each encoding, the periodicity of the encoding must be taken into account.

Consider a codeword, $\boldsymbol{c^i}$, which has length M. Let $R^t(\boldsymbol{c^i})$ denote the cyclic shift of $\boldsymbol{c^i}$ by t places to the right and let $H(\boldsymbol{c^i}) = \{\boldsymbol{c^i}, R(\boldsymbol{c^i}), \ldots, R^M(\boldsymbol{c^i})\}$ be the set of codewords that result from all possible cyclic shifts of $\boldsymbol{c^i}$. Let $p(\boldsymbol{c^i}) = |H(\boldsymbol{c^i})|$ be the number of distinct codewords in $H(\boldsymbol{c^i})$. $p(\boldsymbol{c^i})$ is the **period** of $\boldsymbol{c^i}$.

Let $\boldsymbol{c^a}$ and $\boldsymbol{c^b}$ be two codewords. Then either $H(\boldsymbol{c^a}) = H(\boldsymbol{c^b})$ or $H(\boldsymbol{c^a}) \cap H(\boldsymbol{c^b}) = \emptyset$, the empty set. We call $H(\boldsymbol{c^a})$ the **equivalence class containing** $\boldsymbol{c^a}$. Any member of an equivalence class can be thought of as a "seed" because under cyclic shifts it generates each member of the class. If one seed can generate all members of a large set, this reduces manufacturing costs. In contrast, when core-shell NWs are used (see Section 5.4) shifting does not produce different NWs and thus each codeword corresponds to a separately manufactured encoding.

Given a code, suppose that a single seed encoding is selected for each equivalence class, and that K copies of each seed are present in the ensemble from which NWs are selected. If $p(\boldsymbol{c^i}) = M$ for all codewords, then each seed will generate exactly M codewords and each codeword will be equally likely.

On the other hand, suppose some seeds have period $\frac{M}{2}$. These seeds will generate only $\frac{M}{2}$ codewords, and as a result these codewords will be generated twice as often as those generated from seeds with period M. If the ensemble contains K copies of each seed with period M, it should only contain $\frac{K}{2}$ copies of each seed with period $\frac{M}{2}$. More generally, if codewords have length M, a seed with period p will generate p codewords and should be have multiplicity $\frac{Kp}{M}$, where K is some large constant. This condition ensures that all codewords are equally likely.

To summarize, we can generate ensembles of C codewords, where all codewords are individually addressable and equally likely. In the absence of misalignment, the probability that any two NWs connected to an OC are individually addressable is simply the probability that they have distinct codewords, $1/C$. Since C depends on M, this gives us the probability that two codewords are distinct in terms of M. For both BRCs and (h, M)-hot codes, $\log_2 C < M \leq 2 \log_2 C$.

5.2 Analysis of Encoded NW Decoders

As in the previous chapter on RCDs, we first bound the expected number of individually addressable NWs, N_A, per OC as a function of M. We then bound the number of MWs required for many or all NWs to be individually addressable with high probability. These results are used to determine the number of MWs required to implement various addressing strategies.

5.2.1 Bounds Using Expectation

As in Section 4.2, we bound the average number of individually addressable NWs in a simple encoded NW decoder with M MWs and N NWs connected to a single OC.

Theorem 5.2.1 *In a simple encoded NW decoder in which each of the N NWs is assigned one of C individually addressable codewords with equal probability, let N_a be the number of individually addressable NWs. The expected number of individually addressable NWs is*

$$E[N_a] = N(1 - 1/C)^{N-1}$$

where $C = 2^{M/2}$ when binary reflected codes are used, and $C = \binom{M}{\lceil M/2 \rceil}$ when $(\lceil M/2 \rceil, M)$-hot codes are used.

Proof Let x_i be a zero-one random variable that takes value 1 iff NW $\boldsymbol{n_i}$ is individually addressable. $N_A = \sum_{i=1}^{N} x_i$, which gives

$$E[N_A] = \sum_{i=1}^{N} E[x_i] = \sum_{i=1}^{N} P(x_i = 1) = N \cdot P(x_1 = 1)$$

since the x_i are identically distributed.

The probability that a NW is individually addressable is the probability that no other NW shares its codeword, thus $P(x_i = 1) = (1 - 1/C)^{N-1}$. ■

The following corollary follows immediately from Theorem 5.2.1.

Corollary 5.2.1 *Let N'_a be the total number of addressable NWs in a compound encoded NW decoder with g contact groups, N NWs per contact group, $N' = gN$ NWs in total and M MWs.*

$$E[N'_a] = gE[N_a] = N'(1 - 1/C)^{N-1}$$

where $C = 2^{M/2}$ when binary reflected codes are used, and $C = \binom{M}{\lceil M/2 \rceil}$ when $(\lceil M/2 \rceil, M)$-hot codes are used.

We can now use Hoeffding's inequality (see Section 4.2.2) to bound the total number of individually addressable NWs, N'_a, in a compound encoded NW decoder with high probability. As in Section 4.2.2, we begin with the following theorem.

Theorem 5.2.2 *Let N'_a be the total number of individually addressable NWs in a compound encoded NW decoder with g contact groups, N NWs per contact group, and $N' = gN$ NWs in total.*

$$P(N'_a \leq E[N'_a] - N'k) \leq e^{-2k^2 N' N/(N-1)^2} = e^{-2k^2 g^*}$$

for any $k \geq 0$ where $g^* = g(N/(N-1))^2$.

Proof See proof of Theorem 4.2.2 in the previous chapter. The proof applies to encoded NW decoders as well. ■

From this we obtain a corollary.

Corollary 5.2.2 *Let N'_a be the total number of addressable NWs in a compound encoded NW decoder with g contact groups, N NWs per contact group, $N' = gN$ NWs in total and M MWs.*

$$P(N'_a > \kappa N') \geq 1 - \epsilon$$

if $\kappa \leq 1 - \sqrt{-\ln \epsilon/(2g^*)} - (N-1)/C$ where $g^* = g(N/(N-1))^2$ and $C = 2^{M/2}$ when binary reflected codes are used or $C = \binom{M}{\lceil M/2 \rceil}$ when $(\lceil M/2 \rceil, M)$-hot codes are used.

Proof From Corollary 5.2.1 we have $E[N'_a] = N'(1-1/C)^{N-1}$ and by the above theorem,

$$P(N'_a \leq N'(1-1/C)^{N-1} - N'k) \leq e^{-2k^2 g^*}$$

Thus, if $k = (1-1/C)^{N-1} - \kappa$, then

$$P(N'_a \leq \kappa N') \leq e^{-2g^*((1-1/C)^{N-1}-\kappa)^2}$$

When $e^{-2g^*((1-1/C)^{N-1}-\kappa)^2} \leq \epsilon$, the desired conclusion follows. This occurs when $\ln \epsilon \geq -2g^*((1-1/C)^{N-1} - \kappa)^2$ or $\sqrt{-\ln \epsilon/(2g^*)} \leq (1-\kappa) - (1-1/C)^{N-1}$.

Since $(1-1/C)^{N-1} \geq 1-(N-1)/C$, The requirement that $\kappa \leq (1-1/C)^{N-1} - \sqrt{-\ln \epsilon/(2g^*)}$ can be replaced with the stricter requirement that $\kappa \leq 1 - \sqrt{-\ln \epsilon/(2g^*)} - (N-1)/C$. ■

The above theorem and corollary describe how many individually addressable NWs, N'_A, are present with probability $1-\epsilon$, given C codewords, g OCs, and N NWs per OC. To produce a bound on N_A in terms of the number of MWs, we need only express C in terms of M, which of course depends on which code is used to encode NWs.

Example 5.2.1 *Corollary 5.2.2 is useful in bounding the number of MWs required to implement the "Take What You Get" (TWYG) addressing strategy (see Section 3.5.2) in which a significant fraction of all NWs are individually addressable. As an example, suppose* $g = 157$, $N = 8$, $N' = 1256$, $\epsilon = .01$, *and* $\kappa = .820$. *When* $C = 64$ *and* $\kappa = .820 \leq 1 - \sqrt{-\ln .01/(2 \cdot 157 \cdot (8/7)^2)} - 7/64$. *Thus at least* $\lceil .820 \cdot 1256 \rceil = 1030$ *NWs are addressable with probability .99 using* $M = 12$ *MWs when a BRC is used, or* $M = 8$ *MWs when an* $(8,4)$*-hot code is used.*

Besides the total number of individually addressable NWs, N'_A, we may also be interested in the total number of individually addressable codewords, N'_C, or the number of OCs in which each codeword appears. Both quantities can be bounded using a similar approach.

In the case of N'_C, the following theorem bounds the expected number of unique codewords across g OCs. As motivation, consider two NWs in a particular contact group with same codeword. Neither will be individually addressable, but it is possible to address both NWs simultaneously, effectively treating them as a single NW. As such, rather than only bound the total number of individually addressable NWs, N'_A, it makes sense to also bound the total number of distinct codewords per contact group, summed over all contact groups, N'_C.

Theorem 5.2.3 *Let* N'_C *be the total number of individually addressable codewords in a compound encoded NW decoder with* g *contact groups,* M *MWs,* N *NWs per contact group, and* $N' = gN$ *NWs in total. The expected number of individually addressable codewords is*

$$E[N'_C] = gC(1-(1-1/C)^N) \geq N'(1-N/(2C))$$

where $C = 2^{M/2}$ *when binary reflected codes are used, and* $C = \binom{M}{\lceil M/2 \rceil}$ *when* $(\lceil M/2 \rceil, M)$*-hot codes are used.*

Proof For a particular contact group, let x_i be a zero-one variable that takes value 1 iff the i^{th} codeword is present. Then the total number of codewords in the contact group is $N_C = \sum_{i=1}^{C} =$

x_i, and

$$E[N_C] = C \cdot Prob(x_i = 1) = C(1 - (1 - 1/C)^N)$$

since $(1 - 1/C)^N$ is the probability that a particular codeword does not appear among N NWs. Finally, by a taylor series expansion, $C(1-(1-1/C)^N) \geq N - N^2/(2C)$. Since $E[N'_C] = gE[N_C]$, multiplying by g yields the desired conclusion. ■

From the above theorem and Theorem 5.2.2 above, it is straightforward to derive the equivalent of Corollary 5.2.2 with N'_C in place of N'_A.

Corollary 5.2.3 *Let N'_C be the total number of individually addressable codewords in a compound encoded NW decoder with g contact groups, M MWs, N NWs per contact group, and $N' = gN$ NWs in total.*

$$P(N'_C > \kappa N') \geq 1 - \epsilon$$

if $\kappa \leq 1 - \sqrt{-\ln \epsilon/(2g^*)} - N/(2C)$ where $g^* = g(N/(N-1))^2$ and $C = 2^{M/2}$ when binary reflected codes are used or $C = \binom{M}{\lceil M/2 \rceil}$ when $(\lceil M/2 \rceil, M)$-hot codes are used.

Proof Since $E[N'_C] \geq N'(1 - N/(2C))$, by Theorem 5.2.2

$$P(N'_a \leq N'(1 - N/(2C)) - N'k) \leq e^{-2k^2 g^*}$$

Thus, if $k = 1 - N/(2C) - \kappa$, then

$$P(N'_a \leq \kappa N') \leq e^{-2g^*(1-N/(2C)-\kappa)^2}$$

When $e^{-2g^*(1-N/(2C)-\kappa)^2} \leq \epsilon$, the desired conclusion follows. This occurs when $\ln \epsilon \geq -2g^*(1 - N/(2C) - \kappa)^2$ or $\sqrt{-\ln \epsilon/(2g^*)} \leq 1 - \kappa - N/(2C)$. ■

Example 5.2.2 *Corollary 5.2.3 is useful in bounding the number of MWs required to implement the "Take What You Get" (TWYG) addressing strategy (see Section 3.5.2) when it is acceptable if NWs with the same codeword are addressed at once. As an example, suppose $g = 160$, $N = 8$, $N' = 1280$, $\epsilon = .01$, and $\kappa = .805$. When $C = 32$ and $\kappa = .805 \leq 1 - \sqrt{-\ln .01/(2 \cdot 160 \cdot (8/7)^2)} - 8/64$. Thus at least $\lceil .804 \cdot 1280 \rceil = 1030$ NWs are addressable with probability .99 using $M = 10$ MWs when a BRC is used, or $M = 7$ MWs when an $(7, 3)$-hot code is used.*

The above bounds on N'_A and N'_C in terms of g, N, M and ϵ are useful in bounding the area required to implement the TWYG addressing strategy. In order to bound the area required to implement the "Address Sets Across Groups" (ASAG) addressing strategy (see Section 3.5.2), it is necessary to bound, G_i, the number of contact groups in which codeword $\boldsymbol{c^i}$ appears. To do this, first recall from the proof of Theorem 5.2.3 that the probability a particular codeword appears in a particular contact group is $(1-(1-1/C)^N)$. The expected number value of G_i is thus $E[G_i] = g(1 - (1 - 1/C)^N)$. While Hoeffding's inequeality can be used to obtain a bound on G_i in terms of g, N, M and ϵ, a Chernov bound gives better results.

Let $S = x_1 + x_2 + ... + x_t$ be the sum of t independent binary random variables, and let p_i denote $Pr(x_i = 1)$. The well-known Chernov bound for this sum [61, p. 64] states that for any $0 < \delta < 1$

$$Pr(S \leq (1-\delta)E[S]) \leq (e^{-\delta}/(1-\delta)^{1-\delta})^{E[S]}$$

where $E[S] = \sum_{i=1}^{t} p_i$.

It is also possible to put the above bound in a weaker, but more convenient form [61, p. 64]

$$Pr(S \leq (1-\delta)E[S]) \leq e^{-E[S]\delta^2/2}$$

where again $E[S] = \sum_{i=1}^{t} p_i$.

This bound yields the following theorem.

Theorem 5.2.4 *In an encoded NW decoder with g OCs and N NWs per contact group, let G_i be the total number of contact groups in which the i^{th} codeword appears. Then*

$$Pr(G_i > \kappa g) \geq 1 - \epsilon$$

where $\kappa = (1 - \sqrt{(-2\ln\epsilon)/g(1-(1-1/C)^N)})(1-(1-1/C)^N)$

Proof A given codeword appears in a given contact group with probability $(1-(1-1/C)^N)$, thus $E[G_i] = g(1-(1-1/C)^N)$. From the above Chernov bound, substituting $\kappa/(1-(1-1/C)^N) = (1-\delta)$ we have $Pr(G_i \leq \kappa g) \leq e^{-g(1-(1-1/C)^N)(1-\kappa/(1-(1-1/C)^N))^2/2}$

Thus $Pr(G_i > \kappa g) \geq 1 - \epsilon$ if $\kappa \leq (1 - \sqrt{(-2\ln\epsilon)/g(1-(1-1/C)^N)})(1-(1-1/C)^N)$ ■

Example 5.2.3 *Corollary 5.2.3 is useful in bounding the number of MWs required to implement the "Address Sets Across Groups" (ASAG) addressing strategy (see Section 3.5.2) in which each codeword is guaranteed to appear in at least κg contact groups. As an example, let $g = 256$, $N = 8$, $N' = 1856$, $\epsilon = .000625$ and $\kappa = .25$. When $C = 16$, $\kappa = .25 \leq (1-\sqrt{-2\ln .000625/(256 \cdot (1-(1-1/16)^8))}) \cdot (1-(1-1/16)^8)$. This means that all C codewords appear in at least $.243g$ with probability at least $1 - C * .000625 = .99$. Thus at least $\lceil .25 \cdot 16 \cdot 256 \rceil = 1024$ NWs are addressable with probability .99 using $M = 8$ MWs when a BRC is used, or $M = 6$ MWs when an $(6,3)$-hot code is used.*

5.2.2 Bounds Using Inclusion-Exclusion

In this section we derive upper and lower bounds on the number of codewords, C, and number of MWs, M, required for all NWs in an encoded NW decoder to be individually addressable with probability $1 - \epsilon$. These bounds are based on the principle of inclusion-exclusion (see Section 4.1).

Theorem 5.2.5 *In a simple encoded NW decoder in which N NWs are each assigned one of C codewords with equal probability, let ϵ be the probability that all NWs are not individually addressable. Then ϵ satisfies the following bounds*

$$Q(1 - Q/2) \leq \epsilon \leq Q$$

where $Q = \binom{N}{2}(1/C)$. Here $C = 2^{M/2}$ when binary reflected codes are used, and $C = \binom{M}{\lceil M/2 \rceil}$ when $(\lceil M/2 \rceil, M)$-hot codes are used.

Proof Let $E_{a,b}$, where $a < b$, be the event that $\boldsymbol{c^a} = \boldsymbol{c^b}$. The probability that all NWs are not individually addressable is

$$\epsilon = P(\bigcup_{a<b} E_{a,b})$$

where the union is over all pairs of NWs. By expressing ϵ as a union, we can now use inclusion-exclusion to obtain upper and lower bounds on ϵ in terms of C. This gives

$$\sum_{a<b} P(E_{a,b}) - \sum_{(a,b)\neq(c,d)} P(E_{a,b} \cap E_{c,d}) \leq \epsilon \leq \sum_{a<b} P(E_{a,b})$$

For the inclusion portion of the bound, we simply note that $P(E_{a,b}) = 1/C$, and thus

$$\sum_{a<b} P(E_{a,b}) = \sum_{a<b} 1/C = \binom{N}{2}(1/C)$$

For the exclusion portion of the bound, we must compute $\sum_{(a,b)\neq(c,d)} P(E_{a,b} \cap E_{c,d})$ where $a < b$ and $c < d$. Here we have two cases.

In case (1), a, b, c and d are all different. There are $\binom{N}{4}$ ways of selecting them, and since $E_{a,b}$ and $E_{c,d}$ are independent, $P(E_{a,b} \cap E_{c,d}) = 1/C^2$.

In case (2), either $a = c$ or $b = d$. In both of these two subcases, there are $\binom{N}{3}$ ways of selecting a, b, c and d, and $P(E_{a,b} \cap E_{c,d}) = 1/C^2$.

This gives

$$\sum_{(a,b)\neq(c,d),a<b,c<d} P(E_{a,b} \cap E_{c,d}) = \binom{N}{4}1/C^2 + \binom{N}{3}1/C^2$$

which by the inclusion-exclusion bound above gives

$$\binom{N}{2}(1/C) - \binom{N}{4}1/C^2 - \binom{N}{3}1/C^2 \leq \epsilon \leq \binom{N}{2}(1/C)$$

Since $\binom{N}{2} = N(N-1)/2$, $\binom{N}{3} = N(N-1)(N-2)/6$ and $\binom{N}{4} = N(N-1)(N-2)(N-3)/24$, we have $\binom{N}{4} < \binom{N}{2}^2/6$, and $\binom{N}{3} < \binom{N}{2}^2/4$, which gives

$$\binom{N}{2}(1/C) - (1/2)\binom{N}{2}^2 1/C^2 \leq \epsilon \leq \binom{N}{2}(1/C)$$

Factoring out $\binom{N}{2}(1/C)$ on the left hand side gives the desired result. ■

This theorem implies upper and lower bounds on C (and hence M) in terms of N and ϵ. For the cases when Q and ϵ are small, these bounds are tight, meaning the upper and lower bounds they imply on M agree.

Corollary 5.2.4 *In a compound encoded NW decoder with g contact groups and N NWs per contact group, the minimum value of C such that all $N' = gN$ NWs are individually addressable with probability $1 - \epsilon$ satisfies the following bounds. $C \geq N'(N-1)/(2\epsilon)$ where $C = 2^{M/2}$ when binary reflected codes are used, and $C = \binom{M}{\lceil M/2 \rceil}$ when $(\lceil M/2 \rceil, M)$-hot codes are used.*

Proof Let $\delta = \epsilon/g$. Suppose all N NWs connected to any given OC fail to be individually addressable with probability at most δ. Then the probability that any of the g OCs fail to have all N NWs be individually addressable is at most $g\delta = \epsilon$.

Now let γ be the actual probability that all NWs in a contact group of N NWs are not individually addressable. From Theorem 5.2.5, we have $\gamma \leq \binom{N}{2}(1/C)$. Thus if $C \geq N(N-1)/(2\delta)$, $\gamma \leq \delta$. ■

Example 5.2.4 *To bound the area required to control a crossbar-based memory using a compound encoded NW decoder, we wish to bound the number of MWs required to implement the "All Wires Addressable" addressing strategy (see Section 3.5.2).*

In a compound encoded NW decoder with $g = 128$ OCs, $N = 8$ NWs per OC and $N' = 8 \cdot 128 = 1,024$ total NWs, Corollary 5.2.4 asserts that all N' NWs are individually addressable with probability $1 - \epsilon = .99$ or better when $C \geq 358,400$. When BRC codes are used, this value of C is achieved with $M = 38$ MWs. If $M/2$-hot codes are used, only $M = 22$ MWs are required.

What's more, evaluating Theorem 5.2.5 numerically for these values of N and M shows this threshold value of M is exact.

Example 5.2.5 *In the previous example, the number of MWs can be reduced if we don't require that all NWs in each contact group be individually addressable. As an alternative, we can implement the "Almost All Wires Addressable" addressing strategy (see Section 3.5.2) in which all NWs are individually addressable in almost all (as opposed to all) contact groups.*

Corollary 5.2.4 implies that a failure rate of at most $\epsilon = .01$ can be within a single contact group when $N = 8$ if $C \geq 2,800$ (as noted in the above example, this threshold value of C is in fact exact). When BRC codewords are used, this requires $M = 24$ MWs. When $(M/2)$-hot codes are used, $M = 14$ MWs suffice.

The number of individually addressable NWs in each OC is statistically independent. If all N NWs in a particular contact group are individually addressable with probability $1-\epsilon$, the probability that f or fewer contact groups fail to have all NWs addressable is $\phi(\epsilon, f, g) = \sum_{i=0}^{f} \binom{g}{i} \epsilon^i (1-\epsilon)^{g-i}$.

Let $\epsilon = .01$, $g = 133$ and $f = 5$. Because $\phi(.01, 5, 133) \geq .99$, at least 128 of $g = 133$ OCs have all NWs addressable with probability 0.99. Thus when $M = 14$, $g = 133$, and $N = 8 \cdot 133 = 1064$, $N'_a = 8 \cdot 128 = 1,024$ NWs are individually addressable with probability 0.99.

5.2.3 Area Estimates

To compare addressing strategies, we estimate their area when used to produce a memory with a given storage capacity. In our comparison, we fix ϵ, the probability of failure, and N, the number of NWs per contact group. Given these values, we would ideally like to also fix N'_a, the number of addresses along each dimension of the crossbar, then estimate A_T for all three strategies (see Section 3.5.1). Unfortunately, for a given strategy, it is difficult to choose M and N' to yield an exact value for N'_a. In all three cases below we choose M and N' so that close to $1,024$ NWs are individually addressable along each dimension.

To compare addressing strategies, we draw on the examples of the previous two sections. To minimize area, we assume that $(\lceil M/2 \rceil, M)$-hot codes are used.

- **All Wires Addressable**:

  Here when $g = 128$, $N = 8$, and $M = 22$ ($C = 358,400$) all $N' = N'_a = 1,024$ NWs are addressable with probability at least .99. The ATC requires $\beta = N'_a\lceil \log C \rceil = 22,538$ bits. This gives

$$A_T \approx 22,538\chi + \lambda^2_{meso}1,792 + (\lambda_{meso}22 + \lambda_{nano}1,024)^2$$

- **All Wires Almost Always Addressable**:

  Here when $g = 133$, $N = 8$, and $M = 14$ ($C = 2,800$), all N NWs in at least $g' = 128$ contact groups are individually addressable with probability at least .99. In this case, at least $N'_a = 1,024$ NWs out of the $N' = 1,064$ are individually addressable and the ATC requires $\beta = g'\lceil \log g - g' \rceil + N'_a\lceil \log C \rceil = 12,672$ bits.

$$A_T \approx 12,672\chi + \lambda^2_{meso}1,792 + (\lambda_{meso}14 + \lambda_{nano}1,064)^2$$

- **Take What You Get**:

  Here when $g = 160$, $N = 8$, and $M = 7$ ($C = 32$) at least $N'_A = 1030$ addresses are present among the $N' = 1,280$ NWs with probability at least .99. The ATC requires $\beta = N'_a(\lceil \log C \rceil + \lceil \log g \rceil) = 13,390$ bits. This gives

$$A_T \approx 13,390\chi + \lambda^2_{meso}2,560 + (\lambda_{meso}7 + \lambda_{nano}1,280)^2$$

- **Address Sets Across Groups**:

  Here when $g = 256$, $N = 8$, and $M = 6$ ($C = 16$) all codewords appear in at least 64 contact groups with probability at least .99. This gives $N'_A = 64 * C = 1,024$ addresses and requires an ATC with $\beta = N'_a(\lceil \log g \rceil) = 8,192$ bits. This gives

$$A_T \approx 8,192\chi + \lambda^2_{meso}4,096 + (\lambda_{meso}6 + \lambda_{nano}2,048)^2$$

Unlike for RCDs in Chapter 4, the "All Wires Almost Always Addressable" addressing strategy outperforms all others in terms of area. Unlike for RCDs, a relatively small number of MWs, $M = 14$, are required to ensure that all NWs within a contact group are individually addressable with probability $1 - \epsilon$.

Code Size

We note that area is not the only issue worth considering when manufacturing encoded NW decoders. The size of the code is also a consideration. As noted in Section 5.1.4, when a pattern of lightly and heavily doped regions is repeated along the length of a NW, each NW type generates at most M different codewords. Even when the number of MWs required for a particular addressing strategy is modest, say 15, there is still be a need to grow hundreds of different NW types off chip. This is particularly true for radially encoded NWs (discussed in Section 5.4 below), because here each NW type generates only a single codeword. Furthermore, growing a large number of types of radially encoded NWs likely requires that each NW have many shells, and thus be relatively large compared to axially encoded NWs.

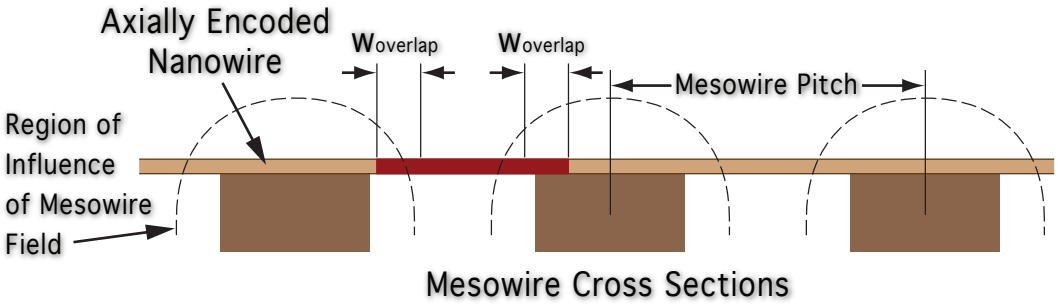


Figure 5.3: In [45], the above schema is used for calculating the probability of misalignment errors occurring. The NW's lightly doped region (shown in red) is controllable, meaning its resistance increases in the presence of a sufficiently strong electric field. When the overlap of the region and a MWs electric field is $W_{overlap}$ or less, the NW is not sufficiently controlled by the MW. The MW, when activated, will only partially increase the NWs resistance, resulting in an error. Notice that the lightly doped region is wide enough so that it is always controlled by at least one MW, regardless of axial misalignment.

5.3 Misalignment of Axial Codes

As discussed in Section 5.1 and illustrated in Figure 5.1, axially encoded NWs are not guaranteed to align with MWs when deposited on chip. Once deposited, let δ denote the distance between a NW's endpoint and some MW. As noted in Section 5.1.2, if δ increases or decreases by a factor of λ_{meso} (the pitch of the MWs), this displacement has the effect of cyclically shifting the NWs codeword. Nanoscale displacement, however, can cause some MWs to only partially control certain lightly doped regions. In other words, certain values of $\delta_{nano} = \delta \bmod \lambda_{meso}$ result in some NW/MW junctions being in error.

In [45], it is shown that when NWs are axially encoded to produce a particular set of codewords, $\mathcal{C}$, there is always a chance that a particular MW activation pattern, $\boldsymbol{a} = \overline{\boldsymbol{c^i}}$, will cause certain NWs to be in error. Specifically, since $\boldsymbol{a}$ will address NWs with codeword $\boldsymbol{c^i}$, there is always a chance that some NWs with that codeword will be shifted by an amount that causes them to be only partially conducting when $\boldsymbol{a}$ is applied. As a result, the best possible approach to coping with nanoscale misalignment appears to be to make sure that each lightly doped region is wide enough such that it is always controlled by at least one MW. In other words, no region should be small enough so that it falls between two adjacent MWs, and isn't controlled by either MW. This ensures that each NW is addressed by either one or zero activation patterns of the form $\boldsymbol{a} = \overline{\boldsymbol{c^i}}$. It also ensure that at most one such activation pattern causes the NW to be in error.

To quantify the probability of misalignment errors, let $w_{overlap}$ be the minimal length overlap needed between the field of a MW and a NW's lightly doped region to reduce the region's conductivity to a satisfactory level (see Figure 5.3). Also, let λ_{meso} be the pitch of MWs (i.e. their center to center distance). Since we assume NWs can shift by large amounts, we treat $\delta_{nano} = \delta \bmod \lambda_{meso}$ as a uniform random variable ranging from 0 to λ_{meso}. It follows that the probability, p_f, that

misalignment errors occur is $p_f = (1 - 2w_{overlap}/\lambda_{meso})$ [45]. This implies that some small fraction of NWs will have codewords that contain errors and suggests that it is not reasonable to require all NWs be addressable. This is not a significant setback, however, since other addressing strategies use less area regardless (see Section 5.2.3).

When an axially encoded NW is misaligned, many MW/NW junctions are likely to be in error simultaneously. This is in contrast to randomized-contact decoders (RCDs), in which errors occur with statistical independence. If one did wish to design an encoded NW decoder such that all NWs are addressable, one approach would be to encode NWs in in a sufficiently random fashion such that the RCD analysis of Section 4.1 applies. Although this would increase the number of required MWs by a small constant factor, it would also allows the encoded NW decoder to tolerate manufacturing errors that are not caused by misalignment. As noted in [44], and analyzed in [63], it is also possible to explicitly design NW encodings that are capable of tolerating random errors.

5.4 Radially Encoded Nanowire Decoders

This section describes how encoded NW decoders can be produced using core-shell NWs in place of modulation-doped NWs. As described at the beginning of Section 5.1, core-shell NWs are encoded radially, whereas modulation-doped NWs are encoded axially. Core-shell NWs consist of a lightly doped silicon core surrounded by a sequence of insulating shells. Shells are comprised of a thin layer of separately-etchable materials. We refer to shell materials as "shell types". As with modulation-doped NWs, many differently encoded core-shell NWs are grown off-chip using chemical vapor deposition, then deposited fluidically. Once deposited, however, a distinct etching sequence is applied to the region under each MW. If etching sequences are chosen properly, NWs with each shell sequence will be given an individually addressable codeword.

Given a shell type, x, we use $E(x)$ to denote the etching process that removes only shells of type x. If two NWs, $\boldsymbol{n_a}$ and $\boldsymbol{n_b}$, have materials x and y in their outer shells respectively, $E(x)$ removes the outer shell of $\boldsymbol{n_a}$, but leaves $\boldsymbol{n_b}$ unaffected. In order to ensure that multiple shells are not removed by a single etching operation, it is assumed that adjacent shells are always of different shell types. If a particular etching sequence exposes a given NW's core (i.e. removes all of its shells) under a particular MW, that MW will control the NW.

By growing NWs with appropriately chosen shell sequences, then applying specific etching sequences, it is possible to produce an encoded NW decoder in which codewords are drawn from BRC or 1-hot codes. Furthermore, since the etched away regions of NWs are guaranteed to align with MWs, these encoders do not suffer from axial misalignment. This section presents the shell sequences and etching sequences required to efficiently construct radially encoded NW decoders from core-shell NWs. At the end of this section, error-correcting codes are discussed as a way to protect against potential etching errors.

5.4.1 The Linear Radially Encoded NW Decoder

Assume that t shell types are used to produce radially encoded NWs with k shells. Since all adjacent shells must be of different types, $C = t(t-1)^{k-1}$ different shell sequences are possible. If NWs

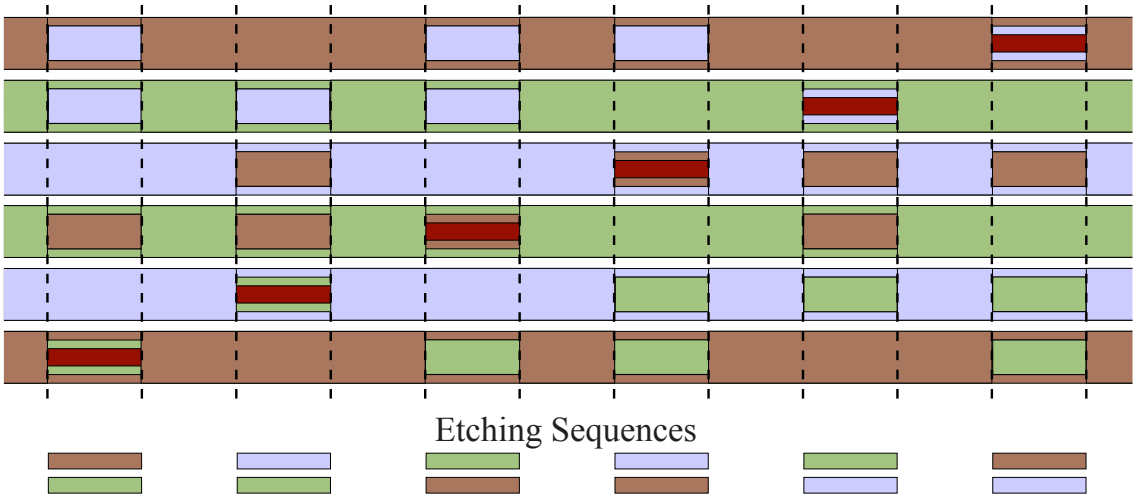


Figure 5.4: In a linear radially encoded nanowire decoder, one MW controls each type of NW. Given t shell types, and k shells per NW, $C = t(t-1)^{k-1}$ shell sequences are possible. Each shell sequence also corresponds to an etching sequence applied under one of the MWs. The result is that the core of NWs with a particular shell sequence are exposed under exactly one MW. In the above figure, $t = 3$ and $k = 2$. Dashed lines indicate the regions under MWs, to which all 6 possible etching sequences have been applied. All 6 NW types are present, but in an actual radially encoded NW decoder, a randomly selected subset of NW types would be present.

with all C shell sequences are grown and deposited to form a radially encoded NW decoder, 1-hot codewords can be produced as follows:

1. Consider t shell types $s_1, \ldots, s_t$. Let $M = C$ and associate a different shell sequence with each of the M MWs. Let $\boldsymbol{s_i} = s_{a(i,1)}, s_{a(i,2)}, \ldots, s_{a(i,k)}$ be the sequence associated with MW $\boldsymbol{m_i}$, where $a(i,j)$ simply indexes shells to shell types. Here $s_{a(i,1)}$ denotes the material of the inner most shell, and $s_{a(i,k)}$ denotes the material of the outermost shell.

2. Before laying down MW $\boldsymbol{m_i}$, apply the etching sequence $E(s_{a(i,k)}), \ldots, E(s_{a(i,1)})$ to the region under $\boldsymbol{m_i}$. Here $E(s_{a(i,k)}), \ldots, E(s_{a(i,1)})$ denotes the application of each of individual etchants, one after the other. This k-step etching process will expose only the cores of NWs with shell sequence $\boldsymbol{s_i}$. If a NW has a different shell sequence, at least one shell will remain.

3. By associating a MW with each shell sequence, each NW is given a 1-hot codeword, since exactly one MW controls it (see Figure 5.4). If NW $\boldsymbol{n_a}$ has a different shell sequence then all other NWs connected to a its OC, it will be individually addressable.

Since $M = C$, this decoder is called a **linear radially encoded NW decoder**. Fortunately, as demonstrated in Section 5.2.1, C need not be large when the TWYG addressing strategy is employed. For example, if $k = 2$ and $n = 4$, $C = 12$, which is sufficient. In [47] four separately etchable shell types have been proposed.

Etching Time

The etching procedure for the linear radially encoded NW decoder requires that a k-step etching sequence be applied under each of the $M = C = t(t-1)^{k-1}$ MWs. If each of the M etching sequences were carried out sequentially, $kC = kt(t-1)^{k-1}$ total etching operations would be required. By parallelizing the etching sequences, however, this number can be reduced to kt. In a **parallel etching operation**, a particular etchant, $E(x)$, is applied to the regions under multiple MWs simultaneously.

To understand how etching operations can be parallelized, consider the first step of each of the M etching sequences. When steps correspond to the same etching operation, they can be performed in parallel. As such, only t parallel etching operations are required to execute the first step of each of the M etching sequences. At this point, the second steps can also parallelized, and so on, until all k-steps have been carried out using kt total parallel etching operations.

5.4.2 The Linear-Logarithmic Radially Encoded NW Decoder

As explained in Section 5.1.1, an axially encoded NW decoder can be constructed such that $M = O(\log_2 C)$. In fact, the same bound on M can be obtained for radially encoded NW decoders. The linear decoder above uses $M = C$ MWs to select one of C NW types. In this section we describe how to construct a **linear-logarithmic radially encoded NW decoder** in which $M = kt$ and $C = (t/2)^k$. In this case, M is logarithmic in C if t is treated as a constant. In the following section, we show how M can be made logarithmic in t, and then explain how the two constructions can be easily combined.

In the linear decoder, the only requirement imposed on the NW shell sequences is that consecutive shells be of different types, and thus $C = n(n-1)^{k-1}$. In this decoder, we divide the t shell types into two equal sized sets (if t is odd, it is acceptable for the sets to be different sizes). In all NW shell sequences, the odd numbered shells are chosen from the first $t/2$ shell types, and the even numbered shells are chosen from the remaining shell types. This allows for $C = (t/2)^k$ possible NW types, and allows for more powerful etching operations.

1. Let $s_1^i, s_2^i, \ldots, s_{t/2}^i$ denote the $t/2$ shell types permitted to appear in a NWs i^{th} shell. Let E_i denote the etching process $E(s_1^i), E(s_2^i), \ldots, E(s_{t/2}^i)$ that removes all material in the i^{th} shell (the order in which the $t/2$ etchants are applied is unimportant).

2. Consider each etching process of the form $E(i,j) = E_k, \ldots, E_{i+1}, E(s_j^i), E_{i-1}, \ldots, E_1$. $E(i,j)$ removes the $k-i$ outermost shells of every NW, a single shell from every NW with shell type s_j^i in its i^{th} shell, then the remaining $i-1$ shells of those NWs. Only these NWs have all k shells removed and their cores exposed.

3. Associate each etching process, $E(i,j)$ with a different MW, $\boldsymbol{m_{i,j}}$, where $1 \leq i \leq k$, $1 \leq j \leq t/2$. Apply the etching processes $E(i,j)$ to the region under $\boldsymbol{m_{i,j}}$. As a result, $\boldsymbol{m_{i,j}}$ controls only the NWs with shell type s_j^i in their i^{th} shell (see Figure 5.5).

4. MWs are grouped into k groups of $t/2$ MWs each, and each of a NWs k shells determines k bits of its codeword. Each NW type is controlled by exactly k of the $k(t/2)$ MWs and all

$C = (t/2)^k$ distinct codewords are individually addressable.

The linear-logarithmic decoder controls $C = (t/2)^k$ NW types with $M = k(t/2)$ MWs. Of course, as k increases, the pitch of core-shell NWs increases as well. As a result, it appears more desirable to increase t, if possible. The next section describes how M can be reduced to $k \log_2 t$, thus accommodating large values of t.

Etching Time

As with the linear radially encoded NW decoder, etching operations for the linear-logarithmic radially encoded NW decoder can be parallelized. In this decoder, a k-step etching sequence is applied under each MW. $k-1$ of these steps involve the application of $t/2$ etchants in an arbitrary order and one step involves the application of a single etchant. This implies that the i^{th} step of all etching sequences can be carried out using $t/2$ parallel etching operations. As a result, $kt/2$ parallel etching operations are required in total.

5.4.3 The Fully Logarithmic Radially Encoded NW Decoder

In order to understand how the number of MWs, M, in a radially encoded NW decoder can be made logarithmic in both the number of shell types, t, and the number of shells, k. First consider the case where $k = 1$.

1. Assign each shell type a unique L-bit binary number. L need be no larger than $\lceil \log_2 t \rceil$. Let x^i be the binary number associated with shell type s_1, and $c_i = x^i\overline{x^i}$ be the "codeword" associated with that shell type.

2. Let S_l, $1 \leq l \leq 2L$ be the set of all shell types assigned a codeword with a 1 as its l^{th} bit. Let $E(S_l)$ be an etching process that removes each shell type in S_l, one after the other, in some arbitrary order.

3. Associate each $E(S_l)$ with a different MW. If MW $\boldsymbol{m_l}$ is associated with $E(S_l)$ it will control only NWs with shell types in S_l.

4. Since a different MW is associated with each bit of c_i, NWs are assigned codewords from a BRC (see Figure 5.6)

This decoder controls t shell types with $\lceil \log_2 t \rceil$ MWs. It is readily incorporated into the linear-logarithmic radially encoded NW decoder, creating a **fully logarithmic radially encoded NW decoder**. To do this, the etching processes $E(i,j) = E_k, \ldots, E(s_j^i), \ldots, E_1$ are replaced with the processes $E(i,l) = E_k, \ldots, E(S_l), \ldots, E_1$. Since shell types are divided into two sets, S_l can be restricted to the $t/2$ shell types that appear in the i^{th} layer. As such, it is acceptable for a shell type in the first set to use the same codeword, c_i, as those in the second set.

The resulting decoder controls $N = (t/2)^k$ NW types with $2 \log_2 C = 2k\lceil \log_2 t/2 \rceil$ MWs. In this decoder, each NWs codeword is the concatenation of the of the codewords associated with each of the shell types in each layer. If MWs are reordered appropriately, this codeword will be of the

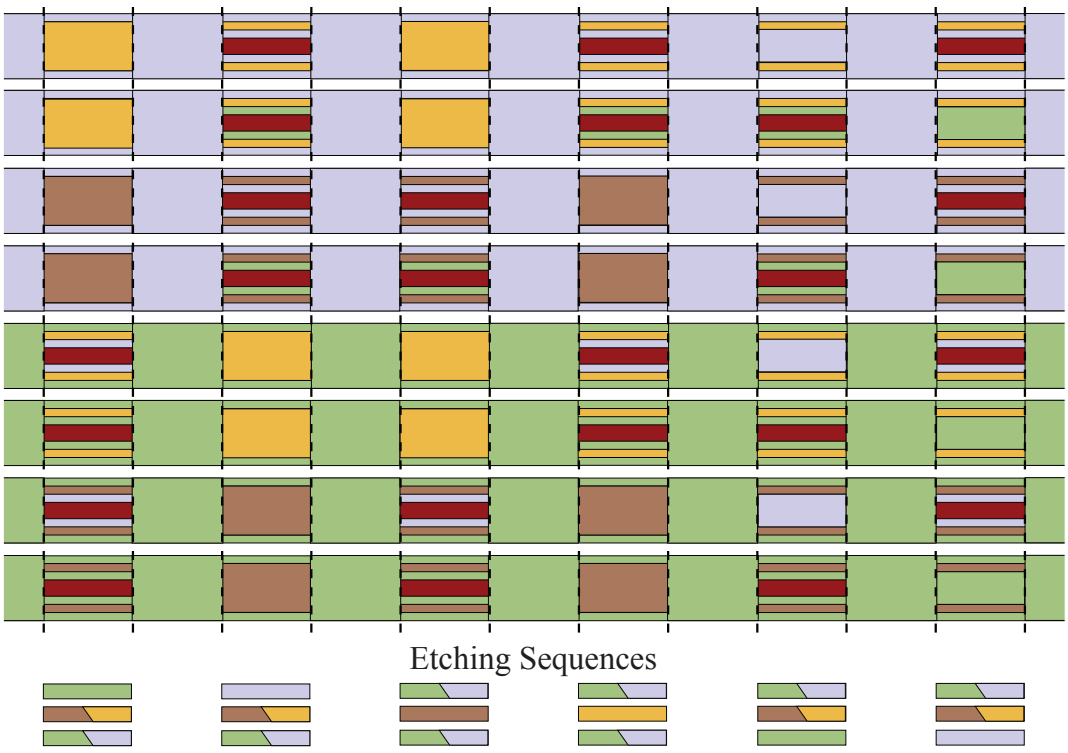


Figure 5.5: In a linear-logarithmic radially encoded nanowire decoder, each MW controls NWs with a particular shell type in their i^{th} layer. To construct this decoder, shell types are partitioned into to two sets. In each NW's shell sequence, the odd shells are drawn only from the first set (here green and purple), while the even shells are drawn only from the second set (here orange and brown). Given t shell types, and k shells per NW, $C = (t/2)^k$ shell sequences are possible. Here $t = 4$ and $k = 3$.

Under MW $\boldsymbol{m_{i,j}}$ ($1 \leq i \leq k$, $1 \leq j \leq t/2$) an etching sequence is applied that removes the $k - i$ outermost shells of all NWs, an additional shell from only NWs with the j^{th} shell type in their i^{th} shell, and then the $i - 1$ remaining shells from these NWs. In the above figure, dashed lines indicate the regions under MWs to which all 6 etching sequences have been applied. Each NW is controlled by exactly 3 of the 6 MWs. All 8 NW types are shown, but in an actual radially encoded NW decoder, a random subset of NW types would be selected.

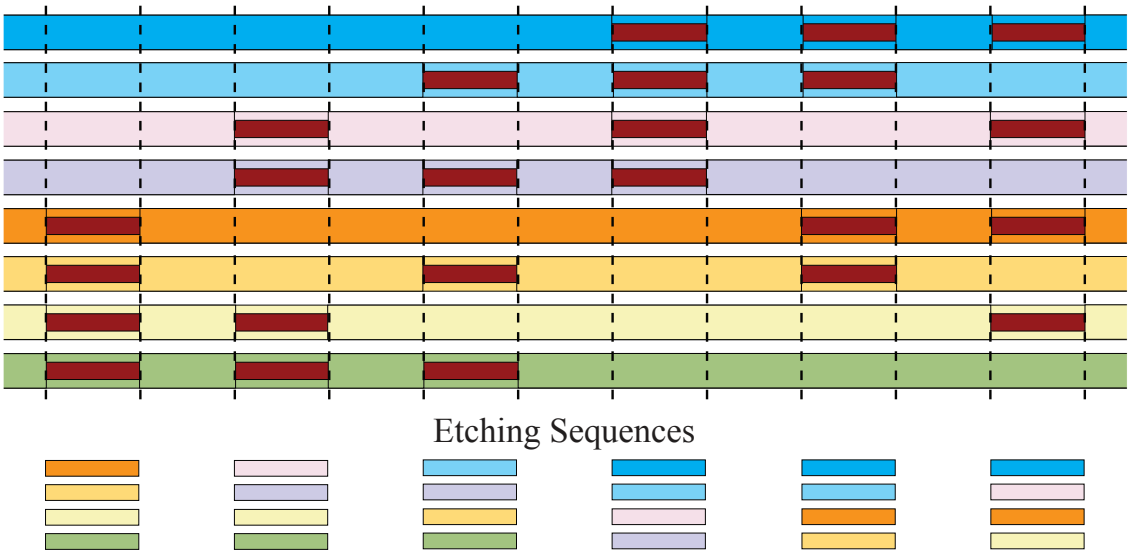


Figure 5.6: A logarithmic radially encoded NW decoder in which NWs only have a single shell. A codeword, c_i, from a length 6 BRC is associated with each of the 8 shell materials. These codewords are used to define the etching sequences that are applied to the regions under the MWs. In this way, each NW type is itself assigned a codeword from a BRC. This approach can be generalized to NWs with multiple shells.

form $\boldsymbol{x}\overline{\boldsymbol{x}}$. Note that the codeword associated with each shell material, c_i, can be drawn from an h-hot code instead of a BRC. This will result in a slightly more efficient decoder.

Etching Time

The etching operations used to construct a fully logarithmic radially encoded NW decoder can be parallelized using the same approach as described for the linear-logarithmic decoder. In the fully logarithmic decoder, a k-step etching sequence is applied under each MW. Each of these steps involves the application of at most $t/2$ etchants in an arbitrary order. As a result, the i^{th} step of all etching sequences can be carried out using $t/2$ parallel etching operations. Once again only $kt/2$ parallel etching operations are required in total.

5.4.4 Fault-tolerant Etching Error Correction

The etching processes we have described may behave imperfectly. Shells which should remain may be removed, and shells that should be removed may remain. Either error can alter a NW's codeword. Consider, for example two NWs, $\boldsymbol{n_a}$ and $\boldsymbol{n_b}$, with distinct error-free codewords, and a third NW, $\boldsymbol{n_c}$, with an erroneous codeword caused by an etching error that turned a 1 to a 0. If, as a result, $\boldsymbol{n_c}$ now implies $\boldsymbol{n_a}$ and $\boldsymbol{n_b}$, neither $\boldsymbol{n_a}$ or $\boldsymbol{n_b}$ will be individually addressable. In other words, etching errors have the potential to significantly reduce the number of useable NWs.

From the end of Section 3.4.1, recall that when the symmetric hamming distance between

codewords is greater than 1, manufacturing errors can be tolerated. Specifically, if the symmetric distance between codewords is d, at least $d-1$ etching errors can be tolerated. In the fully logarithmic decoder, such codeword can be produced by replacing the L-bit strings assigned to shell types with codewords from an error-correcting code. If the strings all have minimum distance d, so will the resulting NW codewords.

In the case of a linear axially encoded NW decoder, a less efficient approach can be considered. In the linear decoder, as described in Section 5.4.1, each of the $M = C$ MWs corresponds to a different k-step etching sequence. All $t(t-1)^{k-1}$ k-step etching sequences are used, but unfortunately a single etching error could render all NWs unaddressable, since it could cause a single NW to type to not be controlled by any MW. As a more fault-tolerant alternative, one could use $C' = t(t-1)^{k'}$ MWs, for some $k' > k$, each corresponding to a different k'-step etching sequence.

Using this approach, each of the C NWs types would be controlled by many of the C' MWs. More importantly, any two NWs with different shell sequences will now be controlled by many different MWs. For example, if $k' = k+1$, the number of MWs is increased by a factor of $t-1$, but each NW is now controlled by $C/C' = t-1$ MWs. Furthermore, if two NWs differ in at least 1 position, their addresses only have two MWs in common. Thus the symmetric distance between NWs is increased from 1 to $t-3$

5.4.5 Hybrid Nanowire Codes and Decoders

In this chapter, we have described how core-shell NWs can be encoded radially and how modulation-doped NWs can be encoded axially. It is natural to ask if these two approaches can be combined. A NW with **a hybrid encoding** has a core that has been grown with an axial encoding surround by shells that form a radial encoding. As with axially encoded NWs, the axial encoding is repeated along the lengths of hybrid encoded NWs to cope with axial misalignment.

As described in [47], axial and radial encoded NW decoders can be efficiently combined to form a hybrid encoded NW decoder. Consider an axially encoded NW decoder with M_A MWs using an length M_A BRC, and a radially encoded NW decoder that uses M_B MWs. Let $M_A = 2M_B$ (or alternatively, let $2M_B$ be a multiple of M_A). If the axially encoded NW decoder controls C_A NW codewords, and the radially encoded NW decoder controls C_B NW codewords, a hybrid encoded NW decoder can be constructed to control $C = C_A \cdot C_B$ codewords.

A **hybrid encoded NW decoder** is implemented using $M_A + 2M_B$ total MWs. The first set of M_A MWs forms an axially encoded NW decoder. Under these MWs, all shells are removed, and thus the MWs can be used to address NWs with a particular axial encoding. The next set of $2M_B$ MWs forms two copies of a radially encoded NW decoder. In other words, among the $2M_B$ MWs, the same etching sequence is applied under j^{th} MW, and the $(j+M_B)^{th}$ MW. Since the core of any given NW is encoded using a BRC, it is guaranteed that one of these MWs (in the absence of axial misalignment) lies over a lightly doped region. Thus, regardless of a NW's axial encoding, the two sets of M_B MWs can be used in tandem as a radially encoded NW decoder to address NWs with a particular radial encoding.

Although the hybrid encoded NW decoder is efficient in terms of MWs, as noted in [47], it requires that NWs be manufactured with multiple shells, and it is still susceptible to misalignment

errors. As such, it may in fact represent the worst of both worlds, thicker NWs, and the possibility of axial misalignment. Recall from Section 3.2.1, however, that axially encoded NWs may still be constructed from NWs with a single shell in order to prevent adjacent NWs from coming into contact. If multiple shell types are used, the hybrid encoded NW decoder offers a way of increasing the number of NW types without requiring longer axial encodings.

5.5 Summary of Results

In this chapter we have analyzed the area and number of NW types required to control N_A out of N NWs using encoded NW decoders. As explained at the beginning of the chapter, an encoded NW decoder is constructed such that each of the N NWs is assigned one of C possible codewords independently at random. A NW's codeword is determined by its encoding, which is defined either axially or radially. Section 5.1 describes several encoding schemes such that all NW codewords (in the absence of manufacturing errors) are both individually addressable and equally likely.

Increasing C, the total number of possible codewords, increases the probability that each NW is individually addressable. Increasing C, however, also increases M, the number of required MWs, as well as the total number of NW types that must be separately manufactured. Section 5.1.2 shows that M can be minimized for a particular value of C by employing $(\lceil M/2 \rceil, M)$-hot codes. In this case $M < \log_2 C + \frac{1}{2}\log_2 \log_2 C + \frac{1}{2}\log_2 \pi$. Section 5.1.3 defines a second potentially useful family of codes, binary reflected codes, for which $M = 2\log_2 C$.

Section 5.2 demonstrates that small values of M and C suffice to construct efficient encoded NW decoders. In Section 5.2.2, Corollary 5.2.4 shows that in a simple encoded NW decoder with M MWs and a single OC connected to N NWs, all N NWs are individually addressable with probability at least $1-\epsilon$ when $C \geq N(N-1)/(2\epsilon)$. Section 5.2.1 shows that C, and hence M, can be significantly reduced if we eliminate the requirement that all NWs be individually addressable. In a compound encoded NW decoder comprised of M MWs, g OCs, and N NWs per OC, Corollary 4.2.2 bounds C such that at least $\kappa N'$ of the $N' = gN$ total NWs are individually addressable with probability at least $1-\epsilon$. As g increases, this bound on C approaches $(N-1)/(1-\kappa)$. As such, it is reasonable to consider constructing encoded NW decoders capable of individually addressing most NWs using, say, 10 or fewer different NW types.

When NWs are encoded axially, Section 5.3 explains that some small fraction of all NWs will suffer from nanoscale axial misalignment. These faulty NWs will need to be detected and ignored. Section 5.4 demonstrates that radially encoded NWs provide a potential solution to this problem. In a radially encoded NW decoder, NWs are encoded using a sequence of separately etchable shell materials. In order to minimize the number of shell materials required, Section 5.4.1 explains how radially encoded NW decoders can be constructed using $(1, M)$-hot codes. Sections 5.4.2 and 5.4.3 go on to explain how these decoders can employ a range of more efficient encodings. Both axially and radially encoded NW decoders can potentially be constructed using fewer than 10 MWs and still be able to individually address most NWs. A comparison between encoded NW decoders, RCDs and mask-based decoders is given at the end of Chapter 6.

Chapter 6

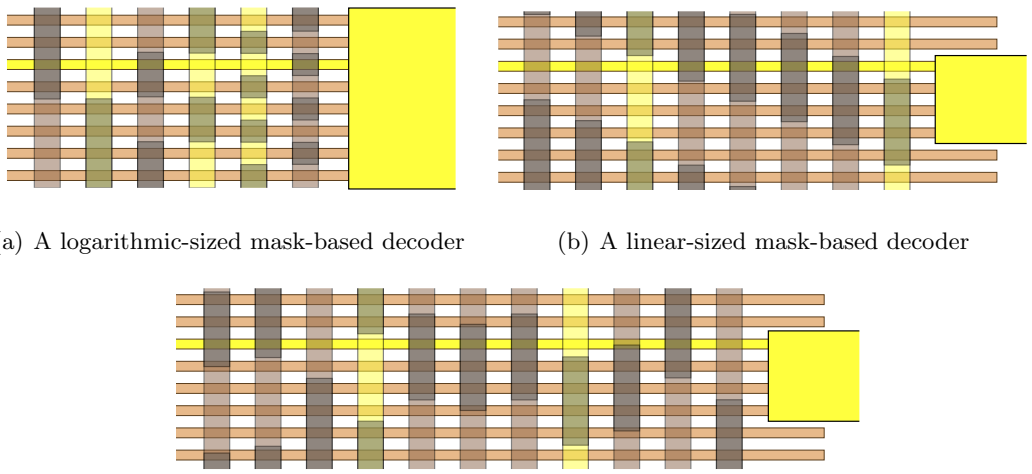
Masked-Based Decoders

In this chapter, we model and analyze the masked-based NW decoder. As described in Section 3.2.2, a masked-based decoder is a NW decoder in which lithographically-defined rectangular regions of high-K dielectric (LRs) are deposited between NWs and MWs. These regions of high-K dielectric focus the field strength of adjacent MWs, thereby causing the lightly doped NWs sitting under each region to turn off when the MW laid on top of the region is turned on. If LRs could be as small as the pitch of NWs and placed with nanometer precision, a mask-based decoder with only $M = 2\log_2 N$ MWs would suffice to individually address N NWs (see Figure 6.1(a)).

Due to the limits of photolithography, it is not realistic to assume that nanoscale regions can be lithographically defined. As an alternative, $2N$ copies of the smallest manufacturable LR could theoretically be arranged in a step-like pattern which we refer to as a "cycle" (see Figure 6.1(b) and Section 6.1.2 below). Unfortunately, this too is unrealistic, as it still assumes that all regions can be placed with nanoscale precision. As a plausible alternative, it has been proposed that many copies of the smallest manufacturable LR be deposited in a roughly cyclic pattern while natural randomness in region placement still permits MWs to gain control over individual NWs with high probability [7, 49, 50] (see Figure 6.1(c)).

Even though LR placement is subject to random variation, the intended location of each region is defined lithographically using a mask. The boundaries of each region are effectively "targeted" at a specific location via openings in the mask. A model for how LR boundaries vary about their intended location is the focus of Section 6.1. Section 6.2 then describes how this model of targeted region placement can be analyzed to determine how many MWs are required to control all N NWs connected to a single OC.

This analysis rests on a novel variant of the well-known **coupon collector problem**. In the classic problem, one of C "coupons" is collected independently at random for each of T trials. One asks how large T must be so that all C coupons are collected with high probability. Section 6.3 analyzes a modified version of this problem in which each trial is able to target a certain coupon (rather then selecting from all coupons with equal probability). The theorems of Section 6.3 are applied in Section 6.4 to bound the number of MWs required for a mask-based NW decoder to individually address all NWs. Additional practical considerations for the size and design of masked-based decoders are presented in Section 6.5. Section 6.6 provides a summarizing comparison between masked-based decoders, encoded NW decoders and RCDs.



(a) A logarithmic-sized mask-based decoder

(b) A linear-sized mask-based decoder

(c) A randomized mask-based decoder

Figure 6.1: Masked-based NW decoders in which regions of high-K dielectric allow each MW to control a different subset of NWs. These regions are illustrated as dark gray rectangles under each MW. When a MW is turned on (indicated with yellow) all NWs under its adjacent high-K dielectric regions are turned off. (a) If arbitrarily small high-K dielectric regions could be manufactured and placed with nanoscale precision, $2\log(N)$ MWs would suffice to address each of N NWs connected to each OC (here $N = 8$). (b) Even if regions with nanoscale width are not possible, $2N$ copies of the smallest manufacturable regions can still be shifted cyclically in order to control all N NWs connected to an OC (here $N = 4$). (c) Using only $2N$ MWs still requires that the lithographically defined regions be placed with nanoscale precision. Since this is not realistic, it has been proposed that many randomly shifted copies of the smallest manufacturable region be used to gain control over individual NWs with high probability. In this way, NW codewords are assigned stochastically. Notice that random shifting may result in MWs that only partially control certain NWs.

6.1 Modeling Decoder Manufacture

Unlike the randomized-contact and encoded NW decoders, the NW codewords in a masked-based decoder are not assigned independently. Two NWs which are adjacent to each other are more likely to have the same codeword (i.e. be controlled by the same MWs), since they are more likely to be covered by the same set of LRs. Before we can analyze masked-based decoders, we must develop a reasonable model for how LRs are placed, and hence how codewords are assigned.

6.1.1 LR Manufacture

To deposit LRs on a chip using photolithography one or more masks are constructed containing multiple rectangular openings. When openings are first made in masks, a one-time process, the separation between the rectangles as well as their size can vary somewhat from their intended values. Additionally, when a mask is used, it is difficult to control the precise alignment of its openings with the NWs already on chip. The offset of a mask from its intended location may be large.

Once a mask is positioned over the NWs, light is passed through the openings in the mask onto a photoresist. This controls an etching process that either removes the lithographically defined regions (positive photoresist) or their complement (negative photoresist). The duration of the etching process, which cannot be precisely controlled, causes variation in the length and width of the LRs.

When a mask is in position above the NWs, we refer to the intended location of a given LR's right or left boundary, relative to the NWs, as its nominal location. Variation in mask manufacture and mask application both cause a LR's endpoint to vary from its nominal location. In the absence of variation, $2N$ left and $2N$ right LR boundaries would suffice to control N NWs connected to an OC (see Figure 6.1(b) as well as Lemma 6.2.1 below). Random variation in LR placement, however, introduces the need for additional LRs, and hence additional MWs.

E-beam lithography is currently too expensive for mass production, but it sets a limit on the best possible conditions for LR manufacture. Using it a) mask placement relative to NWs can vary by 50 to 100nms, b) the length and relative placement of rectangular mask openings can vary by 5 to 10nms from their intended locations on a mask, and c) etching of photoresist can increase the length of LRs by up to 5nms on a chip [64]. If photolithography is used, the longer wavelength of the radiation results in larger variations in these parameters. Uncertainty in mask placement and variation in mask manufacture are independent of the type of lithography employed.

6.1.2 Modeling Variation in Mask Placement

In producing a masked-based decoder, we assume that one or more masks are used to produce one or more cycles of LRs. If each OC is connected to N NWs, each **cycle** contains $2N$ rows of LRs. Let $\boldsymbol{\rho}$ denote the pitch (center-to-center distance) of NWs. In each cycle, the nominal location of the row of LRs under each MW are offset by ρ from those under the previous row (see Figure 6.2). When n cycles are produced we refer to the resulting decoder as an **n-cycle mask-based decoder**. These cycles can be placed using either one or multiple masks.

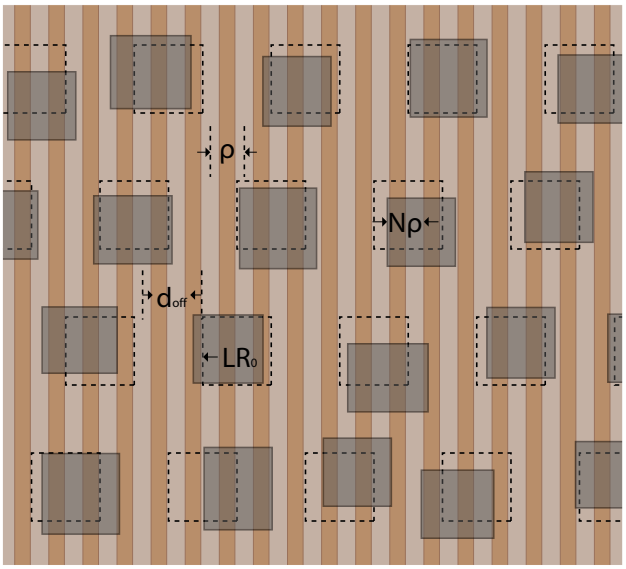


Figure 6.2: In a compound mask-based decoder with g OCs and N NWs per OC, LRs are placed in cycles of $2N$ rows. In this figure, $N = 2$, which is unrealistically small ($N = 8$ would be more plausible). Once MWs are laid down, each row of LRs allows a particular MW to gain control over NWs within each contact group. Once a mask is in position over the NWs, each LR has a nominal location, indicated here by dashed lines. Once deposited, an LR's actual endpoints vary randomly about these nominal locations. The location of a mask is specified by the offset, d_{off}, of some canonical LR, denoted LR_0, from its ideal location. Since NWs have periodicity ρ, and each row of LRs is offset by ρ from the previous row, a change in d_{off} by $\pm\rho$ has no effect on the addressability of NWs. Instead, we are concerned only with the "phase difference" between the NWs and LRs, $\theta = d_{off} \bmod \rho - \rho/2$. In this figure $\theta = -\rho/4$.

Let $\boldsymbol{d_{off}}$ be the offset of a mask from its ideal location, which we assume places the nominal locations of LR boundaries at the midpoint between NWs. d_{off} is defined in terms of the location of a particular but arbitrary LR boundary that we call the **canonical LR boundary, $\mathbf{LR_0}$**. We assume that d_{off} can be large relative to NW pitch, ρ. The cyclic placement of LRs, however, means that if d_{off} increases or decreases by a multiple ρ, it has no effect on the probability that any particular NW is individually addressable. This observation allows us to replace d_{off} by the **phase difference $\boldsymbol{\theta}$**, where $\theta = d_{off} \bmod \rho - \rho/2$ is restricted to the interval $-\rho/2 \leq \theta \leq \rho/2$. Note that $\theta = 0$ corresponds to the nominal position of LR_0 being at the middle of the space between two NWs. It is not important which two NWs it lies between.

Because we assume that the variation of d_{off} is large relative to ρ, *we model θ as a uniform random variable* over the interval $[-\rho/2, \rho/2]$. If the variation in d_{off} is small, as would be the case when the spacing between NWs is large, a non-uniform distribution for θ would be appropriate. We do not consider this case.

6.1.3 Modeling Variation in LR Boundary Placement

When θ is fixed, uncertainties in LR boundary locations result from uncertainties in a) the inscribing of rectangles on masks, b) the exposure of photoresist by electromagnetic radiation through mask rectangles, and c) the photoresist etching time. We collect all these variations in a random variable, d, associated with each LR boundary. The actual location of a LR boundary is determined by θ, the offset of the nominal location of the boundary relative to the adjacent NWs, and d, the change in the position of the boundary relative to its nominal location.

We assume that d has a symmetric probability distribution $f(d)$ that decreases monotonically with d from $d = 0$. This reflects the fact that small variations in d are expected and variations are equally likely to be positive or negative. We also assume that the $\{d_i\}$ associated with LR left and right boundaries are statistically independent and identically distributed.

6.1.4 InterNW Regions

In a mask-based decoder, the locations of the LRs determine which MWs control which NWs. Consider adjacent NWs $\boldsymbol{n_a}$ and $\boldsymbol{n_b}$ where $\boldsymbol{n_a}$ is to the left of $\boldsymbol{n_b}$. If a LR under a MW has a left boundary between $\boldsymbol{n_a}$ and $\boldsymbol{n_b}$, $\boldsymbol{n_b}$ is controlled by the MW and $\boldsymbol{n_a}$ is not controlled. As the LR's left boundary shifts rightward there is a point at which $\boldsymbol{n_b}$ goes from being controlled to partially controlled. Similarly, as the boundary moves leftward there is a point at which $\boldsymbol{n_a}$ goes from being not controlled to partially controlled. The region between these two limits is called the **interNW region**.

In the following section, Lemma 6.2.1 shows that for all NWs to be addressable, a LR right and left boundary must fall in the interNW region between each pair of consecutive NWs. If a LR boundary does not fall into an interNW region, the LR boundary is said to **fail**. If a boundary does not fail, it may fall in the interNW region closest to its nominal location, or some other interNW region. We refer to the interNW region closest to the nominal location as the **targeted interNW region**.

For each LR boundary we let $\boldsymbol{p_i(\theta)}$ be the probability, given a mask phase difference of θ, that a LR boundary moves i regions to the right (left) from its targeted interNW region, when i is positive (negative). Because the random variables $\{d_i\}$ are statistically independent when θ is fixed, the conditional joint probability that LR boundaries on a given mask fall into particular interNW regions is the product of the $p_i(\theta)$.

The facts cited in Section 6.1.1 suggest that a LR boundary will vary by at most a few NW pitches when the mask offset d_{off} is fixed. That is, $p_i(\theta)$ will be non-zero only for small absolute values of i. We assume, $p_i(\theta) = 0$ for $i \geq N$. Since the nominal locations of the right (and left) boundaries of LRs under the same MW are separated by 2 NWs, only one such boundary has a nonzero probability of landing in any particular interNW region.

6.1.5 Additional Sources of LR Boundary Variation

LRs can also be placed using a stamping process [64]. The LRs in a stamp could then be inscribed using E-beam lithography and the stamp used multiple times. Two issues arise in the use of a stamp,

a) uncertainties in the length and separation of LRs grow with the number of stampings and b) large uncertainties arise in the angular orientation of a stamp relative to NWs. It is estimated that the latter could be as large as 20 degrees. E-beam lithography may also introduce a small amount of angular uncertainty.

We do not explicitly model either the degradation of stamps nor the angular uncertainty introduced by both stamping and E-beam lithography in this paper. We believe, however, that these sources of variation can still be analyzed using our methods. Both have the effect of increasing the length of LRs, and decreasing the amount of space between NWs. As a result, the width of an interNW region shrinks because sections of NWs that would otherwise be noncontrollable become partially controllable. This reduction in interNW region width would then be reflected by reducing each $p_i(\theta)$, as defined in the previous subsection.

6.2 Analyzing the n-Cycle Mask-Based Decoder

During operation, the n-cycle mask-based decoder uses standard CMOS to activate a contact group of N NWs. The high-K dielectric regions are then used to turn off all but one NW in a group. As described in Section 6.1.1 the regions are arranged in n cycles where a cycle contains $2N$ rows of LRs and thus requires $M = 2N$ MWs (one over each row). We assume the n-cycle decoder is designed to be able to individually address all N NWs in a contact group with high probability. As shown in the following lemma, this requires both a left and right LR boundary fall into each of the $N - 1$ interNW regions.

Lemma 6.2.1 *Assume that the length of and separation between LRs both span at least N NWs. All NWs N in a contact group are individually addressable if and only if the left boundary and right boundary of two different LRs fall in the interNW region associated with each of the $N-1$ pairs of consecutive NWs.*

Proof A NW $\boldsymbol{n_i}$ is individually addressable if and only if there exists a subset of MWs, denoted S_i, such that no MW in S_i controls or partially controls $\boldsymbol{n_i}$ and all $N-1$ other NWs are controlled by at least one MW in S_i.

For the "if" case assume all consecutive pairs of NWs have left and right LR boundaries in the interNW regions between them and consider an arbitrary NW $\boldsymbol{n_a}$. There exists a MW $\boldsymbol{m_1}$ that lies on top of a LR whose left boundary is in the interNW region to the right of $\boldsymbol{n_a}$. Since the LR must have a length spanning at least N NWs, MW $\boldsymbol{m_1}$ controls all NWs in question to the right of $\boldsymbol{n_a}$. Similarly, there exists a MW $\boldsymbol{m_2}$ that lies on top of a LR whose right boundary is in the interNW region to the left of $\boldsymbol{n_a}$. This MW controls all the NWs in question to the left of $\boldsymbol{n_a}$. The set $S_a = \{\boldsymbol{m_1}, \boldsymbol{m_2}\}$ individually addresses $\boldsymbol{n_a}$.

For the "only if" case, assume all NWs are independently addressable. Consider any two adjacent NWs, $\boldsymbol{n_a}$ and $\boldsymbol{n_b}$, where $\boldsymbol{n_a}$ is to the left of $\boldsymbol{n_b}$ and I_{ab} is the interNW region between them. If $\boldsymbol{n_a}$ is individually addressable, there must be a MW in S_a that controls $\boldsymbol{n_b}$ but not $\boldsymbol{n_a}$. This implies that the LR under this MW has its left boundary in I_{ab}. Similarly, since $\boldsymbol{n_b}$ is individually addressable, there exists a MW that controls $\boldsymbol{n_a}$ but not $\boldsymbol{n_b}$, and thus some LR has its right boundary in I_{ab} as well. ■

This lemma proves that N consecutive NWs are controllable when right and left LR boundaries lie in each of $N-1$ interNW regions. As explained in Section 6.1.1 below, LR boundaries are placed stochastically. Consequently, many rows of LRs are necessary to ensure that these conditions hold with high probability. We also note that if LRs do not span N NWs, all NWs will still be individually addressable, provided that the $N-1$ left (and right) boundaries that lie in the interNW regions all lie under different MWs.

The requirement that LR boundaries fall within interNW regions closely resembles the classic **coupon collector problem** in which a random "coupon" (here an interNW region) is collected at each of T trials (here a row of LRs). One then asks how large T must be for each of C coupons to be collected with high probability. It is well-known that T must be proportional to $C \ln C$. In Section 6.3 we introduce variants of the coupon collector problem that are directly relevant to the analysis of mask-based decoders.

6.2.1 Models for Decoder Analysis

During manufacture, the n cycles of the mask-based decoder are placed using some number of masks. Associated with each mask is a phase difference, θ. The θ's are uniformly distributed independent random variables. Given θ, we know the nominal positions of all LR boundaries produced by that mask. We assume that each LR boundary varies independently about its nominal position according to some unimodal symmetric distribution centered at 0, $f(d)$.

We consider two models for assignment of cycles to masks. In the first, the **coarse-grained model**, we assume that the row of LRs under each MW are on separate masks. This model has $2Nn$ different masks. In the second, the **fine-grained model**, we assume each mask places one or more cycles of $2N$ MWs each. In both models, an independent phase difference random variable, θ_i, is associated with each mask. Given m masks, we use $\underline{\boldsymbol{\theta}}$ to denote the sequence of m phase differences, $\theta_1, \ldots, \theta_m$, associated with those masks.

A n-cycle mask-based decoder has g contact groups of N NWs. The decoder controls all $N' = gN$ NWs if each NW in each group of N NWs is individually addressable. For $1 \le i \le g$, let $F_i(\underline{\theta})$ denote the failure to control all N NWs associated with the i^{th} group of NWs given a value for $\underline{\theta}$. Conditioned on any given value of $\underline{\theta}$, the $F_i(\underline{\theta})$ are assumed independent.

Let $F(\underline{\theta})$ be the event that some NW in some group of g NWs is not individually addressable. It follows that $F(\underline{\theta})$ is the union of the events $F_i(\underline{\theta})$, $1 \le i \le g$. That is,

$$F(\underline{\theta}) = F_1(\underline{\theta}) \cup \cdots \cup F_g(\underline{\theta})$$

The unconditional probability of failure to control all N' NWs, $P(F)$, is the average of $P(F(\underline{\theta}))$ over all the m values of the phase difference.

$$P(F) = \left(\frac{1}{\rho}\right)^m \int_{-\rho/2}^{\rho/2} \cdots \int_{-\rho/2}^{\rho/2} P(F(\underline{\theta}))\, d\theta_1 \cdots d\theta_m$$

Below we use the principle of inclusion-exclusion to bound $P(F)$.

Theorem 6.2.1 *The probability $P(F)$ has the following bounds when $Q \le 1/2$.*

$$Q(1 - Q/2) < P(F) \le Q$$

where $Q = \rho^{-m} g \int_{-\rho/2}^{\rho/2} \cdots \int_{-\rho/2}^{\rho/2} P(F_1(\underline{\theta}))\, d\theta_1 \cdots d\theta_m$.

Proof By the principle of inclusion-exclusion (see Section 4.1), the conditional probability $P(F(\underline{\theta}))$ has the following bounds.

$$Q(\underline{\theta}) - \sum_{i<j} P(F_i(\underline{\theta}) \cap F_j(\underline{\theta})) \leq P(F(\underline{\theta})) \leq Q(\underline{\theta})$$

where $Q(\underline{\theta}) = \sum_{l=1}^{g} P(F_i(\underline{\theta}))$. Because the conditioned events $F_i(\underline{\theta})$ are assumed to be statistically independent, $P(F_i(\underline{\theta}) \cap F_j(\underline{\theta})) = P(F_i(\underline{\theta}))P(F_j(\underline{\theta}))$.

Let Q be the average of $Q(\underline{\theta})$, that is, $Q = \rho^{-m} \int_{-\rho/2}^{\rho/2} \cdots \int_{-\rho/2}^{\rho/2} Q(\underline{\theta})\, d\theta_1 \cdots d\theta_m$. Because the events $F_i(\underline{\theta})$ are identically distributed, $Q = g\overline{P(F_1(\underline{\theta}))}$ where $\overline{P(F_1(\underline{\theta}))}$ is as follows.

$$\overline{P(F_1(\underline{\theta}))} = \rho^{-m} \int_{-\rho/2}^{\rho/2} \cdots \int_{-\rho/2}^{\rho/2} P(F_1(\underline{\theta}))\, d\theta_1 \cdots d\theta_m$$

The sum $\sum_{i<j} P(F_i(\underline{\theta}) \cap F_j(\underline{\theta}))$ in the above lower bound has $g(g-1)/2$ terms. Each term $P(F_i(\underline{\theta}))P(F_m(\underline{\theta}))$ is a product of statistically independent and identically distributed random variables. Thus, its average over $\underline{\theta}$ is $g(g-1)\left(\overline{P(F_1(\underline{\theta}))}\right)^2/2$. Because $Q = g\overline{P(F_1(\underline{\theta}))}$, this average becomes $((g-1)/g)\, Q^2/2$ which is less than $Q^2/2$, giving the desired result. ■

Since the goal is to make Q very small, Q and $P(F)$ are very close. *In the remainder of this chapter we approximate the probability of failure to control all N' NWs by Q.*

Finally, recall that $F_i(\underline{\theta})$ is the event that between every pair of N NWs we collect at least one left LR boundary and one right LR boundary given the phase differences $\underline{\theta}$. Let L (R) be the event that some left (right) LR boundary fails to be collected. Then $P(L \cup R)$ is the probability that one or the other type of boundary fails to be collected. It follows that

$$\max(P(L), P(R)) \leq P(L \cup R) \leq P(L) + P(R)$$

Since there is no reason why left boundaries should be more or less difficult to collect than right boundaries, we can reasonably assume that $P(L) = P(R)$. This gives the following lemma.

Lemma 6.2.2 *The probability of a failure to collect both left LR and right LR boundaries between every pair of N' NWs is within a factor of two of the probability of a failure to collect just left (or right) LR boundaries between every pair of N' NWs.*

In light of this lemma, we can reasonably consider only the collection of only left LR boundaries. If $P(L)$ is small, then $P(R)$ is also small, as is $P(L \cup R)$. In Section 6.4 we model the collection of left LR boundaries as variants of the coupon collector problem. When there is one mask for each LR under each MW, this problem is modeled by the coupon collector problem with failures (Section 6.3.1). When all LRs are produced by one mask, this is modeled by the targeted coupon collector problem (Section 6.3.2). In the final case when multiple cycles are produced by multiple masks, the problem is a multi-stage version of the latter problem (Section 6.3.3).

6.3 Coupon Collection

In this section we analyze three increasingly general variants of the standard coupon collector problem: a) the coupon collector problem with failures, b) the targeted coupon collector problem, and c) the multi-stage targeted coupon collector problem. These generalizations are motivated by the cyclic placement of LRs in mask-based decoders. They are used in Section 6.4 to analyze the n-cycle mask-based decoder.

6.3.1 The Coupon Collector Problem with Failures

In the classic coupon collector problem, one of C coupons is randomly collected during each of T trials. Trials are independent and each coupon is selected with probability $1/C$. We introduce the **coupon collector problem with failures (CCF)** in which on each trial either a coupon fails to be collected with probability p_f (this models a LR boundary that falls outside of an interNW region) or a coupon is collected with probability $(1-p_f)/C$. T is chosen so that all coupons are collected with high probability.

Theorem 6.3.1 *Consider the coupon collector problem with failures, in which each trial has probability of failure p_f, and the probability of selecting the i^{th} coupon is $p_i = p_s/C$ for $1 \leq i \leq C$, where $p_s = 1 - p_f$. Let Γ_{CCF} be the probability of failing to collect all C coupons in T trials. Then, Γ_{CCF} and T satisfy the following bounds:*

$$z(1 - z/2) \leq \Gamma_{CCF} \leq z$$

where $z = C(1 - p_s/C)^T$. Let $\phi_{CCF} = -C\ln(1 - p_s/C)$. When z is small, minimizing z minimizes the bound on Γ_{CCF}. Then,

$$\frac{C}{\phi_{CCF}} \ln\left(\frac{C}{\Gamma_{CCF}(1+\Gamma_{CCF})}\right) \leq T \leq \frac{C}{\phi_{CCF}} \ln\left(\frac{C}{\Gamma_{CCF}}\right)$$

when $\Gamma_{CCF} \leq \sqrt{2} - 1$. ϕ_{CCF} satisfies $p_s \leq \phi_{CCF} \leq p_s(1 + p_s/C)$ if $C \geq 2$.

Proof Theorem 6.3.1 is a special case of Theorem 6.3.2 below. When $p_r = p_s/C$ for all r, z and ϕ are the same as defined above. ■

6.3.2 The Targeted Coupon Collector Problem

We further generalize the coupon collector problem by allowing each trial to "target" a certain coupon. We call this the **targeted coupon collector problem**. As in the coupon collector problem with failures, trials fail with probability p_f, but when a failure does not occur, each coupon is collected with a probability that is a function of the distance of the coupon from the targeted location. Let $p_0, p_1, ..., p_{C-1}$ be these probabilities, where $p_f + \sum_{r=0}^{C-1} p_r = 1$. The targeted coupon collector problem reduces to the coupon collector problem with failures when $p_r = p_s/C$ for all r.

Associated with each trial is a targeted coupon t_j, $1 \leq j \leq T$. The probability that the j^{th} trial collects the i^{th} coupon is $p_{r(i,j)}$, where $r(i,j) = (i - t_j) \bmod C$. This has the effect of targeting the coupons in a cyclic fashion.

To better understand this model of coupon collection, imagine C bins arranged in a circle. At each of T trials, a ball is thrown from directly overhead. A trial collects the i^{th} coupon if it lands in the i^{th} bin. Each throw is aimed at a particular bin, t_j. The likelihood that a ball hits its target is always p_0. The probability that a ball deviates one bin to the right is p_1. The probability that a ball deviates one bin to the left is p_{C-1}. The probability that a ball fails to land in any bin at all is p_f, which is independent of t_j.

As before, we wish to know how large T must be so that all coupons are collected with high probability. We are free to assign any value to each t_j, but we require these values to be chosen in advance. Each t_j cannot be based on the outcomes of previous trials. In our analysis we assume that each value of t_j is chosen an equal number of times and that T is a multiple of C. This is equivalent to cycling through all C coupons multiple times. Thus, we let $t_j = j \bmod C$ and call this the **cyclic targeted coupon collector** problem (CCC).

Theorem 6.3.2 *Consider the cyclic targeted coupon collector problem, in which each trial has probability of failure p_f and the probability of collecting the i^{th} coupon on the j^{th} trial is $p_{r(i,j)}$, where $r(i,j) = (i-j) \bmod C$. Let Γ_{CCC} be the probability of failing to collect all C coupons in T trials, where T is a multipole of C. Then, Γ_{CCC} and T satisfy the following bounds*

$$z(1 - z/2) \leq \Gamma_{CCC} \leq z$$

where $z = C\prod_{r=0}^{C-1}(1-p_r)^{T/C} = Ce^{-\phi_{CCC}T/C}$ and $\phi_{CCC} = -\sum_{r=0}^{C-1}\ln(1-p_r)$. When z is small, minimizing z minimizes the bound on Γ_{CCC}. Then,

$$\frac{C}{\phi_{CCC}}\ln\left(\frac{C}{\Gamma_{CCC}(1+\Gamma_{CCC})}\right) \leq T \leq \frac{C}{\phi_{CCC}}\ln\left(\frac{C}{\Gamma_{CCC}}\right)$$

when $\Gamma_{CCC} \leq \sqrt{2}-1$. $p_s \leq \phi_{CCC} \leq p_s + \sum_{r=0}^{C-1} p_r^2$ when $p_r \leq .5$ where $p_s = \sum_{r=0}^{C-1} p_r = 1 - p_f$. The bounds on T are minimized by maximizing ϕ_{CCC}.

Proof We use the principle of inclusion-exclusion (see Section 4.1). Let E_i be the event that i^{th} coupon is not collected after T trials and let $\Gamma_{CCC} = P(E_0 \cup \ldots \cup E_{C-1})$.

Let E_i' be the event that the i^{th} coupon is not collected after C trials. The probability that the i^{th} coupon is not collected on the j^{th} trial is $(1 - p_{r(i,j)})$, where $r(i,j) = (i-j) \bmod C$. In C consecutive trials, $r(i,j)$ will take on every value from 0 to $C-1$. Since trials are independent,

$$P(E_i') = \prod_{r=0}^{C-1}(1-p_r)$$

Now let E_i be the event that the i^{th} coupon is not collected in any of the T trials, T a multiple of C. Since $P(E_i) = P(E_i')^{T/C}$,

$$P(E_i) = \prod_{r=0}^{C-1}(1-p_r)^{T/C}$$

which is independent of i.

Now bound $P(E_h \cap E_i) = P(E_h' \cap E_i')^{T/C}$. Observe that the h^{th} and i^{th} coupons are not collected on the j^{th} trial with probability $(1 - p_{r(h,j)} - p_{r(i,j)})$. Since $(1-a-b) \leq (1-a)(1-b)$,

$(1 - p_{r(h,j)} - p_{r(i,j)}) \leq (1 - p_{r(h,j)})(1 - p_{r(i,j)})$. As before, over C consecutive trials, $r(h,j)$ and $r(i,j)$ range over all values from 0 to $C-1$. Reordering terms allows us to write,

$$P(E_h \cap E_i) = P(E'_h \cap E'_i)^{T/C} \leq \left[\prod_{r=0}^{C-1}(1-p_r)\prod_{r=0}^{C-1}(1-p_r)\right]^{T/C} \leq P(E_i)^2$$

Applying the principle of inclusion-exclusion we have that

$$\sum_{i=0}^{C-1} P(E_i) - \sum_{h<i} P(E_i)^2 \leq \Gamma_{CCC} \leq \sum_{i=0}^{C-1} P(E_i)$$

and since $\sum_{h<i} P(E_i)^2 \leq \left(\sum_{i=0}^{C-1} P(E_i)\right)^2/2$, this yields the bound

$$z(1 - z/2) \leq \Gamma_{CCC} \leq z$$

where $z = \sum_{i=0}^{C-1} P(E_i)$.

The inequality $z(1-z/2) \leq \delta$ implies that $z \leq 1 - \sqrt{1-2\delta}$. This in turn implies that $z \leq \delta(1+\delta)$ when $\delta \leq \sqrt{2}-1$ (since in this case $\sqrt{1-2\delta} \geq 1-\delta-\delta^2$). Thus, if $\Gamma_{CCC} \leq \sqrt{2}-1$

$$\Gamma_{CCC} \leq z \leq \Gamma_{CCC}(1+\Gamma_{CCC})$$

Substituting in $z = C\prod_{r=0}^{C-1}(1-p_r)^{T/C} = Ce^{-\phi_{CCC}T/C}$, where $\phi_{CCC} = -\sum_{r=0}^{C-1}\ln(1-p_r)$, gives,

$$\frac{C}{\phi_{CCC}}\ln\left(\frac{C}{\Gamma_{CCC}(1+\Gamma_{CCC})}\right) \leq T \leq \frac{C}{\phi_{CCC}}\ln\left(\frac{C}{\Gamma_{CCC}}\right).$$

Finally since $-x(1+x) \leq \ln(1-x) \leq -x$ when $x \leq .5$, $\sum_{r=0}^{C-1} p_r \leq \phi_{CCC} \leq \sum_{r=0}^{C-1}(p_r + p_r^2)$. Thus, $p_s \leq \phi_{CCC} \leq p_s + \sum_{r=0}^{C-1} p_r^2$ when $p_r \leq .5$, where $p_s = \sum_{r=0}^{C-1} p_r = 1 - p_f$. ■

It is of interest to know how sensitive the bounds on T are to the probability distribution $\{p_0, p_1, \ldots, p_{C-1}\}$. When all probabilities are the same, that is, $p_i = p_s/C$, the cyclic targeted coupon collector problem is equivalent to the coupon collector problem with failure described in Section 6.3.1. In this case, $\phi_{CCC} = \phi_{CCF} = -C\ln(1 - p_s/C)$.

Now consider a distribution that is far from uniform, one that is concentrated on just 3 points. If $p_0 = p_1 = p_2 = 1/4$ and $p_s = 3/4$, then $\phi_{CCC} = 3\ln(4/3) = .86$. On the other hand, for even small values of C, $\phi_{CCF} = -C\ln(1 - p_s/C) \approx p_s$. Since $p_s = 3/4$, ϕ_{CCF} and ϕ_{CCC} differ by only a small factor close to 1. In other words, even when each trial's ability to target a specific coupon is relatively good, the bounds on T continue to grow as $C\ln(C/\Gamma)$, where Γ is the probability of failing to collect all coupons.

6.3.3 The Multi-Stage Targeted Coupon Collector Problem

The cyclic targeted coupon collector problem is now generalized to m "stages" where each stage captures the variation introduced by using a new mask. In this problem, for some integer T_μ divisible by C, a **stage** is a set of T_μ trials where the j^{th} coupon, t_j, is targeted T_μ/C times. Associated with each stage is a uniformly distributed random variable $\theta \in [-\rho/2, \rho/2]$ such that the probability

of collecting a coupon targeted at a location i places away is $p_i(\theta)$, $0 \leq i \leq C-1$, a continuous function of θ. Also, $p_s(\theta) = 1 - p_f(\theta) = p_0(\theta) + \cdots + p_{C-1}(\theta)$ where $p_f(\theta)$ is the failure to collect any coupon on one trial. The random variables associated with the stages, $\underline{\theta} = (\theta_1, \theta_2, \ldots, \theta_m)$, are statistically independent. We call this the **multi-stage targeted coupon collector problem**.

Because this problem models an n-cycle mask-based decoder, we are free to consider putting either one or multiple cycles on one stage. Thus, we would like to know how the failure probability $\Gamma_{MM} = P(E_0 \cup E_1 \cup \ldots \cup E_{C-1})$ depends on the number of cycles per stage. We demonstrate that it is smallest when each stage contains one cycle.

Theorem 6.3.3 *Let Γ_{MS} be the probability of failure to collect all coupons in the multi-stage targeted coupon collector problem with m stages in T trials when there are T_μ cycles in the μ^{th} stage, T_μ a multiple of C, $1 \leq \mu \leq m$, and $T = T_1 + \cdots T_m$ where the stage random variables $\underline{\theta}$ are statistically independent. Then, Γ_{MS} and T satisfy the following bounds.*

$$z(1 - z/2) \leq \Gamma_{MS} \leq z$$

where $z = Ce^{-\phi_{MS}T/C}$,

$$\phi_{MS} = -\ln \prod_{\mu=1}^{m} \left(\frac{1}{\rho} \int_{-\rho/2}^{\rho/2} \left(\prod_{r=0}^{C-1} (1 - p_r(\theta_\mu)) \right)^{T_\mu/C} d\theta_\mu \right)$$

and

$$\frac{C}{\phi_{MS}} \ln \left(\frac{C}{\Gamma_{MS}(1 + \Gamma_{MS})} \right) \leq T \leq \frac{C}{\phi_{MS}} \ln \left(\frac{C}{\Gamma_{MS}} \right)$$

when $\Gamma_{CCC} \leq \sqrt{2} - 1$.

When z is small, minimizing z (by maximizing ϕ_{MS}) minimizes the bound on Γ_{MS}. The quantity z is minimized by placing each cycle in a separate stage in which case z satisfies the following bound.

$$z \geq C \left(\frac{1}{\rho} \int_{-\rho/2}^{\rho/2} \prod_{r=0}^{C-1} (1 - p_r(\theta))\, d\theta \right)^{T/C}$$

Proof We use the principle of inclusion-exclusion in which E_i is the event that the i^{th} coupon is not collected after T trials and we let $\Gamma_{MS} = P(E_0 \cup \ldots \cup E_{C-1})$.

We derive bounds on the failure event conditioned on the random variables $\underline{\theta}$, namely, $\Gamma_{MS}(\underline{\theta}) = P(E_0 \cup E_1 \cup \ldots \cup E_{C-1} \mid \underline{\theta})$ and then average the bounds over all values of $\underline{\theta}$.

Let E_i^μ be the event that the i^{th} coupon fails to be collected during T_μ trials in the μ^{th} stage. It follows that $E_i = E_i^1 \cap \cdots \cap E_i^m$ where $E_i^1, E_i^2, \ldots, E_i^m$ are statistically independent given the parameters $\underline{\theta}$. Thus we have

$$P(E_i \mid \underline{\theta}) = P(E_i^1 \mid \theta_1) \cdots P(E_i^m \mid \theta_m)$$

To employ the principle of inclusion-exclusion we must derive a bound on the conditional probability $P(E_h \cap E_i \mid \underline{\theta})$. Using the definition of these two events and the reasoning employed in the proof of Theorem 6.3.2 we have the following bound.

$$P(E_h \cap E_i \mid \theta_1, \theta_2, \ldots, \theta_m) \leq \prod_{\mu=1}^{m} P(E_i^\mu \mid \theta_\mu)^2$$

Here $P(E_i^\mu \mid \theta_\mu)$ is independent of i although it is dependent on θ_μ.

Averaging the bounds over $\underline{\theta}$ and applying the reasoning of the proof of Theorem 6.3.2, we have that $z(1 - z/2) \leq \Gamma_{MS} \leq z$ where

$$\begin{aligned} z &= \left(\frac{1}{\rho}\right)^m \int_{-\rho/2}^{\rho/2} \cdots \int_{-\rho/2}^{\rho/2} \sum_{i=1}^{C} P(E_i \mid \underline{\theta})\, d\underline{\theta} \\ &= \sum_{i=1}^{C} \prod_{\mu=1}^{m} \left(\frac{1}{\rho} \int_{-\rho/2}^{\rho/2} P(E_i^\mu \mid \theta_\mu)\, d\theta_\mu \right) \\ &= C \prod_{\mu=1}^{m} \left(\frac{1}{\rho} \int_{-\rho/2}^{\rho/2} \left(\prod_{r=0}^{C-1} (1 - p_r(\theta_\mu)) \right)^{T_\mu/C} d\theta_\mu \right) \\ &= C e^{-\phi_{MS} T/C} \end{aligned}$$

The latter result follows because $P(E_i^\mu \mid \theta_\mu)$ is independent of i.

A lower bound to z follows from a lower bound to $\frac{1}{\rho} \int_{-\rho/2}^{\rho/2} \left(\prod_{r=0}^{C-1} (1 - p_r(\theta_\mu)) \right)^{T_\mu/C} d\theta_\mu$. Holder's inequality is stated below where $1/p + 1/q = 1$ and $p, q \geq 1$.

$$\int_X |f(y)g(y)|\, dy \leq \left(\int_X |f(y)|^p\, dy \right)^{1/p} \left(\int_X |g(y)|^q\, dy \right)^{1/q}$$

Let $X = [-\rho/2, \rho/2]$, $f(y) = \left(\prod_{r=0}^{C-1} (1 - p_r(\theta_\mu)) \right)$ and $g(y) = 1/\rho$. Then, the inequality becomes the following.

$$\begin{aligned} \int_{-\rho/2}^{\rho/2} \frac{1}{\rho} f(y)\, dy &\leq \left(\int_{-\rho/2}^{\rho/2} f(y)^p\, dy \right)^{1/p} \left(\int_{-\rho/2}^{\rho/2} \rho^{-q}\, dy \right)^{1/q} \\ &= \left(\int_{-\rho/2}^{\rho/2} \frac{1}{\rho} f(y)^p\, dy \right)^{1/p} \end{aligned}$$

Here we have used the fact that $(1/q) - 1 = -1/p$. Consequently, when $p = T_\mu/C$

$$\frac{1}{\rho} \int_{-\rho/2}^{\rho/2} \left(\prod_{r=0}^{C-1} (1 - p_r(\theta_\mu)) \right)^{T_\mu/C} d\theta_\mu \geq \left(\frac{1}{\rho} \int_{-\rho/2}^{\rho/2} \prod_{r=0}^{C-1} (1 - p_r(\theta_\mu))\, d\theta_\mu \right)^{T_\mu/C}$$

which implies the following lower bound to z.

$$\begin{aligned} z &\geq C \prod_{\mu=1}^{m} \left(\frac{1}{\rho} \int_{-\rho/2}^{\rho/2} \prod_{r=0}^{C-1} (1 - p_r(\theta_\mu))\, d\theta_\mu \right)^{T_\mu/C} \\ &= C \left(\frac{1}{\rho} \int_{-\rho/2}^{\rho/2} \prod_{r=0}^{C-1} (1 - p_r(\theta))\, d\theta \right)^{T/C} \end{aligned}$$

This is the bound that applies when each cycle is placed on a separate stage. ■

6.4 Performance of the n-Cycle Mask-Based Decoder

In this section we bound the number of MWs required to control all NWs in a n-cycle mask-based decoder. As explained in Section 6.2.1, we consider two models for the random placement of LRs a) the coarse-grained model in which each row of LRs is placed independently using a separate mask, and b) the fine-grained model in which rows of LRs are placed using masks that contain one or more cycles. The course grained-model provides a very conservative upper bound on the number of MWs required to control all NWs with high probability. The fine-grained model provides an upper bound on the number of MWs required using more optimistic assumptions.

6.4.1 The Coarse-Grained Model

In the coarse-grained model each row of LRs is placed using a separate mask. We assume that mask displacement, d_{off}, can be at least 50-100nms. Since this is comparable to the width of each contact group, we model LR boundaries as untargeted. In other words, within a given contact group, the left (and right) LR boundary under a particular MW is equally likely to fall between any of the $N-1$ pairs of NWs.

During operation, the n-cycle mask-based decoder activates one of g contact groups, then uses M MWs to individually address one of the N NWs within that contact group. Theorem 6.2.1 provides tight bounds on the probability, $P(F)$, that not all $N' = gN$ NWs are individually addressable. $P(F)$ is upper bounded by $gP(F_i)$, where $P(F_i)$ is the probability that all N NWs in a particular contact group fail to be individually addressable.

By Lemma 6.2.1, all N NWs in a particular contact group are individually addressable if a left and a right LR boundary falls within each of the $N-1$ interNW regions. As explained at the end of Section 6.2, we can reasonable consider only the left LR boundaries. The requirement that there be a left LR boundary boundary within all $N-1$ InterNW regions is well-modeled by the coupon collector problem with failures. Here the probability of failing to collect all C coupons is bounded in Theorem 6.3.1. This allows us to upper bound M, the number of MWs required to individually address all N' NWs with probability $1 - P(F_{cg})$.

Theorem 6.4.1 *In the coarse-grained model, let $P(F_{cg})$ be the probability that all $N' = gN$ NWs in a masked-based decoder fail to be individually addressable. $P(F_{cg}) \leq \epsilon$ when M, the number of MWs in the decoder, is chosen as follows.*

$$M = \frac{N-1}{p_s} \ln \left[\frac{2(N'-g)}{\epsilon} \right]$$

Here $p_s = 1 - p_f$, where p_f is the probability that a particular LR boundary fails to fall within an InterNW region.

Proof As shown in Section 6.2 $P(F_{cg})$ is at most twice the sum of the probabilities of failing to collect all LR left boundaries within each of the g groups of N NWs. That is, $P(F_{cg}) \leq 2g\Gamma_{CCF}$ where Γ_{CCF} is the probability of failure to collect $C = N-1$ coupons when the i^{th} coupon is collected with probability $p_i = p_s/C$. Here $p_s = 1 - p_f$, where p_f is the probability of failing to collect any coupon.

If M is chosen so that $\Gamma_{CCF} = \epsilon/2g$, then $P(F_{cg}) \leq \epsilon$. We use the bounds of Theorem 6.3.1 to bound M when $\Gamma_{CCF} = (\epsilon)/(2g)$. In particular, if $M = (C/p_s)\ln(C/\Gamma_{CCF})$, $P(F_{cg}) \leq \epsilon$. ■

In the proof of this theorem, notice that when ϵ, and hence Γ_{CCF}, is small, Theorem 6.3.1 implies an upper bound on M that is very close to the implied lower bound.

Performance of the Model

In the coarse-grained model, the minimum number of MWs, M, required to ensure that all N' NWs in a masked-based decoder are individually addressable is very close $((N-1)/p_s)\ln(2(N'-g)/\epsilon)$. Here $p_s = 1 - p_f$, where p_f is the probability that, under a particular MW, no left LR boundary falls within one of the $N-1$ interNW regions contained in a particular contact group. We can reasonably assume that p_s is between 1/4 and 1/2. To see why, notice that under approximately half of the MWs, no left boundary appears within a given contact group (recall that the width and spacing of LRs both span N NWs). When a left boundary does lie within a contact group, it falls within an interNW region with probability close to 1/2. If contact groups are made to contain more than N NWs (say $2N$), or LRs could span fewer than N NWs (say $N/2$) then p_s would be closer to 1/2.

We now consider a concrete example in which $N = 8$, $g = 128$ and $N' = 1024$ and $\epsilon = .01$. In this case, if $p_s = 1/4$, $M = 339$, and if $p_s = 1/2$, $M = 170$. In either case, the number of MWs is large. In contrast, Chapters 4 and 5 demonstrated that randomized-contact and encoded NW decoders are able to control all $N' = 1024$ NWs using fewer than 50 MWs. Under the fine-grained model, the mask-based decoder is also able to employ fewer MWs.

6.4.2 The Fine-Grained Model

In the fine-grained model each mask contains one or more cycles of LRs. As in the coarse-grained model, the phase differences, θ_i, associated with the m masks are independent uniformly distributed random variables. In the fine-grained model, however, the cyclic placement of LRs ensures that each InterNW region is targeted by one LR left boundary, and one LR right boundary per cycle. As explained in Sections 6.1.3 and 6.1.4, the stochastic displacement of an LR boundary from its targeted location, d, is relatively small, and thus the fine-grained model is well-modeled by the multi-stage targeted coupon collector problem.

Before analyzing the fine-grained model using the multi-stage targeted coupon collector problem, we must address a small discrepancy. An n-cycle mask-based decoder, as defined thus far, contains N NWs (and hence $N-1$ interNW regions), but $2N$ MWs per cycle. As in our analysis of the course grained-model in Theorem 6.4.1, interNW regions correspond to coupons and MWs correspond to trials. In the multi-stage targeted coupon collector problem, however, cycles are supposed have the same number of trials as coupons. As such, we can either modify our definition of the n-cycle mask-based decoder so that OCs contain $2N+1$ NWs, or assume that the periodicity of LRs can be reduced to $N-1$ NWs. The former assumption is conservative, it makes it more difficult to individually address all NWs, the latter assumption is optimistic, it makes gaining control of NWs easier. We choose the latter.

In either case, the probability of failing to individually address all N NWs within a contact group is well-approximated as Γ_{MS}, the probability of failing to collect all C coupons in Theorem 6.3.3. As with the coarse-grained model, the probability of all N' NWs not being individually addressable, $P(F_{fg})$, is very close to $2g\Gamma_{MS}$. This allows us to upper bound M, the number of MWs required to individually address all N' NWs with probability $1 - P(F_{cg})$.

Theorem 6.4.2 *In the fine-grained model, let $P(F_{cg})$ be the probability that all $N' = gN$ NWs in a masked-based decoder fail to be individually addressable. As in Section 6.1.4, let $p_{i \bmod N-1}(\theta)$ be the probability, given a mask phase difference of θ, that a LR boundary moves i regions to the right (left) from its targeted interNW region, when i is positive (negative). $P(F_{cg}) \leq \epsilon$ when M, the number of MWs in the decoder, is chosen as follows,*

$$M = \frac{N-1}{\phi_{MS}} \ln \left[\frac{2(N'-g)}{\epsilon} \right]$$

where ϕ_{MS} is defined as

$$\phi_{MS} = -\ln \left(\frac{1}{\rho} \int_{-\rho/2}^{\rho/2} \prod_{r=2}^{N-2} (1 - p_r(\theta))\, d\theta \right)$$

and each cycle is placed on a separate mask.

Proof As in the proof of Theorem 6.4.1, we observe that $P(F_{cg})$ is at most twice the sum of the probabilities of failing to collect all LR left boundaries within each of the g groups of N NWs. That is, $P(F_{cg}) \leq 2g\Gamma_{MS}$ where Γ_{MS} is the probability of failure to collect $C = N - 1$ coupons in the multi-stage coupon collector problem with displacement probabilities $p_i(\theta)$.

If M is chosen so that $\Gamma_{MS} = \epsilon/2g$, then $P(F_{cg}) \leq \epsilon$. We use the bounds of Theorem 6.3.3 to bound M when $\Gamma_{MS} = \epsilon)/2g$. In particular, if $M = (C/\phi_{MS}) \ln(C/\Gamma_{MS})$, $P(F_{cg}) \leq \epsilon$. As noted in Theorem 6.3.3, ϕ_{MS} is maximized (and M minimized) by placing all cycles on separate masks. ■

The bound on M for the fine-grained case is identical to that given for the coarse-grained model except that the denominator term p_s is replaced by ϕ_{MS}. We approximate ϕ_{MS} using the following lemma.

Lemma 6.4.1 *The factor ϕ_{MS} satisfies the following bound where $p_s(\theta)$ is the probability that a LR left boundary falls into an interNW region.*

$$\phi_{MS} \leq -ln \left(\frac{1}{\rho} \int_{-\rho/2}^{\rho/2} 1 - \sum_{r=1}^{N-1} p_r(\theta) d\theta \right) = -ln \left(1 - \frac{1}{\rho} \int_{-\rho/2}^{\rho/2} p_s(\theta) d\theta \right)$$

where $p_s(\theta) = \sum_{r=0}^{N-2} p_r(\theta)$.

Proof The proof follows from the fact that $(1-a)(1-b) \geq (1-a-b)$. ■

Performance of the Model

As in Section 6.4.1, we assume that the width and spacing between NWs are equal, and thus that $\frac{1}{\rho}\int_{-\rho/2}^{\rho/2} p_s(\theta)d\theta$, the average value of p_s, is close to $1/2$. In this case, ϕ_{MS} is close to $\ln 2 = .7$. Recall that p_s, from the bound for the coarse-grained model, is about .5. Given that Theorems 6.4.1 and 6.4.2 are identical otherwise, we conclude that under the fined-grained the number of required MWs is reduced by a factor of approximately $(.5/.7) \approx 0.7$. Even under this much more optimistic model of decoder manufacture, $M = 122$ MWs are still required. The mask-based decoder appears inefficient even when the location of LR boundaries can be tightly controlled.

6.5 Additional Considerations

The mask-based decoder requires a large number of MWs to control all NWs with high probability. In this section we describe several practical considerations, relating to address translation, that may make mask-based decoders more attractive.

6.5.1 Address Translation Circuitry

To use a NW crossbar as a memory, each external binary address must be mapped to a different pair of orthogonal NWs. This requires programmable address translation circuitry (ATC) to map binary addresses to subsets of MWs (see Section 3.3.2). The ATC along each dimension of the NW crossbar maps the supplied binary string, B, to a contact group σ and a subset of MWs, $\boldsymbol{a}$.

For each B, the ATC must store a value for $\boldsymbol{a}$. The number of bits required for each $\boldsymbol{a}$ is at most M, since any subset of M MWs can be specified using M bits. M bits are necessary if most of the 2^M possible subsets are used with approximately equal frequency. This holds for both encoded NW decoders and randomized-contact decoders, which require $\Omega(M) = \Omega(\log N)$ bits of ATC storage per address.

In mask-based decoders, however, each NW can be addressed using just two MWs, one MW to turn off the NWs to its left and the other to turn off the NWs to its right. Since each $\boldsymbol{a}$ is a subset of two MWs, it can be stored using $2\log(M) = \Omega(\log N)$ bits. Even though mask-based decoders require a large number of MWs, they do not require significantly larger ATC than other decoders.

6.5.2 Alternative Addressing Strategies

In Section 6.4 M was bounded such that every NW in every contact group is individually addressable with probability $1 - \epsilon$. In previous chapters, this was referred to as the "All Wires Addressable" addressing strategy (see Section 3.3.2). As explained in Section 6.4, it implies that all N NWs in a particular contact group are individually addressable with probability approximately ϵ/g. As demonstrated in Chapters 4 and 5, the number of required MWs can be reduced if we relax this requirement.

All Wires Almost Always Addressable

The "All Wires Almost Always Addressable" addressing strategy (see Section 3.3.2) requires that all N NWs in *most* contact groups be individually addressable. To bound the number of MWs this strategy requires, we recall examples 4.1.2 and 5.2.5 of Chapters 4 and 5, respectively. If $g = 133$, and all N NWs in a particular contact group are individually addressable with probability .99, then all NWs in at least 128 of the contact groups are individually addressable with probability at least .99. In this case, a small fraction of the decoder's N' total NWs go unused, and the amount of ATC required increases slightly, but the number of required MWs is significantly reduced. The bounds in Theorems 6.4.1 and 6.4.1 are both reduced by a factor of .6 (compare $\ln(2*896/.01)$ to $\ln(2*7/.01)$). Of course this still implies that over 70 MWs are required, whereas for randomized-contact and encoded NW decoders 30 MWs or fewer suffice.

Take What You Get (TWYG)

A second addressing strategy worth considering is TWYG (see Section 3.3.2), which simply requires that N'_a addresses exist across all OCs, but places no limit on the number of addresses per OC. This increases the amount of ATC required, as an OC, σ, and MW activation pattern, $\boldsymbol{a}$, must be stored for each of the N_a addresses. Also $N' - N'_a$ of the NWs go unused. Even so, the number of required MWs can be reduced substantially. As in Chapters 4 and 5, Hoeffding's Inequality can be applied to bound N'_a in terms of g and M (see Section 4.2). To apply the Hoeffding's inequality, we must first establish a bound on $E[N_a]$, the expected number of addresses per contact group.

To bound $E[N_a]$, let $s_1, \ldots, s_{N_a}$, be the separately addressable groups of NWs, from left to right, within an OC. Notice that the rightmost NW of each s_i (with the exception of s_{N_a}) must have the left boundary of an LR immediately to its right. Similarly leftmost of each s_i (with the exception of s_1) must have the right boundary of an LR immediately to its left. Hence if we consider the interNW regions from left to right, a new separately addressable group appears only after the appearance of a left boundary, then a right boundary. Now to bound $E[N_a]$ we need only bound the expected number of interNW regions, $E[r]$, required for a left boundary followed by a right boundary to appear, since $E[N_a] = 1 + (N-1)/E[r]$.

Taking a conservative approach, we can look to the course grained model, where each LR boundary is equally likely to appear in any interNW region. Here the probability that a given LR's right (or left) boundary appears in an interNW region is $p_s/(N-1)$, and thus the probability that some LR's right boundary appears in the interNW region at least $1 - (1 - p_s/(N-1))^M$. This implies that if M is at least $(N-1)/p_s$, most interNW regions contain a right and left boundary, which in turn implies that $E[r] \leq 2$ and $E[N_a] \geq 1 + (N-1)/2$.

We now bound N_A in terms of $E[N_a] \geq N/\alpha$. First recall Theorem 4.2.2 in Section 4.2.

Theorem 6.5.1 *Let N'_a be the total number of addressable NWs in an NW decoder with g contact groups, N NWs per contact group, and $N' = gN$ NWs in total.*

$$P(N'_a \leq E[N'_a] - N'k) \leq e^{-2k^2 N' N/(N-1)^2} = e^{-2k^2 g^*}$$

for any $k \geq 0$ where $g^* = g(N/(N-1))^2$.

Proof See proof of Theorem 4.2.2 in Chapter 4. The proof applies to masked-based NW decoders as well. ■

From which we obtain a corollary

Corollary 6.5.1 *Let N'_a be the total number of addressable NWs in a NW decoder with g contact groups, N NWs per contact group, $N' = gN$ NWs in total, M MWs and at least an average of αN addresses per contact group.*

$$P(N'_a > \kappa N') \geq 1 - \epsilon$$

if $\kappa \leq 1 - \sqrt{-\ln \epsilon/(2g^*)} - \alpha$ where $g^* = g(N/(N-1))^2$.

Proof $E[N'_a] = gE[N_a] \geq N'\alpha$, and by the above theorem,

$$P(N'_a \leq N'\alpha - N'k) \leq e^{-2k^2 g^*}$$

Thus, if $k = \alpha - \kappa$, then

$$P(N'_a \leq \kappa N') \leq e^{-2g^*(\alpha-\kappa)^2}$$

Thus, when $e^{-2g^*(\alpha-\kappa)^2} \leq \epsilon$ the desired conclusion follows. This occurs when $\ln \epsilon \geq -2g^*(\alpha-\kappa)^2$ or $\sqrt{-\ln \epsilon/(2g^*)} \leq \alpha - \kappa$. ■

As an example, $g = 260$, $N = 8$, $N' = 2080$, $\epsilon = .01$, and $\kappa = .493$. As discussed above, when $M = 16$, we might conservatively assert that $\alpha > 1/2$, in which case $\kappa = .493 \leq 1 - \sqrt{-\ln .01/(2 * 260 * (8/7)^2)} - 1/2$. Thus at least $\lceil .493 * 2000 \rceil = 1025$ NWs are addressable with probability .99.

In this context, it is clear that the mask-based decoder looks significantly more appealing, but still less efficient than other NW decoders. Also, we have allowed the number of NW per address to vary, which may make decoder operation less reliable.

6.6 Summary of Decoder Analysis

In this and the previous two chapters, we have analyzed the number of MWs required to control N_A out of N NWs using masked-based, encoded NW, and randomized-contact decoders. In all three cases we have obtained tight bounds on the number of MWs, M, required to individually address all N NWs connected to a single OC. The bound obtained on M for masked-based decoders, $M \approx 1.4(N-1)\ln(2(N-1)/\epsilon)$, is by far the largest, as it is super-linear in N (see Section 6.4.2). In contrast, randomized-contact decoders require $M \approx 3.5 \ln(N(N-1)/\epsilon)$ MWs (see Section 4.4), and encoded NW decoders require $M \approx \log_2 C + 1/2 \log_2 \log_2 C + 1$ MWs, where $C = N(N-1)/(2\epsilon)$ (see Section 5.5). The number of MWs required by encoded NW decoders is smallest, although there is the additional challenge of manufacturing a large number of NW types.

The area required by all three types of NW decoders can be reduced by relaxing the requirement that all NWs connected to all (or almost all) OCs be individually addressable. We have bounded the number of MWs required to individually address $\kappa N'$ of the $N' = gN$ total NWs in compound NW decoders comprised of M MWs, g OCs, and N NWs per OC. In this case, the above discussion

following Corollary 6.5.1 reveals that mask-based decoders no longer requires M to be super-linear in N. Instead M may be a small multiple of N, provided the decoder's interNW regions are sufficiently large relative to the pitch of NWs. Even so, encoded NW decoders and randomized-contact decoders both appear more promising. For randomized-contact decoders $M = 3.5\ln((1-\kappa)/(N-1))$ MWs are sufficient as g increases (see Section 4.4). For Encoded NW decoders, M approaches $\log_2 C + 1/2\log_2\log_2 C + 1$, where $C = (N-1)/(1-\kappa)$ (see Section 5.5). Not only is this quite efficient in terms of M, but it also means that only a small number of NW types are required. Both randomized-contact and encoded NW decoders appear to be very viable options for providing efficient control over NW crossbar-based memories using the TWYG addressing strategy (defined in Section 3.3.2).

Chapter 7

Nanowire Addressing for Crossbar-based Logic

The previous three chapters have bounded the overhead associated with the stochastic assembly of NW decoders for use in crossbar-based memories. In this chapter we refer to such decoders as **NW memory decoders**. In a crossbar-based memory, NW memory decoders along each dimension of the crossbar must be able to individually address N_A NWs (or alternatively, address N_A disjoint subsets of NWs). As discussed in Section 3.1.2, however, the requirement imposed on NW decoders used to supply inputs to crossbar-based logic circuits is substantially stricter. These **NW logic decoders** are the focus of this chapter. We bound the area of stochastically assembled NW logic decoders and compare this area to that of a deterministic construction proposed by DeHon in [4].

Section 7.1 reviews the conditions that a NW logic decoder must satisfy to provide an interface to a nanoscale logic circuit with N_A inputs. Section 7.2 begins by quantifying the area of DeHon's proposed construction, then demonstrates how stochastically assembled RCDs and encoded NW decoders can use less area. Section 7.3 presents a novel lower bound on the area stochastically assembled NW logic circuit decoders require. Finally, Section 7.4 explains how the bounds of the previous two sections also apply to the area associated with the stochastically-assembled inversion and buffering layers that may be incorporated into programmable nanoscale logic circuits (see Section 2.2.3).

7.1 NW Decoders for Logic

In programmable NW crossbar-based logic circuits, as described in Section 2.2.3, there are two different ways in which NW decoders are used. First, the programmable NW interconnect within the crossbar-based logic must be configured. This can be accomplished by addressing pairs of orthogonal NWs using NW memory decoders. Second, when the logic circuit is used to compute, its N_A inputs must be supplied via MWs. In order to simultaneously specify all N_A inputs, a NW logic decoder is required.

As with memory decoders, NW logic decoders can be constructed stochastically. For a stochastically assembled memory decoder, we require that N_A NWs (or alternatively, N_A codewords) be individually addressable with high probability. In a logic decoder, a stricter condition is needed.

Consider a nanoscale circuit that takes N_A bits as input. The NW decoder used to supply those inputs must be able to address *arbitrary subsets* of a set of N_A NWs. If this condition holds, the set of N_A NWs is said to be **fully addressable**. As a generalization, one could also consider circuits that do not require all 2^{N_A} possible inputs.

In a nanoscale circuit, if N_A out of N input NWs are fully addressable, the remaining $N - N_A$ NWs should be ignored. In crossbar-based logic, this can be accomplished by configuring the programmable WIRED-OR portion of the circuit (see Section 2.2.3 and Figure 2.6) such that the $N - N_A$ input NWs are disconnected from all perpendicular output NWs. Initially, all N input NWs are disconnected from all output NWs. Each of the N_A fully addressable input NWs can then be connected to the appropriate output NWs via write operations. As noted above, a memory decoder is used to address individual output NWs during these write operations.

Arguably, one might expect that the logic decoder itself could be constructed via similar post-assembly configuration [40]. In this scenario, a stochastically assembled memory decoder would first be constructed to individually address N_A input NWs, then write operations would permanently couple each of the NWs to a different MW. If this approach proves technologically feasible, it would be quite efficient. The resulting logic decoder would consist of exactly N_A NWs and N_A MWs, and the only additional overhead required would be that associated with the stochastically assembled memory decoder. In the remainder of this chapter, we assume that connections between MWs and NWs are not programmable. If they are programmable, and the above construction is feasible, then the analysis of the previous three chapters bounds the required overhead. Furthermore, the bounds derived in the following two sections remain applicable to stochastically-assembled inversion and buffering layers (as discussed in Section 7.4).

7.2 Stochastic Assembly of NW Logic Decoders

In our analysis of memory decoders we considered compound NW decoders in which g OCs are each connected to w NWs. Here w is chosen to be as small as manufacturing constraints allow (approximately 10). As noted in Section 3.1.3, one approach for producing a set of N_A fully addressable NWs is to simply connect N_A OCs to groups of w NWs and use only a single NW from each group. This is the deterministic approach proposed by DeHon in [4]. It uses $M = N_A$ MWs to fully address N_A out of $N = 2wN_A$ NWs. Here the factor of 2 is approximate, it accounts for the space between the groups of w NWs. Depending on how tightly the lithographically produced OCs can be spaced, the factor may be closer to 1. Assuming MWs are at right angles to NWs, the deterministic approach to NW addressing uses area

$$A \approx (2wM\lambda)(Nw\lambda) \approx 4w^2N_A^2\lambda^2$$

where λ is the pitch of NWs and $2w\lambda$ is the pitch of MWs.

It is natural to ask whether a stochastically assembled NW decoder can use significantly less area. In such a decoder, multiple NWs from each OC would be used to form a subset of N_A fully addressable NWs. Notice, however, that this implies that when the decoder addresses most subsets of the N_A NWs, all or almost all OCs must be turned on. As a result, the g OCs could be replaced with a single larger OC without a significant reduction in the number of fully addressable NWs

(this reasoning is clarified at the end of the following subsection). In other words, if a simple NW decoder, consisting of a single OC, cannot outperform the deterministic construction described above, a compound NW decoder will not offer a significant improvement either. Consequently the analysis of this chapter focuses on simple NW decoders.

7.2.1 Unique Couplings

In order to analyze the probability that a simple decoder contains a set of N_A fully addressable NWs, we begin with a simple lemma. Given a set of N_A NWs and a set of N_A MWs, we say the sets are **uniquely coupled** if each of the N_A NWs is controlled by a unique MW. In other words, each of the N_A MWs provides individual control over a distinct NW. This directly relates to the criteria NW logic decoders must satisfy.

Lemma 7.2.1 *In a simple NW decoder, a set of N_A NWs is fully addressable if and only if their exists a set of N_A MWs to which it is uniquely coupled.*

Proof Let S be the fully addressable set of N_A NWs. For each NW $\boldsymbol{n_i}$, consider the set $S_i = S - \boldsymbol{n_i}$. Since S_i is addressable, there must be a MW that uniquely controls $\boldsymbol{n_i}$, but no other NW in S. For the other direction, simply note that any subset of S can be addressed by activating the MWs that uniquely control the NWs not in the set. ■

In order to bound the area required for stochastically assembled NW logic decoders, we wish to bound the number of MWs, M, and NWs, N, such that there exists a set of N_A fully addressable NWs with probability at least $1 - \epsilon$. By the above lemma, the probability that such a set exists is equal to the probability that there exists N_A NWs and N_A MWs such that each of the NWs is controlled by a unique MW. Furthermore, this condition applies even to decoders that contain errors (i.e. MWs that only partially control certain NWs).

Before proceeding with our area analysis, let us briefly revisit the assertion that it suffices to consider simple NW decoders. First notice that if a set of N_A NWs is uniquely coupled to N_A MWs, the N_A NWs are fully addressable whether or not they are all connected to a single OC. Now consider a compound NW decoder with g OCs, $N = gw$ total NWs and a set, S, of N_A fully addressable NWs. If NWs $\boldsymbol{n_i}, \boldsymbol{n_j} \in S$ are both connected to the same OC, then for the decoder to address the set $S_i = S - \boldsymbol{n_i}$ there must be a MW that controls $\boldsymbol{n_i}$, but no other NW in S. Thus for every NW in S that shares an OC with some other NW in S, there must be a MW that controls only that NW in S. It follows that if most NWs in S share an OC with at least one other NW in S, there must exist close to N_A uniquely coupled NWs and MWs.

This observation implies that if the g OCs were replaced with a single larger OC, the resulting simple decoder would still have close to N_A fully addressable NWs. Also, by replacing g OCs with a single OC, the added space between the g OCs can be eliminated. If additional NWs are then added such that N_A (as opposed to close to N_A) fully addressable NWs exist with probability at least $1 - \epsilon$, we can expect the resulting simple decoder to have close to the same area as the compound NW decoder it replaced. For this reason, the remainder of this chapter is focused on bounding the area of simple NW decoders.

7.2.2 Area Bounds

From the beginning of this section, recall that $2wMN\lambda^2$ is the area of a NW decoder with N NWs controlled by M perpendicular MWs. To minimize this area, our goal is to minimize MN while choosing M and N (along with any other decoder assembly parameters) such that their exists a set of N_A fully addressable NWs with probability at least $1 - \epsilon$. To outperform the deterministic construction proposed by DeHon, MN must be less than wN_A^2 (or $2wN_A^2$ if we assume there are w NWs between the N_A contact groups of w NWs).

To bound MN, we first consider the two extreme cases in which either $N = N_A$, or $M = N_A$. As we explain shortly, neither of these represent particularly efficient choices (and hence the accompanying analysis is kept relatively brief). We then consider a more efficient hybrid approach in which $N = M = \beta N_A$ for some relatively small value of β. This approach uses less area than the deterministic construction for realistic values of w.

In this section we consider both RCDs and encoded NW decoders. Mask-based decoders are not considered because they require that each MW controls (on average) groups of w consecutive NWs. This implies that close to $N = wN_A$ NWs are required before a stochastically assembled masked-based decoder even has a chance at fully addressing a set of N_A NWs.

Randomized-Contact Logic Decoder

From Chapter 4, recall that an RCD refers to any NW decoder in which NW/MW junctions are modeled as independent random variables. Each NW/MW junction is controlling with probability p, noncontrolling with probability q and in error with probability $r = 1 - p - q$. In an RCD, the probability that a particular NW is controlled by exactly one MW is $p_s = Mpq^{M-1}$.

Now consider the case when $M = N_A$. If we optimistically assume that $r = 0$, we see that $p_s = N_A p(1-p)^{N_A - 1}$ is maximized when $p = 1/N_A$. If we wish to account for errors, we can instead set $p = \alpha/N_A$ and $r = (1-\alpha)/N_A$, for some α relatively close to 1. This gives

$$p_s = \alpha(1 - 1/N_A)^{N_A - 1}$$

As N_A increases, p_s rapidly approaches $\alpha e^{-1}(1 - 1/N_A)^{-1}$ which in turn approaches αe^{-1} (and since N_A denotes the number of input bits to the nanoscale circuit, we expect it to be reasonable large).

So when $p = \alpha/N_A$ each NW is controlled by exactly one MW with constant probability, p_s, close to αe^{-1}. Since each NW is equally likely to be controlled by any of the N_A MWs, and NWs are coupled to MWs independently, we can bound the probability that the N_A MWs are uniquely coupled to some subset of N_A NWs using the "coupon collector problem with failures" (see Section 6.3.1 of the previous chapter). Here MWs correspond to coupons, NWs correspond to trials, and the probability that a trial fails to collect a coupon is $p_f = 1 - p_s = 1 - \alpha e^{-1}(1 - 1/N_A)^{-1}$. Theorem 6.3.1 immediate reveals that the desired unique coupling exists with probability $1 - \epsilon$ when $N = (N_A/p_s)\ln(N_A/\epsilon)$, which gives

$$N = (eN_A^2/\alpha(N_A - 1))\ln(N_A/\epsilon)$$

Furthermore, the above analysis is unchanged if $N = N_A$. The rolls of NWs and MWs are merely reversed. Instead of MWs acting as coupons, NWs act as coupons and MWs act as trials. In this case $M = (eN_A^2/\alpha(N_A - 1)) \ln(N_A/\epsilon)$, and in either case we have

$$MN \approx (e/\alpha)N_A^2 \ln(N_A/\epsilon)$$

Even if no codeword errors occur, in which case $\alpha = 1$, this will not outperform the deterministic construction (for which $MN \leq 2wN_A^2$) unless $2w > e \ln(N_A/\epsilon)$. Setting $w = 10$ and $\epsilon = .01$, requires that $N_A < 16$.

Encoded NW Logic Decoder

In an encoded NW decoder, the distribution with which codewords are assigned is determined by how NWs are encoded (see Chapter 5). As explained in Section 5.1, $(1, M)$-hot encodings ensure that each NW is controlled by exactly one MW (in the absence of a misalignment error). When $M = N_A$ MWs are used, this is an ideal choice. Once again we have an instance of the coupon collector problem with failures in which MWs correspond to coupons and NWs correspond to trials. In the absence of misalignment errors, each trial collects a coupon and $p_f = 0$. Otherwise p_f denotes the probability of a misalignment error (see Section 5.3). In either case we have

$$N = N_A/p_s \ln(N_A/\epsilon)$$

where $p_s = 1 - p_f$. When p_f is relatively small (e.g. less than .25) this does out perform the deterministic construction for reasonable values of N_A, but the gains are modest. For example, when $N_A = 128$, $p_s = .9$, and $\epsilon = .01$, $N = 10.5N_A$. This gives $NM = 10.5N_A^2$, whereas for the deterministic construction, when $w = 10$, $MN < 20N_A^2$.

Now consider the case when $N = N_A$. In contrast with the analysis of RCDs, this case is not identical. For instance, it is no longer guaranteed that each MW is coupled to exactly one NW. Also, since MWs now correspond to trials and NWs to coupons, trials can no longer be viewed as independent. Even so, inclusion-exclusion can still be used bound the number of trials, M, required to collect all coupons with probability ϵ. The key observation is that the relevant probabilities (the probability that a particular coupon fails to be collected, and the probability that a particular pair of coupons fail to be collected) both approach that of an RCD as N_A increases (here $p = h/M$ and $q = 1 - h/M$, assuming (h, M)-hot codes are used).

A Hybrid Approach

For both RCDs and encoded NW decoders, setting $N = N_A$ or $M = N_A$ yields a decoder area that is $O(\ln(N_A/\epsilon)N_A^2)$. This offers, at best, a modest improvement over DeHon's proposed deterministic construction, for which $MN = O(w^2N_A^2)$. Fortunately, it is possible to obtain a better area bound by employing a hybrid approach in which $N = M = \beta N_A$ for some relatively small value of β.

First, consider an encoded NW decoder in which $(1, M)$-hot codes are used. In the absence of misalignment errors, each NW is controlled by exactly one MW. As above, we can view MWs as coupons and NWs as trials, but now only require that C/β of the $C = M = \beta N_A$ coupons be

collected. To see the promise of this approach, let t_i be the number of trials required to collect the i^{th} coupon after the $(i-1)^{th}$ coupon has been collected (here $t_1 = 1$). When $i-1$ coupons have been collected, the probability that a trial collects a new coupon is $(C-i+1)/C$, so $E[t_i] = C/(C-i+1)$.

The number of trials required to collect κC coupons is $T_\kappa = \sum_{i=1}^{\kappa C} t_i$, and thus expected number of trials is

$$E[T_\kappa] = \sum_{i=1}^{\kappa C} C/(C-i+1) = C(1/C + \ldots + C/(C-\kappa C+1)) = C(\mathcal{H}(C) - \mathcal{H}(C-\kappa C))$$

where $\mathcal{H}(N) = 1 + 1/2 + \ldots + 1/N$. It is well-known that $\ln n \leq \mathcal{H}(n) \leq \ln n + 1$ [61, p. 33] and that $\mathcal{H}(n) - \mathcal{H}(\alpha n)$ approaches $\ln n - \ln \alpha n = -\ln \alpha$ as n increases. This reveals that $E[T_\kappa] \approx -C\ln(1-\kappa)$. Thus for fixed $\kappa < 1$, only $O(C)$ trials, on average, are required to collect κC coupons.

In the case of an encoded NW logic decoder for which $M = \beta N_A$, this bounds the expected number of NWs required to collect $(1/\beta)M = N_A$ coupons. To bound the number of NWs required with probability $1-\epsilon$, we use the following lemma.

Lemma 7.2.2 *Consider the classic coupon collector problem in which one of C coupons is collected independently at random during each of T trials. Each coupon is collected with equal probability.*

Let S_C denote the number of distinct coupons collected after $T = C$ trials. Then for $0 > \kappa > 1$

$$P[S_C < \kappa C] \leq (e^{-\delta}/(1-\delta)^{1-\delta})^{C(1-\kappa+\kappa^2/2)}$$

where $\delta = (1 - \kappa/(1-\kappa+\kappa^2/2))$.

Proof Let x_i be a 0-1 random variable that denotes whether a new coupon is collected during the i^{th} trial. Notice that $p(x_i = 1) \geq (C-i+1)/C$, since by the i^{th} trial at most $i-1$ coupons have already been collected. Furthermore, if fewer than κC coupons have been collected by the i^{th} trial, $p(x_i = 1) > 1-\kappa$.

We wish to bound the probability that $S_C = \sum_{i=1}^{C} x_i < \kappa C$. To do this, we instead consider the sum of C *independent* 0-1 random variables, y_i. Here $p(y_i = 1) = (C-i+1)/C$ for $i \leq \kappa C$ and $p(y_i = 1) = 1-\kappa$ for $i > \kappa C$. Let $S'_C = \sum_{i=1}^{C} y_i$. By the logic of the previous paragraph, $P[S_C < \kappa C] \leq P[S'_C < \kappa C]$.

Since S'_C is the sum of independent random variables, $P[S'_C < \kappa C]$ can be bounded using a Chernov bound (see Section 5.2.1):

$$Pr(S'_C \leq (1-\delta)E[S'_C]) \leq (e^{-\delta}/(1-\delta)^{1-\delta})^{E[S'_C]}$$

where $\delta = 1 - \kappa C/E[S'_C]$.

Here $E[S'_C] = \sum_{i=1}^{C} E[y_i] = \sum_{i=1}^{\kappa C}(C-i+1)/C + \sum_{i=\kappa C+1}^{C} 1-\kappa > C(1-\kappa/2)\kappa + C(1-\kappa)(1-\kappa) = C(1-\kappa+\kappa^2/2)$. This gives $\delta = 1 - \kappa/(1-\kappa+\kappa^2/2)$, the desired result. ■

Using the above lemma, we can easily consider the case when $M = N = (5/2)N_A$, in which case $\kappa = 2/5$, $1-\kappa+\kappa^2/2 = 17/25$ and $(1-\kappa/(1-\kappa+\kappa^2/2)) = 7/17$. This gives

$$P[S_C < 1/2C] \leq (e^{-7/17}/(10/17)^{10/17})^{17C/25}$$

which is less than 0.0131 when $C = N_A \geq 64$. As N_A increases further, a larger value of κ can be used. When $\kappa = 1/2$, we have $1 - \kappa + \kappa^2/2 = 5/8$ and $(1 - \kappa/(1 - \kappa + \kappa^2/2)) = 1/5$. This gives

$$P[S_C < 1/2C] \leq (e^{-1/5}/(4/5)^{4/5})^{5C/8}$$

which is less than .01 when $C \geq 343$.

Thus if we consider an error-free encoded NW decoder where $M = N = \beta N_A$, and choose β such that a set of at least N_A MWs exists with probability $1 - \epsilon$, then $MN < (5/2)^2 N_A^2$ when $N_A \geq 64$, and $MN < 2^2 N_A^2$ when $N_A \geq 343$. This outperforms DeHon's deterministic construction, for which $MN > w^2 N_A^2$, if $w > (2.5)^2 = 6.25$ in the first case and $w > 4$ in the second case. Lemma 7.2.2 also yields the following asymptotic result.

Theorem 7.2.1 *Consider an error-free encoded NW decoder with M MWs and N NWs using $(1, M)$-hot encodings. For any $\epsilon > 0$ and $\beta > 1/(2 - \sqrt{2})$ there exists a threshold $N_{\epsilon,\beta}$, such that if $N_A \geq N_{\epsilon,\beta}$ and $M = N = \beta N_A$, then there exists uniquely coupled sets of N_A MWs and N_A NWs with probability at least $1 - \epsilon$.*

Proof In the encoded NW decoder, each NW is controlled by exactly one randomly selected MW. As such, each NW can be thought of as collecting one of $C = M = \beta N_A$ coupons independently at random and with equal probability. We wish to guarantee that at least C/β distinct coupons are collected among the $N = C$ independent trials, given that each trial collects each coupon with probability $1/C$.

Let S_C denote the number of distinct coupons collected after C trials, and let $\kappa = 1/\beta$. In the proof of Lemma 7.2.2, it is shown that

$$Pr(S_C \leq (1 - \delta)E[S'_C]) \leq (e^{-\delta}/(1 - \delta)^{1-\delta})^{E[S'_C]}$$

where $E[S'_C] = C(1 - \kappa + \kappa^2/2)$ and $\delta = 1 - \kappa C/E[S'_C]$. This implies that for any fixed $\delta > 0$, $Pr(S_C \leq (1 - \delta)E[S'_C])$ approaches 0 as C increases. Requiring that $\delta > 0$ is equivalent to requiring that $1 - \kappa/(1 - \kappa + \kappa^2/2) > 0$, or $1 > \kappa/(1 - \kappa + \kappa^2/2)$. This becomes $\kappa^2 - 4\kappa + 2 > 0$, which holds when $\kappa < 2 - \sqrt{2}$. ■

Note: The bounds on β given above apply to an encoded NW decoder that is error free. If misalignment errors occur with probability p_f, then for any particular value of N_A, the corresponding required value of β scales by a factor of at most $1/(1 - p_f)$. To see why, notice that in the proof of Lemma 7.2.2, $E[x_i]$ and $E[y_i]$ are simply scaled by a factor of $1/(1 - p_f)$, as are $E[S_C]$ and $E[S'_C]$. This same scaling bound on β applies to RCDs, for which $p_f \approx 1 - \alpha e^{-1}$ when $p = \alpha/N$.

7.3 Lower Bounding β

As above, consider a stochastically assembled NW decoder with M MWs and N NWs, in which each NW is controlled by exactly one NW with probability $1 - p_f$. The previous section demonstrated that setting $M = N = \beta N_A$, where $\beta > 1/((1 - p_f)(2 - \sqrt{2}))$, allows for a logic decoder with N_A outputs and area $O(MN) = O(\beta^2 N_A^2)$. This section provides an information theoretic lower bound

on β as N_A increases. Part of the appeal of our approach is that it can potentially be applied to other stochastically assembled structures as well.

To begin, let the **configuration**, $\mathcal{C}$, of a decoder denote the state of its MN MW/NW junctions (i.e. codewords $\boldsymbol{c^1} \ldots \boldsymbol{c^N}$). A configuration is **successful** if it contains sets of N_A NWs and N_A MWs, $\mathcal{N}$ and $\mathcal{M}$, that are uniquely coupled. When a NW decoder is stochastically assembled, let $1-\epsilon$ be the probability that the resulting configuration is successful.

The basic approach used to lower bound β is relatively straightforward. Before a NW decoder is assembled, there is a probability distribution associated with $\mathcal{C}$. Depending on the parameters of the assembly process, there is a certain amount of entropy (i.e. uncertainty), denoted $h(\mathcal{C})$, associated with $\mathcal{C}$. For RCDs and encoded NW decoders $h(\mathcal{C})$ is easy to compute. Given β, it is possible to upper bound the entropy of $\mathcal{C}$ given that $\mathcal{C}$ is successful. This bound implies a lower bound on β. Specifically, β must be large enough so any upper bound on the entropy of all successful configurations is at least $(1-\epsilon)h(\mathcal{C})$.

More formally, imagine the repeated assembly of a stochastically assembled NW decoder with M MWs and N NWs. Here $h(\mathcal{C})$ represents the minimum number of bits, on average, required to specify $\mathcal{C}$ after each assembly process, among all possible configurations. In other words, suppose that after each decoder is assembled its configuration, $\mathcal{C}$, is recorded in binary using a predetermined encoding scheme. For any such scheme, the average number of bits required per decoder is at most $h(\mathcal{C})$. Also, the bound is asymptotically achievable [65].

If $\mathcal{C}$ is successful with probability $1-\epsilon$, Shannon's source coding theorem implies that as ϵ shrinks and $MN = \beta N_A^2$ increases, the entropy of $\mathcal{C}$, when restricted to only successful configurations, approaches $(1-\epsilon)h(\mathcal{C})$ [65]. This in turn implies that for arbitrarily large values of N_A, the average number of bits required by an encoding scheme that describes only successful configurations is at least $(1-\epsilon)h(\mathcal{C})$.

In a successful decoder, let $\mathcal{S}$ denote the set of the N_A^2 junctions of the uniquely coupled sets $\mathcal{M}$ and $\mathcal{N}$. Also let $\mathcal{C}-\mathcal{S}$ denote the set of the remaining $MN-N_A^2$ junctions. To obtain a lower bound on β, we observe that the average number of bits required to specify a successful configuration is at most the average number of bits required to specify $\mathcal{S}$, denoted $h(\mathcal{S})$, plus the average number of bits required to specify $\mathcal{C}-\mathcal{S}$ given $\mathcal{S}$, denoted $h(\mathcal{C}-\mathcal{S}|\mathcal{S})$. As explained above, the average number of bits required to specify a successful configuration is at least $(1-\epsilon)h(\mathcal{C})$. Thus

$$(1-\epsilon)h(\mathcal{C}) \leq h(\mathcal{S}) + h(\mathcal{C}-\mathcal{S}|\mathcal{S}) \tag{7.1}$$

which we now apply to both RCDs and encoded NW decoders. For simplicity we consider the bound (which holds for all $\epsilon > 0$) as $\epsilon \to 0$. This gives:

$$h(\mathcal{C}) \leq h(\mathcal{S}) + h(\mathcal{C}-\mathcal{S}|\mathcal{S}) \tag{7.2}$$

7.3.1 A Lower Bound for RCDs

To lower bound β for RCDs we can assume decoders are error-free. Here $c_j^i = 1$ with probability p and $c_j^i = 0$ with probability $1-p$. The c_j^i are independent random variables, so $h(\mathcal{C}) = MNh(p)$, where $h(p) = -p\log p - (1-p)\log(1-p)$ is the binary entropy function [65] (log is base 2).

$h(\mathcal{S})$ is upper bounded below. $h(\mathcal{C}-\mathcal{S}|\mathcal{S}) \leq (MN-N_A^2)h(p^*)$, where p^* is the probability that a given junction in $\mathcal{C}-\mathcal{S}$ is controlling. From inequality 7.2, this gives

$$MNh(p) \leq h(\mathcal{S}) + (MN-N_A^2)h(p^*) \tag{7.3}$$

which implies a bound on β in terms of ϵ and N_A, since $h(\mathcal{S})$ and p^* are both functions of β, N_A and ϵ.

To compute p^*, note that in $\mathcal{C}-\mathcal{S}$ exactly N_A controlling junctions are removed from $\mathcal{C}$. Thus $p^* = (MNp-N_A)/(MN-N_A^2)$. Since $MN = \beta^2 N_A^2$ this gives $p^* = (\beta^2 N_A^2 p - N_A)/(N_A^2(\beta^2-1)) = (\beta^2 p - 1/N_A)/(\beta^2-1)$. Thus for fixed p, p^* approaches $p\beta^2/(\beta^2-1)$ as N_A increases. Also $p \leq p^* \leq p\beta^2/(\beta^2-1)$ when $p \geq 1/N_A$.

The average number of bits required to specify $\mathcal{S}$, $h(\mathcal{S})$, is at most the number of bits required to specify $\mathcal{N}$ and $\mathcal{M}$ plus the number required to give an ordering of one of the sets (this specifies which MW is coupled to each NW). This requires $\log\binom{M}{N_A} + \log\binom{N}{N_A} + \log N_A!$ bits. Inequality 7.3 becomes

$$MNh(p) \leq \log\binom{M}{N_A} + \log\binom{N}{N_A} + \log N_A! + (MN-N_A^2)h(p^*)$$

Stirling's approximation tells us $\log N_A!$ rapidly approaches $N_A \log N_A - N_A \log e + \frac{1}{2}\log(2\pi N_A)$. This implies that $\log\binom{\beta N_A}{N_A} = \log(\beta N_A)! - \log(\beta N_A - N_A)! - \log N_A! < \beta N_A h(1/\beta)$. From this we get $MNh(p) - MNh(p^*) + N_A^2 h(p^*) \leq 2\beta N_A h(1/\beta) + N_A \log N_A - N_A \log e + \frac{1}{2}\log(2\pi N_A)$. Since $MN = \beta^2 N_A^2$ we have

$$N_A\left(\beta^2 h(p) - (\beta^2-1)h(p^*)\right) \leq 2\beta h(1/\beta) + \gamma(N_A) \tag{7.4}$$

where $\gamma(N_A) = \log N_A - \log e + \frac{1}{2N_A}\log(2\pi N_A)$. This implies a lower bound β given N_A and p. To obtain an explicit bound, we show the implied bound is weakest when p is small.

Observe that as N_A increases, $p \to 0$ if β remains constant (i.e. it is not possible for $MN = O(N_A^2)$ unless $p \to 0$). To see why, notice that for fixed β the right-hand side of the above inequality increases logarithmically in N_A. The left-hand side, however, increases linearly in N_A unless the coefficient $\beta^2 h(p) - (\beta^2-1)h(p^*)$ goes to zero. This implies that as N_A increases, $\beta^2 h(p) - (\beta^2-1)h(p^*)$ must approach 0 for the inequality to hold. Since $p^* \to p$ as N_A increases, $h(p^*) \to h(p)$ and $\beta^2 h(p) - (\beta^2-1)h(p^*) \to h(p)$. Thus as N_A increases $h(p)$, and hence p, must go to 0.

Having established that p goes to zero as N_A increases, we now consider two cases: $p \geq 1/N_A$ and $p \leq 1/N_A$. The second case is considered below. In the first case $p \leq p^* \leq p\beta^2/(\beta^2-1)$ and as we now show, the expression $C(p) = \beta^2 h(p) - (\beta^2-1)h(p^*)$ is smallest when $p = 1/N_A$.

When $p = 1/N_A$ we have $p^* = p$ and $C(p) = h(1/N_A)$. Since $p^* = (\beta^2 p - 1/N_A)/(\beta^2-1)$, $\frac{dp^*}{dp} = \beta^2/(\beta^2-1)$. Now consider the derivative of $C(p)$. $C'(p) = \beta^2 h'(p) - \frac{dp^*}{dp}(\beta^2-1)h'(p^*) = \beta^2 h'(p) - \beta^2 h'(p^*) = \beta^2(h'(p) - h'(p^*))$. Here $h(p) = -p\log p - (1-p)\log(1-p)$ and $h'(p) = \log((1-p)/p)$, so $h'(p) > h'(p^*)$ when $p < p^* < 1/2$. Thus for $p > 1/N_A$, $C'(p) > 0$ and when $p \geq 1/N_A$, $C(p)$ is smallest when $p = 1/N_A$. Inequality 7.4 now becomes

$$N_A h(1/N_A) \leq 2\beta h(1/\beta) + \log N_A - \log e + (1/2N_A)\log(2\pi N_A)$$

Since $\log(1-1/N_A) \leq -1/N_A$, we have $h(1/N_A) \geq 1/N_A \log N_A + 1/N_A - 1/N_A^2$ and thus

$$1 + \log e \leq 2\beta h(1/\beta)$$

which reveals that $\beta > 1.25$ if $p \geq 1/N_A$ and N_A increases.

Finally, we return to the case when $p \leq 1/N_A = \beta/N$. The probability that a NW isn't controlled by any MW is $(1-\beta/N)^N$, which rapidly approaches $e^{-\beta}$ as N_A, and hence N, increases. Since on average $e^{-\beta}$ NWs are not controlled by any MW, $(1-e^{-\beta})N \geq N_A$ for a unique coupling to exist with high probability. Since $N = \beta N_A$ we have $(1-e^{-\beta})\beta \geq 1$, which implies that $\beta > 1.349$. For an RCD with $M = N = \beta N_A$, we have demonstrated that $\beta > 1.25$ as $\epsilon \to 0$.

7.3.2 A Lower Bound for Encoded NW Decoders

The above approach can also be adapted for encoded NW decoders with (h, M)-hot codes. Here, in the absence of misalignment errors, $\mathcal{C}$ has entropy $N \log \binom{M}{h}$, $\mathcal{S}$ once again has entropy at most $\log \binom{M}{N_A} + \log \binom{N}{N_A} + \log N_A!$ and $\mathcal{C}-\mathcal{S}$, given $\mathcal{S}$, has entropy at most $(N-N_A)\log \binom{M}{h} + N_A \log \binom{M}{h-1}$. Inequality 7.2 then yields

$$N_A \log \binom{M}{h} \leq \log \binom{M}{N_A} + \log \binom{N}{N_A} + \log N_A! + N_A \log \binom{M}{h-1}$$

or

$$\log \binom{M}{h} - \log \binom{M}{h-1} \leq 2\beta h(1/\beta) + \log N_A - \log e + \frac{1}{2N_A} \log(2\pi N_A)$$

and since $\log \binom{M}{h} - \log \binom{M}{h-1} = \log \binom{M}{h} / \binom{M}{h-1} = \log(M-h+1)/h$, we have

$$\log(\beta N_A - h + 1)/h - \log N_A + \log e \leq 2\beta h(1/\beta) + \frac{1}{2N_A} \log(2\pi N_A)$$

Finally, since $\log(\beta N_A - h + 1)/h - \log N_A = \log(\beta/h - 1/N_A + 1/hN_A)$, as N_A increases we have

$$\log(\beta/h) + \log e \leq 2\beta h(1/\beta) \tag{7.5}$$

If $h = 1$, this becomes $\log e \leq 2\beta h(1/\beta) - \log \beta$, which implies that $\beta > 1.24$. When $h = 2$ or more, $\log(\beta/h)$ changes sign and the bound becomes quite weak. Still, as N_A increases, we would not expect $h \geq 2$ to outperform $h = 1$ when $M = N$. It is also possible to consider a mix of $(1, M)$-hot and $(2, M)$-hot encoded NWs, but it is unclear if this yields a smaller value of β.

7.4 Stochastic Crossbar Interconnect

A stochastically assembled logic decoder requires sets of N_A MWs and N_A NWs such that the MWs are uniquely coupled to the NWs. This is exactly the same condition required for stochastically assembled inversion or buffering layers within NW crossbar logic (see Section 2.2.3). The only difference is that instead of considering MWs coupled to NWs, we now consider input NWs coupled to a second set of output NWs.

To construct inversion and buffering layers, DeHon has suggested the use of $(1, M)$-hot encodings [4]. The analysis of the previous two sections shows that the overhead associated with stochastic

assembly of the inversion or buffering layer is a small constant factor, even when NW misalignment errors occur with probability p_f. Specifically, if N input NWs are coupled to N output NWs, and $N = \beta N_A$, β can be close to $(1 - p_f)^{-1}/(2 - \sqrt{2}) \approx 1.71$, if not smaller. Here p_f denotes the probability that an output NW is misaligned.

Chapter 8

Nanowire Address Discovery

Previous chapters have bounded the area required to implement a stochastically assembled NW decoder along with the mesoscale address translation circuity (ATC) required to control it. In order to provide a consistent external interface to the NW decoder, the ATC's programmable storage must contain information about which MWs control which NWs (see Section 3.3.2). Recall from Section 3.3, however, that an address discovery procedure is require before the ATC can be properly configured. In other words, testing is required after a NW decoder is stochastically assembled in order to determine which NW codewords are present.

This chapter investigates two general approaches to address discovery. The first approach, discussed in Section 8.1, relies on read/write operations to test whether a particular codeword is present. The second approach, which is the focus of the remainder of the chapter, relies only on measuring whether the current flowing across all N NWs within a contact group is above some preselected threshold. Although the read/write approach to testing is simple and efficient, its reliance of nanoscale storage is problematic. Tests using read/write operations are relatively time consuming and possibly unreliable (since the nanoscale storage may itself require testing). More importantly, read/write operations are not even possible if the NWs being tested are not connected to nanoscale storage.

To avoid a reliance on nanoscale storage, most of this chapter focuses on conductance-based tests. Here a voltage is applied across the N NWs within a single contact group, a subset of MWs is activated, then current is measurement to determine if any NW remains conducting [55, 56] (i.e. if the N NWs collectively carry a current that is above some threshold). Such a test does not reveal which NW is on, nor does it reveal if multiple NWs are on. Nonetheless, as explained in Section 8.2, it is sufficiently powerful to determine which subsets of MWs address individual NWs. The main challenge surrounding this approach is to bound the number of current measurements it requires. Section 8.3 shows that when encoded NWs are used, it is possible to discover each NW's codeword with optimal efficiency. Section 8.4 then considers the more challenging problem of discovering each NW's codeword when arbitrary codewords may be present. We demonstrate through asymptotic analysis, as well as experimental simulation, that efficient testing remains possible. We also connect this work to research being done on the learning of monotone DNFs.

8.1 Address Discovery via Read/Write Operations

In this section, we explain how NW codewords can be discovered via read/write operations. We demonstrate that if read/write-based testing is deemed feasible, the resulting address discovery algorithm is simple and efficient. To discover the N codewords within a particular contact group via read/write operations, two contact groups must be addressed, one in each dimension of the crossbar-based memory. In one of the contact groups all N NWs can be addressed whenever a read or write operations is performed (i.e. none of the controlling MWs are activated). In the other contact group progressively larger subsets of MWs are activated, allowing individual NW codewords to be identified. This is accomplished as follows.

When testing begins, a "1" can be written to all N^2 NW/NW junctions. This sets all junctions to a conductive state, and ensures that when any of the N NWs being tested is addressed, current flows from one dimension of the crossbar to the other (i.e. a 1 is read). Now suppose codeword $\boldsymbol{c^i}$ has been discovered (using the procedure described below). That codeword can be addressed and a "0" can be written to the N NW/NW junctions controlled by each NW with that codeword. This disconnects the NW from the other dimension of the crossbar, which in turn ensures that subsequent testing will not reflect whether or not the codeword in question is addressed. In other words, the discovered codeword is effectively "removed" from future testing.

We now describe the discovery of individual codewords. Let activation pattern $\boldsymbol{a}$ be an M-bit binary vector denoting the set of activated MWs. At the start of each run of the testing algorithm, $\boldsymbol{a}$ is all 0's. For each of the M MWs, in some fixed order, MW $\boldsymbol{m_i}$ is turned on (i.e a_i is set to 1), and a read operation is performed to test if at least one of the "unremoved" NWs is addressed by $\boldsymbol{a}$. If no NW is addressed, a_i is set back to 0. Otherwise, a_i remains activated as subsequent MWs are turned on. In other words, we have the following procedure.

$DiscoverCodewordViaReadWriteTesting()$:
  $\boldsymbol{a} = \mathbf{0}$
  for $i = 1$ **to** M **do**
    $a_i = 1$
    if $read(\boldsymbol{a}) == 0$ **then** $a_i = 0$
  $write(\boldsymbol{a}) = 0$

At the end of a run of the above procedure, once all M MWs have been turned on and tested, $\boldsymbol{a}$ addresses at least one NW and turning on any additional MWs will result in no NWs being addressed. Such an activation pattern is referred to as **maximal**. Any maximal activation pattern corresponds to the complement of one of the NW codewords which are present. Since the codeword $\boldsymbol{c^i} = \overline{\mathbf{a}}$ is now discovered, it can be removed via the write operation described above.

Each run of our procedure requires M read tests and results in a new codeword being discovered. Thus all M bits of N codewords are discovered after MN read operations, which is optimal. It is noteworthy that the above procedure even discovers codewords that are not individually addressable. To see why, notice that if codeword $\boldsymbol{c^i}$ implies codeword $\boldsymbol{c^j}$, $\boldsymbol{c^i}$ will be discovered once $\boldsymbol{c^j}$ is removed.

8.1.1 Coping with Errors

In the read/write-based test procedure described above, we can consider two classes of errors. First, the NW/NW junctions being written to and read from may be defective. Second, the NW codewords may contain errors (see Section 3.4).

Storage Errors

Storage-related errors are relatively easy to protect against. First consider the case when some NW/NW junctions are permanently stuck at 0. Since each read operation is actually reading from N junctions in parallel, they still function properly (i.e. measure a sufficiently large current if at least one NW is being addressed) if some of the junctions are stuck at 0.

Now consider the case when some junctions are permanently stuck at 1. These errors do effect the ability of write operations to "remove" NWs. Fortunately, such errors can be detected by initially writing a 0 to all N^2 junctions. If current still flows between the two OCs after the 0's have been written, some junction is stuck at 1. If such an error is present, a different patch of N^2 junctions can be used.

Finally, in order to protect against transient, or one-time errors, each write operation can be followed by a read operation to verify it succeeded. Also, all read operations can be performed multiple times for added redundancy.

Codeword Errors

If a MW/NW junction is in error, the MW provides only partial control over that NW. As a result, some MW activation patterns can leave the NW in a partially conductive state (see Section 3.4). Recall that in read/write-based testing, MWs are turned on one at a time, while read operations are used to verify that some NW remains addressed (if no NW is addressed, the MW is turned back off). When certain MWs provide only partial control over certain NWs, it may be the case that a particular activation pattern does not address any NWs, but does leave multiple NWs in a partially conductive state. In this case, a read operation that applies that activation pattern could incorrectly return a 1 even though no NW is addressed.

If such an error occurs, the discovery algorithm will incorrectly proceed as if some NW were being addressed. Once all MWs have been turned on and tested, the activation pattern, $\boldsymbol{a}$, that is returned will not correspond to the complement of any codeword. Fortunately, such an error can be detected by attempting to write, then read, data to the NW (or NWs) that $\boldsymbol{a}$ is supposed to address. If $\boldsymbol{a}$ does not address any NW, and instead leaves multiple NWs in a partially conductive state, the write operation will not succeed. If, on the other hand, the write operation does succeed, $\boldsymbol{a}$ does in fact correspond to the complement of a codeword.

In the event that $\boldsymbol{a}$ does not correspond to a codeword, two possible workarounds can be used during subsequent run of the discovery algorithm. First, the algorithm can continue by activating MWs in different order. As discussed in Section 8.4, the order in which MWs are activated determines the order in which codewords are discovered. By activating MWs in a different order, additional codewords can be discovered. Alternatively, the algorithm can proceed by reactivating

MWs in the same order, only now write operations can be performed after each read operation to verify that some NW is, in fact, being addressed. The only complication with this approach is that after write operations are performed and verified, it may be necessary to re-remove some of the previously removed codewords via additional write operations (since writing a 1 to the activation pattern being tested may have unremoved them). This can potentially increase the discovery algorithm's runtime by a factor of N.

8.2 Exhaustive Search

In the absence of read/write operations, current measurements can be used to determine which codewords are present. The simplest such address discovery algorithm is exhaustive search. For each contact group, all 2^M MW activation patterns can be applied, and in each case, we can test if the total current flowing across all N NWs is above some preselected threshold (see Sections 3.1.1 and 3.4.2 for a discussion of this threshold). In the absence of decoder errors, the threshold can be chosen such that for any activation pattern, $\boldsymbol{a}$:

1. If the current flowing is below the threshold, no NW is addressed.
2. If the current flowing is above the threshold, at least one NW is addressed.

When all 2^M activation patterns are applied and tested, the binary outputs of all 2^M tests can be reviewed offline to determine which addressable codewords are present (unlike for read/write-based tests, a codeword that is implied by another codeword, will not be discovered). To detect the addressable codewords, we need only identify the maximal activation patterns. As defined above in Section 8.1, activation pattern $\boldsymbol{a}$ is maximal if it addresses at least one NW and if the activation of any additional NWs turns off all NWs. Activation pattern $\boldsymbol{a}$ is maximal if and only if $\boldsymbol{c^i} = \overline{\mathbf{a}}$ is individually addressable.

The runtime of exhaustive search is exponential in M, but as demonstrated in previous chapters, M may be relatively small. In our analysis of the "Take What You Get" addressing strategy for RCDs, for example, we demonstrated that $M = 13$ suffices (see Section 4.2.2). Smaller values of M are possible if one is willing to tolerate a smaller fraction of individually addressable NWs. Smaller values of M are also possible for encoded NW decoders (see Section 5.2.1).

8.2.1 Parallel Exhaustive Search

When M is small, performing 2^M current measurements is acceptable. Furthermore, this exponential runtime can be amortized across contact groups if the ATC is designed to allow for current measurements to be performed across all g contact groups simultaneously. The following argument illustrates why performing this type of parallel exhaustive search, if technologically feasible, is superior to a more efficient serial search algorithm when M is small.

Consider testing all contact groups in parallel in a randomized-contact decoder (RCD) when $M = 13$, $N = 8$, $g = 175$ (as in the example in Section 4.2.2). Using a parallel exhaustive search, the number of tests per contact group is $2^M/g = 8192/175 < 47$. We show that more tests are required by any discovery algorithm that performs binary tests on contact groups one at a time.

Any address discovery procedure must produce the codeword for each individually addressable NW in a contact group. As shown in Theorem 4.2.1 of Chapter 4, the expected number of addressable NWs in a contact group is at least $N(1-(N-1)(1-pq)^M) = 1-7(3/4)13 > 6.6$ (here $p = q = 1/2$). This demonstrates that at least six NWs are addressable at least 1/2 the time. (Given that $N = 8$, if less than six NWs are addressable half the time, the average number of addressable NWs is at most $(5+8)/2 = 6.5$). There are 2^{MN} assignments of M-bit codewords to the N NWs. We refer to each of these assignments as a "configuration". Since all configurations are equally likely, at least $(1/2)2^{MN}$ of these have six individually addressable NWs. The codewords of these NWs must be produced by any address discovery algorithm.

Now let σ be the maximum number of configurations that contain any fixed set of six individually addressable codewords. When a discovery algorithm produces six or more codewords as output, one of at most σ configurations is present. $\sigma \leq 6!\binom{8}{2}2^{2M}$. Here $6!\binom{8}{2}$ bounds the number of ways a set of six codewords can be assigned to $N = 8$ NWs, and 2^{2M} bounds the number of codewords that can be appear on the remaining 2 NWs.

It follows that any discovery algorithm must be able to identify at least $(1/2)2^{MN}/\sigma$ configurations. Let T be the number of tests required to identify a set of codewords. Then, since each test produces a binary outcome, T is at least $\log_2[(1/2)2^{MN}/\sigma] = MN - 1 - 2M - \log_2(6! \cdot 28)$. When $M = 13$ and $N = 8$, $T \geq 63$. Thus, an algorithm that examines one contact group at a time will need to perform at least 63 tests per group. This is more than required by a parallel exhaustive search.

8.2.2 Coping With Codeword Errors

As explained in Section 8.1.1, codeword errors can cause certain activation patterns that do not address any NW to leave multiple NWs in a partially conductive state. Even though no individual NW is addressed, the combined current flowing across the partially conducting NWs may be above the designated threshold. As with read/write based testing, this can cause certain activation patterns to be incorrectly designated as maximal.

One approach to solving this problem is to use a slightly more robust current measurement procedure. Instead of a single threshold, current can be measured with regard to two thresholds, t_1 and t_2. As before, each current measurement test returns "false" if the current is below t_1, but now "true" is returned only if the current is above t_2 (otherwise an error is returned). In this case, an output of true ensures that some NW is carrying a current of at least t_2/N. If t_2 is sufficiently large, this in turn ensures that some NW is addressed.

Let the union of two activation patterns be their bitwise OR. Using this dual-threshold test, if two activation patterns return true, but their union returns false, they must address different NWs. Two such activation patterns are said to be "disjoint".

In order to discover N NW addresses after all 2^M activation patterns are tested, it suffices to identify N patterns that are all disjoint. One method for identifying these patterns from the testing data is to construct a graph, G, with a vertex associated with each activation pattern that returned true. An edge is placed between any two vertices that are disjoint. A clique of N vertices in G corresponds to a set of N activation patterns that all address disjoint sets of NWs.

The disadvantage of this approach is that it requires slightly more complex testing circuitry (since two thresholds are used instead of one). More importantly, the on/off ratio of addressed to nonaddressed NWs must be sufficiently large not only for the decoder to function properly, but for the dual-threshold testing procedure to succeed (since t_2 is specifically chosen to be N times larger than necessary.) An alternative approach to tolerating errors based on the minimum distance between codewords is described in section Section 8.4.

8.3 Encoded NW decoders

In an encoded NW decoder, as discussed in Chapter 5, codewords are determined by how NWs have been encoded during off-chip growth. This allows NW codewords to be restricted such that all codewords are drawn from a code in which no codeword implies any other. One such code is a binary reflected code (BRC), in which all codewords are of the form $\boldsymbol{x}\overline{\boldsymbol{x}}$ where $\boldsymbol{x} \in \{0,1\}^k$ and $M = 2k$ (see Section 5.1.3). Using such a code allows codewords to be discovered via a binary search algorithm within each contact group.

To begin, consider testing for a particular codeword, $\boldsymbol{c^i} = \boldsymbol{x}\overline{\boldsymbol{x}}$ within a particular contact group. By applying the activation pattern $\boldsymbol{a} = \overline{\boldsymbol{c^i}}$, only NWs with that codeword will be addressed. Now let $\boldsymbol{x(b)}$ denote the first b bits of x and let $\boldsymbol{w(x,b)} = \boldsymbol{x(b)1\ldots1}\overline{\boldsymbol{x(b)}}\boldsymbol{1\ldots1}$ where the two groups of 1's are of length $k-b$, and thus $\boldsymbol{w(x,b)}$ is of length $2k$. Notice that the activation pattern $\boldsymbol{a} = \overline{\boldsymbol{w(x,b)}}$ addresses all codewords of the form $\boldsymbol{x(b)} * \ldots * \overline{\boldsymbol{x(b)}} * \ldots *$, where $*$ indicates either a 0 or 1. As noted in [12], this same approach, known as "wildcarding", can be used to write to multiple locations in memory simultaneously.

8.3.1 Binary Search

Using wildcarding, we can test whether any codewords with a particular prefix exist. This in turn allows all N codewords to be discovered in $M/2$ stages. At stage one, we test whether any codewords that begin with a "0" and a "1" are present. If both prefixes are present, stage two continues the binary search within both sets. In other words, stage two checks for codewords that being with "00", "01", "10" and "11". Once it has been determined that no codewords with a particular prefix exist, subsequent stages no long include tests with that prefix. Since at most N codewords are present, no more than $2N$ prefixes will be tested at each of stage of this algorithm. Since the algorithm will discover all bits of a BRC codeword by the $M/2^{th}$ stage, at most MN total tests are required. Here M, the number of MWs, is equal to $2\log_2 C$, where C is the total number of possible BRC codewords.

8.3.2 Searching Across Contact Groups

The above search procedure can be employed to discovery the N NWs within each of the g contact groups, one at a time. In order to discover $N' = gN$ NW codewords across all g contact groups, this approach requires $2N'\log_2 C$ current measurements. Impressively, even fewer measurements are needed if a binary search is conducted across contact groups.

Suppose the g contact groups are partitioned into groups of g', and each of these groups is searched separately. For each codeword, $\boldsymbol{c^i}$, activation pattern $\boldsymbol{a} = \overline{\boldsymbol{c^i}}$ can be applied *all* g' contact groups simultaneously. A binary search can then be used to determine which contact group, or groups, the codeword appears in. We note that this binary search procedure does not assume multiple current measurements can be taken in parallel (as in Section 8.2.1). Instead, it merely assumes that multiple contact groups can be activated simultaneously, and their collective current flow measured.

Using the above procedure, each NW's codeword is discovered using at most $2\log_2 g'$ current measurements (by the same reasoning employed in the previous section). All $g'N$ NW codewords are discovered using at most $2g'N\log_2 g'$ current measurement operations, plus C' additional measurement operations for the codewords that are tested, but not present on any of the $g'N$ NWs. For maximum efficiency, g' should be selected so that C' is small. This is accomplished if $g' = C/N$, in which case each codeword is expected to occur one time. Furthermore, since $C' < C = g'N$, we can immediately assert that at most $2g'N\log_2 C/N + g'N = 2g'N\log_2 2C/N$ current measurements are required.

When summed over all g/g' groups of g' contact groups, the above algorithm uses at most $2N'\log 2C/N$ tests to discovery all $N' = gN$ NW codewords. Also note that the algorithm does not rely on codewords being drawn from a BRC, since no wildcarding is employed. The algorithm instead relies on all codewords being individually addressable, thus (h, M)-hot codes can also be used (see Section 5.1.2). In the next section, we show that using $2N'\log 2C/N$ tests is within a factor of optimal.

8.3.3 A Lower Bound

We wish to lower bound the number of binary test operations required to identify the codewords on N NWs within a contact group. Let αN be the average number of distinct codewords per ohmic region. It follows that the NW codewords can be chosen in at least $\binom{C}{\alpha N}$ different ways in each contact group, and at least $\binom{C}{\alpha N}^g$ ways across all contact groups. This means that at least $g\log\binom{C}{\alpha N}$ binary tests are required to distinguish between these possibilities. Since

$$\binom{C}{\alpha N} = \frac{C!}{(C-\alpha N)!(\alpha N)!} = \frac{C}{\alpha N}\cdot\frac{C-1}{\alpha N-1}\cdot\ldots\cdot\frac{C-\alpha N+1}{1} \geq \left(\frac{C}{\alpha N}\right)^{\alpha N}$$

$g\log\binom{C}{\alpha N} \geq g\alpha N\log(C/\alpha N)$. Since it does not make sense to design the decoder such that α is very small, this lower bound is very close to the number of tests used by our algorithm.

8.3.4 Coping With Misalignment Errors

The discovery algorithm described above relies on the assumption that all NW codewords are drawn from a set of C individually addressable codewords. This ensures that each activation pattern of the form $\boldsymbol{a} = \overline{\boldsymbol{c^i}}$ addresses only NWs with codeword $\boldsymbol{c^i}$. This is a reasonable assumption, since encoded NWs can given binary reflected or (h, M)-hot codewords (see Section 5.1), but as explained in Section 5.3, axially encoded NWs are susceptible to misalignment errors. Such errors may cause certain NWs to be only partially conducting when certain activation patterns are applied.

Fortunately, even when misalignment errors can occur, it is still possible to guarantee that each axially encoded NW is addressed by at most one activation pattern of the form $\boldsymbol{a} = \overline{\boldsymbol{c^i}}$ (see Section 5.3). Furthermore, if a misalignment error does occur, the misaligned NW will be made partially conducting by at most one such activation pattern, while all other activation patterns turn off that NW. As such, the presence of misaligned NWs does not prevent the discovery algorithms described in this section from discovering the codeword of each properly aligned NWs.

The only remaining concern, which was already highlighted in Section 8.2.2, is that multiple misaligned NWs within a single OC may be made partially conducting by the same activation pattern. These NWs may collectively carry the same amount of current as a single, fully conducting NW. To protect against false positives, it makes sense to explicitly test each alleged codeword (perhaps using a single a write operation) after all codewords have been discovered. The dual-threshold test proposed in Section 8.2.2 could also be employed.

8.4 Arbitrary Codes

When M is large, enumerating over all codewords becomes prohibitively slow. Section 8.3.1 provided a more efficient approach to determining which codewords are within a given OC, but this approach does not work when arbitrary codewords may be present, as in an RCD. In this section we consider a simple alternative, which is similar to the read/write-based algorithm presented in Section 8.1, and bound the number of current measurements it requires. A less efficient version of this algorithm was introduced in [55]. As explained in [56], however, the accompanying analysis was based on faulty assumptions.

As in Section 8.1, the objective of each run of the discovery procedure is to generate a maximal activation pattern, $\boldsymbol{a}$. Each run of the procedure, sketched below, begins by selecting a random permutation of the MWs, π. MWs are then activated one at a time, in the order specified by π, until no current is produced (i.e. all NWs are turned off). When current is turned off, the last MW to be turned on is deactivated, and the process continues. In other words, we have the following codeword discovery procedure.

procedure *DiscoverCodewords*()
  $\boldsymbol{a} = \mathbf{0}$
  $\pi = RandomPermutation(1, 2, \ldots, M)$
  for each i **in** π **do**
    $a_i = 1$
    if $test(\boldsymbol{a}) == 0$ **then** $a_i = 0$

After each execution of the *DiscoverCodewords*() procedure, above, a maximal activation pattern is identified (assuming no codeword errors are present), and its complement yields the discovered codeword (see Section 8.1). The procedure is to be executed repeatedly until all (or almost most) individually addressable codewords have been discovered. For ease of simulation, it is convenient to note that the discovered codeword is the codeword that comes first when all codewords are sorted lexicographically according to π. Also, as an optimization, we note that it is not actually necessary to activate subsets of MWs when they do not turn off all of the codewords that have

already been discovered. In this case, the outcome of the test would already be known (a current will be measured). This observation was also made in [66], which evaluates a similar codeword discovery algorithm through simulation.

Each execution of *DiscoverCodewords*() requires M tests. After each test some codeword is discovered. The total time required for codeword discovery thus depends on the relative likelihood of discovering each codeword. If all codewords are equally likely to be discovered, the classic coupon collector problem (see Chapter 6) shows that close to $N_a \log(N_a/\epsilon)$ runs are required to discover all N_a individually addressable codewords with probability $1 - \epsilon$. Unfortunately, as explained below, all codewords are not guaranteed to be equally likely to be discovered.

This is the faulty assumption made in [55]. In fact, experiments indicate that for small values or medium-sized values of M, some codewords will often be much less likely to be discovered than others. For example, when M is 30, all NWs in a contact group of $N = 8$ NWs are addressable with very high probability. If all NWs were equally likely to be discovered, $N \log(N/.01) = 69$ runs of *DiscoverCodewords*() are required with probability .01. Our simulations reported in Section 8.4.2 show this value to be approximately 270. When M is 100, however, the value shrinks to 72.

The reason for this discrepancy is that, when M is small, some NWs that are addressable are much less likely to be discovered than others. For example, when $M = 30$, more than 1/10 of the time there was at least one NW that had only a 1/70 chance being discovered on each run. For intuition as to why this occurs, consider the following four codewords: $\boldsymbol{c^1} = 111100000000, \boldsymbol{c^2} = 000011110000, \boldsymbol{c^3} = 000000001111, \boldsymbol{c^4} = 011101110111$. By symmetry, $\boldsymbol{c^1}$, $\boldsymbol{c^2}$ and $\boldsymbol{c^3}$ are equally likely to be discovered, but $\boldsymbol{c^4}$ can only be discovered if at least two of MW_1, MW_5 and MW_9 are activated before any of the other MWs. This observation reveals that $\boldsymbol{c^4}$ is discovered is with probability $3/12 * 2/12 * 1/4 = 1/96$, where as all other codewords are discovered with probability $(1 - 1/96)/3 = 95/288$. When M is small, these sorts of extreme examples are much more likely to occur.

8.4.1 Asymptotic analysis

As suggested in the previous section, and demonstrated in [56], the number of runs of the procedure *DiscoverCodewords*() required to discover N individually addressable codewords can be bounded probabilistically by establishing a lower bound on the minimum probability with which any particular codeword is discovered. Specifically, consider a NW decoder with N individually addressable codewords. If each codeword is discovered by a particular run of our discovery algorithm with probability at least p_{min}, then the following lemma bounds the expected number of runs required to discover all codewords.

Lemma 8.4.1 *Consider a coupon collector problem in which coupons are not collected with equal probability, but each trial still collects one of N coupons independently at random. Let T be the number of trials before all N coupons get collected. If, on each trial, each coupon is collected with probability at least p_{min}, the expected number of trials required to collect all coupons is at most*

$$E[T] \leq 1 + \frac{1}{p_{min}} \mathcal{H}(N-1)$$

where $\mathcal{H}(N-1) = 1 + \frac{1}{2} + ... \frac{1}{N-1}$.

Proof The average time to collect N coupons is $E[T] = \sum_{i=1}^{N} E[x_i]$, where x_i is the number of trials needed to collect the i^{th} new coupon once $i-1$ coupons have been collected. Let $p_1, p_2, \ldots, p_N$ denote the probabilities with which coupons are collected and let $j_1, j_2, \ldots, j_N$ be the order in which coupons end up being collected.

Because the first new coupon is always collected on the first trial, $E[x_1] = 1$. For $i \geq 2$ the probability distribution for x_i is geometric with probability $1 - (p_{j_1} + p_{j_2} + \ldots + p_{j_{i-1}})$. Thus, $E[x_i] = 1/(1 - (p_{j_1} + p_{j_2} + \ldots + p_{j_{i-1}}))$.

It follows that $E[T]$ is maximized by maximizing $(p_{j_1} + p_{j_2} + \ldots + p_{j_{N-1}})$. Since $p_N \geq p_{min}$, $E[x_N]$ is largest when $p_N = p_{min}$. Similarly, the remaining terms in the sum for $E[T]$ are maximized by setting $p_j = p_{min}$ for $2 \leq j \leq N$ and setting $p_1 = 1 - (N-1)p_{min}$. This yields the desired result. ■

Theorem 8.4.1, given below, was proven in [56] to apply the above lemma to RCDs. To make sense of this theorem, it should be explicitly understood that when a randomized discovery procedure is applied to a stochastically assembled decoder, there are really *two distinct* sets of random events under consideration. First, within each contact group, N codewords are randomly assigned to N NWs. Each possible resulting decoder configuration, $\mathcal{C}$, has a probability associated with it. Second, with the decoder's configuration held fixed, the procedure $DiscoverCodewords()$ is repeatedly applied. During each run, there is a probability, p_i, associated with each of the N codewords being discovered.

Our goal is to bound the probability that our randomized algorithm discovers all codewords efficiently. To do this, we let $\mathbb{C}_{p_{min}}$, be the set of decoder configurations (within a single contact group) such that for each $\mathcal{C} \in \mathbb{C}_{p_{min}}$, $p_i \geq p_{min}$ for all N codewords. We then let $Q(p_{min})$ denote the probability that a decoder's stochastic assembly process yields a configuration in $\mathbb{C}_{p_{min}}$. In the case of RCDs, recall from Chapter 4 that each NW/MW junction becomes controlling with probability p, noncontrolling with probability q and in error with probability $r = 1 - p + q$ (the following theorem assumes $r = 0$). If p_{min} is close to $1/N$, and $Q(p_{min})$ is close to 1, then Lemma 8.4.1 implies that with high probability, $O(N \log N)$ runs of $DiscoverCodewords()$ are needed to discover the individually addressable NWs within almost all contact groups (since $\mathcal{H}(N) \leq \ln N + 1$).

The following theorem, proven in [56], bounds $Q(p_{min})$ in terms of p_{min}, M, N, p and q.

Theorem 8.4.1 *Consider RCD configurations consisting of N codewords of length M in which 0s (1s) occur independently with probability q (p). Let $Q(p_{min})$ be the probability that all N codewords are discovered by $DiscoverCodewords()$ with probability at least p_{min}. Then p_{min} satisfies*

$$p_{min} \leq \frac{1}{2}(4N)^{-\frac{1}{\gamma}(\ln q - k_1/M)} e^{-\left(\frac{(\ln 4N)^2}{\gamma^2 M}\right)\left(\frac{1}{q-k_1/M}-1\right)}$$

when $\gamma = (Mq - k_1)/(Mq^2 + k_2) > 1$ and k_1 and k_2 are chosen so $k_1 \geq \sqrt{2Mq \ln(2N/(1 - Q(p_{min})))}$, and $k_2 \geq \sqrt{2M(1-q^2)\ln(N^2/(1 - Q(p_{min})))}$.

Unfortunately, the above bound is rather weak numerically. Choosing $k_1 = 10$ and $k_2 = 12$ when $N = 8$ yields $Q(u) = 0.93$ when $u = .005$. That is, for 93% of RCDs each codeword is

discovered with probability of at least 1/2 of one percent. In practice a much higher value of p_{min} is achieved, as the simulations in the following demonstrate.

Connection to PAC Learning

In this section we have bounded the number of MWs, M, required so that, with high probability, each NW codeword in an RCD is discoverable with probability at least u after M current measurements are taken. This is similar to the well-known "probably approximately correct" (PAC) learning framework, as applied to learning of random monotone DNFs [67]. In PAC learning of random monotone DNFs, the goal is to give an efficient algorithm (in terms of the number of required queries) capable of identifying, with high probability, most randomly generated t-clause DNFs. As described in Section 3.4.1, our model of binary NW codewords with errors can also be described in terms of N-clause monotone DNFs. Here activating a subset of the MWs, and then measuring whether at least one NW conducts, is equivalent to querying the DNF.

The results of [67] can potentially be applied to error-free NW decoders with a sufficiently large number of NWs and MWs. Unfortunately [67], like most work on PAC learning, does not deal with the possibility that certain inputs are "in error", as defined in Section 3.4.1. In our model, codeword errors effectively cause the N-clause monotone DNF that represents a particular NW decoder to behave like a monotone function that is merely close to some N-clause monotone DNF. As a result, queries near the boundary of the would-be N-clause monotone DNF are unreliable, in that they no longer correspond to the value of an N-clause DNF (here the term "boundary" refers to those inputs at which the DNF switches from 0 to 1). This model of learning in the presence of unreliable boundary queries is presented and analyzed by Blum *et al* in [68] with regard to monotone DNFs, as well as other classes of functions, but their results do not yield tight bounds, particularly when N is small.

8.4.2 Experimental Results

In Matlab, 2000 runs of the Discover_Codewords procedure on each of 5000 randomly generated, error-free contact groups. Each contact group had 8 NWs. Figure 8.1 plots the cumulative distribution of the number of runs before all individually addressable codewords were discovered for both 30 and 100 MWs. Also shown is the cumulative distribution of the fraction of runs that discovered whichever codeword was discovered least often, that is, an empirical estimate of u, the minimum probability with which a codeword is discovered.

As discussed at the beginning of this section, as the number of MWs increases from 30 to 100, the minimum probability with which a codeword is discovered increases. Similarly, the number of runs to discover nearly all codewords with high probability decreases as M increases. In fact, approximately 270 runs are needed to discover all codewords with probability 0.99 when $M = 30$ and approximately 72 when $M = 100$. The latter number is very close to the number predicted when all codewords are equally likely to be discovered using the coupon collector problem. This is further illustrated by the right-hand plots in Figure 8.1, which show that when $M = 100$, u is usually close to 1/8. In other words, when more MWs are used, it is usually the case that

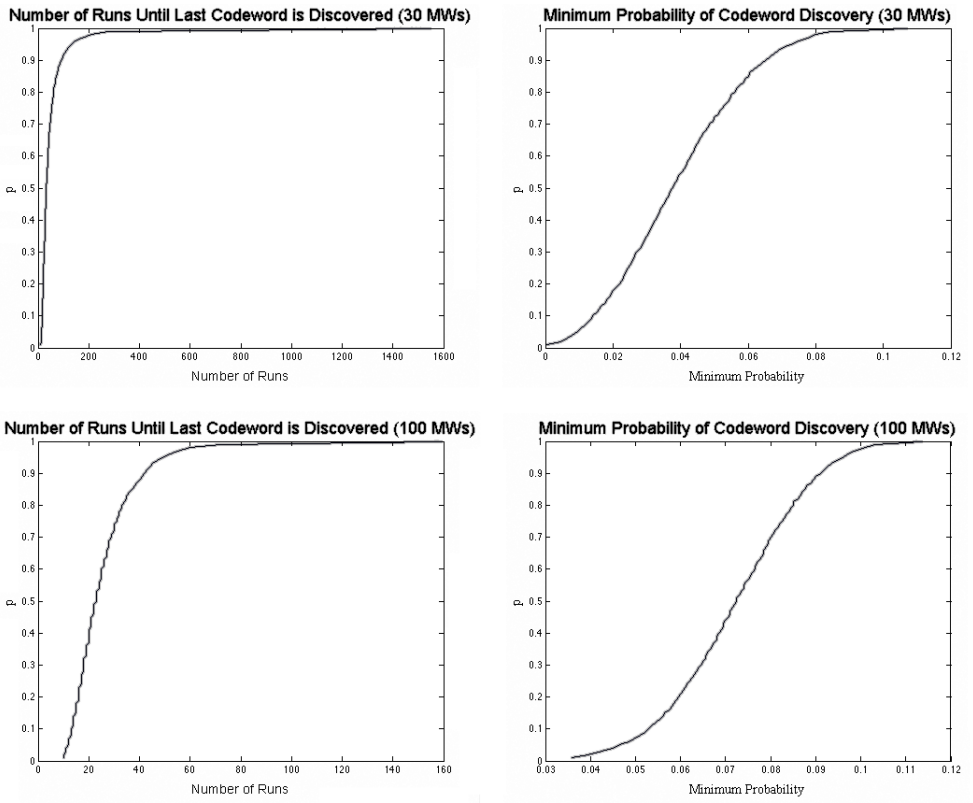


Figure 8.1: Shown are empirical plots obtained by simulating 2,000 runs of Discover_Codewords on 5,000 randomly generated, error-free contact groups each of which has 8 NWs. The plots show the cumulative distribution of the number of runs before all individually addressable codewords are discovered and the fraction of the runs in which the least frequently discovered codeword were found.

each codeword has an approximately equal chance of being discovered on each run of the discovery algorithm.

Chapter 9

Coded Computation

Previous chapters have illustrated that stochastically assembled nanoscale architectures are well-equipped to cope with post-assembly variation. In NW decoders, compensating for permanent decoder-to-decoder variation requires a small constant factor redundancy, along with configurable, highly reliable mesoscale address translation circuitry (see Section 3.3.2). A similar approach can be used to compensate for randomly varying interconnect within crossbar-based logic, as well to route around permanent defects within both logic and memories.

A more daunting challenge, that has not yet been resolved, is whether nanoscale architectures can efficiently cope with transient faults. In the case of nanoscale memories, the use of traditional error-correcting codes is a viable option [69]. By encoding data before it is stored, transient faults can be periodically detected and corrected. In digital logic, however, the traditional approach to fault-tolerance is simply to repeat each computing element multiple times (see Section 9.1.1 below). Unfortunately, this approach, often referred to as "modular redundancy," requires a prohibitive amount of overhead. If each nanoscale gate needs to be repeated many times, it could sensibly be replaced with a single, highly reliable CMOS gate.

It is natural to ask whether there is a more efficient approach to tolerating transient runtime errors within digital logic. This chapter investigates a two-tiered approach to reliable computing that is similar in spirit to the approach used to tolerate assembly-time nanoscale variation. Specifically, we consider using reliable mesoscale CMOS to interface with less reliable nanoscale devices. We refer to this model, in which different computing elements operate at different levels of reliability, as **two-tiered reliability**. By employing this model, we are able to pursue an approach to reliable computation that is markedly different from the bulk of theoretical work that has been pursued previously (see Section 9.1.1). Though the concept of two-tiered reliability has been used implicitly in algorithm-based fault-tolerance (see Section 9.1.2), we use it here to investigate fault-tolerance using error-correcting codes. This approach is referred to as coded computation.

In **coded computation**, the input and output to a lengthy computation are encoded in an error-correcting code. Highly reliable logic gates can perform the encoding and decoding, while noisy gates perform successive steps of computation on the encoded data. As long as the encoded data isn't corrupted by too many errors, the data can be corrected and eventually decoded. Such an approach to fault-tolerance is analogous to using a reliable encoder and decoder to transmit data over a noisy channel. As we demonstrate, coded computation introduces a wide range of design

possibilities.

Section 9.1 briefly reviews previous work on reliable computing and motivates the use of error-correcting codes. Section 9.2 provides a model of "regular" computation that is amenable to encoding. Section 9.3 outlines a general approach for making regular computations reliable via error-correcting codes. Sections 9.4, 9.5 and 9.6 discuss the details of how codewords can be manipulated in order to compute on the encoded data. Section 9.7 presents specific codes that can be used for coded computation. Finally Section 9.8 bounds the overhead associated with code-based fault-tolerance.

9.1 Approaches to Reliable Computation

In this section we review the traditional model of reliable computation, originally proposed by von Neumann in 1956. His model, in which all gates fail with constant probability, has been the basis for the majority of subsequent theoretical work on reliable computation. We also briefly discuss a more practical, but less systematic approach to fault-tolerance known as "algorithm-based fault-tolerance". Finally we outline a more novel approach to fault-tolerance, coded computation, that has potential to yield both theoretical and practical results. Describing a framework for performing reliable coded computations is the focus of the remainder of this chapter.

9.1.1 Modular Redundancy

Early digital computers relied on vacuum tubes, which were unreliable. This suggested the same type of scaling challenge we face today: As computers became increasingly complex (i.e. used more logic gates), the probability that some component fails during a given computation approaches 1. This motivated von Neumann, in his well-known 1956 paper [14], to propose a systematic approach to building logic from unreliable gates. He described how an arbitrary circuit, $\mathcal{C}$, built from perfectly reliable gates could be converted to a fault-tolerant circuit $\mathcal{C}'$, constructed from potentially faulty gates. Here "fault-tolerant" means that, regardless of the size of $\mathcal{C}$, the error rate of each output of $\mathcal{C}'$ is constrained to be no more than a constant multiple of the failure rate of each gate.

To model gate failures, von Neumann assumed that each unreliable gate's binary output could flip independently at random from 0 to 1, or 1 to 0, with probability p_f. His goal was to construct a new circuit $\mathcal{C}'$ from an arbitrary fault-free circuit, $\mathcal{C}$, using unreliable gates. His construction ensured that the output of $\mathcal{C}'$ was incorrect with probability $O(p_f)$. As von Neumann noted, any such circuit $\mathcal{C}'$ could not only tolerate transient faults but also permanent failures.

Von Neumann used a randomized construction, assembling $\mathcal{C}'$ by repeating each gate in $\mathcal{C}$ r times, then after each group of repeated gates, using r randomly connected majority gates to suppress errors. This approach is very similar to protecting transmitted data using a repetition code. The main subtlety is that the majority gates can themselves fail. Von Neumann's key observation was that r potentially faulty, constant-sized majority gates would still correct a fixed fraction of errors with high probability. As such, the total number of errors present after each group of r majority gates would only rise above some fixed threshold, αr, with a probability that is exponentially small in r.

Thirty years later, Pippenger successfully analyzed von Neumann's construction and made it deterministic using expander graphs [15]. As von Neumann hypothesized, for arbitrary $\mathcal{C}$ the size of $\mathcal{C}'$, denoted $|\mathcal{C}'|$, need only be $O(|\mathcal{C}| \log |\mathcal{C}|)$. Similarly, when $r = O(\log |\mathcal{C}|)$ the probability that a particular group of r majority gates produces more than αr erroneous outputs is $O(1/|\mathcal{C}|)$. An excellent description of this analysis can be found in [70]. Unfortunately this analysis also suggests that the constant associated with the $O(\log |\mathcal{C}|)$ bound is large.

After Pippenger obtained an upper bound on $|\mathcal{C}'|$, he and others obtained matching lower bounds for a number of simple functions that are "sensitive" to all inputs, for example, XOR [71, 72, 73]. In some sense, this showed that von Neumann's approach was optimal. The derivations of these bounds, however, also highlighted a shortcoming of the von Neumann fault model. Since all gates fail with probability p_f, the gates at the input and output of a circuit always have probability p_f of being incorrect. Thus, in order to compute the XOR of N inputs, each input must be connected to $O(\log N)$ gates simply to ensure that information about its correct value reaches the output with high probability.

Since inputs must be repeatedly sampled, they are effectively encoded using a repetition code with rate $O(1/\log N) = O(1/\log |\mathcal{C}|)$. Notice that this repetition-based approach to reliable computation contrasts sharply with results from digital communication theory. Since the time of Shannon, it has been known that repetition is a highly inefficient error control mechanism. To achieve efficient fault-tolerant communication, a reliable encoder and decoder are used to send information across a noisy channel. When data is encoded, information about each input to the channel is "spread" across many check symbols. By allowing these transmitted check symbols to be functions of a large number of inputs, only constant factor overhead is required to protect against random bit flips. It is only natural to ask whether similar ideas can be applied to reliable computation.

9.1.2 Two-Tiered Reliability and Coded Computation

In order to allow for more efficient error protection, and to overcome the lower bounds referenced above, we must alter the von Neumann model. Using the concept of two-tiered reliability we consider a more general, but also a more realistic model of noisy computation in which gates operate at different levels of reliability. Some gates can be larger, but highly reliable (much like today's CMOS gates), while most gates are small and susceptible to transient failures (an anticipated characteristic of nanoscale devices). By allowing two levels of gate reliability, expensive, power-hungry, but highly-reliable mesoscale gates can potentially "supervise" less reliable nanotechnology. This two-tiered model allows us pursue designs that cannot be considered under the von Neumann model.

Perhaps the most straight-forward application of two-tiered reliability, which we do not focus on here, is to exploit the fact that many algorithms have relatively simple checks. After a computation is performed by noisy gates, it may be possible to use a small number of highly reliable gates to check the computation's output. This type of algorithm-specific approach to tolerating faults is sometimes referred to as "algorithm-based fault-tolerance" [74]. Although promising for specific problems, it is difficult to know how broadly such techniques can be employed. To draw a loose analogy, the problem of making an algorithm fault-tolerant, through periodic check computations,

has a similar flavor as trying to parallelize an algorithm, in that it is currently more art than science. Furthermore, unlike parallelization, fault-tolerant versions of many widely used algorithms are not known, and thus there is a shortage of standard techniques on which to draw.[1]

Instead of focusing on algorithm-specific solutions to fault-tolerance, we employ two-tiered reliability in the context of coded computation. Here highly reliable logic gates encode the inputs (and decode the output) of a lengthy computation. The computation itself is performed by noisy gates. Each step of the computation is performed on encoded data. After each step, error correction takes places, as well as "transcoding" and "data-movement" operations (see Section 9.2). In this way, a long computation is divided into a series of steps, each of which is made fault-tolerant using (ideally) a small amount of overhead. The overhead associated with coded computation depends both on the code being used and the amount of additional operations needed to compute on encoded (as opposed to unencoded) data. Both sources of overhead are bounded in subsequent sections.

9.1.3 Previous Work on Coded Computation

The earliest work on coded computation considered only bitwise operations performed on pairs of codewords [75, 76, 77]. This is overly restrictive. Later certain algorithm-specific encodings were considered. For example, arithmetic codes can be used for addition and multiplication [78], and check sums for linear matrix operations [79].

More recent work by Spielman [80] suggests that a much more general approach to code-based fault-tolerance is possible. Spielman's work is based on encoding the operations performed by a hypercube. We work with a somewhat more general of model of computation that is better tailored to circuits. The key differences between hypercubes and circuits being the possibility of arbitrary data-movement, and the use of boolean logic gates in place of processors. Although processors, unlike gates, can have memory, Spielman doesn't actually use this fact (i.e. processors are assumed to be memoryless).

In Spielman's approach to coded computation, data is encoded using 2D Reed-Solomon codes. He provides a way of computing on encoded data such that the result of each computation step is also encoded data in Reed-Solomon code with smaller error correction capability than the original code [80]. This necessitates that the differently encoded data be "transcoded" back to the original code so that subsequent computation steps can be performed. Both computation and transcoding steps are done in a noisy environment.

The overhead of the Spielman approach is quite large. Reed-Solomon codes, along with the use of processor-based hypercube networks both introduce significant overhead. In our framework we demonstrate that this overhead can be reduced. We describe how other codes and network topologies can be employed. Overall, we explore a much wider range of design possibilities.

[1] Although most work on algorithm-based fault-tolerance does not explicitly mention the notion of tiered reliability, we suspect that future work in this area will benefit from explicitly assigning a reliability level to each operation within an algorithm. This model would be particularly well-suited to multicore architectures in which all cores may not operate at the same level of reliability. In this context, one can attempt to quantify the fraction of reliable operations a particular algorithm requires.

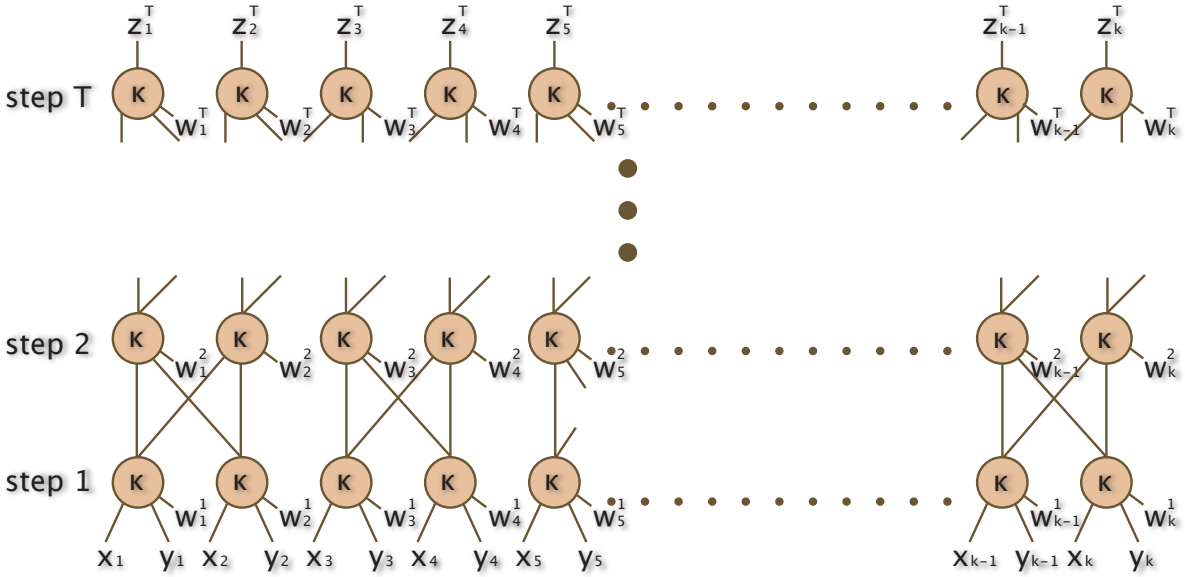


Figure 9.1: A T-step regular network in which each computation step consists of k 3-input functions applied in parallel. In step 1, functions $\kappa_1, \kappa_2, \ldots, \kappa_k$ are applied to inputs $x_1, x_2, \ldots x_k$ and $y_1, y_2, \ldots, y_k$. Here κ_j is the result of applying a generic function κ with input (instruction) w_j. The resulting output vector is duplicated and each copy is permuted. The two permuted copies provide the input vectors to step 2, in which a potentially different set of functions are applied. The alternation between data-movement and computation continues until step T, at which point k outputs are produced.

9.2 A Model of Computation

Before we examine how a computation can be made fault-tolerant through coding, we must define the computation itself. The model of computation used in this paper is the **regular computing network**, defined below. This model, as illustrated in Figure 9.1, divides a computation into T steps. A **computation step** consists of applying k m-input functions in parallel to m sequences of k symbols. Each of these sequences is called an **input vector**, and the resulting sequence of k outputs is called an **output vector**. The k functions that are applied to the m input vectors can be thought of as representing k processors or logic gates.

The network's input is supplied to the first computation step, and the network's output is produced by the final computation step. With the exception of the first step, all of a step's input vectors are permuted output vectors from previous steps.

A **regular computing network** meets the following conditions:

1. Each of the m input vectors to a computation step are permutations of the output vectors from previous computation steps. It is acceptable for one output vector to be supplied to multiple subsequent steps as an input vector. It is also acceptable if the input vectors to a particular step come from multiple previous steps.

2. All input and output vectors are length k. Each computation step consists of the application of k m-input function in parallel. Here the k functions are applied component-wise to the m input vectors.

3. We refer to the permuting of a step's output vector as a **data-movement operation**. During data-movement operations, the set of permutations that are applied are restricted to some set, Π (discussed below).

These regularity conditions are general enough to model computations performed by both circuits and structured networks (e.g. hypercubes and meshes). Examples of both are given in Section 9.2.2.

9.2.1 Formalizing the Model

We now make explicit the computational model when $m = 2$. Let F denote some finite field, for example $GF(2) = \{0, 1\}$. Let $\boldsymbol{x}, \boldsymbol{y} \in F^k$, $\mathbf{x} = (x_1, x_2, \ldots, x_k)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_k)$, denote two k-element input vectors. In a computation step, k two-input function, $\phi_i : F^2 \mapsto F$, are applied in parallel to the k pairs of inputs (x_i, y_i). The use of two input functions (as opposed to, say, m-input functions) is not required, but it is sufficiently illustrative for this discussion. The output vector of a computation step, $\boldsymbol{z} \in F^k$, is $\mathbf{z} = (\phi_1(x_1, y_1), \ldots, \phi_k(x_k, y_k))$.

For a given computation step, let $H = \{h_1, h_2, \ldots, h_{|H|}\}$ denote the set of distinct functions among $\phi_1, ..., \phi_k$, and let $W = \{1, ..., |H|\}$ denote a set of **instructions** used to index into H. Now each ϕ_i can be indexed to one of the $|H|$ functions in H. For a given computation step, we define its **instruction vector** as $\boldsymbol{w} \in W^k$, where w_i is the index of ϕ_i in H, or more succinctly, where $h_{w_i} = \phi_i$. Finally we define the step's **computation function** as $\kappa : F \times F \times W \mapsto F$ where $\kappa(x, y, w_i) = h_{w_i}(x, y) = \phi_i(x, y)$.

The output vector of a computation step, $\boldsymbol{z}$, can now be expressed as $z_i = \kappa(x_i, y_i, w_i)$, $1 \leq i \leq k$. Thus instead of applying k distinct two input functions to the input vectors, $\mathbf{x}$ and $\mathbf{y}$, we are now applying a single three input function to the input vectors plus an instruction vector, $\mathbf{w}$. To describe this component-wise application of κ more concisely we will frequently use the notation $\kappa^{(\mathbf{k})}$ where

$$\mathbf{z} = \kappa^{(\mathbf{k})}(\mathbf{x}, \mathbf{y}, \mathbf{w}) = (\kappa(x_1, y_1, w_1), \ldots, \kappa(x_k, y_k, w_k))$$

Each computation step, as defined by a particular κ and $\boldsymbol{w}$, is followed by a data-movement operation. Data-movement is described using permutations drawn from a set, Π, of k-element permutations. In a data-movement operation, the output of some previously executed computation step, $\mathbf{z}$, is copied and permuted according to some $\pi \in \Pi$. This allows us to express $\mathbf{z}_t$, the output vector of computation step t, in terms of the output vectors of two previous steps, $\mathbf{z}_a$ and $\mathbf{z}_b$ (where $a, b < t$). To express $\mathbf{z}_t$ we select the appropriate κ, $\boldsymbol{w_t} \in W^k$ and $\pi_{t,1}, \pi_{t,2} \in \Pi$ then write

$$\mathbf{z}_t = \kappa^{(\mathbf{k})}(\pi_{t,1}(\mathbf{z}_a), \pi_{t,2}(\mathbf{z}_b), \mathbf{w}_t) \tag{9.1}$$

For our purposes, namely providing fault-tolerance through coding, it is acceptable if H, W and κ vary from step to step (hence we could have reasonably written H_t, W_t and κ_t above). Π, by contrast, must remain fixed. The actual choice of Π, which is illustrated in the following

subsection, depends on the computation being implemented, as well as the code one intends to use for fault-tolerance. The motivation behind restricting Π, as well imposing our other two regularity conditions, will be made apparent in Section 9.3.

9.2.2 Examples

We now illustrate how regular computing networks can implement the same computations as logic circuits and common network topologies. In both cases, our task is to modify the computation performed by the network of computing elements (either gates or processors) so that it meets our regularity conditions. Once this is accomplished it is straightforward to define, H, W, and κ for each computation step. We also discuss the choice of Π.

Circuits

First consider how a regular computing network can implement the computation performed by a logic circuit. The basic approach is to "levelize" the circuit, then add additional buffer gates so that each level contains the same number of gates. The level of a gate is the length of the longest path between it and some input to the circuit.

To model a circuit let $F = \{0, 1\}$ and let H be a set of functions such as $\text{AND}(x, y)$, $\text{OR}(x, y)$ and $\text{NOTX}(x, y) = 1 - x$. In $\text{NOT}(x, y)$ the second unused input is needed to maintain our regularity conditions.

Let H also contain the **buffer functions** $\text{BUFFX}(x, y) = x$ and $\text{BUFFY}(x, y) = y$. These functions are also important for maintaining our regularity conditions, as demonstrated by the following five-step construction.

In our construction an arbitrary circuit is converted to a regular computing network in which data-movement is unrestricted, meaning Π can contain all k-element permutations. After the construction we describe how Π can be restricted.

1. Given an arbitrary boolean circuit, represented as a directed acyclic graph, the fan-out of each gate is reduced to two by adding additional one-input buffer gates, $\text{BUFF}(x) = x$ [81, p. 395]. These gates are replaced with two input buffer gates in step 4.

2. Partition the gates of the circuit into **levels**. The level of a gate is the number of edges on the longest path from that gate to one of the circuit's inputs. Add one-input buffer gates along any wires that pass through a level. This ensures that the outputs of gates in level i are supplied as inputs only to gates in level $i - 1$.

3. Add disconnected one-input buffer gates to the circuit so that each level has the same number of gates.

4. Replace all of the one-input BUFF and NOT gates with equivalent two-input gates BUFFX and NOTX (defined above) in which the second input (which is ignored) is disconnected.

5. One at a time, connect the disconnected inputs of the newly added two-input gates to any gate at the previous level which does not already have fan-out two.

To see that all disconnected inputs can be connected in this fashion, notice that each level contains the same number of gates, all gates have two inputs, and each gate's inputs come for the outputs of the previous level. This implies that if no gate is given fan-out greater than two, all gates can be connected with fan-out exactly two.

The circuit resulting from this five-step construction contains the same number of gates on each level, each of which has fan-in and fan-out 2. Hence it is readily represented as a regular computing network in which Π is unrestricted and both of computation step's input vectors are permutations of the previous step's output vector. Each computing step corresponds to a level of the circuit, each ϕ_i corresponds to a logic gate, and $H = \{\text{AND}, \text{OR}, \text{NOTX}, \text{BUFFX}\}$. Since Π is unrestricted, the data-movement operations that occur between computation steps can involve arbitrary permutations.

In order to restrict Π, these arbitrary permutations can be implemented using a structured switching network. For example, a Beneš network [82], which consists of two back-to-back butterfly (or FFT) graphs [81, p. 310], can permute its k inputs arbitrarily using $2 \log k + 1$ levels of k switches (each with fan-in and fan-out 2). This network, when placed between levels of our newly constructed circuit, maintains the other two regularity conditions given in Section 9.2, while allowing Π to contain only $\log k$ permutations. Furthermore, the newly added levels of switches, which implement a fixed permutation, only requires computation steps in which $H = \{\text{BUFFX}, \text{BUFFY}\}$, and hence $\kappa(x, y, w) = xw + y(1 - w)$.

It is also possible to restrict Π further and implement each butterfly network using only cyclic shifts via a shuffle exchange protocol [83, 84]. In this case, $|\Pi| = 2$, as it need only contain one place cyclic shifts in either direction. In the next section we show how restricting Π makes it easier to implement a regular network's computation on encoded data.

Networks

Now consider the regular computing network model as applied to structured networks of processors. Computations performed on structured parallel machines, such as a 2D mesh or hypercube, can be mapped directly to our model. These machines consist of k processors connected according to some k vertex graph, G. Before each step of computation, processors exchange data with their neighbors in G.

For simplicity, assume that data is exchanged with only a single neighbor before each computation step. Also assume that, before the start of the computation, each processor, p_i, has an initial state, s_i. These assumptions allow us to easily represent the k operations performed in parallel on a given computation step as k two-input functions $\phi_1, \ldots, \phi_k$. The input to each ϕ_i is the state of processor p_i and the state of a neighboring processor p_j (alternatively, partial information about states can be used). The output of ϕ_i determines the new state of p_i. Here states are represented as elements in some (possibly large) finite field, F. It is also acceptable if states are represented as s-tuples in F^s, in which case output vectors from s different computation steps collectively represent the state of each processor (this approach allows processors to have memory).

By representing the k operations performed during each computation step as two-input functions over states, our first and second regularity conditions are satisfied. To illustrate how the third

regularity condition can be satisfied, namely that data-movement permutations are restricted to some set Π, we consider some specific network topologies.

In a 2D mesh of processors, G is an m-by-m grid. Instead of writing p_i to denote a particular processor, we write $p_{i,j}$ to refer to the processor in row i and column j. Let $z_{i,j}$ denote the state of processor $p_{i,j}$. In G, each processor has 4 neighbors (with the exception of processors on the periphery), so on the next computation step $\phi_{i,j}$ is applied to $z_{i,j}$ and either $z_{i+1,j}$, $z_{i-1,j}$, $z_{i,j+1}$ or $z_{i,j-1}$.

In order to implement this data-movement in a regular computing network, the output vector $\mathbf{z}$ can be copied four times, and each copy differently permuted by a cyclic shift along its rows or columns. Before the next computation is performed, an intermediate computation step can be added in which m^2 4-input buffer gates are used to select the appropriate entries from the four shifted copies of $\mathbf{z}$. Each $\phi_{i,j}$ can then be applied to $\mathbf{z}$ and the output of this newly added selection step. Successive steps of the mesh's computation can be implemented using the same approach, thus $|\Pi| = 4$. In order to avoid the need for 4-input buffer gates, the selection step can also be implemented in stages using 2-input buffer gates.

Consider next data-movement in a hypercube. In a $k = 2^b$-processor hypercube, each processor is indexed using $b = \log_2 k$ bits. Without loss of generality, we can consider *normal algorithms* on the hypercube. These are algorithms for which, after each computation step, data is only swapped between pairs of processors whose b-bit indices differ in one particular position. (This is the class of computations for which Spielman defined coded computation in [80].) In a normal algorithm, data-movement always corresponds to applying one of b permutations to the values stored at the processors, hence when implemented using a regular computing network, $|\Pi| = b$. As with a mesh, each ϕ_i corresponds to the operation performed by a processor at a given step of computation.

9.3 The Coded Computation Framework

Now that we can describe computations in terms of regular computing networks, we can return to our goal of making these computations fault-tolerant though coding. To accomplish this, we would ideally find a way to add redundancy to each computation step by increasing the width of a regular computing network from k to n so that the input and output vectors of each computation step belong to the same efficient error-correcting code, C. This would allow us to periodically detect and suppress errors before they propagate too much and corrupt the entire computation (i.e. before the number of errors in some step's output vector surpasses the error-correction capability of C).

Unfortunately, this simple approach to fault-tolerant computation is not viable. As proven in Section 10.1, if each computation step consists of n constant-depth functions, applied in parallel, C cannot be an asymptotically good code (meaning as n increases, the code's rate must go to 0). Furthermore, if the n functions are applied component-wise to the encoded input vectors, C cannot outperform basic repetition in which each symbol is repeated r times. As a result we consider a modified approach.

1. First, the input vectors to a regular network are encoded in an error-correcting code, C. We

limit our attention to linear systematic[2] codes.

2. Second, an **encoded computation step** is performed on the encoded input vectors by applying n copies of a constant-depth function component-wise in parallel. The resulting encoded output vector provides a fault-tolerant encoding of the original, unencoded computation step's output. This encoded output vector, however, is *no longer a codeword in* C. Instead it is a codeword in another linear systematic code C^* of reduced error correction capability.

3. Third, a **transcoding operation** is performed which, in the case of error-free computation, projects the encoded output vector from C^* to the codeword in C that encodes the results of the computation on the input data. During transcoding, some errors in the encoded output vector can also be corrected.

4. Data-movement operations are implemented by permuting the encoded output vector either before, after, or during transcoding. The set of allowed permutations, Π^*, will depend on C and C^*.

5. After transcoding and data-movement operations have been performed on an encoded output vector, the vector can be supplied as an encoded input vector to a subsequent encoded computation step.

This process of performing an encoded computation step, followed by transcoding and data-movement operations, is repeated for each of the original regular network's computation steps. In this way, the regular network is "encoded", and its computation becomes fault-tolerant (see Figure 9.2).

9.3.1 One Step of Coded Computation

In Section 9.2.1, one step of unencoded computation is described in (9.1), as shown below.

$$\mathbf{z}_t = \kappa^{(\mathbf{k})}(\pi_{t,1}(\mathbf{z}_a), \pi_{t,2}(\mathbf{z}_b), \mathbf{w}_t)$$

To describe one step of coded computation, let $\mathbf{x}_t = \pi_{t,1}(\mathbf{z}_a)$ and $\mathbf{y}_t = \pi_{t,2}(\mathbf{z}_b)$. Let $E : F^k \mapsto G^n$, where $F \subseteq G$, be the encoding function for a linear, systematic error-correcting code C used to encode the inputs to a computation step. To compute, the function $\Phi : G^3 \mapsto G$ is applied component-wise to the encodings of $\mathbf{x}_t$, $\mathbf{y}_t$ and $\mathbf{w}_t$, namely, the n-tuples $E(\mathbf{x}_t)$, $E(\mathbf{y}_t)$ and $E(\mathbf{w}_t)$. Let $\Phi^{(n)} : G^{3n} \mapsto G^n$ denote the component-wise application of $\Phi : G^3 \mapsto G$.

Since we want the result of a computation step to be a codeword in a linear systematic code, $\Phi : G^3 \mapsto G$ should satisfy that the condition that $\Phi(u, v, w) = \kappa(u, v, w)$ when $u, v, w \in F$. This condition doesn't specify how Φ should be defined when its inputs are from G but not $F \subseteq G$. This is done using an idea due to Spielman [80], namely, by letting Φ be a polynomial interpolation of $\kappa : F^3 \mapsto F$ over G^3. In other words, Φ is a multivariate polynomial chosen to agree with κ over $F^3 \subseteq G^3$. A detailed discussion of Φ is given Section 9.4.

[2]In a systematic code the input data to be encoded appears in the output codeword

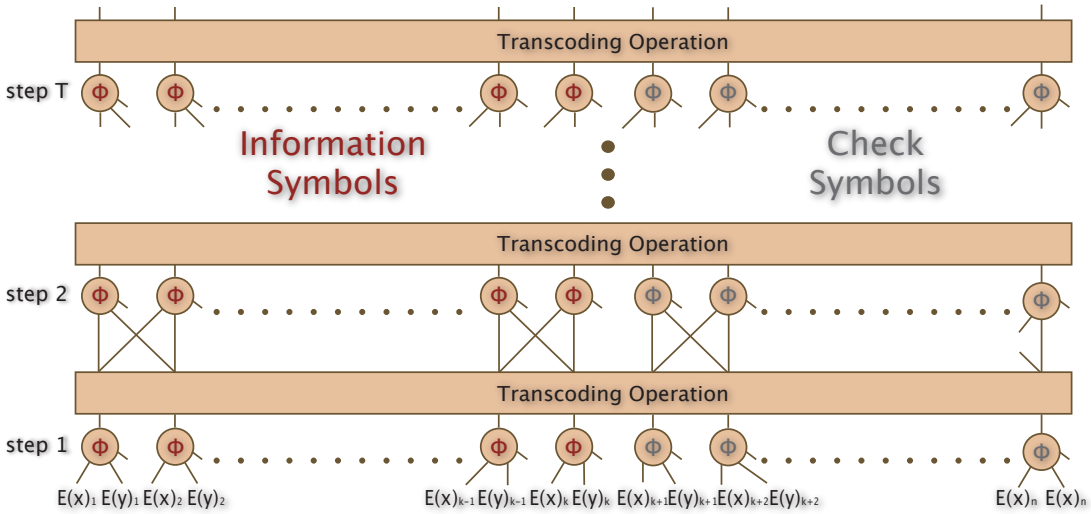


Figure 9.2: A T-step coded computation in which each step of a T-step regular computation has been encoded.

Given C and Φ an encoded computation step is expressed as

$$\Phi^{(n)}(E(\mathbf{x}_t), E(\mathbf{y}_t), E(\mathbf{w}_t))$$

This results in *some* encoding of $\mathbf{z}_t$ in a new code C^*. It is useful to denote this encoding as $E^*(\mathbf{z}_t)$, and thus write

$$E^*(\mathbf{z}_t) = \Phi^{(n)}(E(\mathbf{x}_t), E(\mathbf{y}_t), E(\mathbf{w}_t)), \tag{9.2}$$

We note that given some C and Φ, any given output vector $\mathbf{z}_t$ generally has multiple representations in C^*, even in the absence of errors. In other words, we typically cannot choose for C^* a code with only k information symbols, and thus $E^*(\mathbf{z}_t)$ may not be well defined unless, as in Equation 9.2, it is clear how the encoding of $\mathbf{z}_t$ is computed.

With this consideration in mind, we can safely define C^* as follows. First we note that $\Phi^{(n)}$ is chosen so as to be "difference preserving" with regard to $\kappa^{(k)}$ and C, meaning that for any two sets of inputs such that, $\kappa^{(\mathbf{k})}(\mathbf{x}, \mathbf{y}, \mathbf{w}) \neq \kappa^{(\mathbf{k})}(\mathbf{x}', \mathbf{y}', \mathbf{w}')$, we will have $\Phi^{(n)}(E(\mathbf{x}), E(\mathbf{y}), E(\mathbf{w})) \neq \Phi^{(n)}(E(\mathbf{x}'), E(\mathbf{y}'), E(\mathbf{w}'))$. Given such a Φ and C, let C^* be the smallest possible linear code that contains all possible vectors that result from applying $\Phi^{(n)}$ to three encoded input vectors in C. The relationship between C and C^*, along with their potential error-correction capabilities, is the focus of Sections 9.4.2 and 9.7.

In the remainder of this chapter, when we denote an encoded output vector as $E^*(\mathbf{z}_t)$ it is clear from the context how the encoded output vector is defined. Equation 9.2, for example, makes it clear that $E^*(\mathbf{z}_t)$ denotes the encoded output vector that results from applying $\Phi^{(n)}$ to a particular set of encoded input vectors.

9.3.2 Transcoding the Output

As explained at the beginning of this section, the transcoding step implements a mapping T_{C,C^*} : $G^n \mapsto G^n$ that in the absence of errors maps the output of a computation step, a codeword $E^*(\mathbf{z}_t) \in C^*$, to $E^*(\mathbf{z}_t) \in C$. This allows the encoding of $\mathbf{z}_t$ to be supplied to subsequent steps as input. If the encoded output of a computation step is close to a codeword $E^*(\mathbf{z}_t)$ (i.e. certain positions are in error), then T_{C,C^*} should map it to a word that is equal or close to $E(\mathbf{z}_t)$. When there is equality (meaning no errors occurred during transcoding) the output of a coded computation step followed by transcoding is expressed as follows.

$$E(\mathbf{z}_t) = T_{C,C^*}(\Phi^{(n)}(E(\mathbf{x}_t), E(\mathbf{y}_t), E(\mathbf{w}_t))) = E(\kappa^{(\mathbf{k})}(\mathbf{y}_t, \mathbf{y}_t, \mathbf{w}_t)). \tag{9.3}$$

9.3.3 Conditions on Permutations

As defined above $\mathbf{x}_t = \pi_{t,1}(\mathbf{z}_a)$ and $\mathbf{y}_t = \pi_{t,2}(\mathbf{z}_b)$, that is, the input vectors to computation step t result from the application of permutations $\pi_{t,1}$ and $\pi_{t,2}$ to output vectors $\mathbf{z}_a$ and $\mathbf{z}_b$. As we have explained, each $\pi \in \Pi$ is a permutation over k elements. Given a permutation π^* over n elements, we say that π^* is an **extension** of π if π^* applies π to the first k of the n elements. Given a set of permutations Π, our objective for the purpose of encoding a regular network's computation is to define a set of n-element permutations, Π^*, that can be used to realize the permutations in Π. We note that in the encoded version of a regular computing network a permutation, $\pi^* \in \Pi^*$ can be applied either before, during, or after a transcoding step.

Codes are often closed under some set of permutations. For example, cyclic codes are closed under cyclic shifts. Suppose C and C^* are closed under some set of permutations Π_C and Π_{C^*} respectively. Then any permutations from Π_C can be applied to an output vector after it has been transcoded, and any permutation from Π_{C^*} can be applied to the output vector prior to transcoding. Furthermore, the transcoding operation which projects some $E^*(\mathbf{z}_t)$ to $E(\mathbf{z}_t)$ typically involves decoding codewords in C^*, then encoding these results in C (see Section 9.5 for details). During this decoding/reencoding process there generally is some set of permutations, Π_T, that can be applied to permute $\mathbf{z}_t$ such that the transcoded output remains in C.

If Π_C and Π_{C^*} are permutations that preserve the codes C and C^*, and Π_T are the permutations that can be applied during transcoding.[3] Then, $\Pi^* = \Pi_C \bigcup \Pi_{C^*} \bigcup \Pi_T$ is a set of permutations that can be applied to the encoded output of a computation step to implement data-movement. Consider an unencoded computation step after which some $\pi \in \Pi$ is applied. We wish to realize an equivalent data-movement operation on the encoded data. If C and C^* are systematic, and if a particular $\pi \in \Pi$ has an extension $\pi^* \in \Pi^*$, then π can be realized simply by applying π^*. If $\pi^* \in \Pi_{C^*}$ is an extension of π, then

$$E(\pi(\mathbf{z}_t)) = T_{C,C^*}(\pi^*(\Phi^{(n)}(E(\mathbf{z}_a), E(\mathbf{z}_b), E(\mathbf{w}_t))))$$

since π^* is applied prior to transcoding to permute the information symbols in accordance with π. If $\pi^* \in \Pi_C$ is an extension of π, then

[3] During transcoding it may be possible to permute the information symbols, $\mathbf{z}_t$, while modifying (as opposed to permuting) certain check symbols. Though this type of data-movement operation is not technically a permutation over n-elements, it can still be thought of as a data-movement operation contained in Π_T.

$$E(\pi(\mathbf{z}_t)) = \pi^*(T_{C,C^*}(\Phi^{(n)}(E(\mathbf{z}_a), E(\mathbf{z}_b), E(\mathbf{w}_t))))$$

This latter case is advantageous because multiple different $\pi^* \in \Pi_C$ can be applied to different copies of a transcoded output, avoiding the need to transcode each differently permuted copy separately.

Sometimes we may want to implement a data-movement operation, $\pi \in \Pi$, which does not have an extension $\pi^* \in \Pi^*$. In this case we can **compose** several permutations in Π^* to produce an extension of π. We say that $\pi_1^*, \pi_2^* \in \Pi^*$ are composed to produce a third permutation, π^*, if the results of applying π_1^* and π_2^* separately to $E(\mathbf{z}_t)$ can be combined via an additional step of coded computation such that the result is $\pi^*(E(\mathbf{z}_t))$. It may also be useful to compose π_1^* and π_2^* in this way such that the result is $E(\pi(\mathbf{z}_t))$. In other words, permutations $\pi_1^*, \pi_2^* \in \Pi^*$ can be potentially be composed via an additional encoded computation step to produce a permutation in Π^* or a permutation in Π. A more detailed look at permutation composition is given in Section 9.6. In that section we also consider how Π can be chosen so as to accommodate a particular Π^*.

9.3.4 Spielman's Model

The framework for coded computation described in this section is directly inspired by the approach used by Spielman in [80]. Spielman's implementation of coded computation applies to normal algorithms on a k-processor hypercube. As stated in Section 9.2.2 these are algorithms in which, before each computation step, data is permuted by moving it between processors whose addresses differ in the j^{th} bit position. In this case it is easy to extend a particular permutation over k elements, π_j, to a permutation over n elements, π_j^*. To do this one can apply the equivalent π_j^* to an n-vertex hypercube. As Spielman observed, this approach is very well-suited to Reed-Solomon codes over $GF(2^q)$, for $q \geq \log_2 n$, since these codes are closed under this class of data-movement. The use of Reed-Solomon codes as C and C^* is discussed in Section 9.7.

9.4 Interpolation Polynomials

As explained in Section 9.3.1, a single step of coded-computation consists of applying an interpolation polynomial, $\Phi : G^3 \mapsto G$, component-wise to two encoded input vectors, $E(\mathbf{x})$ and $E(\mathbf{y})$, and an encoded instruction vector, $E(\mathbf{w})$. This produces an encoded output vector. Recall that the input and instruction vectors are encoded in a linear code C, and the output vector is encoded in *a different linear code*, C^*. For a particular C we must select C^* and Φ such that the encoded output vector encodes the correct output (i.e. the output produced by the unencoded computation step, $\mathbf{z} = \kappa^{(\mathbf{k})}(\mathbf{x}, \mathbf{y}, \mathbf{w})$).

In order to simplify this task, we assume that C and C^* are systematic linear codes. In this case, the first k symbols of $E(\mathbf{x})$, $E(\mathbf{y})$ and $E(\mathbf{w})$ are $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{w}$ respectively. This allows us to choose $\Phi : G^3 \mapsto G$ to be a polynomial interpolation of $\kappa : F^3 \mapsto F$ over G^3. In other words, Φ is a multivariate polynomial in G chosen to agree with κ over $F^3 \subseteq G^3$. Thus the first k symbols of the encoded output vector will necessarily be $\mathbf{z} = \kappa^{(\mathbf{k})}(\mathbf{x}, \mathbf{y}, \mathbf{w})$.

We can formally define an interpolation polynomial $\Phi(r,s,t) : G^3 \mapsto G$ for a function $\kappa : F^3 \mapsto F$ using the function $M_{x,X}(r) = \prod_{\rho \in X - \{x\}} \frac{r-\rho}{x-\rho}$, where $X \subseteq G$, $x \in X$, and arithmetic over G. Notice that $M_{x,X}$ is constructed so as to take value 1 when $r = x$ and 0 for all other $r \in X$. For other values of $r \in G - X$, $M_{x,X}(r)$ can take values in G other than 0 or 1. Also, $M_{x,X}(r)$ is a degree $|X| - 1$ polynomial.

Now let $X, Y, W \subseteq F$ denote the sets of values of x, y, and w, respectively, for which $\kappa(x,y,w)$ is defined. Then, $\Phi(r,s,t)$ is defined as follows.

$$\Phi(r,s,t) = \sum_{x \in X, y \in Y, w \in W} \kappa(x,y,w) M_{x,X} M_{y,Y} M_{w,W}$$

It follows that $\Phi(r,s,t) = \kappa(r,s,t)$ when $r \in X$, $s \in Y$ and $t \in W$. Unlike $\kappa(x,y,z)$, however, $\Phi(r,s,t)$ is also defined for all $r,s,t \in G$. The degrees of $\Phi(r,s,t)$ in r, s and t are $|X|-1$, $|Y|-1$, and $|Z|-1$, respectively.

By defining Φ in this way, we ensure that if C is systematic, C^* is systematic. We also ensure that if C is linear, C^* is linear over some larger basis. This observation is explained in detail in Section 9.4.2 below.

9.4.1 Examples of interpolation polynomials

In this section we consider some practical examples of Φ and illustrate that there is considerable flexibility in how Φ is implemented. First, consider a coded computation that corresponds to a regular computation comprised entirely of NAND gates. A single step of computation consists of k NAND functions applied in parallel. The instruction vector is constant (say all 0's), and the input alphabet is $F = GF(2)$. The interpolation polynomial, which doesn't depend on the instruction t, is $\Phi(r,s) = (1 - rs)$ for all choice G. Over $GF(2)$ $\Phi(r,s)$ has the same value as NAND.

Now consider the more realistic case where the computation contains both NAND and BUFFX gates (these act as pass-through gates that produce their first input as output). Now the interpolation polynomial becomes $\Phi(r,s,t) = (1-rs)t + r(1-t)$. When $r,s \in \{0,1\}$, this polynomial returns the value of the NAND(r,s) when the instruction is $t = 1$. When the instruction is $t = 0$ it returns r unchanged.

In practice, interpolation polynomials need not have only one variable for each input and instruction. For example, when encoding circuits with AND, OR and NOT, we can let $F = GF(3)$ and apply the polynomial $\Phi(r,s,t) = rs(1-t)(2-t)/2 + (r+s-rs)t(2-t) + (1-r)t(t-1)/2$, where $x,y \in \{0,1\}$, $w \in \{0,1,2\}$ and arithmetic is over $GF(3)$. Notice that $\Phi(r,s,0) = rs = \text{AND}(r,s)$, $\Phi(r,s,1) = r + s - rs = \text{OR}(r,s)$ and $\Phi(r,s,2) = 1 - r = \text{NOT}(r)$. Thus $\Phi(r,s,t)$ is an interpolation polynomial for the function $\kappa(x,y,w)$ that computes either AND, OR, and NOT when $t = 0, 1$ or 2, respectively.

Now suppose we wish to use a binary code, and thus do not want $t \in \{0,1,2\}$. As an alternative, we can add a second instruction variable and use $F = GF(2)$. In this case we have $\Phi(r,s,t_0,t_1) = rst_0t_1 + (r+s+rs)t_0(1-t_1) + (1-r)(1-t_0)(1-t_1)$, where $r,s,t_0,t_1 \in \{0,1\}$ and arithmetic is over $GF(2)$. Now $\Phi(r,s,1,1) = \text{AND}(r,s)$, $\Phi(r,s,1,0) = \text{OR}(r,s)$ and $\Phi(r,s,0,0) = \text{NOT}(x)$. The advantage of this construction, which refer to as a **binary expansion of** Φ, is that a computation with more than two instructions can still be encoded using binary codes.

If, instead of a boolean circuit with several types of logic gates, we wish to encode a computation performed by a hypercube, then Φ depends on the processors being employed. Complex processors, for which F is large, can potentially require very high degree interpolation polynomials. In later sections it will become clear that applying a sufficiently high degree Φ eliminates any error correction capability. In other words, when the degree of Φ is sufficiently high, C^* must contain all codewords of length n, and thus encoded outputs have a minimum distance of 1.

To address this limitation, a high degree interpolation polynomial can be implemented through successive applications of lower degree polynomials, with a transcoding operation performed after each application. For example if $\Phi(r,s,t) = r^4s^4t^4$, Φ has total degree 12. Instead of applying Φ directly, Φ can be realized by first applying $\Phi'(r,s,t) = rst$ component-wise to all three codewords in C, then transcoding the output, applying $\Phi''(r) = r^2$ to the result (which is once again a codeword in C), transcoding once more, and finally applying $\Phi''(r)$ a second time. By computing Φ in stages, C^* avoids the need to accommodate the component-wise application of a Φ with total degree 12.

More generally, the idea of decomposing Φ can be used to avoid ever applying a polynomial of total degree greater than 2. Consider our earlier example, $\Phi(r,s,t) = (1-rs)t + r(1-t) = t - rst + r - rt$. Each product term of this polynomial can be implemented in stages using successive applications of the polynomial $\Phi'(x,y) = xy$. After each application of $\Phi'(x,y)$ a transcoding operation is performed. This avoids the need for C^* to accommodate any polynomial with degree greater than two. Once each product term is computed the terms can be summed. We refer to this approach for computing Φ in stages as the **product decomposition of** Φ. Since C and C^* are linear no transcoding is required when terms are summed (although if a large number of terms are being summed, there may be a need for intermediate error correction).

To conclude, we note that the product decomposition and binary expansion of an arbitrary polynomial can be applied simultaneously to avoid ever applying any Φ other than $\text{AND}(x,y) = xy$ or $\text{XOR} = x + y$. In the later case, no transcoding is required.

9.4.2 Applying Polynomials to Linear Codes

Assume C is linear, has generator matrix M, and is closed under some set of permutations. In this section we examine the properties of C^*, the code that results when we apply a polynomial Φ to codewords in C (see Section 9.3.1 for a formal definition of C^* in terms of C and Φ).

We first express the encoded input vectors, and encoded instruction vector, as $E(\mathbf{x}) = \mathbf{x}M$, $E(\mathbf{y}) = \mathbf{y}M$ and $E(\mathbf{w}) = \mathbf{w}M$. Let $B_0 = \{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_k\}$ be the set of **basis** n**-vectors** of M. Then,

$$E(\mathbf{x}) = (\ldots, \sum_{i=1}^{k} x_i b_{i,j}, \ldots), \quad E(\mathbf{y}) = (\ldots, \sum_{i=1}^{k} y_i b_{i,j}, \ldots) \text{ and } E(\mathbf{w}) = (\ldots, \sum_{i=1}^{k} w_i b_{i,j}, \ldots)$$

When $\Phi(r,s,t)$ is applied component-wise to the encoded inputs, it produces the encoded output $E^*(\boldsymbol{z})$ whose jth component is given below.

$$E^*(\boldsymbol{z})_j = \Phi(\sum_{i=1}^{k} x_i b_{i,j}, \sum_{i=1}^{k} y_i b_{i,j}, \sum_{i=1}^{k} w_i b_{i,j})$$

If $\Phi(r,s,t)$ is a multivariate polynomial, it contains products of powers of r, s, and t plus a constant term. Thus $E^*(\boldsymbol{z})_j$ is the sum of products of powers of $b_{i,j}$. This allows us to express $E^*(\boldsymbol{z})$ as the sum of products of basis vectors. The **total degree of a term** in an interpolation polynomial $\Phi(r,s,t)$ is the sum of the degrees of the factors containing r, s, and t. The **degree** of an interpolation polynomial $\Phi(r,s,t)$ is the degree of a maximal degree term.

Let $\boldsymbol{a} \wedge \boldsymbol{b} = (a_1 * b_1, a_2 * b_2, \ldots, a_m * b_m)$ denote the **parallel product** of basis vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ where $a_i * b_i$ is multiplication in the field G. Then, the output codeword $E^*(\boldsymbol{z}) \in C^*$ can be expressed as a linear combination of the parallel product of powers of the basis vectors in B_0. For example, if $\Phi(r,s,t) = rst^2 + rt + s$, then $E^*(\boldsymbol{z})$ is the sum of parallel products of up to four basis vectors.

To express this categorization of C^* more clearly, we define the parallel product of a basis.

Definition 9.4.1 *Let $B_1 = \wedge(B_0)$ denote the set of all possible parallel products of pairs of basis vectors in B_0 (including the parallel product of a basis vectors with itself). Let $B_i = \wedge(B_{i-1}) \bigcup B_{i-1}$. Thus, B_i consists of the parallel products of up to i vectors, selected with repetition, from B_0. Finally, let C_i be the code consisting of all linear combinations of vectors in B_i.*

Lemma 9.4.1 *The size of B_i, denoted $|B_i|$, satisfies the following bound.*

$$|B_k| \leq (|B_0| + k)^{2^k}$$

Proof The proof is by induction. The basis for induction is that the bound has value $|B_0|$ when $k = 0$. Let the inductive hypothesis be that $|B_i| \leq (|B_0| + i)^{2^i}$ for $i \leq k-1$. We show that it holds when $i = k$.

It follows from the definition of B_i that

$$\begin{aligned} |B_i| &\leq |B_{i-1}|^2 + |B_{i-1}| = |B_{i-1}|(|B_{i-1}| + 1) \\ &\leq (|B_{i-1}| + 1)^2 \\ |B_k| &\leq ((|B_0| + k - 1)^{2^{k-1}} + 1)^2 \\ &= (|B_0| + k - 1)^{2^k} + 2(|B_0| + k - 1)^{2^{k-1}} + 1 \end{aligned}$$

On the other hand, using the binomial theorem and discarding terms we have the following expansion.

$$\begin{aligned} (|B_0| + k)^{2^k} &= ((|B_0| + k - 1) + 1)^{2^k} = \sum_{j=0}^{2^k} \binom{2^k}{j} (|B_0| + k - 1)^j \\ &\geq (|B_0| + k - 1)^{2^k} + \binom{2^k}{2^{k-1}} (|B_0| + k - 1)^{2^{k-1}} + 1 \end{aligned}$$

Because $\binom{2^k}{2^{k-1}} \geq 2$ for $k \geq 1$, the result follows. ■

Lemma 9.4.2 *Consider m codewords in C_i in B_i, $\mathbf{c_1} \ldots \mathbf{c_m}$, and Φ, a degree d polynomial in m variables. The result of applying Φ component-wise to the m codewords in C_i is a codeword in C_{di}.*

Proof Each $\mathbf{c_i}$ is a linear combination of vectors in B_i. As a result, $\wedge(\boldsymbol{c_i}, \boldsymbol{c_j})$ is a linear combination of vectors in B_{i+1}. It follows that the codeword resulting from the application of Φ is in B_{di}. ■

9.5 Transcoding

Recall from Section 9.3.2 that in the absence of errors the transcoding operation that follows a step of coded computation is described as follows:

$$E(\mathbf{z}_t) = T^{(n)}_{C,C^*}(E^*(\mathbf{z}_t))$$

where $\mathbf{z}_t$ and $E^*(\mathbf{z}_t)$ are defined below (here $a, b < t$).

$$\begin{aligned} \mathbf{z}_t &= \kappa^{(\mathbf{k})}(\pi_1(\mathbf{z}_a), \pi_2(\mathbf{z}_b), \mathbf{w}_t) \\ E^*(\mathbf{z}_t) &= \Phi^{(n)}(\pi_1^*(E(\mathbf{z}_a)), \pi_2^*(E(\mathbf{z}_b)), E(\mathbf{w}_t)) \end{aligned}$$

Here the permutation π_i^* is an extension of the permutation π_i and $\Phi^{(n)}$ is an extension of the function $\kappa^{(\mathbf{k})} : F^k \mapsto F^k$, defined through interpolation. The input and instruction vectors, $\mathbf{z}_a$, $\mathbf{z}_b$ and $\mathbf{w}_t$, are encoded in the code C and the output vector $\mathbf{z}_t$ is in the code C^*. Transcoding is needed to project $E^*(\mathbf{z}_t)$ to $E(\mathbf{z}_t)$.

If the number of errors that occur during a computation step and the subsequent transcoding step is small enough, the result of transcoding will be equal to or close to $E(\mathbf{z})$. For the transcoding operation $T^{(n)}_{C,C^*}$ to be fault-tolerant it must satisfy the following requirements.

1. Transcoding must be able to tolerate (i.e. correct) a small fraction of errors in $E^*(\mathbf{z}_t)$. These errors will have been introduced during component-wise application of Φ, or during the previous transcoding operation.

2. Since errors can occur during transcoding, these errors should be correctable by a subsequent transcoding operation with very high probability.

The first requirement is met if C^* has a sufficiently large error-correcting capability and an efficient error-correction algorithm. As we explain below, the second requirement can be met if C and C^* are linear (and possibly multidimensional) and the transcoding operation is structured appropriately.

To better understand the second requirement, consider the naive approach to transcoding: Simply decode $E^*(\mathbf{z}_t)$ to $\mathbf{z}_t$ then re-encode the result as $E(\mathbf{z}_t)$. The problem with this approach is that *a single error* in $\mathbf{z}_t$ will corrupt $E(\mathbf{z}_t)$. Such an error will not be correctable, thus additional fault-tolerance is required. One approach to fault-tolerant transcoding would be to develop code-specific fault-tolerant transcoding algorithms. In this section, however, we present two much more general techniques for tolerating transcoding errors based on the use of two-dimensional linear codes.

9.5.1 Transcoding Using 2D Codes

Spielman [80] addresses the need for fault-tolerant transcoding by letting C and C^* be two-dimensional Reed Solomon (RS) codes. In a 2D RS code both the rows and columns of each

2D codeword belong to a 1D RS code. Using such a code, or more generally *any 2D linear code*, $E^*(\mathbf{z}_t) \in C^*$ can be transcoded as follows.

1. Assume C and C^* are 2D linear codes. Let C_R and C_C denote the row and column codes of C, and let C_R^* and C_C^* be the row and column codes of C^*.
2. To transcode $E^*(\mathbf{z}_t)$ to $E(\mathbf{z}_t)$, first error-correct and decode each row of $E^*(\mathbf{z}_t) \in C^*$ using an arbitrary decoding algorithm for C_R^*.
3. Encode each of the decoded rows in the row code of C, C_R. The result is a codeword in an intermediate 2D code in which the columns belong to the column code C_C^*, but the rows belong to the row code C_R.
4. Error-correct and decode each column of the intermediate 2D codeword using an arbitrary decoding algorithm for C_C^*.
5. Encode each of the decoded columns in the column code of C, C_C. In the absence of errors this results in $E(\mathbf{z}_t) \in C$.

This transcoding algorithm can, in fact, be described very succinctly as "Transcode the rows of $E^*(\mathbf{z}_t)$, then transcode the columns". Here the transcoding of a row or a column simply entails decoding it, then re-encoding it in a different linear code. We emphasize that no assumption is made about the decoding or encoding algorithms being employed. We also note that when a row or column is re-encoded, it can potentially be placed in a new code. In other words, transcoding can be used to allow C to vary from one coded computation step to the next.

When a row is transcoded an error may occur which causes many symbols within that row to be in error. In this case, the row may no longer be close to the correct codeword in C_R. Notice, however, that any such error affects only one position in each column, hence column transcoding can correct these errors (as long as not too many rows become corrupted). Similarly, if a column transcoding error occurs, it results in at most one error per row. As such, these errors can be corrected during the row transcoding operations that take place after the next computation step. This analysis is quantified in Section 9.8.3.

This approach to fault-tolerant transcoding can be generalized to higher dimensional codes (e.g. 3D codes instead of just working with rows and columns), although it is unclear if there is any benefit. It is also possible to apply this technique recursively if C_R, C_L, C_R^* and C_L^* are themselves 2D linear codes. In this case C_R^* and C_L^* could themselves be transcoded (in steps 2 and 4, respectively) first by rows, then by columns.

9.5.2 Transcoding Using Checksums

In the previous section C and C^* are assumed to be 2D linear codes. During each computation step $\Phi^{(n)}$ is applied to codewords in C to produce a codeword in C^*. This implies that both the row and column codes of C and C^* must be chosen so as to accommodate component-wise application of Φ. In this section we propose an approach to fault-tolerant transcoding that relaxes this requirement.

1. Instead of assuming that C and C^* are 2D linear codes, assume that each encoded k-tuple is divided into m words, each of length k/m (assume m divides k). Each word is encoded in a linear code, C_1. A codeword in C consists of the m separately encoded codewords. Similarly a codeword in C^* consists of m separately encoded codewords in C_1^*.

2. To transcode $E^*(\mathbf{z}_t) \in C^*$, form a 2D codeword from $E^*(\mathbf{z}_t)$ by treating the m separately encoded codewords in C_1^* as rows in a 2D code, then employ a second linear code, C_2, to encode the columns of the m rows.

3. As in Section 9.5.1 decode each row using an arbitrary decoding algorithm for C_1^*, then encode the result in C_1.

4. To correct row transcoding errors, decode each of column using an arbitrary decoding algorithm for C_2.

Notice that during coded computation steps Φ is only applied to codewords in C_1, not C_2. As long as this application of Φ does not introduce too many errors, the resulting m codewords in C_1^* will not contain too many errors. Furthermore, when the columns of these codewords are encoded to produce codewords in C_2, not too many of the columns will be corrupted by these errors. As such, if some row transcoding operations fail, the columns will still be able to correct these failures. This analysis is formalized in Section 9.8.4.

The advantage of this approach is that C_2 can be an arbitrary linear code, and thus it can have better error-correcting capabilities and more efficient encoding and decoding algorithms than C_1. The disadvantage of this approach is that, since columns are encoded in C_2 after Φ is applied, any column that contains an error prior to transcoding will contain incorrect check symbols. Still, as long as not too many such errors exists, the row transcoding will correct these errors. At that point, the column codes will successfully protect against row transcoding errors. Since row transcoding is much more likely to fail than applications of Φ (since many more logic gates are involved) this appears to be a reasonable tradeoff.

Finally, we note that both transcoding procedures we have described allow for permuting of encoded data. In both cases the encoded columns can be rearranged when rows are decoded, but have not yet been re-encoded. Similarly encoded rows can be rearranged when the columns are decoded. As noted in Section 9.3.3 these permutations of the information symbols can be used to implement data-movement.

9.5.3 Transcoding in Parallel Architectures

In order to make transcoding operations fault-tolerant, multiple codewords are transcoded in parallel. This might be taken to imply that any given computation step must be large enough to accommodate a sufficiently large number of codewords. We emphasize that another alternative would be to consider parallel (or pipelined) architectures in which the transcoding operations for multiple computation step, of multiple concurrent computations, are transcoded in parallel. For example, in a 2D code, each row codeword could correspond to completely different computation step in a fault-tolerant parallel architecture. This is an important point, because otherwise coded

computation could only be applied if each individual computation step were sufficiently wide to accommodate extremely large codewords. It also suggests designing a fault-tolerant architecture in which a number of centrally located "transcoding units" are used to transcode the outputs of a number of concurrent computation steps. Checksum-based transcoding is particularly well-suited to this approach, since here m distinct codewords come together as rows in 2D code only during transcoding.

9.6 Codeword Permutations

This section examines the role of codeword permutations, first discussed in Section 9.3.3, in greater detail. As explained previously, an encoded computation must not only implement the computation steps of a regular computing network (through component-wise application of interpolation polynomials), but also implement the network's data-movement operations. This is accomplished by applying some restricted set of permutations to the encoded data.

Recall from Section 9.2 that the output vector, $\mathbf{z}_t$, of each step of a regular network's computation is permuted by some $\pi \in \Pi$ before it is supplied as input to a subsequent computation step. In the corresponding encoded computation, this data-movement operation may be implemented by permuting the encoded output vector either before, during, or after the transcoding operation that follows each computation step. As defined in Section 9.3.3, we use Π^* to denote the set of permutations that can be applied to the encoded data. In this section we discuss how permutations in Π^* can be used to realize permutations in Π. Section 9.6.1 considers the case when each $\pi \in \Pi$ has an extension (defined below) in Π^*. Section 9.6.2 then describes how permutations in Π^* can be composed via a small number of additional computation steps to produce extensions of $\pi \in \Pi$.

9.6.1 Direct Application of Extension Permutations

In this section we assume that each $\pi \in \Pi$ has an extension $\pi^* \in \Pi^*$. Recall from Section 9.3.3 that a permutation over n elements, π^*, is an **extension** of a permutation over $k < n$ elements, π, if π^* applies π to the first k of the n elements. If π^* is an extension of π, any data-movement operation that applies π after a step of unencoded computation can be realized through application of π^* after a step of encoded computation. At first glance, it may appear overly optimistic to expect that every permutation in Π has an extension Π^*. Fortunately, the following examples suggest otherwise.

First, consider the case described by Spielman in [80] (see Section 9.3.4). Here the regular computing network being encoded corresponds to a normal algorithm on a hypercube. In this case all data-movement permutations correspond to swapping data along a single dimension of a k-processor hypercube. When data is encoded, each information symbol corresponds to the data located at a particular processor. As such, each information symbol is indexed by a binary string of length $\log_2 k$, and each permutation in Π is extended by any permutation that flips the appropriate bit of each information symbol's index string.

Spielman's construction encodes data using length n Reed-Solomon (RS) codes constructed over a field $GF(2^q)$. As Spielman observes, when $n = 2^q$ (or a smaller power of two) these codes are

closed under permutations that correspond to swapping data along the dimensions of an n-processor hypercube. More specifically, when $n = 2^q$ each codeword symbol can be indexed using a binary string of length q and Π^* contains all permutations that corresponding to flipping the same bit in each binary index (Section 9.7.1 provides an explanation of why this statement holds). Since k is also a power of two, each permutation in Π has an extension in Π^*, provided that the information symbols are located in the positions indexed by binary strings in which the first $q - \log_2 k$ bits are 0 (in other words, the vertices in a k-vertex hypercube are embedded in an n-vertex hypercube by padding each vertex's index with 0's). The same holds true if multidimensional RS codes are used, and if Reed-Muller codes are used (see Section 9.7.2).

As a second example in which Π^* contains an extension of each $\pi \in \Pi$, suppose Π contains only cyclic shifts (as noted at the end of Section 9.2.2 this is still sufficient to perform arbitrary computations with limited overhead). If n is a multiple of k, Π^* can contain permutations that simultaneously shift the k information symbols, and $n - k$ information symbols, in blocks of k. For example, suppose C is chosen to be a length n RS code over the field $GF(p^2)$, where p is an arbitrary prime. Let $n = cp$, where c is a constant less than p. If $k = p$ information symbols are encoded, this information can be indexed by the tuples $(0, 0), \ldots, (0, p-1)$ and the check symbols indexed by the tuples $(i, 0), \ldots, (i, p-1)$ for $1 \leq i < c$. As explained in Section 9.7.1, this code is closed under any permutation that adds a constant to the second position of each symbol's index. This means that the p information symbols can be cyclicly shifted if each block of p check symbols is also cyclicly shifted by the same amount. For intuition as to why such permutations are permitted, note that RS codes are constructed by evaluating a polynomial, $p(x)$, of bounded degree at a n locations. Adding a constant to x shifts the polynomial but does change its degree.

As a final very general example of how Π^* can effectively contain an extension of each $\pi \in \Pi$, recall the observation at the end of Section 9.5.2. Namely, during the transcoding of 2D codes, it is possible to rearrange both the rows and columns of the encoded data (i.e. the columns containing the information symbols can be permuted, followed by the rows). This type of data-movement, though not technically a permutation (some check symbols may be modified instead of permuted) is sufficient to implement two-dimensional cyclic shifts. This in turn allows the encoded network to implement the data-movement required by a 2D mesh (see Section 9.2.2). Also, if the number of encoded information symbols is a power of two, shuffling their rows or columns during transcoding is sufficient to implement the data-movement required by a normal hypercube algorithm.

9.6.2 Composing Permutations to Realize Extensions of Π

Sometimes it may be necessary to implement a data-movement operation for which $\pi \in \Pi$ does not have an extension in Π^*. To do this, the permutations in Π^* must be combined, via computation steps, to form permutations that are extensions of those in Π. This problem is very similar to implementing an arbitrary computation on a regular computing network for which Π is restricted.

Section 9.2.2 described how arbitrary permutations over k elements can be performed by a switching network of logarithmic depth. A Beneš switching network [82], for example, consists of two back-to-back butterfly (or FFT) graphs [81, p. 310]. At each stage of this network two pieces of data are moved to a node and potentially swapped (i.e. each node acts as a switch). The necessary

data-movement at each stage of the network can be implemented by performing two cyclic shifts, one to the right and one to the left, each by a number of places that is a power of two.

Let $E(\boldsymbol{z})$ be the transcoded output of some stage of the encoded network. As long as Π^* contains cyclic shifts, a Beneš network can be implemented as an encoded computation and used to realize an arbitrary permutation of $\boldsymbol{z}$. To implement each stage of the Beneš network, $E(\boldsymbol{z})$ can be copied twice, and each copy shifted by the appropriate power of two. The computation step performed by the switching network simply corresponds to each node selecting the appropriate entry from either $E(\boldsymbol{z})$, or one of the two shifted copies of $E(\boldsymbol{z})$. This selection operation can be implemented using two applications of the polynomial $\Phi(r, s, t) = rt + s(1 - t)$, where t is used to determine which element is selected.

After each application of Φ a transcoding operation is required, but even if Π contains arbitrary permutations, the entire switching network will require at most $4 \log_2 n$ transcoding operations. In other words, arbitrary permutations can be carried out with only logarithmic overhead provided Π^* contains cyclic shifts. Furthermore, one suspects that Π would not contain arbitrary permutations and that Π^* may contain more than just cyclic shifts. This may allow for a shallower switching network.

Composing Multiple Permutations Via Masking

In the above example, encoded output vectors are copied, permuted in two different ways, and then the differently permuted copies are combined via an interpolation polynomial that acts as a switch. It is worth noting that this approach can be extended to an arbitrary number of differently permuted copies of the output vector. To see how, call a codeword, $E(\boldsymbol{m}) \in C$, a "mask" if its information symbols, $\boldsymbol{m}$, consist only of 0's and 1's. Now suppose that an arbitrary codeword $E(\boldsymbol{x}) \in C$ is multiplied by $E(\boldsymbol{m})$, meaning the polynomial $\Phi(r, s) = rs$ is applied component-wise, and the product transcoded. The result, $E(\mathbf{z}) = T^{(n)}_{C,C^*}(\Phi^{(n)}(E(\mathbf{x}), E(\mathbf{m})))$, is a codeword C in which some of the information symbols in $\boldsymbol{x}$ are now set to 0 while the others remain unchanged.

Now consider m copies of $E(\boldsymbol{x})$, each of which has been permuted by a different permutation $\pi^*_i \in \Pi^*$. To combine these m differently permuted copies, each copy can be multiplied by a different mask, $E(\boldsymbol{m_i})$. The m products, $E^*(\mathbf{z}_i) = \Phi^{(n)}(\pi^*_i(E(\mathbf{x})), E(\mathbf{m_i}))$, are codewords in C^*. Since C^* is linear, these m codewords can be summed and their sum transcoded. The resulting codeword in C is equal to $\sum_{i=1}^{m} E(\mathbf{z}_i)$ (even though only a single transcoding operation was required). Finally, suppose we have chosen the masks such that at most one $\boldsymbol{m_i}$ has a 1 in each position. Then each information symbol of $\sum_{i=1}^{m} E(\mathbf{z}_i)$ comes from a differently permuted copy of $E(\mathbf{x})$, as specified by which ever $\boldsymbol{m_i}$ has a 1 in each position.

This general idea of "masking" different positions in differently permuted copies of $E(\boldsymbol{x})$, then combining these copies via addition, allows us to realize a substantial range of permutations using only a single additional transcoding step. For example, the switching operations described in the previous section can be implemented using masks. This avoids the need to perform two transcoding operations to implement each stage of the switching network, thus reducing the total number of transcoding operations from $4 \log_2 n$ to $2 \log_2 n$. What's more, when masking is used, the masks themselves can be pre-computed and hard-wired into a circuit. Finally, masking makes it easy to

consider encoding each step computation in multiple blocks. If each computation step is extremely wide (i.e. k, and hence n, are very large), it may make sense to simply divide the steps up into b blocks and encode each block separately. In this case, each encoded input vector corresponds to b separately encoded blocks of information symbols. The b encoded blocks can be computed on and transcoded separately, but there is still the need to move data between blocks. Masking allows for this. To exchange data between several blocks, a different mask can be applied to each block and the results combined via addition.

9.6.3 Data-Movement Overhead

The constructions described in this section demonstrate that a wide range of data-movement operations can be implemented efficiently during coded computation. The exact overhead associated with realizing an arbitrary permutation depends on Π^* and the overhead associated with transcoding. As we have noted Π^* can contain a fairly wide range of permutations simply by virtue of how information symbols can be permuted during transcoding. Furthermore, the technique of masking provides significant flexibility regarding how permutations are composed.

In general, the overhead associated with realizing arbitrary data-movement permutations is at most logarithmic. We note, however, that today's hardware employs circuits that are embedded on a chip. As such, they do not implement arbitrary data-movement. We suspect that the data-movement used in practice would require only constant factor overhead. Furthermore, it is often possible to design circuits and algorithms with specific data-movement constraints in mind. We thus conclude that transcoding, and not data-movement, appears to be the primary source of overhead associated with coded computation. Even so, it will still be necessary going forward to investigate how the constructions of this section can be applied to specific embedded architectures.

9.7 Families of Codes

In this section we examine specific families of parallel product codes and consider the overhead associated with using these codes for coded computation. For a family of codes to allow for efficient use in our coded computation framework, it must have a reasonably good rate and be relatively simple to transcode (i.e. it must have efficient decoding and encoding algorithms). In his work on coded computation, Spielman proposed using **Reed-Solomon (RS) codes** [80]. In this section we consider these and other codes based on polynomial interpolation, such as **Reed-Muller (RM) codes**.

9.7.1 Reed-Solomon Codes

An $[n, k, d]_q$ code is one that has block length n, k information symbols, minimum distance d, and code alphabet of size q. Given a finite field, G, an $[n, k, d]_{|G|}$ RS code, which we denote $RS[n, k, d]_{|G|}$ is defined using degree $d-1$ polynomials over G. A systematic $RS[n, k, d]_{|G|}$ code is defined in terms of an arbitrary subset $H = \{h_0, ..., h_{k-1}\}$ of G. Given the k information symbols, $\boldsymbol{a} = (a_0, a_1, \ldots, a_{k-1})$, $a_i \in G$, let $p_{\boldsymbol{a}}(u)$ denote the degree $k-1$ interpolation polynomial such

that, for each h_i, $p_{\boldsymbol{a}}(h_i) = a_i$. The encoding of $\boldsymbol{a}$ in $RS[n,k,d]_{|G|}$ is simply the value of $p_{\boldsymbol{a}}(u)$ evaluated at all points in G, or if $|G| > n$, a subset S of G of size n where $H \subset S$.

Since any two degree $k-1$ polynomials agree on at most $k-1$ points in G, $RS[n,k,d]_{|G|}$ codewords have minimum distance, $d = n-k+1$. Furthermore, since degree $k-1$ polynomials are closed under addition, the code is guaranteed to be linear. To see that RS codes are examples of parallel product codes, it suffices to observe that one possible basis for $RS[n,k,d]_{|G|}$ is $\boldsymbol{b_i} = (1^i, \alpha^i, ..., \alpha^{(|G|-2)i})$, where α is the generator of G and i ranges from 1 to $k-1$. We note, however, that this is not the basis used in the construction above, since it does not cause RS codes to be systematic.

To better illustrate how RS codes allow for coded computation, consider the computation $E^*(\mathbf{z}_{t+1}) = \Phi^{(n)}(\pi_1^*(E(\mathbf{z}_t)), \pi_2^*(E(\mathbf{z}_t)), E(\mathbf{w}_t))$ on a hypercube using normal algorithms where $\Phi^{(n)}$ has degree δ. Because the codeword $E(\mathbf{z}_t)$ is closed under normal algorithm permutations, the three codewords $\pi_1^*(E(\mathbf{z}_t))$, $\pi_2^*(E(\mathbf{z}_t))$, and $E(\mathbf{w}_t)$ are the values of the polynomials $\tilde{p}_{\boldsymbol{x}}(u)$, $\tilde{p}_{\boldsymbol{y}}(u)$ and $\tilde{p}_{\boldsymbol{z}}(u)$, respectively. The result of applying $\Phi^{(n)}$ to each of the n codeword symbols is a codeword in $C^* = RS[n, \delta(k-1)+1, d^*]_{|G|}$ which corresponds to the degree $\delta(k-1)$ polynomial $\Phi(\tilde{p}_{\boldsymbol{x}}(u), \tilde{p}_{\boldsymbol{y}}(u), \tilde{p}_{\boldsymbol{w}}(u))$ evaluated at the same n elements of G. Here the minimum distance between codewords in C^* satisfies $d^* = n - \delta(k-1)$. In order for C^* to have error correction capability, $d^* \geq 3$ and hence $k \leq (n-3)/\delta + 1$.

Overhead

RS codes have a large minimum distance, they can be encoded with logarithmic overhead (i.e. via an $O(n \log n)$ algorithm) and decoded deterministically using polylogarithmic overhead [85, 86]. Unfortunately, they also have a significant drawback in that their alphabet-size, $|G|$, must be at least as large as their length, n. This means that even a simple extension polynomial, such as $\Phi(r,s,t) = rt + s(1-t)$ (here $\delta = 2$), must be applied to each check symbol in a codeword using finite field arithmetic. When encoding an arithmetic circuit (a circuit in which each gate is a finite field operation), this is not unreasonable, but when encoding a logic circuit, in which gates are simple binary operations, the overhead associated with computing on check symbols is $o(\log(n))$ (since each check symbol computation operates on at least $\log(n)$ bits).

9.7.2 Reed-Muller Codes

Reed-Muller (RM) codes can be used in lieu of RS codes for C and C^*. They are also defined by evaluating a polynomial. However, instead of a using a one-dimensional bounded-degree polynomial over an arbitrary field G, an r-dimensional, bounded-degree polynomial over the field $GF(2)^r$ is used. Each codeword in the Reed-Muller code $RM[n,k,d]_2$ corresponds to an r-dimensional, degree-m polynomial, $p_{\boldsymbol{a}}(u_1, ..., u_r)$ over all $n = 2^r$ values in $GF(2)^r$. Since there are only $\sum_{i=0}^{m} \binom{r}{i}$ such polynomials, we require $k \leq \sum_{i=0}^{m} \binom{r}{i}$. It is not hard to show that $d = 2^{r-m}$ [87] and hence it is desirable for m to be as small as k allows.

Like RS codes, RM codes are linear since the set of r-dimensional degree-m polynomials used to define $RM[n,k,d]_2$ is closed under addition. Furthermore, like RS codes, RM codes can be made systematic by selecting polynomials via interpolation. Also as with RS codes, RM codes allow

for application of a polynomial Φ. For example, when the extension polynomial $\Phi(r, s, t) = rt$ is applied to two codewords in $C = RM[2^r, \sum_{i=0}^{m} \binom{r}{i}, 2^{r-m}]_2$, the result is a codeword in $C^* = RM[2^r, \sum_{i=0}^{2m} \binom{r}{i}, 2^{r-2m}]_2$. Notice that the minimum distance between codewords has been reduced from 2^{r-m} to 2^{r-2m}.

Overhead

In the example above, the rate of C and error-correcting capabilities of C^* are not nearly as good as for RS codes. However, for RM codes Φ can be a simple binary operation (in this case AND), which can greatly reduce the complexity of other operations. RM codes are also simpler to encode and decode than RS codes. Like RS codes, RM codes can be encoded via a simple $O(n \log n)$ algorithm, but now only binary arithmetic is required. They can also be decoded using $O(n \log n)$ arithmetic operations [88]. We also note that both RS and RM codes are closed under a wide range of permutations including, for example, those corresponding to data movement on a hypercube.

9.7.3 Other Polynomial Codes

RS and RM codes are actually two extreme examples of more general polynomial-based codes. RS codes are formed by interpolating over 1-dimensional polynomials over an alphabet G, for which $|G| > k$. RM codes, in contrast, use a binary alphabet and an m-dimensional polynomial, and thus require $k < 2^m$. It is perfectly reasonable to consider codes with an intermediate-sized alphabet, say $|G| > \log k$, constructed from polynomials with intermediate dimension.

In general, an m-dimensional polynomial over G of degree d can interpolate over $k = (d+1)^m$ information symbols. Here d denotes the maximum degree of any given variable, and thus the total degree of the polynomial is at most md. If $p(x_1, \ldots x_m)$ interpolates over k points in G^m, a systematic codeword is generated by evaluating p at n points in G^m. One standard approach for selecting these n points is to consider a set $S \subseteq G$ such that $n = |S|^m$. If two distinct m-dimensional polynomials, p_1 and p_2, of total degree md are evaluated at all n points in S^m, the well-known Schwartz-Zippel Lemma reveals that they will differ in at least $n - nmd/|S|$ locations [89, p. 29].

Now consider the code, C, with codewords produced by evaluating all m dimensional polynomials, p_i, of total degree at most md at all n points in S^m. C can tolerate errors as long as $|S| > md$. Furthermore, if the $\Phi(r, s, t) = rt$ is applied to two codewords in C, the resulting codeword in C^* corresponds to the polynomial $p_1 p_2$, which has total degree at most $2md$. Thus C^* can tolerate errors as long as $|S| > 2md$. Finally, since $n = |S|^m$ and $k = (d+1)^m$, C has rate $k/n = (|S|/(d+1))^m$.

9.7.4 Multidimensional Codes

As explained in Section 9.5, multidimensional codes play a very important role in ensuring that errors can be corrected during transcoding operations. This is an important advantage of using linear codes, such as codes based on interpolation polynomials of bounded degree. If C is a systematic linear $[n, k, d]$ code, a systematic two-dimensional code, $C \times C$, can be formed from C by encoding a grid of k-by-k information symbols first by rows, then by columns. The new code has parameters

$[n^2, k^2, d']$, where $d' \geq d^2$. If one-dimensional codes C and C^* can be used for coded computation (i.e. component-wise application of Φ), so can two-dimensional codes $C \times C$ and $C^* \times C^*$. It is also acceptable if a different pair of codes, C and C^*, are used in each dimension of the 2D codes. Higher dimensional codes can be constructed from C and C^* as well.

9.8 Overhead

In this section we bound the overhead required to implement each level of a regular computing network, $\mathcal{C}_i$, using coded computation. We use $\mathcal{C}'_i$ to denote the corresponding level of the encoded computation. Here each $\mathcal{C}'_i$ consists of a computation step, followed by a transcoding step. Both steps must be implemented in a way that allows them to tolerate gates that fail with probability p_f. This is accomplished through a combination of gate repetition and code-based fault-tolerance.

9.8.1 Reliability via Repetition

As explained in Section 9.1.1, repetition-based fault-tolerance is one method for making an arbitrary network of computing elements fault-tolerant. In this approach each computing element (in our case logic gates) is assumed to fail independently at random with probability p_f. As long as p_f is below some critical threshold, it suffices to repeat each gate $r = O(\log |\mathcal{C}|)$ times, then use r constant-sized majority gates to suppress the number of errors among the r repeated outputs. Here $|\mathcal{C}|$ denotes the total number of gates in the circuit. In the case of a T-step regular computing network $|\mathcal{C}| \leq T \max |\mathcal{C}_i|$, which implies that $r = O(\log \max |\mathcal{C}_i| + \log T)$. Thus if T is much larger than $|\mathcal{C}_i|$, which may well be the case for a lengthy computation, $\log T$ is the dominant term.

Notice that using a repetition-based approach to fault-tolerance is equivalent to encoding each layer of the computing network, $\mathcal{C}_i$, with a repetition code. Using such a code no transcoding is required, merely error suppression via a layer of $|\mathcal{C}_i|r$ constant-sized majority gates. The total overhead is thus $O(r) = O(\log |\mathcal{C}|) = O(\log T + \log \max |\mathcal{C}_i|)$. As we demonstrate, when T or $\max |\mathcal{C}_i|$ is sufficiently large, it is possible to provide more efficient fault-tolerance using coded computation. Since gate repetition can serve as a building block of coded computation (as explained below) we begin with some repetition-related terminology.

In this section we refer to each group of r repeated gates, along with the accompanying r majority gates, as a **cluster**. We also refer to the r wires coming out of a cluster, and the two sets of r wires coming into a cluster as **bundles** (we assume that each repeated gate takes two inputs). For a given circuit we use the terms **output clusters** and **input clusters** to refer to clusters that contain output gates and input gates, respectively. Each cluster takes two bundles as input, and produces one bundle as output. Each bundle is said to be α**-correct** if no more than αr of the values it transmits are in error. For a given value of α, a cluster is said to $fail$ if both of its input bundles are α-correct, its output bundle is not α-correct.

In the context of the von Neumann model, in which all gates can fail with probability p_f, it is standard to take the output bundles of each output cluster and "decode" them to single-bit outputs via tree of unreliable majority gates. If a given bundle of outputs is α-correct, it can be shown that for sufficiently small value of α, the output of the tree of majority gates that takes that bundle

as input will be correct with probability $1 - O(p_f)$ [70]. In this section we do not employ this tree-based construction, since in the context of coded computation it is acceptable for outputs (and inputs) to be encoded in a repetition code (i.e. it is OK if each output remains in repeated form). Furthermore, we will find it extremely useful to compose bit-repetition with other codes. To this end, we consider **r-repeated Reed-Muller** and **r-repeated Reed-Solomon** codes in which each bit of each symbol in a Reed-Muller or Reed-Solomon codeword is repeated r times. This allows a circuit operating on these codewords to employ repetition-based fault-tolerance without needing to worry about decoding (or reencoding) the repeated outputs via trees of unreliable gates.

By employing r-repeated codewords, repetition-based error correction can act as a building block for more efficient code-based fault-tolerance. As such, the overhead associated with coded computation will itself be dependent on the $O(\log |\mathcal{C}|)$ overhead associated with gate repetition. For intuition behind this overhead, notice that the expected number of gate failures among r gates is rp_f, thus the probability that a particular set of r repeated gates introduces more than $cp_f r$ errors, for $c > 1$, grows exponentially small in r (for proof, consider the Chernov bound in Section 5.2.1). As such, if the two input bundles to a cluster are both α-correct, then with very high probability at most $2\alpha r + cp_f r$ errors will be present among the r repeated outputs supplied to the cluster's r majority gates. As long as cp_f is $O(\alpha)$ and α is not too large, it can be shown that the majority gates will be able to correct a large fraction of these errors with very high probability [70]. This shows that when the two input bundles supplied to a cluster are α-correct, the cluster fails to produce an output bundle that is α-correct with probability that is exponentially small in r. Finally, notice that the entire computation performed by all $|\mathcal{C}|$ clusters succeeds with probability at least $1 - \epsilon$ as long as each of the $|\mathcal{C}|$ clusters fails with probability at most $\epsilon/|\mathcal{C}|$. For fixed ϵ this requires that $r = O(\log |\mathcal{C}|)$.

When a tree of majority gates is used to decode each output cluster's output bundle, each single-bit output must fail with probability at least p_f (since the majority gate at the root of each tree fails with probability p_f). As such, there is no particular advantage to choosing r such that ϵ (the probability that some output bundle is not α-correct) is much smaller than p_f. When the majority tree construction is not employed, and the outputs are left in encoded (i.e. repeated) form, there may in fact be a reason to choose r such that each cluster fails with probability much less than $p_f/|\mathcal{C}|$. Suppose, for example, that a particular value of r is sufficient to ensure that each cluster fails with probability $O(p_f/|\mathcal{C}|)$. In this case doubling r will ensure that each cluster fails with probability $O(p_f^2/|\mathcal{C}|^2)$. More generally we can use gate repetition with $r = O(\log |\mathcal{C}|)$ to ensure that each cluster's failure rate is polynomially small in $|\mathcal{C}|$. In other words, von Neumann gate repetition with $r = O(\log |\mathcal{C}|)$ can be used to ensure that all output bundles of the circuit represent the correct value not just with probability $O(p_f)$, but with probability $O((p_f/|\mathcal{C}|)^c)$ for any fixed constant c. If these bundles are decoded using reliable majority gates, the entire computation will succeed with probability $1 - O((p_f/|\mathcal{C}|)^c)$.

In the context of coded computation, we will use gate repetition to ensure that each step of coded computation produces output bundles that have a polynomially small chance of representing an incorrect value. These output bundles will each correspond to a repeated bit in an r-repeated codeword. Coded computation will then be performed on the r-repeated codewords in some code C. If C can tolerate up to e errors, the coded computation will be able tolerate up to e incorrect

bundles. As we explain, the probability of e such failures occurring after any given step of coded computation can be made to decrease exponentially in e. Thus, as demonstrated below, coded computation employed on top of gate repetition is able ensure that any given computation step, $\mathcal{C}_i$, fails to produce a correct output with exponentially small probability using polylogarithmic overhead. In contrast, gate repetition requires $r = O(|\mathcal{C}_i|)$ overhead to achieve an exponentially small failure rate for a given computation step. As such, coded-computation is asymptotically superior to repetition-based fault-tolerance.

9.8.2 Basic Analysis Framework

The computation of a fault-tolerant regular network is divided into T steps. Each step consists of a computation step, followed by a transcoding step. During coded computation the input to a particular computation step, or the output from a particular transcoding step, is said to **correct** if the codeword it is closest to in C is the codeword that would be present in an error-free coded computation. Also, for codes C^* and C, let the set of **correctable codewords** in C^* be those vectors that an error-free transcoding operation will map to the correct codeword in C. For a computation to fail there must be some step in which one of the two events occurs.

1. **A computation step failure:** The inputs to the computation step are correct, but the output is no longer a correctable codeword.

2. **A transcoding step failure:** The input to the transcoding step is a correctable codeword representing the correct value, but the step's output is no longer correct.

We can bound the probability that a coded computation fails by bounding the probability that either of these two events occurs on any given step. Let $P(E_1)$ and $P(E_2)$ denote these probabilities for whichever step has the largest chance of failing (in practice, we expect all steps to have an approximately equal chance of failure). The probability that the computation fails, denoted P_f, is at most $P_f \leq T(P(E_1) + P(E_2))$. Thus our goal is to bound $P(E_1)$ and $P(E_2)$.

By definition, an error-free transcoding step will be able to correct any correctable codeword that the computation step outputs. Thus a transcoding failure occurs only if a sufficient number gate failures occur during the transcoding operation. Too many such failures will corrupt a potentially correctable codeword. Thus $P(E_2)$ depends on how transcoding is implemented. The following subsections consider several approaches to transcoding.

In a computation step, if two correct codewords are supplied as inputs, the output may not be correctable. The probability of a computation step failure will depend on how many, and where, errors within the correctable inputs have occurred. To get a very precise bound on $P(E_1)$, we would need to associate a probability distribution with where errors occur in a transcoding step's correctable output. We would then need to consider the likelihood that a particular distribution of errors, among a pair of correct inputs, results in computation step failure.

This calculation would be extremely cumbersome. As such, it makes sense to designate a subset of all possible encoded inputs as **sustainable codewords**. An input vector to a computation step is sustainable if it is correct (i.e. it is closest to the correct codeword in C), and the errors it

contains fulfill some pre-chosen criteria. This criteria, which will depend on the code being used, must ensure that any two sustainable inputs to a computation step are very likely to produce a correctable output.

Along similar lines, it is also useful to designate some subset of the correctable codewords as **highly correctable codewords**. These are vectors that are very likely to be mapped to a correct output by a transcoding operation in which gates fail. Using the idea of both highly correctable and sustainable codewords we can redefine events E_1 and E_2 as follows.

1. **A computation step failure:** The inputs to the computation step are sustainable, but the output of the computation step is not highly correctable.

2. **A transcoding step failure:** The input to the transcoding step is highly correctable, but the output is not sustainable.

Note that the bound, $P_f \leq T(P(E_1) + P(E_2))$, still holds, since as before, if neither of the two events occurs on any step, the computation succeeds. Now, however, $P(E_1)$ and $P(E_2)$ are much easier to bound.

As a simple example, consider the above events when a repetition code is used (here C and C^* are the same code). If each symbol is repeated r times, all codewords with fewer than $r/3$ errors in each position can be considered highly correctable. Sustainable codewords can then be defined as having fewer than $r/7$ errors in each position. This ensures that any two sustainable codewords, when computed on, are likely to produce a correctable output (provided p_f is sufficiently small). As highlighted above, this is essentially the approach taken in [70], although instead of a transcoding step majority-based error correction is employed. Using groups of r constant-sized majority gates, any highly correctable codeword can be mapped to a sustainable codeword with high probability.

Errors in Computation Steps

During a computation step, n functions, Φ, are computed in parallel. Let ϵ_c denote an upper bound on the probability that any particular application of Φ fails (meaning that an incorrect output is produced). Such a failure is called a **computation error**. Assume Φ takes three inputs, in which case each computation step takes two sustainable codewords as input vectors, and one instruction vector, which we assume is fault-free.[4]

If each of the n functions applied during a computation step is computed by a single gate, $\epsilon_c = p_f$, the failure rate of individual gates. If however, codewords use a non-binary alphabet, Φ will be more complex. If each function is computed using G_c gates, $\epsilon_c < G_c p_f$. When G_c is sufficiently large, $G_c p_f$ will be too large to provide a useful bound. In this case, additional repetition-based fault-tolerance can be added to ensure that ϵ_c remains close to ϵ. By using r-repeated codewords, and repeating each bit of all codewords r times, Φ can be computed by a circuit that employs von Neumann gate repetition. As noted above, this allows $O(\log n)$ overhead to be used to ensure that ϵ_c is polynomially small in n.

[4]We make this assumption because, in practice, instructions can be hard coded into a circuit. If we did not wish to make this assumption, we would simply increasing ϵ_c accordingly.

It is also worth noting that since each computation step will involve finite field arithmetic, it may be practical to employ algorithm-based fault-tolerance in lieu of, or in addition to, gate repetition. For example, if the G_c gates are used to perform finite field arithmetic in $GF(2^q)$, check arithmetic can be performed mod2. Although this alone may not be sufficient to reduce ϵ_c to the desired value, it can be used in conjunction with repetition. Theoretically, coded computation could even be applied recursively to boost the reliability of each application of Φ.

Errors in Transcoding Steps

During each transcoding step, multiple transcoding operations will take place in parallel. As with parallel applications of Φ, discussed above, there will be an error rate, ϵ_t, associated with each of these transcoding operations. When a particular transcoding operation produces an incorrect output, this is called a **transcoding error**. If each transcoding operation uses G_t total gates, than $G_t p_f$ provides a bound on ϵ_t. As G_t increases, however, this fault rate becomes prohibitively high. Furthermore, G_t will typically be much larger than G_c. Once again von Neumann style gate repetition provides a general way to reduce ϵ_t as needed, regardless of the transcoding algorithm being employed. Specifically, ϵ_t can be made polynomially small in n using $O(\log n)$ overhead.

Also, as in applications of Φ, algorithm-based fault-tolerance may also be useful. Since the decoding of linear error-correcting codes typically involves a significant amount of arithmetic, modular arithmetic may once again provide a way of employing intermediate check points. Also, when a codeword is decoded it is typically easy to check that the answer is correct via a less expensive encoding operation. This once again suggests that algorithm-based fault-tolerance may be used in conjunction with gate repetition in order to reduce ϵ_t. Finally, certain codes may allow for encoding and decoding circuitry in which each gate failure only corrupts a limited number of output bits. This idea is examined in [90] with the goal of producing highly fault-tolerant memories. As noted in Section 9.8.4, iterative decoding of low-density parity check codes also has this property.

A Reliable Step of Coded Computation

Each step of coded computation is comprised of a computation step, followed by a transcoding step. The computation step consists of n applications of a function Φ, each of which produces an incorrect output with probability ϵ_c. As noted at the end of Section 9.4.1, it suffices to consider the case when $\Phi(r, s) = rs$, since more complex Φ can be decomposed into a series of multiplications. and additions. After a computation step, the transcoding step consists of $n_t = O(n)$ transcoding operations each of which fails with probability ϵ_t. As discussed above, both ϵ_c and ϵ_t can be made polynomially small in n using gate repetition in which $r = O(\log(n))$. Now suppose the coded computation is designed so that at least e_1 computation errors or e_2 transcoding errors must occur for a particular step of coded computation to fail. As long $n\epsilon_c < e_1$ and $n_t\epsilon_t < e_2$, the probability that each step of coded computation fails will decrease exponentially in n.

In the analysis that follows, we consider several approaches for reliably implementing each step of coded computation using different codes and transcoding methods. The reliability of a coded computation typically increases with its width. As described in Section 9.5, for example, the fault-tolerance of each transcoding step relies on multiple codewords being transcoded in parallel. As

noted in Section 9.5.3, however, this does not necessarily require that each computation step be wide enough so to accommodate multiple large codewords. An alternative approach is to consider either a parallel, or pipelined architecture in which multiple transcoding operations, for multiple different computation steps are carried out in parallel.

9.8.3 Coded Computation Using 2D Codes

This section bounds the overhead required to make a regular computing network, $\mathcal{C}$, fault-tolerant via coded computation when C and C^* are both n-by-n two-dimensional codes. In this case each computation step consists of n^2 applications of Φ in parallel, and each transcoding step consists of n row transcoding operations followed by n column transcoding operations (see Section 9.5.1). For simplicity, we assume that the row and column codes of C are both the same $[n, k, d]$ linear code, and that the row and column codes of C^* are both the same $[n, k^*, d^*]$ linear code. Here $k^* > k$ and $d^* < d$.

As discussed in Section 9.8.2, we can bound the probability that a given step of coded computation fails by providing suitable definitions of "highly correctable" and "sustainable" codewords.

Definition 9.8.1 *When C and C^* are 2D codes, an encoded vector in C^* is called* **highly correctable** *if at most $d^*/3$ rows contain more than $d^*/2$ errors each. An encoded vector in C is called* **sustainable** *if no row contains more than $d^*/6$ errors. If r-repeated codewords are used, and each bit is repeated r-times, the term "error" refers to at least αr of the r copies of a bit being in error (see Section 9.8.1).*

Now consider a computation step in which Φ is applied component-wise to two sustainable inputs. If no computation errors occur, the encoded output will contain at most $2d^*/6 = d^*/3$ errors per row. Thus, in order for a computation step to fail to produce a highly correctable output from two sustainable inputs, more than $d^*/2 - d^*/3 = d^*/6$ computation errors must occur in more than $d^*/3$ rows. Now consider a transcoding step in which a highly correctable input is transcoded row-wise then column-wise, as described in Section 9.5.1. Since at most $d^*/3$ rows represent incorrect codewords the transcoding step can fail to produce a sustainable output only if more than $d^*/2 - d^*/3 = d^*/6$ row transcoding operations produce errors, or more than $d^*/6$ column transcoding operations produce errors.

To bound the probability that a step of coded computation fails, let $B(N, S, p)$ denote the probability that a binomial random variable with parameters N and p takes value greater than S. $B(N, S, p) = \sum_{i>S}^{N} \binom{N}{i} p^i (1-p)^{N-i}$, which grows exponentially small in $\delta^2 Np$ when $S \leq (1-\delta)Np$ for any fixed constant δ (see the Chernov bound in Section 5.2.1 where $E[S] = Np$). Also, recall from Section 9.8.2 that we use ϵ_c to denote the probability that a particular application of Φ results in an error, and ϵ_t to denote the probability that a particular transcoding operation results in an error (when a transcoding error occurs, all positions in the output vector may be incorrect). The probability that a coded computation fails is bounded by the following Lemma.

Lemma 9.8.1 *Consider a T-step coded computation in which C and C^* are 2D linear codes whose row (and column) codes are have parameters $[n, k, d]$ and $[n, k^*, d^*]$, respectively. Also, let ϵ_c and ϵ_t*

denote the probabilities of a computation error and a transcoding error, respectively. Then the probability of a computation step failing is at most $B(n^2, d^{*2}/18, \epsilon_c)$ and the probability of a transcoding step failing is at most $2B(n, d^*/6, \epsilon_t)$. Finally, the probability, P_f, that some step of a T-step coded computation fails is bounded as follows:

$$P_f < T(B(n^2, d^{*2}/18, \epsilon_c) + 2B(n, d^*/6, \epsilon_t))$$

Proof During each computation step Φ is applied n^2 times and each application has probability ϵ_c of producing an error. As discussed above, a computation step with two sustainable inputs can tolerate at least $(d^*/3) \cdot (d^*/6) = d^{*2}/18$ computation errors. Since these errors occur independently, the probability of a computation step failing is at most $B(n^2, d^{*2}/18, \epsilon_c)$.

Similarly, during each transcoding step n row and n column transcoding operations take place. Each operation results in an error with probability ϵ_t. For the transcoding step to fail when given a highly correctable input more than $d^*/6$ row transcoding errors, or $d^*/6$ column transcoding errors must occur. Since transcoding errors occur independently, the probability of a transcoding step failing at less than $2B(n, d^*/6, \epsilon_t)$.

Finally, as explained in Section 9.8.2, a T-step coded computation succeeds if none of the T computation and T transcoding steps fail. Thus $P_f \leq T(B(n^2, d^{*2}/18, \epsilon_c) + 2B(n, d^*/6, \epsilon_t))$ ■

Regardless of the code being used, ϵ_c and ϵ_t should be chosen so that $\epsilon_c < d^{*2}/18n^2$ and $\epsilon_t < d^*/6n$ in order to successfully bound P_f. For example, suppose computation errors occur with probability $\epsilon_c \leq 10^{-7}$ and transcoding errors (which are significantly more likely, due to the added circuity involved) occur with probability $\epsilon_t \leq 10^{-3}$. Here reliable coded computation can be implemented using a family of codes for which d^* is large compared to $10^{-3} \cdot 6n$. Reed-Solomon codes can easily meet this criteria. For instance, a Reed-Solomon code for which $k = 32$ and $n = 100$ yields $d^* = 38$ if Φ has degree 2 (see Section 9.7.1). In this case, the above lemma reveals that each step of coded computation fails with probability at most $P_f \leq B(n^2, d^{*2}/18, \epsilon_c) + 2B(n, d^*/6, \epsilon_t) = B(10^4, 80, 10^{-7}) + 2B(10^2, 6.2, 10^{-3}) < 1.5 \cdot 10^{-11}$. As n increases, the bound on P_f decreases exponentially.

2D Reed-Solomon Codes

Now consider the use of n-by-n 2D Reed-Solomon (RS) codes as C and C^*, the case originally analyzed by Spielman in [80]. As explained in Section 9.7.1, a one dimensional $[n, k, d]$ RS code requires that $n = k + d - 1$ and that the code have alphabet size at least n. Furthermore, $d^* = n - 2(k-1) = d - k + 1$ when Φ has degree 2 (see Section 9.7.1). Also, one dimensional RS codes can be transcoded via decoding, then re-encoding, using $O(n \log^2 n)$ arithmetic operations [91, 92] (also in [80] Spielman proposes a randomized transcoding algorithm using the result of [93]).

As 1D RS codes grow in length, each symbol of each codeword requires $O(\log n)$ bits, and $d^* = O(d) = O(n)$. Thus $B(n^2, d^{*2}/18, \epsilon_c)$, above, is exponentially small in n when $\epsilon_c < d^{*2}/(18n)$, and $B(n, d^*/6, \epsilon_t)$ is exponentially small in n when $\epsilon_t < d^*/(6n)$. In both cases this requires ϵ_c and ϵ_t be held below some constant. To do this r-repeated RS codewords can be used as described in the previous section. This requires $r = O(\log n)$. As such, when 2D RS codes are used, a single computation step is made exponentially reliable in n using at most $O(\log^4 n)$ overhead.

Finally, an additional factor of $O(\log(n))$ overhead may be required during the subsequent data-movement operation in order to realize an arbitrary permutation over the information symbols. Since $n = O(k)$, this implies that making each step of a regular computing network with k^2 operations per step (and potentially arbitrary data movement between steps) exponentially reliable in k using RS codes requires at most $O(k \log^5 k)$ gates.

To review, there are several sources for this logarithmic overhead. First, the alphabet size of RS codes must grow with their length. Second, r-repeated codewords must be employed in order to hold ϵ_c and ϵ_t constant (here the need to hold ϵ_t constant, not ϵ_c, represents the real bottleneck). Third, arbitrary data-movement operations may be required, and fourth, there is a costly transcoding operation after each step. This transcoding, which takes place after each step of coded computation, requires encoding and decoding of $2n$ 1D RS codewords (n rows and n columns). This represents the single largest source of overhead. In practice, it might make sense to consider using RS codes for which n, and hence d and d^*, are $O(k \log k)$. This would allow for $O(\log(k))$ applications of Φ before a transcoding step is required. Alternatively, it may be possible to structure a computation so as to reduce the total number of transcoding steps. For example, if Φ is applied to two codewords, the result can be copied c times, each of these c copies permuted by different permutation Π_{C^*} (as defined in Section 9.3.3), and the c differently permuted copies summed. This summation requires only a single transcoding operation, even though it is considerably more powerful than a single application of Φ (the summation operation can also be combined with "masking", as described in Section 9.6.2).

9.8.4 Coded Computation Using 1D Codes

This section bounds the overhead required to make a regular computing network, $\mathcal{C}$, fault-tolerant via a coded computation that employs checksum-based transcoding, as described in Section 9.5.2. This approach to coded computation no longer requires that C and C^* be 2D codes. Instead, each step of coded computation is performed on m separate codewords in some linear code, C. As before we use $[n, k, d]$ and $[n, k^*, d^*]$ to denote the parameters of C and C^* respectively. Once again $k^* > k$ and $d^* < d$. As explained in Section 9.5.2 m codewords from C^* can be transcoded in a fault-tolerant manner if they are first encoded in a 2D code. To do this the m codewords in C^* are treated as m rows in a 2D code, then a second linear code, C_2, is used to encode their n columns. The result is a 2D codeword in $C^* \times C_2$. Here C_2 can be an arbitrary systematic linear $[n_2, m, d_2]$ code. In the newly generated 2D codeword, we refer to the m original codewords as "information rows" and the $n_2 - m$ newly generated rows as "check rows".

As in Section 9.8.3 we can bound the probability that a given step of coded computation fails by providing suitable definitions of "highly correctable" and "sustainable" codewords.

Definition 9.8.2 *Consider m separately encoded rows in an m-by-n array. When the m rows represent codewords in C^* they are collectively called* **highly correctable** *if no more than $d^*/3$ of their n columns contain any errors (i.e. at least $n - d^*/3$ columns are error free). When the m rows represent codewords in C, they are collectively called* **sustainable** *if at most $d^*/8$ of their columns contain errors. If r-repeated codewords are used, and each bit is repeated r-times, the term "error" refers to at least αr of the r copies of a bit being in error (see Section 9.8.1).*

During a computation step, Φ is applied component-wise to two sets of sustainable codewords. If no computation errors occur, at most $d^*/4$ of the encoded output's columns will contain errors. For the computation step to fail to produce a highly correctable output, more than $d^*/3 - d^*/4 = d^*/12$ computation errors must occur. Since these errors occur independently, the probability that a computation step fails is at most $B(nm, d^*/12, \epsilon_c)$.

Now consider a transcoding step that receives a highly correctable array of m codewords in C^*. Initially the n columns are encoded in C_2. Since at most $d^*/3$ of the columns contain errors, more than $d^*/6$ encoding errors must occur for any row to contain more than $d^*/2$ errors. Next the n_2 rows are transcoded. If no more than $d_2/2$ transcoding errors occur, each column will represent the correct codeword in C_2. Finally the n_2 columns are decoded. If no more than $d^*/8$ decoding errors occur, the resulting array of m codewords in C will be sustainable. Thus a transcoding step will produce a sustainable output from a highly correctable input as long as fewer than $d^*/6$ encoding errors, $d_2/2$ transcoding errors, and $d^*/8$ decoding errors occur.

Since C_2 can be chosen to be an arbitrary linear code, it is reasonable to assume that the row transcoding operations are the most likely to result in errors. Thus we could use ϵ_t as a bound on the probability of all three error types. Alternatively, we can provide a more fine-grained analysis and use ϵ_{t2} to denote the probability that a particular column experiences either an encoding or a decoding error. This yields the following Lemma.

Lemma 9.8.2 *Consider a T-step coded computation that employs checksum-based transcoding, where C and C^* are linear codes with parameters $[n, k, d]$ and $[n, k^*, d^*]$, respectively. Also, C_2 is the linear $[n_2, m, d_2]$ column code used during transcoding. Let ϵ_c and ϵ_t denote the probabilities of a computation error and a transcoding error, respectively. Also, let ϵ_{t2} denote the probability that a particular column's encoding or decoding operation results in an error during transcoding. Then the probability of a computation step failing is at most $B(nm, d^*/12, \epsilon_c)$ and the probability of a transcoding step failing is at most $B(n, d^*/6, \epsilon_{t2}) + B(n_2, d_2/2, \epsilon_t)$. Finally, the probability, P_f, that some step of a T-step coded computation fails is bounded as follows:*

$$P_f \leq T(B(nm, d^*/12, \epsilon_c) + B(n, d^*/8, \epsilon_{t2}) + B(n_2, d_2/2, \epsilon_t))$$

Proof As explained above, a computation step can only fail if more than $d^*/12$ computation errors occur. Thus the probability that a computation step fails is at most $B(nm, d^*/12, \epsilon_c)$. Also, a transcoding step can fail only if more than $d^*/6$ encoding errors, $d_2/2$ transcoding errors, or $d^*/8$ decoding errors occur. This implies that more than $d^*/8$ columns must experience either an encoding or a decoding error, thus the probability that a transcoding step fails is at most $B(n, d^*/8, \epsilon_{t2}) + B(n_2, d_2/2, \epsilon_t)$. Since a T-step coded computation succeeds if none of the T computation and T transcoding steps fail, $P_f \leq T(B(n^2, d^{*2}/18, \epsilon_c) + 2B(n, d^*/6, \epsilon_t))$ ■

In order to obtain a bound on P_f, $\epsilon_c < d^*/12nm$, $\epsilon_t < d_2/2n_2$ and $\epsilon_{t2} < d^*/6n$. We note that since C_2 can be an arbitrary linear code, d_2 can be chosen such that $d_2 = O(n_2) = O(m)$ and such that $d_2 > d^*$. As such, the requirement on ϵ_{t2} is more stringent. Fortunately the overhead associated with keeping ϵ_{t2} small can reduced when C_2 is chosen appropriately. For example, C_2 can be a code that allows for linear-time encoding and decoding [94]. C_2 can also be a low-density parity check code that allows for linear-time encoding and efficient iterative decoding based on

sparse matrices [95]. One advantage of iterative decoding is that it is inherently fault-tolerant against the failure of gates (or in our case clusters of gates). In iterative decoding the number of errors in a codeword is repeatedly reduced during successive stages of the decoding algorithm. If errors are introduced during a particular stage, they can be corrected by subsequent stages.

1D Reed-Solomon Codes

Now consider the use of 1D RS Codes as C and C^*. As explained in Section 9.7.1, a one dimensional $[n, k, d]$ RS code requires that $n = k + d - 1$ and that the code have alphabet size at least n. Also, $d^* = n - 2(k-1) = d - k + 1$ when Φ has degree 2 (see Section 9.7.1). As noted in the previous section, these codes can be transcoded via decoding, then re-encoding, using $O(n \log^2 n)$ arithmetic operations [91, 92].

As in the previous section, we first observe that as 1D RS codes grow in length, each symbol of each codeword requires $O(\log n)$ bits, and $d^* = O(d) = O(n)$. $B(nm, d^*/12, \epsilon_c)$ is exponentially small in n when $\epsilon_c < d^*/12nm$. Similarly $2B(n, d^*/6, \epsilon_{t2})$ is exponentially small in n when $\epsilon_{t2} < d^*/6n$. In both cases, assuming m is polynomial in n, this can be accomplished by using r-repeated RS codewords when $r = O(\log n)$. As explained at the beginning of this section, this value of r is sufficient to ensure that all clusters fail with probability $O(p_f/n^c)$, for any constant c. Similarly, we can ensure that $B(n_2, d_2/2, \epsilon_t)$ is exponentially small in n if we choose C_2 such that $n_2 = O(m) = O(n)$ and $d_2 = O(n)$. As such, we have shown that each computation step can once again be made exponentially reliable in n using $O(\log^4 n)$ overhead, with an additional factor of $O(\log n)$ potentially required to permute data arbitrarily.

As noted above, when checksum-based transcoding is employed there is considerable flexibility when choosing the column code, C_2. For example, C_2 can be chosen such that $d_2 = O(n_2)$, and $n_2 = O(m)$. Notice, however, that it is not necessarily optimal to set $m = O(n)$. For example, in [80], Spielman proposes a transcoding algorithm for RS codes using the result of [93]. The algorithm is relatively efficient (it introduces polylogarithmic overhead), but corrects only $O(\sqrt{n})$ errors. As such, each step of the resulting coded computation is not exponentially reliable in $d^* = O(n)$, but exponentially reliable in $\sqrt{d^*} = O(\sqrt{n})$. If this algorithm is used in the context of checksum-based transcoding, it is sensible to set $m = O(\sqrt{n})$, as opposed to $O(n)$. This reduces the overhead associated with transcoding, but maintains the exponential reliability in $\sqrt{d^*}$. In contrast, if $m = O(n)$, each column decoding operation is exponentially reliably in $m = O(n)$, but the other portions of the coded computation are not any more reliable.

1D Reed-Muller Codes

As explained in Section 9.7.2, Reed-Muller (RM) codes can be used in place of RS codes for C and C^*. The key difference is that RM codes use only a binary alphabet, and provide smaller values for d and d^*. This allows for much simpler computation and transcoding steps, but worse fault-tolerance. If $n = O(k \log k)$, d and more importantly d^*, are not $O(n)$.

When RM codes are used, $d = 2^{r-m}$ and $d^* = 2^{r-2m}$, where r and m are parameters of the particular RM code being used for C. As explained in Section 9.7.2, r and m determine the length of codewords and bound the number of information symbols in C. Specifically, $n = 2^r$ and

$k \leq \sum_{i=0}^{m} \binom{r}{i}$. To ensure that k can be at least as large as some required threshold, k_t, we can set $r \geq \log_2 2k_t$ and $m = \log_2 2k_t/2$. Since $n = 2^r$, we incur at most polylogarithmic overhead as long as $r = \log_2 2k_t + c \log_2 \log_2 2k_t$ for a given constant c. In this case $n = 2k_t \log_2^c 2k_t$ and k can be at least k_t. Unfortunately this gives $d^* = 2^{r-2m} = 2^{c \log_2 \log_2 2k_t} = \log_2^c 2k_t$. Since $d^* = O(n/k)$, this provides no better fault-tolerance than repetition. Thus even though coded computation can be performed using RM codes, it is unclear if the resulting construction can provide better fault-tolerance than repetition-based fault-tolerance. Specifically, choosing m and r such that k is at least as large as some desired threshold, k_t, and n is no larger than $2k_t \log_2^c 2k_t$, appears to imply that $d^* = O(n/k)$. This highlights a very important point. In order for coded computation to provide an exponentially high level of reliability, C and C^* must be chosen such that $d^* = O(n^{1/c})$, for some constant c.

9.8.5 Summary of Results

This chapter offers a framework for performing reliable computation using error-correcting codes. Our framework extends the approach Spielman described in [80] in several significant ways. First, we describe how a range of linear codes can be used, instead of relying purely on 2D Reed-Solomon codes over $GF(2^q)$. Second, we offer multiple approaches for transcoding from C^* to C after each computation step. Third, we explore a range of strategies for permuting data. As we explain, codewords can be permuted before, after or during transcoding, and the permutations one applies need not be limited to those that correspond to data-movement on a hypercube. Finally, we explicitly quantify the big-O overhead associated with performing coded computations using both Reed-Solomon and Reed-Muller codes.

As described in this section, coded computation introduces overhead by 1) encoding data through the addition of check symbols and performing computation operations on those check symbols, 2) performing transcoding operations on encoded outputs, 3) performing data-movement operations, and 4) potentially employing repetition-based fault-tolerance to reduce the error rates associated with computation and transcoding operations. In the case of 1) Reed-Solomon codes introduce $O(\log n)$ overhead, since each check symbol requires at least $\log n$ bits, but it is possible to reduce this overhead by using a code with a smaller alphabet. As an extreme example we consider Reed-Muller codes, which use a binary alphabet and thus introduce only $O(1)$ overhead. Unfortunately, Reed-Muller codes have poor error-correction capabilities. In the future, it may be promising to consider codes that require, say, $O(\log \log n)$ bits per check symbol.

In the case of 2) the overhead associated with transcoding both Reed-Solomon and Reed-Muller codes is determined by the efficiency of their decoding algorithms (in both cases, encoding is significantly simpler than decoding). Decoding these codes involves $O(n \log^2 n)$ and $O(n \log n)$ arithmetic operations, respectively. Additionally, the number of errors that can be tolerated during transcoding determines the reliability of each step of coded computation. We have introduced checksum-based transcoding as a means of employing a second, more efficient error-correcting code in order to boost the reliability, and reduce the overhead, associated with each transcoding step.

In the case of 3) both Reed-Solomon and Reed-Muller codes can directly implement data-movement on a hypercube. As such, performing arbitrary permutations on encoded data introduces

at most $O(\log n)$ overhead. Furthermore, a number of algorithm-specific data-movement optimizations appear feasible in practice. Finally, in order to reduce the $O(\log n)$ overhead associated with 4), it may be possible to apply coded computation recursively. Instead of using r-repeated codewords, this would involve using codewords in which subsets of symbols are themselves encoded in a smaller error-correcting code.

To summarize, Section 9.8.3 demonstrates that each step of a regular computing network with k^2 operations per step (and potentially arbitrary data movement between steps) can be made exponentially reliable in k using 2D Reed-Solomon codes and $O(k \log^5 k)$ gates. In contrast, as explained at the end of Section 9.8.1, repetition codes (i.e. modular redundancy) allow each step of the same computation network to be polynomially reliable in k using $O(k \log k)$ gates. As explained at the end of Section 9.8.4, 1D Reed-Muller codes also allow each step of computation to be polynomially reliable, but they require greater overhead. Even in the absence of data-movement, the overhead associated with adding check symbols, and transcoding codewords, is $O(\log^2 k)$.

To more directly compare 2D Reed-Solomon versus repetition-based fault-tolerance, we can consider a T-step regular computing network with k^2 operations per step. This network has $|C| = Tk^2$ gates in total. If T is exponential in k, then making each step of computation exponentially reliable in k allows the entire computation to be polynomially reliable in k. As stated above, this level of reliability requires a factor of $O(\log^5 k)$ overhead. In contrast, achieving the same level of reliability via repetition-based fault-tolerance requires a factor of $O(\log |C|) = O(\log T + \log k^2) = O(k)$ overhead (see discussion at the end of Section 9.8.1). Thus for a sufficiently long computation, coded computation using Reed-Solomon codes is superior to repetition-based fault-tolerance.

Chapter 10

Exploring the Power of Coded Computation

Building on the framework established in Chapter 9, this chapter offers some examples of the promise and challenges of coded computation. Section 10.1 provides a lower bound that offers important insight into the overhead required to perform a single step of coded computation. Section 10.2 highlights the fact that even though it remains a challenge to encode specific functions efficiently (i.e. with constant factor overhead), most arbitrary boolean functions, which already require an exponential number of gates, are amenable to a simple and efficient encoding scheme. Finally, Section 10.3 provides a concrete example of how parallel prefix computations can be implemented as coded computations.

10.1 Lower Bounds

Recall from Section 9.3.1 that a single step of coded computation is expressed as

$$E^*(\mathbf{z}) = \Phi^{(n)}(E(\mathbf{x}), E(\mathbf{y}), E(\mathbf{w})), \tag{10.1}$$

where Φ is chosen such that $\mathbf{z}_t = \kappa^{(\mathbf{k})}(\mathbf{x}, \mathbf{y}, \mathbf{w})$, and E and E^* are the encoding functions associated with two different error-correcting codes, C and C^*. As defined in Section 9.2.1, $\kappa^{(\mathbf{k})}$ denotes the component-wise of application of some function, κ, to k sets of inputs, $(x_1, y_1, w_1) \ldots (x_k, y_k, w_k)$. Similarly, $\Phi^{(n)}$ denotes the component-wise application of Φ.

As explained in Section 9.3.2, after $\Phi^{(n)}$ is applied a transcoding step is required to project $E^*(\mathbf{z})$ to $E(\mathbf{z})$. This allows $E(\mathbf{z})$ to be supplied as an input to future computation steps. The major overhead associated with coded computation appears to be this need to perform a transcoding operation after each computation step. The process of performing multiple steps of coded computation would be dramatically simplified if no transcoding operation were required and C and C^* were the same code. For example, suppose κ denotes XOR (i.e addition mod 2). In this case, choosing any linear code over $GF(2)$ as C allows us to compute $E(\mathbf{z})$ by computing $E(\mathbf{x}) + E(\mathbf{y}) = E(\mathbf{x} + \mathbf{y}) = E(\text{XOR}^{(\mathbf{k})}(\mathbf{x}, \mathbf{y}))$. Thus Φ is simply addition, and C^* is C.

Since XOR alone does not form a complete basis, it is of great interest whether a similarly simple choice of C and Φ exists for $\kappa = \text{AND}$ or $\kappa = \text{OR}$. Some early work in coding theory, however, pointed

out an immediate difficulty [96, 76, 77].

Theorem 10.1.1 ([77]) *Let $\kappa(x, y)$ be an* AND*-type function, meaning κ either computes* AND*, or computes a function that differs from* AND *in that some of the inputs or outputs are negated. Let C be a binary error-correcting code with encoding function $E : \{0,1\}^k \mapsto \{0,1\}^n$ and minimum distance d. If a function Φ exists such that $\Phi^{(n)}(E(\mathbf{x}), E(\mathbf{y})) = E(\kappa^{(\mathbf{k})}(\mathbf{x}, \mathbf{y}))$, then*

$$n \geq kd$$

Thus C is no more efficient than simply repeating each symbol d times (in which case $\Phi = \kappa$).

This lower bound demonstrates that you can't do better than a repetition code when Φ is applied component-wise with no transcoding. It does not, however, take into account the possibility of applying Φ to more than one symbol per codeword (also it fails to account for non binary codes). The next theorem considers the case which the code alphabet is non-binary and each output symbol can be a function of up to c components of each codeword. The following lower bound, first presented in [97], accounts for both possibilities.

Theorem 10.1.2 *Let $\kappa(x, y)$ be an* AND*-type function, meaning κ either computes* AND*, or computes a function that differs from* AND *in that some of the inputs or output are negated. Let C be an error-correcting code over the alphabet G with encoding function $E : G^k \mapsto G^n$ and minimum distance d. Let $F : G^{2n} \mapsto G^n$ be a function such that $F(E(\mathbf{x}), E(\mathbf{y})) = E(\kappa^{(\mathbf{k})}(\mathbf{x}, \mathbf{y}))$, If each output of F is a function of at most c symbols of $E(\boldsymbol{x})$ and $E(\boldsymbol{y})$, then*

$$n \geq kd/(c \log_2 |G|)$$

Proof Without loss of generality, assume $\kappa(x, y) = \text{AND}(x, y)$. Let $F(\boldsymbol{x}, \boldsymbol{y})_i$ denote the i^{th} component of $F(\boldsymbol{x}, \boldsymbol{y})$. By assumption $F(\boldsymbol{x}, \boldsymbol{y})_i$ depends on at most c components of $\boldsymbol{x}$ and of $\boldsymbol{y}$. Let $S(i)$ denote these components of $\boldsymbol{x}$.

Let $\mathbf{1}_r$ be a k-tuple in which all components are 0 except for the r^{th} component, which has value 1. Also, let $\mathbf{0}$ be the k-tuple in which all components are 0. By definition of F, $F(E(\boldsymbol{x}), E(\mathbf{1}_r)) = E(\mathbf{1}_r)$ or $E(\mathbf{0})$ depending on whether $x_r = 1$ or $x_r = 0$.

Let $\zeta_i^r(E(\boldsymbol{x})) = F(E(\boldsymbol{x}), E(\mathbf{1}_r))_i$ denote the ith component of $F(E(\boldsymbol{x}), E(\mathbf{1}_r))$. $\zeta_i^r(E(\boldsymbol{x}))$ depends on $|S(i)|$ components of $\boldsymbol{x}$. By assumption, $|S(i)| \leq c$.

Because the code has minimum distance d, there are at least d positions at which the codewords $E(\mathbf{1}_{\boldsymbol{r}})$ and $E(\mathbf{0})$ differ. Let $I(\mathbf{0}, r)$ denote these positions. Observe that for each r, we can select i in $I(\mathbf{0}, r)$ and compute $\zeta_i^r(E(\boldsymbol{x}))$. Since $\zeta_i^r(E(\boldsymbol{x})) = E_i(\mathbf{0})$ if and only if $x_r = 0$, knowing $\zeta_i^r(E(\boldsymbol{x}))$ reveals the value of x_r.

Let $E^{S(i)}(\boldsymbol{x})$ denote the components of $E(\boldsymbol{x})$ in positions $S(i)$, $|S(i)| \leq c$. It follows that $E^{S(i)}(\boldsymbol{x}) \in G^c$.

Let R_i denote the values of r such that $\boldsymbol{i} \in I(\mathbf{0}, r)$. Knowing $E^{S(i)}(\boldsymbol{x})$ reveals x_r for any $r \in R_i$. Since $E^{S(i)}(\boldsymbol{x})$ takes at most $|G|^c$ possible values and each variable x_r takes two values, $2^{R_i} \leq |G|^c$ or $|R_i| \leq c \log |G|$.

Pairs (r, i) satisfying $r \in R_i$ if and only if $i \in I(\mathbf{0}, r)$ are called **linked pairs**. The total number of linked pairs, Q, can be counted two ways:

$$Q = \sum_{r=1}^{k} |I(\mathbf{0}, r)| = \sum_{i=1}^{n} |R_i|$$

Since we know $|I(\mathbf{0}, r)| \geq d$, and $|R_i| \leq c \log_2 |G|$, we have $kd \leq cn \log_2 |G|$, the desired bound. ■

The argument of the above proof applies to any AND-like function such as NAND, OR, or NOR and any other Boolean functions of two or more variables that is "partially sensitive", meaning the function can be made either dependent on, or independent of, one of its inputs depending on the values for the remaining variables. More precisely, we say that a binary function of m inputs is **partially sensitive** if there exists some $a_2, \ldots, a_m$ such that $\kappa(0, a_2, \ldots, a_m) = \kappa(1, a_2, \ldots, a_m)$, and some $b_2, \ldots, b_m$ such that $\kappa(0, b_2, \ldots, b_m) \neq \kappa(1, b_2, \ldots, b_m)$. The proof of the above theorem can be applied to any such function. This allows us to apply the lower bound to the case where κ is a function of more than two variables. It also allows us to consider the case where each κ_i in the component-wise application of κ is not the same (but each is partially sensitive).

Implications

Theorem 10.1.2 implies that the only way for C to be asymptotically "good", meaning both n and d are proportional to k, is if the fanout of inputs to outputs is also proportional to k. This immediately implies that a single step of coded computation cannot be constant depth when a good code is being used. This is why the component-wise application of $\Phi^{(n)}$, which is constant depth, must be followed by a transcoding operation, which is not constant depth. This also motivates the use of two codes, C and C', instead of just one.

Given that each step of fault-tolerant coded computation cannot be constant depth, it would be quite interesting to know whether a single step of coded computation can use only $O(n)$ gates. Although no such construction is known, Theorem 10.1.2 is not strong enough to preclude its existence. For example, as demonstrated in [94], asymptotically good error-correcting codes exist that can be both encoded and decoded using $O(n)$ gates. If such a code is chosen as C, then a single step of coded computation, as defined for the purpose of Theorem 10.1.2, can be carried out by first decoding both $E(\mathbf{x})$ and $E(\mathbf{y})$, then computing $\kappa^{(\mathbf{k})}(\mathbf{x}, \mathbf{y})$ directly, and finally reencoding the result. Though this construction would only require $O(n)$ gates, it offers no real fault-tolerance. Specifically, any error that occurs when computing $\kappa^{(\mathbf{k})}(\mathbf{x}, \mathbf{y})$ will not be correctable.

To address this shortcoming, it would be necessary to explicitly add some sort of fault-tolerance condition to the definition of coded computation used in Theorem 10.1.2. For example, in order to help avoid a bottleneck in which a single fault can corrupt an entire computation, one can add the requirement that m disjoint paths exist between any set of m inputs and m outputs. An acyclic directed graph with n inputs and n outputs that fulfills this requirement is known as a "super concentrator" and in fact $O(n)$ size super concentrators exist [98]. It is noteworthy that the linear time encodable/decodable codes in [94] also utilize a construction based on linear-sized super concentrators. It is unclear whether a similar construction is applicable to coded-computation.

10.2 Efficiently Encoding Most-Boolean Function

As explained in Section 9.1.1, von Neumann's repetition-based approach to fault-tolerance relies on replicating each gate in a circuit, C, r times then using r constant-sized majority gates to suppress errors with high probability. The resulting circuit, C', is reliable in the sense that the error rate of each output is at most a constant multiple of the failure rate of individual gates. In [15] Pippenger notes that while this construction requires each gate in an arbitrary circuit be repeated $r = O(\log |C|)$ times (and hence $|C'| = O(|C| \log |C|)$) *most* boolean functions can be computed reliably using only constant factor redundancy (meaning $|C'| = O(|C|)$). In this section we note that Pippenger's observation and his accompanying construction naturally extend to coded computation.

Consider a random boolean function $f(x_1, \ldots, x_m)$, meaning $f : \{0,1\}^m \mapsto \{0,1\}$ is randomly drawn from the set of all possible 2^{2^m} m-input boolean functions. When realized by a circuit, C, all but an exponentially small fraction of such functions require $|C| = \Theta(2^m/m)$ gates to compute [81]. Now suppose we wish to reliably compute f using a circuit, C', constructed from noisy gates that fail independently at random with probability p_f. Pippenger showed that all f for which $|C| = \Theta(2^m/m)$ can still be computed reliably by C' (in the sense described above) using only $|C'| = \Theta(2^m/m)$ gates. To obtain this result Pippenger efficiently modified Lupanov's original $\Theta(2^m/m)$ construction using gate repetition. The key insight used to avoid the $O(\log |C|)$ factor overhead is that not all gates need be repeated the same number of times. As Pippenger explains, most gates in Lupanov's $\Theta(2^m/m)$-sized circuit have a very low probability of influencing the circuits output when they fail, and thus most gates need only be repeated $O(1)$ times (see Section 10.2.1 below).

In order to extend Pippenger's result to coded computation, suppose that instead of evaluating a single random boolean function, $z = f(x_1, \ldots, x_m)$, we wish to evaluate k such functions, $z_1 = f_1(x_1, \ldots, x_m), \ldots, z_k = f_k(x_1, \ldots, x_m)$. In other words, we wish to compute a random function of m inputs and k outputs. With high probability, the k outputs will not be able to share a substantial amount of common logic, and hence most such functions require a circuit, $C^{(k)}$, of size $|C^{(k)}| = \Theta(k2^m/m)$. Such a circuit can be realized via k copies of Lupanov's construction and can be made fault tolerant via k copies of Pippenger's construction.

Now suppose that in addition to computing $z_1, \ldots, z_k$, we wish to compute $n - k$ check bits using the encoding function, $E(z_1, \ldots, z_k)$, of a length n systematic error-correcting code. Each checkbit of $E(z_1, \ldots, z_k)$, denoted $z_{k+1} \ldots z_n$, is itself a function of the original m inputs, and hence the entire codeword can be computed by a circuit, $C^{(n)}$, of size $|C^{(n)}| = \Theta(n2^m/m)$. Furthermore, Pippenger's construction allows the entire codeword to be computed reliably with a noisy circuit, $C'^{(n)}$, of size $|C'^{(n)}| = \Theta(n2^m/m)$. Here each output of $|C'^{(n)}|$ fails with probability $O(p_f)$, and hence if a sufficiently good constant rate code is used the probability that the codeword gets corrupted falls exponentially with n. Since $|C'^{(n)}| = O(|C'^{(k)}|)$, the fault-tolerant computation has been implemented using constant factor overhead.

In the above construction, it is crucial that the m inputs supplied to the computation are reliable. Hence if multiple such computations were to be carried out in series (i.e. $z_1, \ldots, z_k$ are supplied as inputs to a second fault-tolerant computation), reliable gates must be used to decode

$E(z_1, \ldots, z_k)$. Even so, the number of reliable gates required ($O(k)$ if the codes in [94] are used) is a tiny fraction of the total number of gates being used.

A key characteristic of the above construction is that the check bits being computed, $z_{k+1} \ldots z_n$, belong to the same class of functions (namely, randomly selected boolean functions) as the information symbols, $z_1, \ldots, z_k$. This ensures that the total overhead devoted to computing those check bits is proportional to the number gates in $C^{(k)}$. Furthermore the Pippenger construction ensures that $|C'^{(n)}| = |C^{(n)}|$. In order for this general approach to be practical, other classes of functions that meet these criteria must be identified.

Classes of functions that can compute both checkbits and information symbols seem relatively easy to come by. Since each checkbit can be linear sums of the information symbols (meaning $E(z_1, \ldots, z_k)$ is a codeword in a linear error-correcting code), any class of functions closed under addition would suffice. The class of randomly chosen polynomials of k variables and at most degree d, is one appealing example. In this case we once again have $|C^{(n)}| = O(|C^{(k)}|)$ when all gates are reliable. An open question, however, is whether a Pippenger-style construction can be adapted to circuits that compute polynomials of k variables efficiently using noisy gates.

10.2.1 Pippenger's Construction

Pippenger's construction is a modification of the well-known Lupanov construction for computing an arbitrary function using $O(2^n/n)$ gates. His construction relies on the following two theorems, which are combined by composing g_r and h_i appropriately:

THEOREM 4.1: Let $g_r(x_0, \ldots, x_{r-1}, y_0, \ldots, y_{s-1}) = y_t$, where $t = x_0 + 2x_1 + \ldots + 2^{r-1}x_{r-1}$. For every r and $s = 2^r$, g_r can be reliably computed by a network of $O(s)$ noisy gates.

THEOREM 4.2: Let $h_{a,0}(z_0, \ldots, z_{a-1}), \ldots, h_{a,b-1}(z_0, \ldots, z_{a-1})$ denote the b Boolean function of a Boolean arguments. For every a and $b = 2^{2^a}$, $h_{a,0}, \ldots, h_{a,b-1}$ can be reliably computed by a network of $O(b)$ noisy gates.

Interestingly, both of these theorems rely on the functions being computed via switching networks. In both cases, three input switches are used where one input, w, acts to control which of the other two inputs is outputted. In such a network, it is acceptable if the switch also applies some function to the input it outputs. The key criteria is that the switches output is only a function of one of the two inputs.

In the first case, a switching network is a balanced tree. In the second case, the network has an inverted tree-like structure. Each network, however, has only logarithmic depth, the size of each level either grows (or shrinks) exponentially. The result is that von Neumann-style repetition can be applied in different degrees. Small levels of the network are given high levels of redundancy, and hence high levels of reliability, while the gates that make up the largest level need not be repeated at all. To bound the reliability of an output, for any given input, one need only traverse the logarithmic number of switches on which that output depends. If r_l is the redundancy associated with level r, the probability of an output $\delta \leq \sum_{i=1}^{L} O(\epsilon^{r_l})$. The output error is thus bounded by a geometric series. This suggests that it is worth considering what other families of functions can be efficiently computed by switching networks in which only a few of the levels contain most of the switches.

10.3 Coded Prefix Computations

To conclude this chapter, we look at a concrete example of coded computation. This section describes how the framework of Chapter 9 can be employed to make a parallel prefix computation fault-tolerant.

Prefix computations are used to parallelize many operations including integer addition where it is used in the carry-lookahead adder. Let $\otimes : \mathcal{A}^2 \mapsto \mathcal{A}$ be an associate operator. Then, the **prefix function** $\mathcal{P}_{\otimes}^{(n)} : \mathcal{A}^n \mapsto \mathcal{A}^n$ maps input $\boldsymbol{x} = (x_0, x_1, \ldots, x_{n-1})$, $x_i \in A$ to output $\boldsymbol{y} = (y_0, y_1, \ldots, y_{n-1})$ where $y_1 = x_1$ and $y_i = y_{i-1} \otimes x_i$.

Prefix functions can be defined for any **semigroup** $(\mathcal{A}, \otimes)$ consisting of a set $\mathcal{A}$ and an associative operator $\otimes$. If a semigroup contains an identity element e with the property that $e \diamond x = x \diamond e = x$, it is called a **monoid**.Three examples of monoids are a) the natural numbers under addition $(\mathcal{N}, +)$, b) the set of DNA strings under string concatenation $(\{C, G, T, A\}, \cdot)$, and c) the set of pairs of Boolean variables under the diamond operator $\diamond$ that combines two pairs of carry propagate and generate bits $(\{0, 1\}^2, \diamond)$ where $\diamond$ is defined below.

$$(p_1, g_1) \diamond (p_2, g_2) = (p_1 \wedge p_2,\ (g_1 \wedge p_2) \vee g_2)$$

It is a simple exercise to show that $\diamond$ is associative.

Below we describe an algorithm to compute the prefix function on $n = 2^k$ inputs in $O(\log n)$ steps using $O(n \log n)$ operations. It is not as efficient as other $O(n)$-operation algorithms but it has the advantange that it maps exactly to a hypercube and performs the same operation at each processor on each time step after sharing data with another process across a dimension of the hypercube.

A k-hypercube (a.k.a. k-cube) has 2^k vertices each assigned a binary k-tuple. Two vertices are adjacent if their tuples differ in exactly one place. Each k-tuple is associated with an integer in the set $\{0, 1, 2, \ldots, n-1\}$ for $n = 2^k$. Individual vertices form 0-cubes. Two 0-cubes ($(i-1)$-cubes) whose tuples have a common prefix of $k-1$ ($k-i$) bits define a 1-cube (i-cube). For example, $\{v_0, v_1\}$ is a 1-cube and $\{v_0, v_1, v_2, v_3\}$ is a 2-cube.

The algorithm executes k steps on 2^k values $X = \{x_0, x_1, \ldots, x_{n-1}\}$. Initially, vertex v_j contains the pair (x_j, e). These vertices form 0-cubes. After one step two vertices v_j and v_{j+1} that form a 1-cube have their pairs combined to form new pairs $(p_t^{(1)}, s_t^{(1)})$, $t \in \{j, j+1\}$, so as to maintain the property that $p_t^{(1)}$ ($s_t^{(1)}$) is the ordered combination of X values in this 1-cube preceding and including (beyond) x_t (see Table 10.1).

In step 1 vertices whose indices differ in the least significant bit receive the pair $(p_t^{(0)}, s_t^{(0)})$ from their neighbor across the smallest dimension of the hypercube. Since the product $p_t^{(0)} \diamond s_t^{(0)}$ is the ordered combination of all the elements in the other 0-cube, each processor can update its pair to the new value $(p_t^{(1)}, s_t^{(1)})$ where $p_t^{(1)}$ ($s_t^{(1)}$) is the ordered combination of elements in the 1-cube that precede or include (succeed) x_t. Also, for each pair $(p_t^{(1)}, s_t^{(1)})$ the product $p_t^{(1)} \diamond s_t^{(1)}$ is the combination of elements in the 1-cube.

At the second algorithm step processors whose indices differ in the second least significant position receive a pair from their neighbor which contains sufficient information to update the pairs so that the invariant again holds. (See the table.)

v_i	**Step 0**	**Step 1**	**Step 2**	**Step 3**
000	(x_0, e)	(x_0, x_1)	$(x_0, x_1 \diamond x_2 \diamond x_3)$	$(x_0, x_1 \diamond x_2 \diamond x_3 \diamond x_4 \diamond x_5 \diamond x_6 \diamond x_7)$
001	(x_1, e)	$(x_0 \diamond x_1, e)$	$(x_0 \diamond x_1, x_2 \diamond x_3)$	$(x_0 \diamond x_1, x_2 \diamond x_3 \diamond x_4 \diamond x_5 \diamond x_6 \diamond x_7)$
010	(x_2, e)	(x_2, x_3)	$(x_0 \diamond x_1 \diamond x_2, x_3)$	$(x_0 \diamond x_1 \diamond x_2, x_3 \diamond x_4 \diamond x_5 \diamond x_6 \diamond x_7)$
011	(x_3, e)	$(x_2 \diamond x_3, e)$	$(x_0 \diamond x_1 \diamond x_2 \diamond x_3, e)$	$(x_0 \diamond x_1 \diamond x_2 \diamond x_3, x_4 \diamond x_5 \diamond x_6 \diamond x_7)$
100	(x_4, e)	(x_4, x_5)	$(x_4, x_5 \diamond x_6 \diamond x_7)$	$(x_0 \diamond x_1 \diamond x_2 \diamond x_3 \diamond x_4, x_5 \diamond x_6 \diamond x_7)$
101	(x_5, e)	$(x_4 \diamond x_5, e)$	$(x_4 \diamond x_5, x_6 \diamond x_7)$	$(x_0 \diamond x_1 \diamond x_2 \diamond x_3 \diamond x_4 \diamond x_5, x_6 \diamond x_7)$
110	(x_6, e)	(x_6, x_7)	$(x_4 \diamond x_5 \diamond x_6, x_7)$	$(x_0 \diamond x_1 \diamond x_2 \diamond x_3 \diamond x_4 \diamond x_5 \diamond x_6, x_7)$
111	(x_7, e)	$(x_6 \diamond x_7, e)$	$(x_4 \diamond x_5 \diamond x_6 \diamond x_7, e)$	$(x_0 \diamond x_1 \diamond x_2 \diamond x_3 \diamond x_4 \diamond x_5 \diamond x_6 \diamond x_7, e)$

Table 10.1: This table describes a 3-step parallel prefix computation as executed on an 8 vertex hypercube. At each step of the computation, each vertex, v_i, stores only a pair of values. Initially, each vertex contains the pair (x_i, e), where e denotes the identity element with regard to the associative operator $\diamond$. During the j^{th} computation step, each vertex communicates only with its neighbor along the j^{th} dimension of the hypercube. The $\diamond$ operator is applied to subsets of the four values stored at each pair of neighboring vertices in order to update the pair of values stored at each vertex.

After k steps, the tth processor v_t contains p_t^k, which is the combination of the elements in X preceding and including x_t, as well as s_t^k, which is not used.

The running time of this algorithm on a hypercube is $T(n) = O(\log n)$.

10.3.1 Encoding a Parallel Prefix Computation

As shown, a parallel prefix computation $\mathcal{P}_{\otimes}^{(k)}$ over a monoid $\mathcal{M}$ is represented as successive computations by a function $\boldsymbol{\kappa^{(k)}}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{w})$ in which $\boldsymbol{x}$ and $\boldsymbol{y}$ denote data stored at hypercube processors and those adjacent across successive dimensions of the hypercube, respectively.

The computation is encoded with RM codes (see Section 9.7.2) by first representing $\mathcal{M}$ as a set of binary tuples and $\otimes$ as an operation on these tuples, denoted as $\boldsymbol{\kappa'^{(k)}}(\boldsymbol{x'}, \boldsymbol{y'}, \boldsymbol{w'})$. The jth bit here is $\kappa'(x'_j, y'_j, w'_j)$. A binary RM codeword is formed from each bit in $\boldsymbol{x'}$, $\boldsymbol{y'}$ and $\boldsymbol{w'}$. $\kappa'(x'_j, y'_j, w'_j)$ is extended by a polynomial Φ' that is a function of the jth bit in each codeword.

Each computation step produces a result, which in the absence of errors, is in a different code from the input code. Transcoding converts it back to the input code so that a subsequent step can procede.

Consider the prefix computation for the carry-lookahead computation in which data is organized as pairs $(p_\alpha, s_\alpha) \in \mathcal{M}^2$ and pairs are combined with $\otimes$ using the functions $z_0(w)$ and $z_1(w)$ defined above. Here $\mathcal{M} = \{0, 1\}^2$ and $(u_0, u_1) \otimes (v_0, v_1) \equiv (u_0 \wedge u_1,\ (v_0 \wedge u_1) \vee v_1)$. Each of p_α and s_α is represented as a pair (u_i, v_i). Since $z_0(w)$ and $z_1(w)$ are each represented by two bits, four extension polynomials Φ'_i are computed, one for each output bit. Φ'_i is a function of seven bits, two bits for each of the three values from $\mathcal{M}$ on which $z_0(w)$ and $z_1(w)$ depends plus w. (In general, Φ'_i depends on $3 * log_2|\mathcal{M}| + 1$ bits.) Transcoding involves decoding and re-encoding the four output codewords into the input code. For more detail on transcoding see Section 9.5.

The prefix algorithm is implemented with $\log_2(k)$ steps of coded computation, each of which consists of $O(k)$ binary operations and a constant number of transcoding operations. The overhead associated with the coded-computation thus depends on the overhead of transcoding. This is an important area for future research.

Chapter 11

Conclusion

Recall, from Chapter 1, the primary assertion of this thesis:

> Emerging nanoscale computing technologies necessitate fundamental changes in the way computer architectures are designed and analyzed. Significant uncertainty is associated with the assembly and operation of nanoscale devices. This uncertainty must not only be modeled and accounted for, but actively embraced as part of the design process. This is in stark contrast with today's VLSI, where complex, meticulously optimized designs are realized through a deterministic, top-down etching process. For emerging nanoscale architectures, probabilistic modeling and analysis are primary requirements for the successful realization of nanoscale computer architectures.

Section 1.2 described four fundamental characteristics of emerging nanoscale architectures: a) stochastic assembly, b) post-assembly testing and configuration, c) strict assembly constraints, and d) imperfect operation. Chapter 2 then described four broad categories of emergent nanoscale computing technology, all of which embody these characteristics. Of the four categories, semiconductor-based nanowire crossbars are seen as the current frontrunner for near-term nanoscale architectures. As such, they are the focus of the majority of this thesis. Since a range of crossbar-related technology has already been demonstrated, nanowire crossbars offer the opportunity to describe and analyze a simple, realistic model of nanoscale computation. This analysis yields practical results, theoretical insights, and a concrete approach for exploring the general challenges posed by the four fundamental characteristics listed above.

Much of this thesis focuses on the specific problem of how nanowire crossbars can be reliably controlled with mesoscale circuitry. Chapter 3 provides a detailed look at how nanowires can be interfaced with mesoscale address wires via stochastically assembled nanowire decoders. In Section 3.2 a range of proposed nanowire decoders are reviewed, and in Section 3.4 a simple, but general approach to modeling these decoders is presented. This simple model of decoder behavior, the "binary model with errors", is powerful enough to account for manufacturing errors, but avoids the need for mathematically cumbersome physical modeling. Using this model, Chapters 4, 5 and 6 analyze the area, as well as other resources, required to reliably implement stochastically assembled randomized-contact, encoded nanowire, and masked-based decoders, respectively. As explained in Section 3.5 there are a number of possible addressing strategies for using programmable mesoscale

address translation circuitry to provide a consistent external interface to a stochastically assembled decoder. For all three types of nanowire decoder listed above, the "Take What You Get" addressing strategy appears most promising.

In Chapter 4, it is shown that randomized-contact decoders, which can potentially be realized through a wide range of assembly methods, are both efficient and robust. Sections 4.1 and 4.2 provide tight bounds on the number of mesowires required to individually address all, and most nanowires with high probability. In both cases, $O(\log_2 N)$ mesowires wires suffice when each contact group contains N nanowires. Furthermore, the constant associated with this bound is small (between 2.4 and 5), and only increases by a small constant factor when a constant fraction of all mesowire/nanowire junctions are in error. The numerical examples that appear in Section 4.3 suggest that between 10 and 30 mesowires are sufficient to reliably control 1Mb nanoscale memories using randomized-contact decoders.

In Chapter 5, encoded nanowire decoders are investigated as an alternative to randomized-contact decoders. Section 5.1 describes two schemes for encoding nanowires that are applicable to both axially and radially encoded nanowire decoders. Section 5.2 demonstrates that axially encoded nanowire decoders are extremely area efficient. Like randomized-contact decoders, they require $O(\log_2 N)$ mesowires to individually address all or most, of the N nanowires within each contact group, but here the associated constant is smaller (between 1 and 4). Furthermore, the numerical examples at the end of Section 5.2 suggest that between 5 and 20 mesowires are sufficient to control 1Mb nanoscale memories. In Section 5.4 it is shown that similarly efficient decoders can be realized using radially encoded nanowires.

Chapter 6 analyzes masked-based nanowire decoders. These decoders are somewhat different from randomized-contact and encoded nanowire decoders in that nanowire codewords are not assigned independently. Instead stochastically placed high-K dielectric regions couple mesowires to contiguous groups of nanowires. In order to bound the number of mesowires required to individually address N nanowires, Section 6.3 introduces a novel variant of the classic coupon collector problem in which each trial probabilistically targets a particular coupon. Tight bounds are provided on the number of trials required to collect all coupons with high probability. In Section 6.4, these bounds are shown to imply that masked-based decoders require $O(N \log_2 N)$ mesowires to individually address all N nanowires within a contact group. In practice, this suggests that over 100 mesoscale wires are needed to control a 1Mb memory. Fortunately, Section 6.5 demonstrates that substantially fewer mesoscale wires are required when the "Take What You Get" addressing strategy is employed. In this case under 20 mesoscale wires would likely suffice.

To complete our analysis of the size required for stochastically assembled nanowire decoders, Chapter 7 investigates how a decoder's requirements change when it is used to control crossbar-based logic instead of a memory. Section 7.2 demonstrates that the area needed to control N_A nanowire inputs using either a randomized-contact or encoded nanowire decoder is $O(\beta^2 N_A^2)$, where β is a small constant. This outperforms the deterministic construction proposed in [4]. As explained in Section 7.4, it also shows that only small constant factor overhead is needed to accommodate stochastically assembled inversion and buffering layers within nanoscale logic circuits. A novel information theoretic lower bound on β is presented in Section 7.3.

Chapter 8 describes how nanowire codewords can be efficiently discovered after a decoder has been stochastically assembled. This information is needed to configure the decoder's address translation circuitry. Section 8.1 gives an optimal algorithm for discovering all codewords using read/write operations, then describes how the algorithm can cope with both memory and decoder errors. Section 8.2, then describes how codewords can be discovered using current measurement operations in place of read/write operations. As demonstrated in Section 8.3, this approach is particularly well-suited to encoded nanowire decoders, for which an optimal codeword discovery algorithm is given. Section 8.4 then considers the more challenging problem of efficiently discovering each NW's codeword when arbitrary codewords may be present. Asymptotic analysis, along with experimental simulation, demonstrates that efficient codeword discovery remains possible. As explained, the problem of discovering nanowire codewords through current measurements is directly connected to research in PAC learning of monotone DNFs.

Having investigated the challenges associated with stochastic assembly and post-assembly testing, Chapters 9 and 10 focus on the more daunting challenge of coping with transient runtime faults. Once assembled, nanoscale devices are expected to be substantially less reliable than their mesoscale counterparts. To this end, Section 9.1 explores the notion of two-tiered reliability, that is, structuring a computation such that a limited number of gates operate at very high levels of reliability, while most (nanoscale) gates are susceptible to random faults. As explained, tiered reliability is a promising approach to fault-tolerance that offers the flexibility required to move beyond traditional repetition-based fault-tolerance. If some logic gates are highly reliable, while most are potentially faulty, the inputs to a lengthy computation can be reliably encoded using an error-correcting code. Section 9.2 defines a general, but highly regular model of computation which, is amenable to code-based fault tolerance. Section 9.3 describes a general framework for performing fault-tolerant computations on codewords. The flexibility of code-based fault-tolerance, termed coded computation, is discussed in subsequent sections, and its overhead is bounded in Section 9.8. Chapter 10 then provides additional examples of the promise and challenges of code-based fault-tolerance, or coded computation. Coded computation, along with multi-tiered reliability in general, appears to be a ripe area for future research.

This thesis has presented a wide range of probabilistic modeling and analysis in order to demonstrate the feasibility of stochastically assembled crossbar-based architectures. Even in the presence of large amounts decoder-to-decoder variation and randomly occurring manufacturing errors, we have shown that the area overhead associated with designing reliable stochastically assembled nanowire decoders can be kept to a small constant factor. Furthermore, we have shown that the overhead associated with stochastically assembled interconnect is not limited to the interconnect itself. The area and design of the associated control circuitry must be accounted for, as must the need for reliable post-assembly testing. Finally, we have provided a detailed framework for coping with transient faults via error-correcting codes. Coded computation is appealing from both a theoretical and a practical perspective, as it has the potential to significantly outperform traditional modular redundancy. In total, this thesis has provided a robust set of tools for the design and analysis of reliable nanoscale architectures.

Bibliography

[1] K. Eric Drexler. *Engines of Creation: The Coming Era of Nanotechnology.* Anchor Books, New York, NY, USA, 1986.

[2] K. Eric Drexler. *Nanosystems: molecular machinery, manufacturing, and computation.* John Wiley & Sons, Inc., New York, NY, USA, 1992.

[3] Richard E. Smalley. Of chemistry, love and nanobots. *Scientific American*, 285(3):76–77, 2001.

[4] André DeHon. Nanowire-based programmable architectures. *J. Emerg. Technol. Comput. Syst.*, 1(2):109–162, 2005.

[5] S. C. Goldstein and M. Budiu. NanoFabrics: spatial computing using molecular electronics. *Procs. 28th Annl. Int. Symp. on Computer Architecture*, pages 178–189, June 2001. see `http://www.cs.cmu.edu/~seth/papers/isca01.pdf`.

[6] Dongmok Whang, Son Jin, Yue Wu, and C. M. Lieber. Large-scale hierarchical organization of nanowire arrays for integrated nanosystems. *Nano Letters*, 3(9):1255–1259, 2003.

[7] E. Johnston-Halperin, R. Beckman, Y. Luo, N. Melosh, J. Green, and J.R. Heath. Fabrication of conducting silicon nanowire arrays. *J. Applied Physics Letters*, 96(10):5921–5923, 2004.

[8] Yong Chen, Gun-Young Jung, Doublas A. A. Ohlberg, Xuema Li, Duncan R. Stewart, Jon O. Jeppeson, Kent A. Nielson, J. Fraser Stoddart, and R. Stanley Williams. Nanoscale molecular-switch crossbar circuits. *Nanotechnology*, 14:462–468, 2003.

[9] André DeHon. Array-based architecture for FET-based, nanoscale electronics. *IEEE Transactions on Nanotechnology*, 2(1):23–32, Mar. 2003.

[10] Tad Hogg and Greg Snider. Defect-tolerant logic with nanoscale crossbar circuits. Technical report, HP Labs, 2004. see `http://www.hpl.hp.com/research/idl/papers/molecularAdder/`.

[11] André DeHon, Seth Copen Goldstein, Philip Kuekes, and Patrick Lincoln. Nonphotolithographic nanoscale memory density prospects. *IEEE Transactions on Nanotechnology*, 4(2):215–228, 2005.

[12] Benjamin Gojman, Eric Rachlin, and John E. Savage. Evaluation of design strategies for stochastically assembled nanoarray memories. *J. Emerg. Technol. Comput. Syst.*, 1(2):73–108, 2005.

[13] K. K. Likharev and D. B. Strukov. Cmol: Devices, circuits, and architectures. In G. Cuniberti *et al.*, editor, *Introduction to Molecular Electronics*, pages 447–477, 2005.

[14] John von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable componets. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 43–98, 1956.

[15] Nicholas Pippenger. On networks of noisy gates. In *Procs. 26th IEEE FOCS Symposium*, pages 30–38, 1985.

[16] Nicholas A. Melosh, Akram Boukai, Frederic Diana, Brian Gerardot, Antonio Badolato, Pierre M. Petroff, and James R. Heath. Ultrahigh-density nanowire lattices and circuits. *Science*, 300:112–115, Apr. 4, 2003.

[17] Yong Chen, Douglas A. A. Ohlberg, Xuema Li, Duncan R. Stewart, R. Stanley Williams, Jan O. Jeppesen, Kent A. Nielsen, J. Fraser Stoddart, Deirdre L. Olynick, and Erik Anderson. Nanoscale molecular-switch devices fabricated by imprint lithography. *Applied Physics Letters*, 82(10):1610–1612, 2003.

[18] Chen Yang, Zhaohui Zhon, and Charles M. Lieber. Encoding electronic properties by synthesis of axial modulation-doped silicon nanowires. *Science*, 310:1304–1307, 2005.

[19] M.R. Stan, P.D. Franzon, S.C. Goldstein, J.C. Lach, and M.M. Ziegler. Molecular electronics: from devices and interconnect to circuits and architecture. *Proceedings of the IEEE*, 91(11):1940–1957, Nov 2003.

[20] LM Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266(5187):1021–1024, 1994.

[21] Paul W. K Rothemund, Nick Papadakis, and Erik Winfree. Algorithmic self-assembly of dna sierpinski triangles. *PLoS Biol*, 2(12):e424, 12 2004.

[22] Matthew Cook, Paul W.K. Rothemund, and Erik Winfree. Self-assembled circuit patterns. *DNA Computers*, 9(7):91–107, 2004.

[23] Paul W. K. Rothemund. Folding dna to create nanoscale shapes and patterns. *Nature*, 440(7082):297–302.

[24] E. Winfree, F. Liu, L. A. Wenzler, and N. C. Seeman. Design and self-assembly of two-dimensional dna crystals. *Nature*, 394(6693):539–544, August 1998.

[25] Len Adleman, Qi Cheng, Ashish Goel, Ming-Deh Huang, David Kempe, Pablo Moisset de Espanés, and Paul Wilhelm Karl Rothemund. Combinatorial optimization problems in self-assembly. In *STOC '02: Procs. 34th Annual ACM symposium on Theory of Computing*, pages 23–32, New York, NY, USA, 2002. ACM Press.

[26] Erik Winfree and Renat Bekbolatov. Proofreading tile sets: Error correction for algorithmic self-assembly. *DNA Computers*, 9(7):126–144, 2004.

[27] Aaron Sterling. Distributed agreement in tile self-assembly. *CoRR*, abs/0902.3631, 2009.

[28] H. Wang. Proving theorems by pattern recognition ii. *Bell System Technical Journal*, 40:1–42, 1961.

[29] C. S. Lent, P. D. Tougaw, W. Porod, and G. H. Bernstein. Quantum cellular automata. *Nanotechnology*, 4:49–57, January 1993.

[30] Islamshah Amlani, Alexei O. Orlov, Geza Toth, Gary H. Bernstein, Craig S. Lent, and Gregory L. Snider. Digital Logic Gate Using Quantum-Dot Cellular Automata. *Science*, 284(5412):289–291, 1999.

[31] Géza Tóth and Craig S. Lent. Quantum computing with quantum-dot cellular automata. *Phys. Rev. A*, 63(5):052315, Apr 2001.

[32] Cabreve;lin C. Guet, Michael B. Elowitz, Weihong Hsing, and Stanislas Leibler. Combinatorial Synthesis of Genetic Networks. *Science*, 296(5572):1466–1470, 2002.

[33] Ronan Baron, Oleg Lioubashevski, Eugenii Katz, Tamara Niazov, and Itamar Willner. Logic gates and elementary computing by enzymes;. *The Journal of Physical Chemistry A*, 110(27):8548–8553, 2006.

[34] S. Y. Chou, P. R. Krauss, and P. J. Renstrom. Imprint lithography with 25-nanometer resolution. *Science*, 272:85–87, 1996.

[35] Dongmok Whang, Song Jin, and Charles M. Lieber. Nanolithography using hierarchically assembled nanowire masks. *Nano Letters*, 3(7):951–954, 2003.

[36] Zhaohui Zhong, Deli Wang, Yi Cui, Marc W. Bockrath, and Charles M. Lieber. Nanowire crossbar arrays as address decoders for integrated nanosystems. *Science*, 302:1377–1379, 2003.

[37] C. P. Collier, E. W. Wong, M. Belohradský, F. M. Raymo, J. F. Stoddart, P. J. Kuekes, R. S. Williams, and J. R. Heath. Electronically configurable molecular-based logic gates. *Science*, 285:391–394, 1999.

[38] Charles P. Collier, Gunter Mattersteig, Eric W. Wong, Yi Luo, Kristen Beverly, José Sampaio, Francisco Raymo, J. Fraser Stoddart, and James R. Heath. A [2]catenate-based solid state electronically reconfigurable switch. *Science*, 290:1172–1175, 2000.

[39] K. Gopalakrishnan, R. S. Shenoy, C. Rettner, R. King, Y. Zhang, B. Kurdi, L. D. Bozano, J. J. Welser, M. B. Rothwell, M. Jurich, M. I. Sanchez, M. Hernandez, P. M. Rice, W. P. Risk, and H. K. Wickramasinghe. The micro to nano addressing block. In *Procs. IEEE Int. Electron Devices Mtng.*, Dec. 2005.

[40] André Dehon. Deterministic addressing of nanoscale devices assembled at sublithographic pitches. *IEEE Transactions on Nanotechnology*, 4(6):681–687, 2005.

[41] P.P. Sotiriadis. Information capacity of nanowire crossbar switching networks. *Information Theory, IEEE Transactions on*, 52(7):3019–3032, July 2006.

[42] G.S. Snider and W. Robinett. Crossbar demultiplexers for nanoelectronics based on n-hot codes. *Nanotechnology, IEEE Transactions on*, 4(2):249–254, March 2005.

[43] W. Robinett, G.S. Snider, D.R. Stewart, J. Straznicky, and R. Williams. Demultiplexers for nanoelectronics constructed from nonlinear tunneling resistors. *Nanotechnology, IEEE Transactions on*, 6(3):289–254, May 2007.

[44] Eric Rachlin and John E Savage. Nanowire addressing in the face of uncertainty. In J. Becker, A. Herkersdorf, A. Mukherjee, and A. Smailagic, editors, *Procs. 2006 Int. Symp. on VLSI*, pages 225–230, Karlsruhe, Germany, March 2-3, 2006.

[45] André DeHon, Patrick Lincoln, and John E. Savage. Stochastic assembly of sublithographic nanoscale interfaces. *IEEE Transactions on Nanotechnology*, 2(3):165–174, 2003.

[46] Benjamin Gojman, Eric Rachlin, and John E Savage. Decoding of stochastically assembled nanoarrays. In *Procs 2004 Int. Symp. on VLSI*, Lafayette, LA, Feb. 19-20, 2004.

[47] John E. Savage, Eric Rachlin, André DeHon, Charles M. Lieber, and Yue Wu. Radial addressing of nanowires. *J. Emerg. Technol. Comput. Syst.*, 2(2):129–154, 2006.

[48] G. Y. Jung, S. Ganapathiappan, A. A. Ohlberg, L. Olynick, Y. Chen, William M. Tong, and R. Stanley Williams. Fabrication of a 34x34 crossbar structure at 50 nm half-pitch by UV-based nanoimprint lithography. *Nano Letters*, 4(7):1225–1229, 2004.

[49] Eric Rachlin, John E Savage, and Benjamin Gojman. Analysis of a mask-based nanowire decoder. In *Procs 2005 Int. Symp. on VLSI*, Tampa, FL, May 11-12, 2005.

[50] Eric Rachlin and John E. Savage. Analysis of mask-based nanowire decoders. *IEEE Trans. Comput.*, 57(2):175–187, 2008.

[51] R. S. Williams and P. J. Kuekes. Demultiplexer for a molecular wire crossbar network, US Patent Number 6,256,767, July 3, 2001.

[52] Jennifer Long and John E Savage. Nanowire-based crossbar modeling and analysis of a membrane-based randomized-contact decoder. In *Procs. NSTI-Nanotech 2008*, volume 3, pages 80–83, June 1-5, 2008.

[53] X. Ma, D. B. Strukov, J. H. Lee, and K. K. Likharev. Afterlife for silicon: Cmol circuit architectures. In *Procs. IEEE-NANO*, 2005.

[54] N.H. Di Spigna, D.P. Nackashi, C.J. Amsinck, S.R. Sonkusale, and P.D. Franzon. Deterministic nanowire fanout and interconnect without any critical translational alignment. *Nanotechnology, IEEE Transactions on*, 5(4):356–361, July 2006.

[55] Tad Hogg, Yong Chen, and Philip J. Kuekes. Assembling nanoscale circuits with randomized connections. *IEEE Trans. Nanotechnology*, 5(2):110–122, 2006.

[56] Eric Rachlin and John E. Savage. Nanowire addressing with randomized-contact decoders. *Theor. Comput. Sci.*, 408(2-3):241–261, 2008.

[57] Philip J Kuekes, Warren Robinett, Gabriel Seroussi, and R Stanley Williams. Defect-tolerant interconnect to nanoelectronic circuits. *Nanotechnology*, 16:869–882, 2005.

[58] Philip J Kuekes, Warren Robinett, and R Stanley Williams. Improved voltage margins using linear error-correcting codes in resistor-logic demultiplexers for nanoelectronics. *Nanotechnology*, 16:1419–1432, 2005.

[59] Eric Rachlin and John E Savage. Nanowire addressing with randomized-contact decoders. In *Procs. ICCAD*, November, 2006.

[60] Eric Rachlin and John E. Savage. Reliable nanowire addressing via randomized-contact decoders. In *Procs. TECHCON 2007, Semiconductor Research Corporation (September)*, 2007.

[61] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, Cambridge, 2005.

[62] Lincoln J. Lauhon, Mark S. Gudiksen, Deli Wang, and Charles M. Lieber. Epitaxial core-shell and core-multishell nanowire heterostructures. *Nature*, 420:57–61, 2002.

[63] Yeow Meng Chee and Alan C. H. Ling. Limit on the addressability of fault-tolerant nanowire decoders. *IEEE Transactions on Computers*, 58(1):60–68, 2009.

[64] Dr. Rob Beckman of Caltech Department of Chemistry. Personal communication, 2005.

[65] David J. C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003. available from `http://www.inference.phy.cam.ac.uk/mackay/itila/`.

[66] Jia Wang, Ming-Yang Kao, and Hai Zhou. Address generation for nanowire decoders. In *GLSVLSI '07: Proceedings of the 17th Great lakes symposium on VLSI*, pages 525–528, 2007.

[67] Jeffrey C. Jackson, Homin K. Lee, Rocco A. Servedio, and Andrew Wan. Learning random monotone dnf. In *APPROX '08 / RANDOM '08*, pages 483–497, Berlin, Heidelberg, 2008. Springer-Verlag.

[68] Avrim Blum, Prasad Chalasani, Sally A. Goldman, and Donna K. Slonim. Learning with unreliable boundary queries. *Journal of Computer and System Sciences*, 56(2):209 – 222, 1998.

[69] Helia Naeimi and André DeHon. Fault tolerant nano-memory with fault secure encoder and decoder. *Submitted to the International Conference on Nano Networks*, September 2007.

[70] Peter Gacs. Reliable computation. Technical report, Department of Computer Science, Boston University, 2005.

[71] N. Pippenger, G.D. Stamoulis, and J.N. Tsitsiklis. On a lower bound for the redundancy of reliable networks with noisy gates. *Information Theory, IEEE Transactions on*, 37(3):639–643, May 1991.

[72] Gacs and Gal. Lower bounds for the complexity of reliable boolean circuits with noisy gates. *IEEETIT: IEEE Transactions on Information Theory*, 40, 1994.

[73] William S. Evans. Information theory and noisy computation. Technical Report TR-94-057, Berkeley, CA, 1994.

[74] M. Vijay and R. Mittal. Algorithm-based fault tolerance: a review. *Microprocessors and Microsystems*, 21(3):151 – 161, 1997. Fault Tolerant Computing.

[75] Peter Elias. Computation in the presence of noice. *IBM J. Res. Develop.*, 2:346–353, 1958.

[76] W. W. Peterson and M. O. Rabin. On codes for checking logical operations. *IBM Journal of Research and Development*, 3(2):163–168, 1959.

[77] S. Winograd. Coding for logical operations. *IBM Journal of Research and Development*, 6(4):430–436, 1962.

[78] P.G. Neumann and T.R.N. Rao. Error-correcting codes for byte-organized arithmetic processors. *Computers, IEEE Transactions on*, C-24(3):226–232, March 1975.

[79] Kuang-Hua Huang and J. A. Abraham. Algorithm-based fault tolerance for matrix operations. *IEEE Trans. Comput.*, 33(6):518–528, 1984.

[80] Daniel A. Spielman. Highly fault-tolerant parallel computation. In *Procs. 37th IEEE FOCS Symposium*, pages 154–163, 1996.

[81] John E. Savage. *Models of Computation: Exploring the Power of Computing*. Addison Wesley, 1998.

[82] V. E. Beneš. Permutation groups, complexes, and rearrangeable multistage connecting networks. *Bell Syst. Techn. J.*, 43:1619–1640, 1964.

[83] H. S. Stone. Parallel processing with the perfect shuffle. *IEEE Trans. Computers*, C-20:153–161, 1971.

[84] Chuan-Lin Wu and Tse-Yun Feng. The universality of the shuffle-exchange network. *Computers, IEEE Transactions on*, C-30(5):324–332, May 1981.

[85] J. Justesen. On the complexity of decoding reed-solomon codes (corresp.). *IEEE Trans. Information Theory*, 22(2):237–238, 1976.

[86] D. V. Sarwate. On the complexity of decoding Goppa codes. *IEEE Trans. Information Theory*, 23(4):515–516, 1977.

[87] Jacobus H. van Lint. *Coding Theory*. Springer-Verlag, Lecture Notes in Mathematics, Berlin, 1973.

[88] Ilya Dumer and Kirill Shabunov. Recursive error correction for general reed-muller codes. *Discrete Appl. Math.*, 154(2):253–269, 2006.

[89] Steven Rudich and Avi Wigderson. *Computational Complexity Theory*. AMS Bookstore, 2004.

[90] 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.

[91] Shuhong Gao. A new algorithm for decoding reed-solomon codes. In *in Communications, Information and Network Security, V.Bhargava, H.V.Poor, V.Tarokh, and S.Yoon*, pages 55–68. Kluwer, 2002.

[92] Chen Ning and Yan Zhiyuan. Complexity analysis of reed-solomon decoding over gf(2m) without using syndromes. *EURASIP J. Wirel. Commun. Netw.*, 2008:1–11.

[93] Erich Kaltofen and Victor Pan. Parallel solution of toeplitz and toeplitz-like linear systems over fields of small positive characteristic. In *of Lecture Notes Ser. Comput. World Sci. Publishing*, pages 225–233, 1994.

[94] Daniel A. Spielman. Linear-time encodable and decodable error-correcting codes. In *STOC '95: Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 388–397, New York, NY, USA, 1995. ACM.

[95] T.J. Richardson and R.L. Urbanke. Efficient encoding of low-density parity-check codes. *Information Theory, IEEE Transactions on*, 47(2):638–656, Feb 2001.

[96] Peter Elias. Computation in the presence of noise. *IBM Journal on Research and Development*, 2:346–353, 1958.

[97] E. Rachlin and J.E. Savage. A framework for coded computation. *Information Theory, 2008. ISIT 2008. IEEE International Symposium on*, pages 2342–2346, July 2008.

[98] Leslie G. Valiant. Graph-theoretic properties in computational complexity. *J. Comput. Syst. Sci.*, 13(3):278–285, 1976.