UNIVERSITY OF CALIFORNIA, SAN DIEGO

Towards Reliable Nanoelectronic Systems

A dissertation submitted in partial satisfaction of the requirements for the degree
Doctor of Philosophy

in

Computer Science

by

Wenjing Rao

Committee in charge:

Professor Alex Orailoglu, Chair
Professor Peter Asbeck
Professor Chung-Kuan Cheng
Professor Keith Marzullo
Professor Tajana Rosing

2008

The dissertation of Wenjing Rao is approved, and it is acceptable in quality and form for publication on microfilm:

Chair

University of California, San Diego

2008

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

The over six years' of life in UCSD, dedicated to the journey of pursuing a Ph.D degree, has been of significant importance to my life - it has by far brought me more than I had expected before, and will definitely continue its great influences on my future life, more than I am even realizing now. I always feel very lucky of being in such an environment, where I have received so much help, that this small section falls way short of expressing the acknowledgements I owe to all the people.

First of all, many thanks to my advisor, Professor Alex Orailoglu, for the great guidance he provided, without which I couldn't have learned or accomplished as much during my Ph.D years. To name one example, I used to easily believe whatever printed in the books as an obedient student, yet during the process of debating on numerous interesting research problems with Alex, I went through a significant change and started to think independently and critically. Throughout the time, Alex kept frequent communications with me, keeping me updated with my strengths and weaknesses, so that I could always stay focused and improve. Meanwhile, my strong curiosity on a broad range of knowledge has always been highly encouraged.

Working with Alex is challenging, yet inspiring at the same time. I enjoyed very much the research meetings with him over various interesting ideas. Working on hard research topics and making progress on a fast pace, is always challenging. Nonetheless, Alex has always made it so much fun by providing an open atmosphere that encourages active thinking and debates, thus enabling ideas to grow and flourish. I never needed to worry about my questions or statements being shallow or stupid, as Alex has been always very patient to explain, and open to criticisms. Overall, working with Alex has been not only a great learning experience, but also an enjoyable one at the same time.

I also want to thank Professor Keith Marzullo, for being both a great mentor and a good friend of me. I have learned tremendously from all the input he had provided me on my research and teaching activities. In addition, throughout the years, I have benefited to a great extent from our conversations, covering a broad spectrum of topics, including research, teaching, philosophy, culture, sports, music, and even the usages of everyday English idioms.

Many thanks also go to the other professors I have met during the six years. Among them, I want to particularly thank Professor Mihir Bellare, for his guidance on the Ph.D experiences at the early years, and the course on computational complexity that I enjoyed a lot. I also want to thank Professor Ramesh Karri in Polytechnical University, for his help in many of our research papers.

Tracing back to the years before my graduate study, I also have to pay particular acknowledgements to the people who had encouraged me in pursuing a Ph.D degree. Many thanks to the advisor of my undergraduate study, Professor Xiaowei Li. I couldn't have gotten to know the area of EDA and testing, had it not been with his course and advice during my last year of undergraduate study.

Special thanks also go to my family - my parents and grandparents who are always supportive and proud of me. I could never underestimate the influences of my parents on my selection of the pursuit a Ph.D degree. My father has kept motivating me on pursuing higher education ever since I was a child, while my mother has been always supportive for whichever decision I made. Their support has made my years here a worry-free journey.

I also want to thank all my friends, for helping me through all kinds of difficulties, and for sharing with me the unforgettable life in San Diego. Particular thanks go to Kathy Wong, who hosted me upon my arrival and has become a precious friend of me ever since. Many thanks also go to my academic siblings, including Ismet Bayraktaroglu, Sule Ozev, Peter Petrov, Ozgur Sinanoglu, Baris Arslan; my good friends in UCSD, including Shuo Zhou, Jianhua Liu, Jiangsui He, Shengrong Lin, all my friends on the badminton court, and all the friends that have made my life so rich.

In the end, I want to thank my husband, whose deep love has made me strong and mature throughout the years.

VITA

1997–2001	B.S. Beijing University
2001–2008	Ph.D. University of California, San Diego

PUBLICATIONS

W. Rao, A. Orailoglu and R. Karri, “Towards Nanoelectronics Processor Architectures” in Book: Emerging Nanotechnologies: Testing, Defect Tolerance, and Reliability, Vol. 37, Chapter 13, pp. 339–372, Springer, 2008.

W. Rao, A. Orailoglu and R. Karri, “Towards Nanoelectronics Processor Architectures” Journal of Electronic Testing: Theory and Applications (JETTA), Vol. 23, pp. 235–254, 2007.

W. Rao and A. Orailoglu, “Towards Fault Tolerant Parallel Prefix Adders in Nanoelectronic Systems” to appear in IEEE Design, Automation, and Test in Europe (DATE), 2008.

W. Rao, A. Orailoglu and R. Karri, “Fault Tolerance Approaches to Nanoelectronic Programmable Logic Arrays” IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 216–224, 2007.

W. Rao, A. Orailoglu and R. Karri, “Logic Level Fault Tolerance Approaches Targeting Nanoelectronic PLAs” IEEE Design, Automation, and Test in Europe (DATE), pp. 865–869, 2007.

W. Rao, A. Orailoglu and R. Karri, “Topology Aware Mapping of Logic Functions onto Nanowire-based Crossbar Architectures” IEEE/ACM Design Automation Conference (DAC), pp. 723–726, 2006.

W. Rao, A. Orailoglu and R. Karri, “Fault Identification in Reconfigurable Carry Lookahead Adders Targeting Nanoelectronic Fabrics” IEEE European Test Symposium (ETS), pp. 63–68, 2006.

W. Rao, A. Orailoglu and R. Karri, “Nanofabric Topologies and Reconfiguration Algorithms to Support Dynamically Adaptive Fault Tolerance” IEEE VLSI Test Symposium (VTS), pp. 214–221, 2006.

W. Rao, A. Orailoglu and R. Karri, “Architectural-Level Fault Tolerant Computation in Nanoelectronic Processors” IEEE International Conference on Computer Design (ICCD), pp. 533–542, 2005.

W. Rao, A. Orailoglu and R. Karri, “Fault Tolerant Nanoelectronic Processor Architectures” IEEE Asia South Pacific Design Automation Conference (ASPDAC), pp. 311–316, 2005.

W. Rao, A. Orailoglu and G. Su, “Frugal Linear Network-Based Test Decompression for Drastic Test Cost Reductions” IEEE International Conference on Computer Design (ICCD), pp. 721–725, 2004.

W. Rao, A. Orailoglu and R. Karri, “Fault Tolerant Arithmetic with Applications in Nanotechnology based Systems” IEEE International Test Conference (ITC), pp. 472–478, 2004.

W. Rao, I. Bayraktaroglu and A. Orailoglu, “Test Application Time and Volume Compression through Seed Overlapping” IEEE/ACM Design Automation Conference (DAC), pp. 732–737, 2003.

W. Rao and A. Orailoglu, “Virtual Compression through Test Vector Stitching for Scan Based Designs” IEEE Design, Automation, and Test in Europe (DATE), pp. 104–109, 2003.

ABSTRACT OF THE DISSERTATION

Towards Reliable Nanoelectronic Systems

by

Wenjing Rao
Doctor of Philosophy in Computer Science

University of California, San Diego, 2008

Professor Alex Orailoglu, Chair

Nanoelectronics, promising significant boosts in device density, power and performance, has been projected as the new driving force for Moore's law[1] in the post-CMOS era. However, turning the vision of nanoelectronic systems into reality requires solving the fundamental challenge of unreliability. The traditional fault tolerance schemes have focused on low fault rates. Not only do they become impractically expensive, but also they fail to provide the flexibility and resilience necessitated for the significant levels of expected fault rates in the nanoelectronic environment. Furthermore, the particularities of nanofabrics necessitate that the fault tolerance techniques fit certain topological constraints to ensure viability within the expected constraints. Overall, the new realm of nanotechnology imposes manifold challenges on attaining fault resilience.

The work in this thesis addresses the reliability challenge in nanoelectronic systems from the following perspectives. Based on a number of new characteristics exhib-

[1] The observation made by Intel co-founder Gordon E. Moore, that the number of transistors on a chip roughly doubles every two years.

ited in common by the emerging nanoelectronics, a number of peculiar fault tolerance issues are identified and thus focused on throughout the thesis work. First, the two representative genres of fault tolerance approaches, fault masking and online repair, are confronted with unique challenges in nanoelectronic systems. Second, no matter which approach is chosen, building a fault tolerant nanoelectronic system cost-efficiently relies on two main principles: inherent redundancy exploitation, and a hierarchical fault tolerance strategy. Last but not least, the fundamental issue of reliability in nanoelectronics strongly impacts system design, raising a series of new modeling, algorithmic and re-evaluation considerations in the system design perspectives.

The success of nanoelectronic system construction essentially relies on the twin approaches of developing aggressive fault tolerance schemes, and novel system design with reliability considerations. The thesis work expands across multiple design abstraction levels of nanoelectronic systems, including a series of fault tolerance approaches that respond to the new characteristics of nanoelectronics, and a number of new system design perspectives that address the reliability challenge.

Chapter 1

Introduction

Current deep submicron fabrication technology imposes increasingly stringent constraints on VLSI designs. While the effects of such submicron design constraints have been observed in the elevated error occurrence rates, the extent of the challenge is dwarfed compared to the challenges that will be encountered as we enter the era of nanoelectronics [5]. While nanoelectronics promises to provide orders of magnitude improvements in device density, power and performance, turning this vision into reality requires solving the highly challenging problem of converting the nanoelectronic fabric, which is prone to high levels of faulty behavior, into an effectively reliable medium that is capable of engendering the next generation of nanoelectronic circuits.

Despite the existence of multiple nanoelectronic device candidates, they all face in common the challenge of severe unreliability, which is essentially inevitable at their ultimately shrunk device scales. On the one hand, the application requirements on future nanoelectronic systems necessitate that any such techniques continue to ensure that the highest levels of resilience be indeed attained. On the other hand, a number of new characteristics exhibited by the nanofabric, such as the regularity and adaptivity requirements, necessitate that the fault tolerance techniques fit certain topological constraints to ensure viability within the expected constraints of nanofabrics. In a nutshell, the new realm of nanotechnology imposes manifold challenges on attaining fault resilience, and the practical solutions necessitate that the dramatic overhead for achieving reliability be moderated. Whether the vision of nanoelectronic systems can be turned into reality hinges on whether we can establish efficient strategies to overcome the fundamental challenge of unreliability.

1.1 Nanoelectronic device candidates

CMOS technology has moved beyond 45 nanometers in scale, and is projected to reach beyond 22 nanometers in the next several years [14, 36]. At nanometer scale, CMOS devices start to meet the physical limits. Since the energy needed to read or write a bit is limited by the uncertainty principle, the correlated thermal, quantum and power dissipation limits of CMOS are to be reached soon, as frequency and density increase at the current fast pace. Further shrinking in the CMOS feature sizes is checkmated by the insurmountable barriers of quantum effects, leakage current and power consumption. The technology limits will eventually manifest as economical limits, such that CMOS production will cease to continue on Moore's law at nanometer scales.

While non-traditional CMOS including those advanced MOSFETs will provide a path to scaling CMOS to the end of the roadmap, perhaps by using new transistor structural designs with new materials, eventually the continuation of Moore's law relies on the shift to nanoelectronic devices that operate based on quantum physical effects. A number of emerging device candidates have been proposed as highly promising for the following perspectives: 1) functionally scalable by several orders of magnitude beyond CMOS, 2) providing high information and signal processing rate and throughput, 3) exhibiting substantially less energy dissipation, and 4) eventually implementable with minimum cost per function. At the current stage, there is not a single nanoelectronic device expectable to replace CMOS in the near future, while a number of device candidates under active research have been shown as promising for the basis for future nanoelectronic systems.

One-dimensional materials, including nanotube and nanowire, constitute critical elements in constructing nanoelectronic logic gates, as they possess the potential advantage of enhanced mobility and phase-coherent transport of electron wavefunctions, leading to fast transistors and new wave interference devices [2, 16, 34]. Carbon Nanotubes (CNT) [2] are an important subset of 1D nanoelectronic structures, as their semiconducting band structure provides the capability of varying among the three states of metallic, semiconducting, and insulating. Various simple circuits have been demonstrated by CNT based transistors, including inverters, NOR gates, basic flip-flops and ring oscillators [4, 38, 73, 74].

Resonant Tunnel Devices (RTD), including resonant tunnel transistors (RTT) and hybrid devices incorporating resonant tunneling diodes, are two terminal devices that can reach very high switching speed intrinsically, and exhibit negative differential resistance in their I-V curves [55, 89]. These two characteristics make RTD potentially attractive as high speed devices supporting multi-valued logic systems. However, the peak current through an RTD depends exponentially on the barrier thickness, thus making it inherently subject to high process variations and operation uncertainties [36].

Single-Electron Transistors (SET) are three-terminal devices operating on the Coulomb blockade where the number of electrons on an island or dot is an integer number controlled by a gate [11, 42]. Numerous SET based memory elements and circuits have been proposed, yet the operation of SET circuits is generally limited to very low temperatures. SET devices are potential candidates for significantly low power operations. However, they suffer from the two main disadvantages of low error immunity and limited fan-out. The operation basis of SET makes it extremely sensitive to thermal noise and background charges. The limited fanout restricts the logic architecture of SET to be exceedingly local based, so that SET would not have to drive a high capacitance line across the chip.

Quantum Cellular Automata (QCA) represent a new computational paradigm, where a regular array of cells performs computing and signal transferring through the interaction with neighbors [52]. Implementations of the QCA paradigm include electronic QCA and magnetic QCA. The new paradigm of QCA supports novel basic logic gates, for instance, the universal gate of majority function for Boolean algebra. Research on basic arithmetic designs, such as adders, has been carried out for QCA circuits [52]. QCA is employed in a locally interconnected architecture, and the coupling between the neighboring cells is based on their electrostatic interactions. Based on the single electron devices, QCA circuits are highly susceptible to environmental influences such as background charge.

Molecular devices have been proposed based on bistable operations of some molecules with respect to electron transport [13, 22, 46, 79]. Experiments have demonstrated two-terminal molecular devices employed at the crosspoints of a crossbar array operating in parallel, exhibiting functions as digital switches or analog diodes. A number of molecular devices also display a region of negative differential resistance, thus supporting multi-valued logic operations. Molecular devices exhibit potentials in high device densities, reaching 10^{12} bits/cm^2, as well as low power dissipation. However, the orientation of the molecular devices is hard to control during the manufacturing stage, thus resulting in massive defects, low yield, high process variation, and online operation uncertainties.

Spin transistors have been proposed conceptually based on a narrow gap semiconductor FET with ferromagnetic source and drain contacts [91]. However, spin transistors are still in their early research stage, and no viable devices have been demonstrated, despite the research activities in this field.

Overall, although each of these nano-scale device candidates operates on a unique physical basis, they present to a large extent a common set of potential advantages over CMOS, including high speed operation, low power consumption and high device densities. Besides the advantages displayed by the nanoelectronic devices, they share a number of new characteristics due to the nanometer-scale device dimensions. Most importantly, they confront in common the severe reliability challenge.

1.2 Reliability challenge

The high unreliability of nanoelectronic devices exists mainly in two forms. First, manufacturing defects increase significantly, as the fabrication process in nano environments is prone to defects due to the small scale of devices and the bottom-up self-assembly process. In comparison with the defect rates of 10^{-9} to 10^{-7} in current CMOS systems, the defect rates of nanoelectronic systems are projected to be extremely high, of 10^{-3} to 10^{-1} orders of magnitude [62].

The line between defects and process variations becomes blurred in nanoelectronic systems. Defective devices may still be functional, yet not meet the speed and reliability requirements for effective large-scale circuit operation. These effects are going to be particularly acute for the nanoelectronic devices operating on quantum physics basis, as it is well known that making nanoscale circuits operate with any degree of functional certainty is very difficult.

Consequently, a high occurrence of transient faults is expected during run-time [6, 14, 58]. This is essentially caused by device scales and the low voltage utilized in nano transistors, which result in extremely high sensitivity to environmental influences, such as temperature, cosmic ray particles and background noise.

In fact, transient faults have been observed increasingly in current CMOS based systems as the device scales down to the deep sub-micron stage. Single event upset caused by cosmic particles has already been observed in large amounts in memory systems and sequential logic state elements. Similar transient faults have started to be observed in combinational logic as well [39, 41, 75]. The challenge of dynamic transient faults is severely aggravated in nanoelectronics based systems [6, 24]. The ultra low power utilized as well as the quantum effects nanoelectronic devices rely on, both result in significantly reduced noise margins and increased sensitivity to environmental effects. Therefore, significant number of transient faults are expected to be triggered due to variances in temperature, cosmic particles, background noise, and crosstalk effects [6, 14, 24, 58].

Furthermore, not only is the fault rate projected to be high, but also a high variance in the fault rate and a clustered behavior of faults can be expected in the nanoelectronic environment. The functionality of most nanoelectronic devices is extremely sensitive to certain manufacturing parameters [36]. For instance, the peak current through an RTD depends exponentially on the barrier thickness. Similar sensitivity also exists in other devices such as CNT and molecular devices. The bottom-up manufacturing process, however, cannot perfectly control such parameters to be precisely the same across all the devices fabricated, thus resulting in high levels of process variations among the transistors. Such variations inevitably lead to differences in noise immunity among the devices in a chip, thus engendering clustered fault behavior in the system. In addition, many environmental effects, such as elevation in temperature generated when heavy computations cause high power consumption in a particular area, contribute further clustered behavior to the fault distribution in the system.

Overall, regardless of which basic device candidate to choose, future nanoelectronic systems will face severe reliability challenges. Specifically, these challenges include massive existence of manufacturing defects, significant degree of process variation, and high online fault rates with clustered and varying behavior.

1.3 New characteristics of nanoelectronic systems and their impact

1.3.1 Impact of high defect / fault rates

The most significant reliability characteristic shared among all the nanoelectronic device candidates is the exceedingly high level of defect and fault rates. The massive defect and fault occurrences projected in the nanoelectronic environment determine a fundamental difference between nanoelectronic systems and the current CMOS based systems. Basically, defect and fault tolerance is deemed to play an essential role in nanoelectronic systems, as defect-free systems become impractically expensive to fabricate, and high occurrences of run time faults have to be dealt with.

A number of changes are inevitable for the nanoelectronic systems. First of all, for the current CMOS systems, manufacturing test is performed to select defect-free chips and discard defective ones. Nanoelectronics based systems on the other hand have to be constructed with built-in defect tolerance, as massive defects will exist in almost every fabricated chip. Moreover, the current CMOS systems are highly robust to online faults, and the approaches that deal with online faults mainly focus on the detection of a low occurrence of faults. For nanoelectronic systems, the correction of online faults constitutes a major concern beyond fault detection. Furthermore, due to the high fault rates, the single fault assumption, which dominates in current CMOS systems, no longer holds. Multiple fault occurrences need to be examined and dealt with for the nanoelectronic systems.

Basically, the defect and fault tolerance schemes existing for the current CMOS systems fall far from providing a feasible solution to the reliability challenge in nanoelectronic systems. The current defect / fault tolerance approaches are based on the assumption of low defect / fault occurrences. When they are used to maintain correctness in a high defect / fault rate environment, they either fail to guarantee reliability, or demand impractical amounts of redundancy.

1.3.2 Impact of hardware abundance in nanoelectronic systems

The abundant hardware resources facilitated by the boost of device density is one of the main advantages presented in common by the nanoelectronic device candidates. Such a change not only influences the fault tolerance approaches, but also alters the design optimization space of future nanoelectronic systems.

Basically, the abundant amount of hardware supports high performance through parallelism, thus alleviating the constraint on hardware resource in the design optimization space. In other words, for nanoelectronic systems, while performance continues to be a critical optimization dimension, hardware cost is no longer as expensive. In addition, reliability is introduced as a new dimension into the design optimization space of nanoelectronic systems. The abundance in hardware resources provides a means to approach reliability through hardware redundancy. On the other hand, reliability can be approached from by exploiting time redundancy as well, yet at the cost of performance overhead. Since both performance and reliability of the system compete for the hardware resources, nanoelectronic system design needs to consider the utilization of hardware resources for the trade-off between reliability and performance.

1.3.3 Stringent locality constraint

With shrinking device dimensions, increasing device densities and increasing device speeds, wiring delay becomes the performance bottleneck and wiring area overhead becomes excessively large in late-CMOS systems. Long wires add significant overhead to area, power and delay of the system, and furthermore suffer reliability problems themselves.

In a nanoelectronic system with densely packed and highly shrunk scaled devices, such interconnect challenges are only going to be significantly worse. First of all, accessing the extremely small devices is geometrically challenging. Second, the low power operation and highly limited fan-out characteristics of nanoelectronic devices determine that, transferring information and signals at a high speed or with high bandwidth, can be only achieved within the immediate neighborhood. Overall, interconnection and communication become extremely expensive, and are strictly limited within a localized neighborhood in the nanoelectronic systems [6, 36].

The new challenge of highly localized interconnection has inevitably strong impact on the design and fault tolerance approaches for the emerging nanoelectronic systems [6, 14, 36]. In the traditional fault tolerance approaches such as TMR and NMUX, the issue of localized communication and interconnection complexity is not included in the assumption. Consequently, the application of these approaches in the nanoelectronic environment is subject to the topological constraint, and the achievable fault tolerance capability needs to be examined based on such a constraint as well.

1.3.4 Regularity and reconfigurability

Current CMOS systems utilize top-down lithographic fabrication, which is limited in obtaining precision when scaling down to the level of nanometers. Aggressive approaches in lithographic fabrication do exist for the nanoelectronic environment, yet they are too expensive to be applied for massive production. On the other hand, a bottom-up approach is expected to prevail as the basic way to construct nanoscale circuits by building structures in a self-assembly manner.

Bottom-up self-assembly based approaches have been shown for molecular electronics based fabrics with a crossbar-like architecture [12, 13, 22, 46, 53]. The resultant structure from the bottom-up fabrication process contains a number of perpendicular nanowires, forming a grid with nanoelectronic devices located at the crosspoints [34, 47, 76, 77].

Based on the crossbar structure, latch based storage elements and programmable logic array (PLA) like logic blocks have been proposed [18, 21, 47]. These approaches on the development of nanoelectronics based memory and logic, in conjunction with research work on the interface design between CMOS and nanodevices [18, 21, 80, 81], have exhibited a promising potential for the construction of functional nanoelectronic systems.

The main implications of a bottom-up fabrication process are: 1) *regularity in structures* imposed by the self-assembly process, 2) *massive defects* caused during the fabrication process, and 3) *post-fabrication reconfigurability* necessitated to define the circuits and bypass the defects. The high regularity defines a specific topology of the system, thus needing to be considered together with the locality constraint for the fault tolerance schemes. On the other hand, the high regularity opens up the opportunity for developing efficient fault tolerance schemes. For instance, detection and diagnosis of defects and faults can be performed with high efficiency in a regular system.

Reconfigurability supports a new genre of fault tolerance schemes for nanoelectronic systems, namely the online repair based approach. With the capability of online reconfiguration, once the faulty units are identified, a system can be repaired online by replacing the faulty units with spare ones. Overall, online reconfiguration based fault tolerance approaches are highly flexible and efficient in dealing with the massive occurrences of faults. By exploiting the regularity and reconfigurability of nanoelectronic systems, one can perform efficient online detection and diagnosis, and maintain the reliability of the system through a final online repair process.

Chapter 2

Thesis Overview

Reliability has ben identified as a fundamental challenge for the nanoelectronic systems. In fact, throughout the decades of evolution in electronic system, a number of devices exhibiting various promising advantages over CMOS has been proposed, including Germanium transistors, Gallium Arsenide MESFET, various threshold logic, tunneling diodes and other optical logic devices [43]. These devices have nonetheless failed to take over the dominant position of CMOS, to a large extent because they failed to match up the significant advantage offered by CMOS: high gain of the device allowing the circuit to operate reliably with great precision and signal reproducibility. Essentially, the reliability challenge has been a killer for many otherwise highly promising device candidates.

At the current stage, when CMOS is going beyond 45nm scale and reaching at a fast pace towards its physical limit, to succeed as the next generation of electronic system, nanoelectronic based systems need to display significant advantages over the current CMOS. Whether the grand vision of future nanoelectronic systems can be turned into reality directly hinges on whether the fundamental challenge of reliability can be overcome. If reliability cannot be foreseeably guaranteed in a cost-efficient manner for nanoelectronic systems, the continuous scaling down of electronic devices, which has served as the driving force for device density boost for four decades, will eventually terminate in the near future.

The reliability challenge is not only fundamental, but also quite severe. As is presented in the introduction, the unique characteristics of the emerging nanoelectronics determine a set of particularities for the reliability challenge. While fault tolerance has always been a challenging topic in electronic designs, a number of factors prevalent in nanoelectronics elevate further the expected difficulties and challenges.

Foremost are the significant levels of fault rates. The traditional fault tolerance schemes have evolved to deliver techniques that are superior in providing resilience to faults, albeit when the expected fault rates are exceedingly low in proportion to correct operation rates. As fault rates climb, not only does the occurrence of faulty behavior in components start approaching certainty, but the hardware necessary for error checking and correction itself becomes increasingly unreliable. Direct application of these fault tolerance schemes would demand hardware redundancy of three to five orders of magnitude to achieve reliability under the fault rates projected in the nanoelectronic environment [24]. Such a huge amount of redundancy requirement would immediately exhaust the density boost advantage offered by the nanoelectronic devices.

More importantly, the direct application of traditional fault tolerance schemes is not only impractically expensive, but also unsatisfactory under the nanoelectronic environment. Not only do they lack the flexibility in dealing with the fault rates with high variability, but they also fail to address the new characteristics exhibited by nanoelectronics, which have strong impact on fault tolerance strategies in various ways. Consequently, despite the abundance and maturity in fault tolerance research, there is no efficient solution directly applicable to address the reliability challenge in nanoelectronic systems.

In this chapter, we give an overview on a number of novel fault tolerance issues in nanoelectronic systems, which are to be addressed by the various approaches developed in the thesis work. Particularly, chapters 3 to 5 of the thesis focus on developing novel fault tolerance approaches, by focusing on the characteristics of nanoelectronics that are highly relevant to the reliability challenge.

The reliability challenge, accompanied by the new characteristics of the nanoelectronic systems, inevitably introduces new concerns to the system design process. On the one hand, a number of new challenges emerge, necessitating novel models and solutions to be addressed; on the other hand, the characteristics of nanoelectronics also open up new opportunities previously impractical in the system design process. Overall, the design optimization space needs a rethinking, as the tradeoff factors are shifting away from the traditional axes between performance and hardware cost. Certainly, hardware resources continue to be less and less expensive as systems move into the nanoelectronic environment. More importantly, reliability needs to be included into the tradeoff factors with performance and hardware. In chapters 6 to 8, we focus on the new design perspectives at various system hierarchy levels. New mathematical models are therein established, and corresponding algorithms are developed to address these new issues.

2.1 Novel fault tolerance themes for nanoelectronic systems

A number of fault tolerance approaches at various system hierarchical levels are proposed in this thesis. Before presenting these approaches individually, we first identify a number of important fault tolerance issues that are unique to the emerging nanoelectronic systems. Essentially, although each approach focuses on a specific perspective of the system, such as nanoelectronic logic, arithmetic component, or topological constraint, they all try to address the common set of fault tolerance issues identified for nanoelectronic systems.

2.1.1 Existent redundancy exploitation

Any fault tolerance approach relies on certain forms of redundancy, and aggressive schemes targeting the high fault rates necessitate redundancy in a large amount. Introducing extra redundancy into the system is always expensive, no matter which form it takes, since hardware, time and information redundancy each imposes its particular overhead to the system. Developing powerful fault tolerance schemes at a low cost essentially depends on the exploitation of the existent redundancy in the system. One might question the existence of such “exploitable” redundancy in an efficiently designed system. However, in a nutshell, this is achievable from two perspectives.

First, typically a function has multiple implementations with different tradeoffs between performance and hardware cost. The abundant hardware resource in the nanoelectronic environment makes the high performance designs much more favorable, using parallel computations instead of serial ones. In these high performance modules, there typically exist alternative computational paths, which represent a form of redundancy exploitable for fault tolerance purposes.

In addition, it is very likely that various fault tolerance approaches are necessitated in combination, due to their complementary advantages and costs, as well as the requirement for aggressive fault tolerance approaches to maintain reliability in an environment with high fault rates. Particularly, fault masking schemes and online repair based approaches present complementary attributes in terms of performance and flexibility for variable fault rates. The combination of the two can therefore provide a powerful fault tolerance capability for a nanoelectronic system. By exploiting the redundancy employed for one scheme to serve the other one as well, a highly efficient overall fault tolerance approach can be achieved.

2.1.2 Fault masking issues in nano

Fault masking approaches constitute the genre of most general fault tolerance schemes, including the representative N-Modular Redundancy (NMR) and N-Multiplex (NMUX) schemes. Basically, hardware redundancy is used in these approaches to provide multiple redundant copies of the same computation. The correct result can be achieved despite the fault occurrences that may lie below a certain threshold. Fault masking approaches are generally applicable for most computations. Nevertheless, for nanoelectronic systems, a number of important issues emerge and need to be addressed for fault masking approaches to be applicable.

- Due to their inflexibility, fault masking schemes typically have poor efficiency under high and variable fault rates. The redundancy for a fault masking scheme is usually preset to a fixed amount. Consequently, the redundancy amount needs to target the worst case to guarantee reliability under a high fault rate, thus demanding tremendous hardware overhead.

- Fault masking schemes have a certain threshold for tolerating faults. At the presence of multiple faults beyond the preset threshold, the result becomes incorrect. Fault masking schemes do not naturally provide a mechanism to detect such a scenario. thus becoming vulnerable under the high occurrence of faults in a nanoelectronic environment. In other words, an additional layer of fault detection capability needs to be introduced for the fault masking schemes to identify the erroneous output, when the number of fault occurrences reaches beyond the threshold.

- Majority voting based fault masking schemes, such as NMR, rely on a voter in the final stage for attaining a correct result. The implementation of a voter poses a number of issues under the nanoelectronic environment. At a low design hierarchical level, the implementation of a majority voter could be exceedingly expensive, both in terms of hardware and performance. Furthermore, the reliability of the voter poses as an additional concern, since it is crucial to the entire fault masking scheme. The voter's reliability might be hard to guarantee at a low design level, and can be costly at a higher design level.

- The implementation of a fault masking scheme is subject to the stringent locality constraint in the nanoelectronic environment. In order to cover a high rate of faults, both NMR and NMUX based approaches involve considerable number of interconnections, which are presumed available in the general models.

Consequently, the capability of fault masking schemes under the nanoelectronic environment is limited by the interconnect constraints. A reevaluation of the fault masking schemes' capability is necessitated based on the topological information. Furthermore, how to develop powerful fault masking schemes that address the locality issue in the nanoelectronic environment constitutes an important issue.

2.1.3 Online repair

Since online reconfigurability is supported by a large number of nanoelectronic devices, online repair based fault tolerance schemes are envisioned as promising alternatives to fault masking schemes. Online repair approaches provide complementary advantages to the fault masking schemes, due to their high flexibility and efficiency in utilizing the hardware redundancies. Online repair based approaches are best applicable to systems with high regularities and a large number of identical components. The bottom-up fabrication process naturally results in systems with high regularity; furthermore, most high performance arithmetic components which rely on parallelism consist of a large number of identical subcomponents. Consequently, online repair based approaches are highly promising for nanoelectronic systems.

However, online repair schemes involve a relatively long process, consisting of online detection, fault diagnosis and the final stage of reconfiguration based repair, necessitating extra control hardware as well. Whether an online repair based scheme can deliver high fault tolerance capability with efficiency depends on the three individual steps.

- The first stage of *online fault detection* can benefit from the existing research in the CMOS domain, as the problem of online fault detection has been well investigated in the literature, and various online testing approaches have been developed.

- The second stage of *online fault diagnosis* is responsible for identifying the faulty units that are to be replaced in the third stage. Online diagnosis constitutes a crucial stage to the end result of an online repair based approach.

  Such a problem was not a prominent concern previously, since the CMOS based systems are highly robust, and online reconfigurability is not commonly present. In CMOS based systems, fault diagnosis has been mainly performed under offline environment conditions. Performing fault diagnosis online is by far a much more challenging issue, due to the lack of controllability in an online environment.

  Overall, online diagnosis has to be done on the fly at runtime, and its resolution directly influences the amount of hardware needed to perform the repair in the third stage, since a low diagnosis resolution inevitably results in the replacement of a large chunk of hardware in the ambiguity set. Consequently, online diagnosis constitutes a crucial issue for the online repair based fault tolerance approaches. The development of efficient online diagnosis schemes relies on the exploitation of regularity in the system, and reusing the existent redundancies.

- The final stage of *reconfiguration based repair* replaces the faulty component candidates identified in the second stage with spare ones. Therefore, the hardware overhead required in this stage relies heavily on the diagnosis resolution in the previous stage.

  Online repair based schemes can achieve high flexibility through spare unit sharing. However, once again, the locality constraint in the nanoelectronic environment needs to be considered. To address the issue of redundancy sharing under topological constraints for the repair stage, new models need to be established to develop efficient redundancy allocation schemes and perform evaluations with locality considerations.

2.1.4 Hierarchical fault tolerance approaches

Although the massive fault occurrences in a nanoelectronic system stem from the reliability challenge at the device level, it takes a series of hierarchically organized fault tolerance approaches to guarantee the computation correctness of the entire system, and furthermore the mechanism to achieve performance, hardware and reliability tradeoff. Trying to maintain reliability at a single design abstraction level is impractical for the significant fault rates in the nanoelectronic environment, and a hierarchical fault tolerance is the only way to construct reliable nanoelectronic systems efficiently, for the following reasons:

- Not all fault tolerance approaches are applicable at each design hierarchical level.

  The lower design abstraction levels have the advantage of relatively cheap hardware resources, yet cannot afford fault tolerance approaches that require complex control mechanisms. Consequently, achieving the flexibility to deal with the high variances of fault rates, such as applying time redundancy and balancing hardware and performance dynamically is extremely expensive. At the higher design abstractions, each basic component is rather complex. Fault tolerance schemes that are simple yet demand large hardware redundancy, such as NMR and NMUX, are expensive to implement. Certain types of fault tolerance approaches have highly constrained applicability areas. For instance, information redundancy based error checking / correction coding approaches can be only applied to a set of subsystems such as data storage and transferring components. To address the reliability challenge of nanoelectronic systems on a single design abstraction level, no matter which one, would restrict one's choice to a small selection of fault tolerance approaches. Failing to exploit the various benefits provided by the full range of fault tolerance approaches undoubtedly hampers the power one would need to overcome the severe reliability challenge.

- The nature of fault tolerance approaches determines a peculiar curve of hardware cost versus achievable reliability, where the hardware cost does not grow linearly with the achievable reliability [29]. Basically, reliability improvement costs significantly higher when it gets close to 100%. In other words, it costs much less to improve the reliability of a system from 60% to 70% than to strive for the last ten percentage points from 90% to 100%.

  A hierarchical fault tolerance strategy, therefore, exploits lower levels of design abstraction to provide limited fault masking to cover the bulk of expected faults, while the tail end of the fault distribution is left to higher design abstraction levels. The hybrid approach enables low-cost fault handling for most faults while expending control complexity at higher chunk granularity, thus amortizing it.

2.2 Overview of proposed work towards reliable nanoelectronic systems

The thesis work on fault tolerance of nanoelectronic systems includes three distinct pieces. Each piece of the work focuses on fault tolerance issues from a particular perspective of system design hierarchical level. However, each piece of the work addresses multiple fault tolerance issues discussed above and forms a relatively complete approach.

2.2.1 Fault tolerant nano PLA logic

For nano PLA logic based on the highly regular crossbar architecture, a special fault masking scheme is developed based on two-level logic tautology, Traditional fault masking schemes are shown to be highly inefficient at the logic gate design level, under the locality constraint. By exploiting the characteristics of the two-level logic, such a fault masking scheme becomes highly applicable to the regular structure of nano PLAs.

Furthermore, a reconfiguration based online repair scheme is proposed to compensate the fault masking approach. The particular issue of online diagnosis is addressed by exploiting the regularity of nano PLA structure as well as the characteristics of two-level logic functions. The redundancy employed in the proposed fault masking scheme is exploited for the online repair approach as well. Consequently, this approach provides an efficient fault tolerance approach integrating fault masking and online repair for nanoelectronic PLAs.

2.2.2 Reliable parallel adders in nano

An adder is the basic building block for arithmetic components, thus serving as a starting point for research on fault tolerant arithmetic units. In nanoelectronic systems, when the constraint on hardware cost is significantly alleviated, high performance parallel adders easily become the favorable choices.

Parallel adders, including carry lookahead adders (CLA) and parallel prefix adders (PPA), employ parallel computation paths to eliminate carry propagation. Based on the correlations among the carry signals, alternative calculation paths can be constructed for the same computation with highly insignificant hardware and performance overhead. This essentially indicates a way to exploit existing redundancy in the system for fault tolerance purposes.

This piece of research work exploits such redundancy in parallel adders, including CLA and PPA, for fault masking, online fault detection and diagnosis. It turns out that by exploiting the inherent redundancy and the regularity within parallel adders, various fault tolerance schemes can be supported with very low cost, thus delivering high reliability efficiently.

2.2.3 Locality aware redundancy allocation

This piece of work focuses on the locality issue of the nanoelectronic environment and its impact on fault tolerance approaches. Particularly, redundancy allocation for an online repair based approach is tightly connected to the constraint of localized interconnections and communications. Basically, the spare units in a system for repair purposes need to be shared among as many as possible functional units to enhance flexibility and fault tolerance capability. However, such a sharing is not only subject to the compatibility among the functional units and spare units, but also limited due to the interconnection constraint.

The problem of redundancy allocation under a locality constraint is mathematically modeled and investigated. Such a model can be used to evaluate the reliability of a system, based on the amount of redundancy and the interconnection complexity, and is independent of the specific details of the topological layout. Based on the proposed model, reconfiguration based defect and fault tolerance algorithms are developed to maximize the reliability of a nanoelectronic system under manufacturing defects and online fault occurrences, while satisfying the interconnection constraints.

As a case study, a specific flexible NMR fault tolerance scheme is proposed, which is essentially a fault masking scheme supported by an online repair approach. With localized redundancy sharing, the fault masking scheme is enhanced with high flexibility, thus being capable of recovering from massive manufacturing defects, and tolerating high and variable occurrences of online faults.

2.3 New designs of nanoelectronic systems under the reliability challenge

Specifically, the three pieces of research focus on the following:

- The traditional logic synthesis process focuses on the optimization of logic functions. In a PLA structure, thanks to the regular structure, the mapping of a resultant logic function onto the fabric is highly flexible and a trivial process. In a nano PLA, however, the massive defects in the fabric impose strict constraints to the function mapping process. Therefore, the process of function mapping emerges as a new challenge. We address this problem by developing a mathematical model for it using a bipartite graph representation. Based on the model, we develop the corresponding algorithm to perform a defect aware function mapping.

- Despite its outstanding performance, the applicability of carry save arithmetic is traditionally limited due to the requirement of an underlying multi-valued logic system. The implementation of multi-valued logic system through the traditional binary-based CMOS system is quite expensive. Multi-valued logic is naturally supported by a number of nanoelectronic devices, thus enabling the examination of carry-save arithmetic for nanoelectronic systems based on these devices. Applying carry-save arithmetic not only engenders great potential of performance boost, but also opens up the opportunity for developing information redundancy based fault tolerance to the arithmetic components. By exploiting the characteristics of carry-save arithmetics, we develop an error checking code based fault tolerance scheme for arithmetic units. As a result, a unified information redundancy based fault tolerance scheme can be extended to arithmetic units, in addition to its traditional application areas such as buses and memory subsystems.

- For a processor architecture based on unreliable nanoelectronic devices, since any fault tolerance approach demands redundancy either in the form of time or hardware, reliability needs to be considered in conjunction with the performance and hardware tradeoffs. A new computational model for nanoelectronics processor architectures is introduced, which provides flexible fault tolerance to deal with the high and time varying faults.

Chapter 3

Fault Tolerant Nano PLA Logic

Programmable logic arrays (PLAs) are promising as platforms for nanoelectronic logic, since they are highly regular and can be supported by the nano crossbar architectures. This chapter focuses on the fault tolerance of nanoelectronic PLAs, so as to ensure their viability as a foundation for the logic level of nanoelectronic systems.

In this chapter, we present a comprehensive fault tolerance scheme integrating fault masking and online repair for nanoelectronic PLAs. The proposed fault masking scheme is based on two-level logic tautology, thus necessitating no majority voting and displaying significant advantages over the traditional TMR based approach on nano PLAs. Based on the two-level logic characteristics in a PLA structure, we propose an online repair scheme on top of the fault masking scheme for enhanced reliability, when fault occurrences exceed the threshold of the fault masking capability. For the online repair approach, we focus on the crucial stage of online diagnosis. By utilizing the existent redundancy employed in the fault masking scheme, high diagnosis resolution can be achieved with insignificant hardware overhead.

3.1 Motivation

As the crossbar based architectures are placed center stage as promising fundamental basic structures in constructing nanoelectronic systems, nano crossbar based PLAs exhibit significant potential as a logic level basic architecture, since any arbitrary function can be implemented in a two-level logic form. PLA based logic systems in CMOS have enjoyed extensive research attention resulting in mature technologies. To deal with the tremendous increases in online fault rates in the nanoelectronic environment, aggressive fault tolerance techniques need to be developed and integrated into general PLA designs.

In essence, any fault tolerance scheme relies on certain forms of redundancy. It has been shown that the generally applicable fault tolerance schemes demand a tremendous amount of hardware redundancy for the high fault rates in nanoelectronic systems [24, 62], making them infeasible even with the support of the hardware density boost in the nanoelectronic environment.

To overcome the unreliability challenge for nanoelectronic systems, the only applicable way remains the exploitation of the particularity of the system, rather than the simple adoption of generic fault tolerance approaches. For nano PLAs, the high regularity of PLA logic and the online reconfigurability of nano crossbars can be exploited to support efficient fault tolerance schemes.

We focus on developing two genres of fault tolerance approaches, namely the fault masking schemes and online repair based approaches, for nano PLAs.

- Fault masking based schemes introduce low delay penalties and perform best when dealing with transient faults. Fault masking approaches can guarantee the correctness of the computational output for fault occurrences under a certain threshold. However, when the number of fault occurrences exceeds the threshold, fault masking fails to maintain a correct output.

- Online repair based approaches enhance the reliability of the system by detecting the faulty units and performing a reconfiguration process. The nanoelectronic devices located at the crosspoints in a crossbar exhibit two distinct states of connecting and disconnecting the two wires, and these two states can be configured by applying a positive or negative voltage correspondingly [13, 34, 53]. For such a system supporting a highly dynamic reconfiguration capability, online repair is promising for its hardware efficiency in dealing with massive faults.

The two genres of approaches, online repair and fault masking, exhibit complementary advantages in their fault tolerance capabilities. Even though development of fault masking and online repair schemes for nano PLAs can benefit from exploiting the characteristics of nanoelectronics and the regularity of PLA logic, implementing the two approaches independently proves to be highly costly. The only possibility to apply both strategies to guarantee the reliability of nano PLAs is through the overlapping of their redundancy requirements. In other words, when the existent redundancy can be used by both the fault masking and the online repair based approach, a powerful integrated fault tolerance scheme can be achieved efficiently for nanoelectronic PLAs.

3.2 Preliminary: Fault Models for nano PLAs

In nanoelectronics, particularly for the crossbar based PLA structures, a massive number of two-terminal molecular devices are sandwiched at the crosspoints between two orthogonal layers of densely packed parallel nanowires. Due to the highly sensitive devices and their unreliable characteristics, the main fault occurrences are expected at the crosspoints. These online faults include permanent faults such as a device becoming nonprogrammable, as well as transient faults of switching between states due to environmental effects [12, 20, 34, 36]. For the faults occurring at the crosspoints of a nano crossbar architecture, their corresponding fault models in a two-level PLA logic can be represented with four types of faults [1, 8, 78], which are shown in table 3.1.

The four fault types are essentially the combination of two dimensions: the oc-

Table 3.1: Fault models of PLA

fault	type	K-map	cause	effect	output	example
AND $\downarrow$	G	growth	missing device in AND plane	missing variable	$0 \rightarrow 1$	$f = ab + cd$ $\rightarrow b + cd$
AND $\uparrow$	S	shrink	extra device in AND plane	extra variable	$1 \rightarrow 0$	$f = ab + cd$ $\rightarrow abe + cd$
OR $\downarrow$	D	disappearance	missing device in OR plane	missing product term	$1 \rightarrow 0$	$f = ab + cd$ $\rightarrow cd$
OR $\uparrow$	A	appearance	extra device in OR plane	extra product term	$0 \rightarrow 1$	$f = ab + cd$ $\rightarrow ab + cd + e$

currence plane (AND or OR), and the occurrence type (missing or extra device). For instance, for a G type fault, a device is either missing or switched from an "on" state to an erroneous "off" state in the AND plane. The fault effect results in a *growth* in the Karnaugh map. At the logic level, a variable is dropped from a product term, and the outputs connected to the product term change unidirectionally from 0 to 1. In the example shown in table 3.1, a G type fault with a missing variable a changes the original function of $ab + cd$ into the erroneous $b + cd$. We can notice from the table that all these four types of faults lead to unidirectional changes in the output. This is an important attribute for the crosspoint faults in PLA logic, based on which specialized techniques that entail reduced hardware overhead can be developed for fault tolerance purposes.

3.3 Fault Masking for nano PLAs

Fault masking is a general technique that can be applied straightforwardly to arbitrary functions. In the traditional NMR based fault masking approach, to achieve the single fault masking capability, at least TMR with triple the amount of hardware is required, plus the additional overhead of a majority voter.

However, the general fault masking scheme of NMR does not provide an effective solution for nano PLAs. The cost of a voting process in NMR based approaches, both in terms of hardware and performance, is typically amortized by the size of the component and the complexity of the computation itself. At the low design hierarchical level of logic gates, any function implemented by a nano PLA has two logic levels only. Implementing NMR based approaches in a PLA logic form is highly expensive, due to the relative cost of implementing a voter when compared to the size of a typical two-level logic function by itself. Not only is significant area overhead needed for a majority voter itself, but also tremendous additional area is wasted to support the NMR structure with the highly regular PLA architecture. In addition, for any NMR based approach, the crucial issue of voter's reliability needs to be addressed. In the case of nano PLA, this needs to be approached by adding extra redundancy to make a voter itself fault tolerant. Alternatively, a voter might be implemented with more reliable devices such as CMOS. However, this demands a hybrid implementation of nanoelectronic devices and CMOS devices and is expensive. Overall, applying the general fault masking scheme of NMR to nano PLAs, which reside at the logic gate design hierarchical level, is highly inefficient.

Alternatively, by exploiting the characteristics of two-level logic functions, a Boolean logic tautology can be used to develop a class of new fault masking schemes that are particularly favorable to nano PLAs. With tautology based fault masking, the redundancy is integrated within the logic function. Consequently, faults can be masked without an explicit majority voting process and significant improvement can be achieved from both the performance and the hardware utilization perspectives.

3.3.1 Tautology based fault masking examples

In Boolean logic, the AND and the OR functions provide two tautology forms, which we will refer to as $\widehat{f_{AND}}$ and $\widehat{f_{OR}}$:

$$\widehat{f_{AND}} = f \cdot f \equiv f$$

$$\widehat{f_{OR}} = f + f \equiv f$$

These tautology forms represent two redundant designs for a Boolean logic function. By embedding the redundancy within the Boolean logic, a number of faults can be masked without the necessity for a majority voting process at all.

Table 3.2: Tautology form fault masking capability over the 4 fault models

fault	type	f under fault	$\widehat{f_{AND}}$ under fault	$\widehat{f_{OR}}$ under fault
AND ↓	G	$ab \rightarrow a$	$ab \cdot ab \rightarrow aba = f$	$ab + ab \rightarrow a + ab \neq f$
AND ↑	S	$ab \rightarrow abc$	$ab \cdot ab \rightarrow ababc \neq f$	$ab + ab \rightarrow ab + abc = f$
OR ↓	D	$p_1 + p_2 \rightarrow p_1$	$(p_1 + p_2)(p_1 + p_2) \rightarrow p_1(p_1 + p_2) \neq f$	$p_1 + p_2 + p_1 + p_2 \rightarrow p_1 + p_1 + p_2 = f$
OR ↑	A	$p_1 + p_2 \rightarrow p_1 + p_2 + p_3$	$(p_1 + p_2)(p_1 + p_2) \rightarrow (p_1 + p_2)(p_1 + p_2 + p_3) = f$	$p_1 + p_2 + p_1 + p_2 \rightarrow p_1 + p_2 + p_1 + p_2 + p_3 \neq f$

Table 3.2 shows the fault masking capability of the two tautology forms over the four fault types. The AND and OR tautology forms have complementary fault masking capabilities depending on the fault manifestation directions and the fault occurrence plane. Specifically, $\widehat{f_{AND}}$ can mask fault types G and D, while it is susceptible to the S and A types of faults. $\widehat{f_{OR}}$ exhibits exactly the complementary capability and susceptibility.

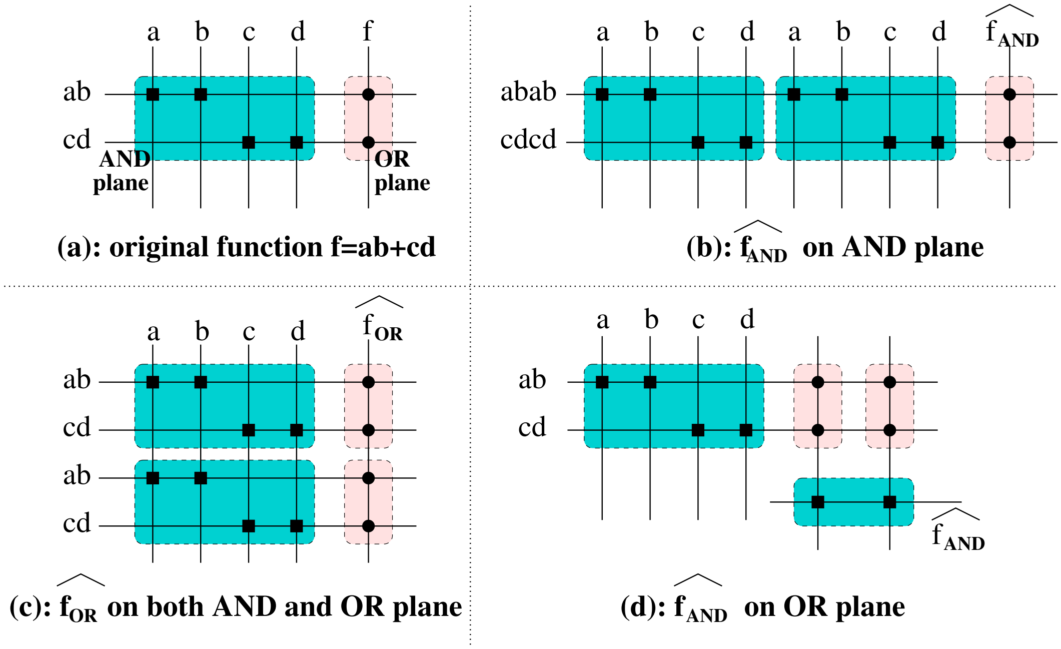
As a further observation, applying a tautology form to the plane of the same function results in a single level logic, while applying it on the other plane necessitates an additional logic level. Basically, neither the application of $\widehat{f_{AND}}$ tautology to an AND function nor of the $\widehat{f_{OR}}$ tautology on an OR function entails any extra logic level. The converse cases, on the other hand, do introduce an additional logic level.

Figure 3.1 illustrates how Boolean tautology can be used on a 2-level nano PLA structure:

- Figure 3.1(a) shows the direct PLA implementation of the original function $f = ab + cd$.

- In figure 3.1(b), the $\widehat{f_{AND}}$ tautology is applied to the AND plane, thus masking the G type faults, namely the AND plane missing device faults.

- In figure 3.1(c), the $\widehat{f_{OR}}$ tautology is applied on both the AND and the OR planes.



(a): original function f=ab+cd

(b): $\widehat{f}_{AND}$ on AND plane

(c): $\widehat{f}_{OR}$ on both AND and OR plane

(d): $\widehat{f}_{AND}$ on OR plane

Fault	(a)	(b)	(c)	(d)
G type: missing device @ AND plane		✓		
S type: extra device @ AND plane			✓	
D type: missing device @ OR plane			✓	
A type: extra device @ OR plane				✓

Figure 3.1: Fault masking examples

As a result, both the S (AND plane extra device) and the D (OR plane missing device) faults are masked. Moreover, the extra level of OR logic introduced by applying $\widehat{f_{OR}}$ to the AND plane is absorbed in the original OR plane of the PLA; therefore, the resulting structure can still maintain a 2-level logic structure.

- In figure 3.1(d), $\widehat{f_{AND}}$ is applied to the OR plane, thus masking the A type of faults, i.e., OR plane extra device faults. In this case, an extra level of AND logic is inevitable since it has to be added after the OR plane in the original PLA.

Overall, as is shown in the examples from figure 3.1, each of the four types of faults can be masked by applying the Boolean tautology forms in nano PLAs.

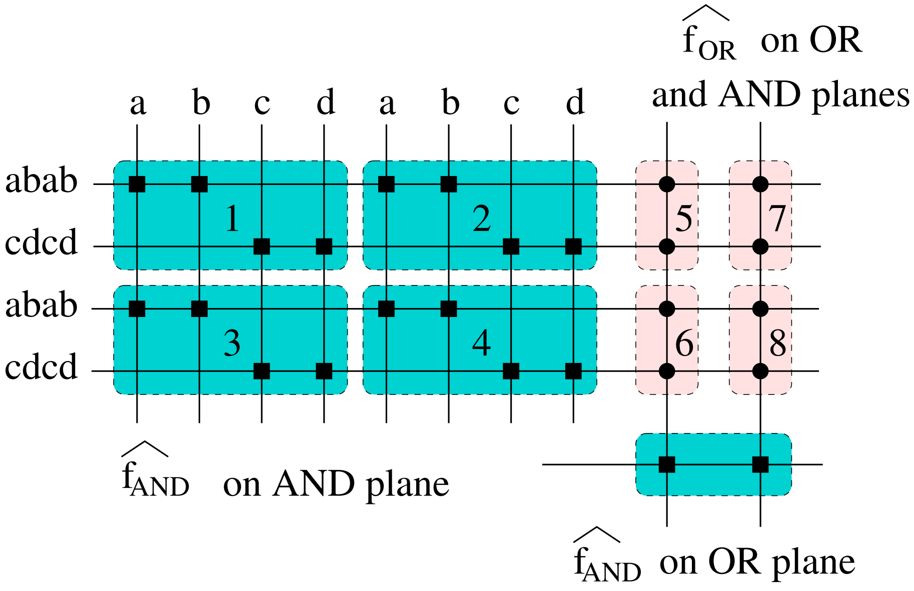


Figure 3.2: Tautology based fault masking for all 4 fault types

3.3.2 Tautology based fault masking in nano PLAs

Single fault masking analysis

To cover all the four types of faults simultaneously, one needs to combine the approaches shown in figure 3.1. We show in figure 3.2 the corresponding example of such a combined approach. The AND planes are labeled from 1 to 4, while the OR planes are labeled from 5 to 7. All four types of faults can be masked according to the following analysis:

G type: a missing device fault in an AND plane is masked by applying an $\widehat{f_{AND}}$ tautology on the AND plane.

For instance, the fault effect of any missing device in plane 1 never reaches the OR planes, due to the redundant copy of plane 2. Consequently, within the plane pair 1 and 2 (as well as plane pair 3 and 4), any missing device fault occurrence can be successfully masked.

S type: an extra device fault in an AND plane is masked by applying $\widehat{f_{OR}}$ tautology on the AND plane.

An extra device fault occurring in AND plane 1 cannot be masked by plane 2, and will manifest in both OR planes 5 and 7. However, the existence of redundant AND planes 3 and 4 guarantees that OR planes 6 and 8 are not affected by the fault. Therefore, the fault manifestation in plane 5 can be masked by plane 6. At the same time, the fault manifestation in plane 7 is masked by plane 8.

D type: a missing device fault in an OR plane is masked by applying the $\widehat{f_{OR}}$ tautology on the OR plane.

Within the plane pair 5 and 6 (as well as plane pair 7 and 8), any missing device fault occurrence can be successfully masked.

A type: an extra device fault in an OR plane is masked by applying $\widehat{f_{AND}}$ tautology on the OR plane.

This is achieved by adding the redundant planes 7 and 8 for planes 5 and 6. With the additional level of AND plane at the end, any extra device fault occurring in the OR planes can be masked at the final output.

The above analysis shows that any single fault occurrence of any fault type can be masked by the tautology based fault tolerance scheme.

Double fault masking analysis

In fact, most of the double fault and multiple fault occurrences can be masked as well by the tautology based approach. To estimate the double-fault masking capability, assuming that the PLA consists of I input, O outputs and P product terms, then there exist $I \times P$ crosspoints in each of the AND planes, and $P \times O$ crosspoints in each of the OR planes. The total number of possible double fault occurrences in all the eight planes, considering the combination of all the four types, is therefore $(2O + 2I) \times 2P \times ((2O + 2I) \times 2P - 1) \times 4 \times 4/2$. We provide an analysis on the double-fault masking capability of the tautology based approach by listing below all the possible double-faults that escape masking.

G+G: (AND $\downarrow\downarrow$) When the first fault is a G type fault (device missing in AND plane) in plane 1, the fault effect will be masked unless a second G type fault occurs in plane 2 at exactly the same location. The number of possible occurrences of this type is $2 \times I \times P$.

S+S: (AND $\uparrow\uparrow$) When the first fault is an S type fault occurring in plane 1, then a second fault of the same type in plane 3 or 4 might escape the fault masking approach. However, the second fault's position in the plane has to be in the same row as the first one in plane 1. In other words, double faults where both faults are of the S type will be masked unless the two faults take place at the copies of the same product term signal. The number of possible occurrences of this type is $2 \times I^2 \times P$.

S+D: (AND $\uparrow$ OR $\downarrow$) Since the fault effect of an S type fault (extra device in the AND plane) always reaches the OR planes and relies on the redundancy in the OR planes to mask, a second fault in the OR plane might result in fault masking failure. For instance, when the first S type fault occurs in plane 1 or 2, the occurrence of a D type fault at the same product line in plane 6 or 8 will make the fault effect unmaskable. Essentially, the D type fault (missing device in the OR plane) prevents the redundant correct copy of the product term signal from reaching the final result, thus enabling the faulty signal to penetrate through and affect the output values. The number of such unmaskable combinations of an S type fault and a D type fault is $4 \times I \times P$.

D+D: (OR $\downarrow\downarrow$) In a manner analogous to the AND $\downarrow\downarrow$ case, a D type fault occurring in an OR plane escapes the fault masking approach only if there is a second fault of the same type occurring in the same location of the corresponding redundant plane (5 with 6, and 7 with 8). The number of possible occurrences of this type is $2 \times P \times O$.

A+A: (OR $\uparrow\uparrow$) In a manner analogous to the AND $\uparrow\uparrow$ case, the double occurrence of A type faults cannot be masked if they occur at the two redundant copies of the same output signal. The number of possible occurrences of this case is $2 \times P^2 \times O$.

An overall rate of the unmaskable double fault occurrence can be calculated by summing up the above cases and dividing it by the total number of possible double fault occurrences. For a non-trivial PLA with I, P, O larger than 2, the percentage of unmaskable double fault occurrence is less than 1%. For larger PLAs, the percentage is far below 1%.

3.3.3 Hardware overhead analysis

The hardware overhead for a fault masking PLA scheme needs to be analysed from both the device and the wiring aspects. We will discuss this issue based on the previous assumption that the nanoelectronic PLA implements a logic function with I input wires, O function outputs and P product terms. Furthermore, we assume also that the number of devices utilized in the original PLA is D_A in the AND plane, and D_O in the OR plane. Figure 3.3 illustrates the overall schematics for the proposed tautology based fault masking scheme, and in comparison, the TMR based fault masking implemented in a nano PLA.

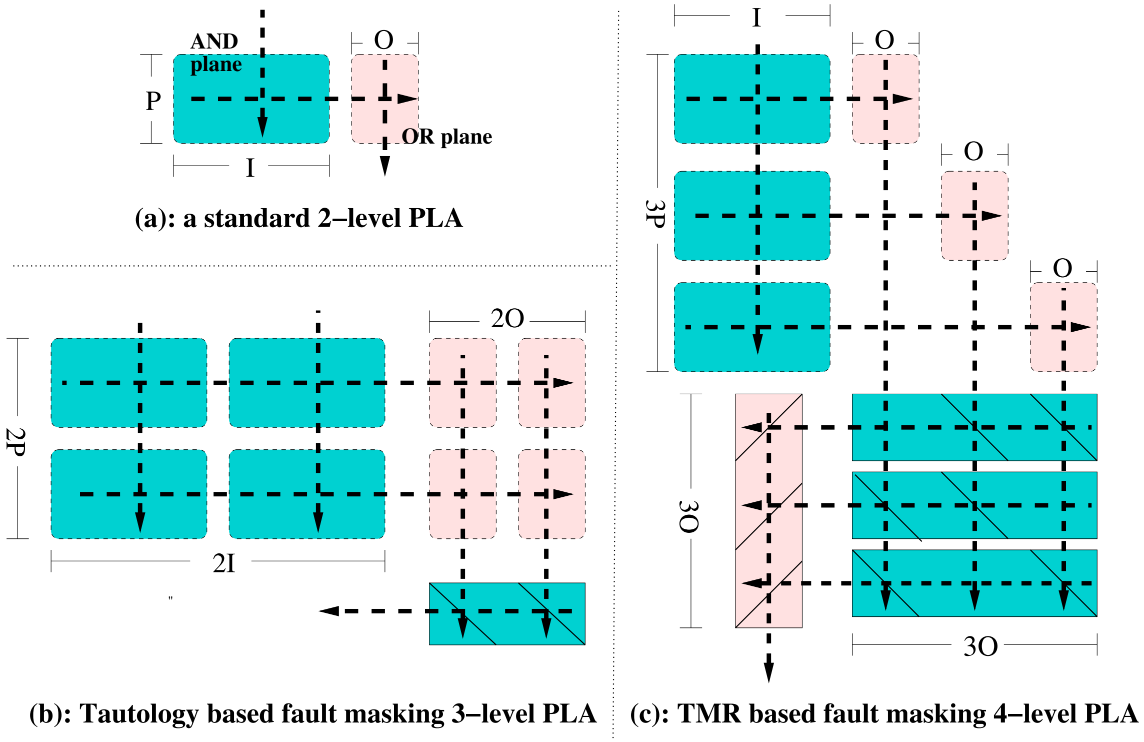
In figure 3.3(a), the original 2-level PLA is shown with one $P \times I$ AND plane and one $P \times O$ OR plane. Figure 3.3(b) illustrates the architecture of the proposed fault masking scheme, in a 3-level PLA implementation. The architecture consists of a $2P \times 2I$ AND plane and a $2P \times 2O$ OR plane. An extra logic level of an AND plane is added with every logic output wire ANDed with its duplicate, thus using two extra devices for each logic output wire. Therefore, the extra level of AND logic uses an additional $2O$ number of devices and O wires. Overall, the tautology based scheme utilizes $4D_A + 4D_O + 2O$ devices with $2I + 2P + 3O$ wires.

Figure 3.3(c) illustrates a TMR fault masking approach in a PLA implementation. In a TMR approach, both the AND plane and the OR plane need three identical copies, thus requiring $3D_A + 3D_O$ devices. In terms of wires, a tripling in the number of product term wires and output wires is necessary; however, one copy of the input wires can be extended to cross the three AND planes placed in a column. For a TMR approach, a majority voting process is required for every final output wire. This in turn imposes two extra logic levels in a PLA structure, due to the majority voting function of the three output copies. This extra voting stage therefore necessitates an additional $9O$ devices, and $7O$ wires.

Table 3.3 lists the hardware comparison of the proposed and the TMR based fault masking schemes. From the hardware aspect, although the proposed tautology based fault masking scheme utilizes one more copy of the AND plane and the OR plane, the TMR approach necessitates significant extra overhead at the voter implementation. Furthermore, the proposed schemes display a highly regular structure, as can be seen in figure 3.3, thus making them compatible with an efficient nano PLA implementation. On the other hand, the TMR approach has a voting structure that necessitates fan-in from three independent modules. Implementing such a function with the highly regular structure of nano PLAs results in a large amount of area overhead for wiring.

The fault masking capability of a TMR implementation is limited to the three copies of AND and OR planes only, and does not naturally cover the voting hardware. The proposed scheme is capable of masking any of the 4 fault types occurring at any position in the whole architecture, since the redundancy is built in under the tautology form, and the fault masking capability covers each individual plane. Performance-wise, the proposed scheme evidently surpasses the TMR approach due to the implementation with one less logic level.

Overall, it can be concluded that fault masking in a nanoelectronic PLA can be efficiently achieved by exploiting redundancy in a tautology form. The traditional representative fault masking scheme of NMR does not necessarily make a good solution for fault tolerance in nano PLA, due to its voting overhead as well as being incompatible



(a): a standard 2–level PLA

(b): Tautology based fault masking 3–level PLA

(c): TMR based fault masking 4–level PLA

Figure 3.3: Comparison of fault masking schemes

with the regular structure of PLA based logic.

3.4 Online Repair for nano PLAs

Online reconfigurability is naturally supported in nano crossbar structures, since the two terminal molecular devices are dynamically configurable into the "on" and "off" states. Therefore, transient faults that have accidentally switched a device between its

Table 3.3: Hardware overhead summary

fault masking scheme	hardware overhead		logic levels
	device	wire	
Original	$D_A + D_O$	$I + P + O$	2
Tautology based	$4D_A + 4D_O + 2O$	$2I + 2P + 3O$	3
TMR based	$3D_A + 3D_O + 9O$	$I + 3P + 7O$	4

"on" and "off" states can be fixed through re-applying the corresponding voltage and resetting it to the correct configuration. For permanent faults, based on the regular structure of nano PLAs, spare rows and columns can be used to perform the repair, which is similar to the traditional built-in self-repair in systems of high regularities such as memories and FPGAs [23, 44, 48] Essentially, reconfiguration based online repair schemes are promising for nano PLA reliability enhancement.

An online repair scheme consists of three phases: 1) fault detection, 2) fault diagnosis, and 3) reconfiguration based repair. Efficient online fault detection approaches for PLA logic have existed for CMOS based systems. Since the reconfiguration phase relies on precise information regarding the faulty devices so as to perform online repair, the key challenge remains in the online diagnosis phase, which is responsible for identifying the faulty devices. Diagnostic resolution of faulty components is crucial to the performance and hardware overhead during the reconfiguration stage, since it determines the spare units necessary during reconfiguration, as well as the amount of repair that needs to be performed. A coarse grain diagnostic resolution leads to replacing a large number of fault-free components with spares, while a fine grain diagnostic resolution with precisely identified faulty components results in efficient utilization of spare hardware. Consequently, we focus on the particular challenge of online diagnosis and how to overcome such a challenge by exploiting the regularity of nano PLAs.

3.4.1 Online diagnosis exploiting nano PLA's regularity

Basically, online diagnosis can be described as the problem of identifying the type(s) and location(s) of the fault(s) given the fact that a fault or multiple faults have been detected. Since it is hard to probe the status of each crosspoint online, one has to infer information from the logic relationship between the functional input and the faulty output to perform diagnosis.

In a typical offline diagnosis, test vectors can be applied to the inputs while the outputs can be compared with the expected ones stored in a diagnosis dictionary. In an online diagnosis process, the vectors which stimulate the fault manifestations are the run time functional inputs, and thus not controllable. Storing all the expected outputs for every possible input is prohibitively expensive. Therefore, although plenty of research approaches exist for offline diagnosis, online diagnosis remains a new challenge.

To eliminate the ambiguity in an online diagnosis process, additional observation points need to be inserted between the AND plane and the OR plane, so as to differentiate the AND plane faults from the OR plane ones. In the two-level PLA structure, the additional observation points basically make the signals in the product term wires observable. Overall, counting in the new observation points, the information accessible in a nano PLA for online diagnosis includes:

- **Input Vector (IV)**: the inputs to the PLA AND plane.
- **Product Term Vector (PTV)**: the outputs of the AND plane; at the same time the inputs to the OR plane.
- **Output Vector (OV)**: the outputs of the PLA OR plane.

To overcome the online diagnosis challenge, we need to exploit the highly regular structure of a PLA. Based on the two-level logic function and the regularity of a PLA, the following information can be utilized to infer the fault type and location:

- the logic relationship among the observable information online, i.e., the three vectors: IV, PTV and OV;
- information obtained from the fault detection phase;

Figure 3.4 shows the information available in an online diagnosis environment for nano PLAs. For AND plane faults (G and S), IV serves as the input and PTV serves as the output; for OR plane faults (D and A), IV serves as the input and PTV serves as the output.

Since each fault in a PLA changes the output in a unidirectional manner, the location candidates can be inferred, upon fault detection, based on the plane input /

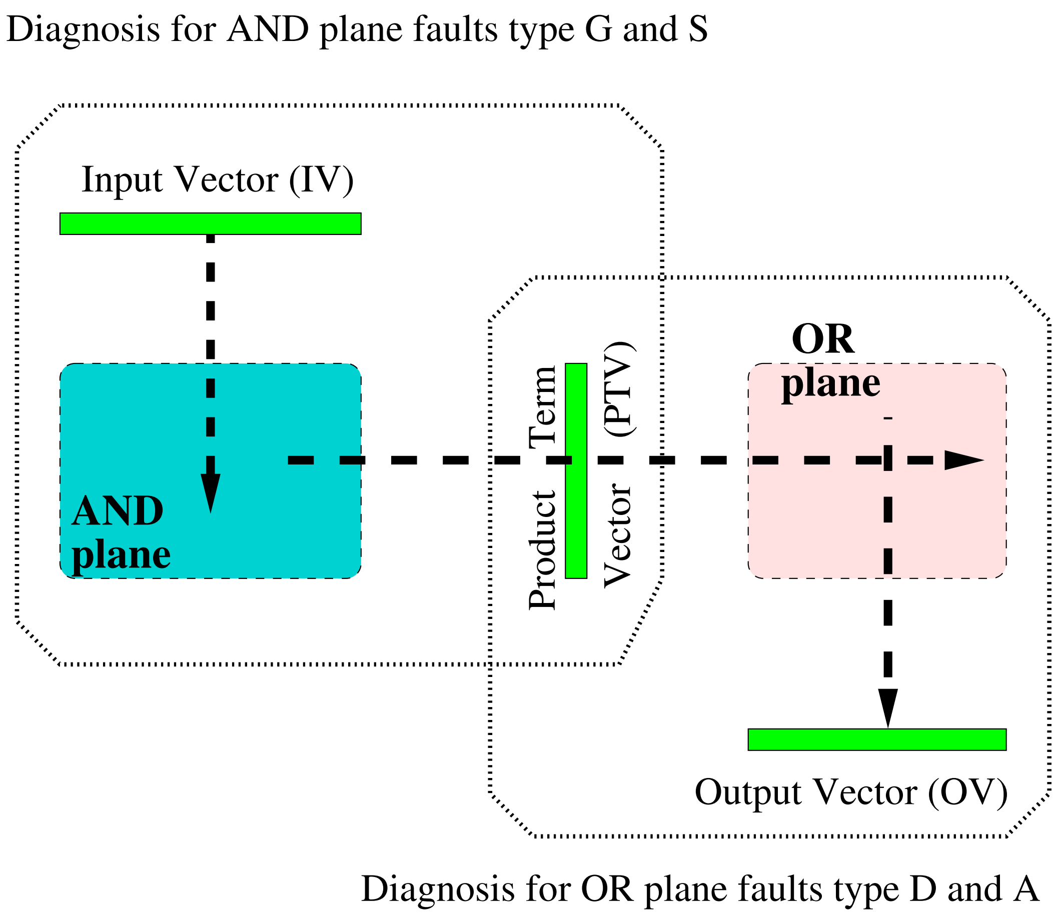

Diagnosis for AND plane faults type G and S
Input Vector (IV)
AND plane
Product Term Vector (PTV)
OR plane
Output Vector (OV)
Diagnosis for OR plane faults type D and A

Figure 3.4: Online fault diagnosis observable points

output for each fault type. For example, a D type fault (missing device in the OR plane) can result in a single bit flip from 1 to 0 in the Output Vector (OV). This is essentially because the missing device fails to *pull up* the output line, when the product term wire carries a value 1.

Due to such a unidirectional change from 1 to 0 caused by a D type fault, the zero bits in OV provide the column candidate set of the fault occurrence. The row candidate set on the other hand, needs to be obtained from PTV, which is the input of the OR plane. Since a D type fault occurs in the OR plane, we only need to focus on the bits with value 1 in PTV, because the product term wires with value 0 would never have stimulated a D type fault. According to the above information from the I/O vectors of the OR plane, the candidate rows and columns of a D type fault can be identified.

A similar analysis can be performed on all the four fault types of the two planes. From the input side of a plane, the controlling values determine the candidate set. This can be easily understood from the perspective of digital testing, where a fault in an AND (OR) gate can only be stimulated when its associated input is set to the controlling value (0 for an AND gate, 1 for an OR gate) and all the other inputs are set to the non-controlling value (1 for an AND gate, 0 for an OR gate). From the output side of a plane, the candidate set is determined by the direction of the fault effect, regardless of whether the plane is AND or OR. Specifically, for a fault that flips the output from 0 to 1 (G and A types), the "1" positions in the output side of the plane constitute the candidate set; on the other hand, for a fault that flips the output from 1 to 0 (S and D types), the "0" positions in the output side of the plane form the candidate set.

Table 3.4 lists the necessary conditions, from the input and output perspective of a plane, for the manifestation of each type of fault at a specific location. These two conditions are marked as C_1 and C_2:

C_1: for the G, S types of faults occurring at location $[i][p]$ in the AND plane, the IV must have a zero at the i'th bit, at the input to the AND plane, to stimulate the fault; similarly, for the D, A types of faults at location $[p][o]$ in the OR plane, the $PTV[p]$ bit as an input to the OR plane must be 1 to stimulate the fault.

C_2: the S, D faults unidirectionally flip the outputs from 1 to 0, forcing the corresponding bit at the output of the plane to be 0; similarly, for the G, A faults, which flip the outputs unidirectionally from 0 to 1, the corresponding bit at the output of the plane must be 1.

The necessary conditions in table 3.4 identify a set of candidate locations for a detected fault based on the input and output values of a plane. To further pinpoint the exact location of a fault, one needs to refer to a "dictionary". Unlike a traditional offline diagnosis dictionary that contains test vectors and expected outputs, the dictionary for online diagnosis has to be small, yet containing all the information needed for precise

Table 3.4: Online diagnosis conditions for the 4 fault models

fault type	fault location	occurring plane	C_1: input to the plane	fault effect	C_2: output of the plane
G	$[i][p]$	AND	$IV[i] = 0$	$0 \rightarrow 1$	$PTV[p] = 1$
S	$[i][p]$	AND	$IV[i] = 0$	$1 \rightarrow 0$	$PTV[p] = 0$
D	$[p][o]$	OR	$PTV[p] = 1$	$1 \rightarrow 0$	$OV[o] = 0$
A	$[p][o]$	OR	$PTV[p] = 1$	$0 \rightarrow 1$	$OV[o] = 1$

diagnosis resolution, nonetheless. Due to the regularity in the PLA structure, the correct configuration of the PLA planes suffices to serve for the purposes of a diagnosis dictionary. The hardware overhead of such a dictionary is comparable to the original PLA size, since the configuration of each crosspoint is stored using one bit.

3.5 Nano PLA with fault masking and diagnosis capabilities

Generally speaking, a fault masking approach can tolerate a certain level of fault occurrences by masking the faulty result from affecting the output. Such a fault masking capability, determined by the amount of redundancy devoted for reliability purposes, has certain limitations. For instance, TMR is capable of masking any single fault occurrence. When two out of the three identical computations are faulty, the correct result cannot be identified. Furthermore, when the two faulty results conform, a faulty output is taken instead as the correct output according to the majority vote.

Consequently, a system guarded by a fault masking approach is not inherently *fault secure*. The *fault-secureness* property of a system guarantees that, once a fault occurs, it can always be identified. Therefore, a faulty output never escapes detection and is never regarded as the correct output [51].

The lack of the fault-secureness property in fault masking approaches has not been addressed previously, since fault occurrence is very low and unlikely to exceed the fault masking threshold. When the fault rate is high, multiple faults beyond the threshold of the fault masking approach might occur at the same time. Consequently, the fault-secureness property needs to be ensured besides the fault masking capability. For nano PLAs, when multiple faults occur and escape the fault masking scheme, it is important to: 1) distinguish such a case from a fault-free scenario, 2) identify the types and locations of the faults, and 3) reconfigure the faulty devices to guarantee the correctness of the output, and the reliability of the system. In other words, an integrated fault tolerance scheme combining online repair and fault masking approaches is needed to enhance the reliability of nano PLAs.

The key challenge in adding online repair capability to the fault masking scheme is the extra hardware requirement. Such an approach will not be affordable if the hardware requirement is costly on top of the redundancy necessitated by the fault masking approach. Whether the overall fault tolerance works essentially hinges on the exploration of an efficient approach that enables the existing redundancy in the fault masking scheme to be exploited for further diagnosis purposes.

The proposed fault masking scheme utilizes redundant copies of the AND and OR planes to implement a Boolean tautology. In fact, the redundancy existing within the tautology based scheme can be exploited to eliminate the necessity for a dictionary in online diagnosis. In other words, precise online diagnosis resolution can be achieved by exploiting the redundant planes in the fault masking scheme, thus eliminating the necessity to store the correct configuration using extra memory elements. In this section, we put the online diagnosis approach into the fault tolerance framework in conjunction with the proposed fault masking scheme, and examine how the overall fault tolerance strategy can work together for nano PLAs. We show that such an online repair based strategy can be implemented with insignificant extra hardware on top of the fault masking scheme.

3.5.1 Exploiting existing redundancy in fault masking

The idea of implementing an efficient fault tolerance approach is to get double mileage out of the redundancy inherent in the fault masking scheme for diagnosis purposes, instead of spending extra hardware on the configuration dictionary. To carry out such a scheme, we need to carefully examine which part of the redundancy in the fault masking scheme is furthermore exploitable, and how much the redundancy contributes to the diagnosis resolution. More importantly, we need to answer the question of how fault diagnosis can be superimposed on top of a fault masking scheme, since the underlying mechanisms for these two fault tolerance schemes are different: the former needs to be built upon the manifestation of a fault, in order to identify the fault location, while the latter focuses on making the fault effect invisible.

Specifically, for an integrated fault tolerance scheme combining fault masking and online repair, there are two considerations:

- Diagnosis of a fault when it is masked successfully by the fault masking approach.

  When masked successfully, the fault effect does not manifest at the output; thus the final function result remains correct. In such a case, if the fault location can be identified, then online repair can be performed ahead of time without performance loss.

- Diagnosis of faults when the fault masking approach fails.

  When the faults manifest and result in an erroneous output, the fault-secureness attribute of the system needs to be examined carefully. Basically, the faulty result needs to be identified, so as not to be taken as a correct one. In such a case, it is crucial to identify the locations of the faults, since online repair has to be performed to recover the fault masking capability of the system.

Figure 3.5 illustrates the observable points needed for diagnosis purposes in the nano PLA with the proposed fault masking approach. As in the previous example, the four copies of the AND plane are labeled 1 to 4, while the four copies of the OR plane are labeled 5 to 8. Particularly, for the two PTV vectors and two OV vectors, we label them correspondingly as follows: PTV-12 consists of product term wires crossing planes

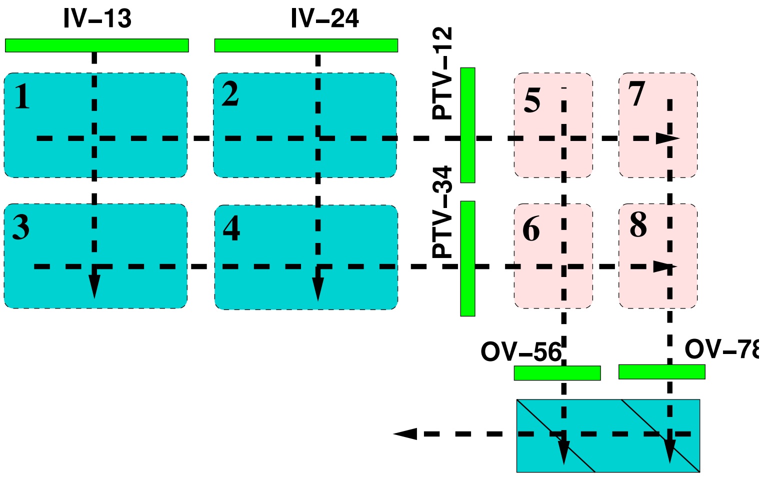


Figure 3.5: Observable points for diagnosis purposes in the fault masking framework

1 and 2, PTV-34 consists of product term wires crossing planes 3 and 4; OV-56 consists of output wires covering planes 5 and 6, and OV-78 consists of output wires covering planes 7 and 8.

From the fault masking approach, it can be easily seen that the two IV vectors are copies of the same input signals. Under the fault-free situation, PTV-12 and PTV-34 represent the same PTV vector, while OV-56 and OV-78 represent the same output vector. Such existing redundancy in the fault masking approach can be exploited for online diagnosis. Specifically, by comparing the PTV pair, the precise row of an AND plane fault location can be identified. By comparing the OV pair, we can obtain the precise column of an OR plane fault location. Overall, the existent redundancy in the fault masking scheme is used to enhance diagnostic resolution, thus eliminating the necessity for a dictionary which stores all the correct configuration information.

Apparently, the comparison of the observation point vectors yields the possibility to detect faults and pinpoint fault locations. Specifically, we want to examine whether the following requirements can be achieved by relying solely on the information from the comparisons:

- fault-secureness of the system, i.e., detecting the situation when fault masking fails;

- full diagnosis resolution under the two situations: 1) fault successfully masked, and 2) unmaskable faults.

Table 3.5: Results of vector comparison for successfully masked faults

fault	type	fault effect example	PTV: 12=34?	OV: 56=78?

AND ↓	G	PTV12=bab;
		PTV34=$abab$
			=	=

AND ↑	S	PTV12=$ababc$;
		PTV34=$abab$
			$\neq$	=

OR ↓	D	OV56=$ab + cd + ab$;
		OV78=$ab + cd + ab + cd$
			=	=

OR ↑	A	OV56=$ab + cd + ab + cd + e$;
		OV78=$ab + cd + ab + cd$
			=	$\neq$

Table 3.5 lists the results of PTV pair comparison and OV pair comparison for single fault occurrences that can be successfully masked. Since the effect of a successfully masked G type fault never reaches the PTV, both the PTV comparison and the OV comparison conform, as is displayed in the first row of table 3.5. On the other hand, although successfully masked in the final output, the fault effect of an S type fault manifests in one of the PTVs. Consequently, PTV12 and PTV34 do not conform, as it can be observed in the second row of table 3.5.

Table 3.6: Results of vector comparison for faults that escape from being masked

fault	type	fault effect example	PTV: 12=34?	OV: 56=78?
AND ↓↓	G+G	PTV12=b; PTV34=ab	$\neq$	=
AND ↑↑	S+S	PTV12=abc; PTV34=abd	$\neq$	=
		PTV12=abc; PTV34=abc	$=$	=

AND ↑ OR ↓	S+D	PTV12=abe; PTV34=ab;
		OV56=$abe + cd$; OV78=cd
			$\neq$	$\neq$

OR ↓↓	D+D	OV56=cd; OV78=$ab + cd$	=	$\neq$

OR ↑↑	A+A	OV56=$ab + cd + e$;
		OV78=$ab + cd + f$
			=	$\neq$

		OV56=$ab + cd + e$;
		OV78=$ab + cd + e$
			=	=

Table 3.6 lists the results of vector comparisons for the unmaskable double-fault

occurrences, which are discussed in the fault masking section. For instance, for a double fault occurrence, both of G type, the scheme fails to mask the fault effect only when two G type faults occur at the same location of both planes 1 and 2 (or 3 and 4). In such a case, the fault effect manifests in PTV-12 (or PTV-34) and cannot be masked in the OR planes either. As a result, PTV-12 and PTV-34 do not conform, yet the output vectors OV-56 and OV-78 conform on an identical erroneous result.

In combination, tables 3.5 and 3.6 illustrate the information provided by the comparison results under the maskable and unmaskable fault occurrences. According to table 3.5, the comparisons of the PTV and the OV pairs can detect S and A type of faults, even when the fault effect is successfully masked. For maskable G and D type of faults, the comparisons in the PTV and the OV do not identify fault occurrences. According to table 3.6, the comparisons of the PTV and the OV pairs can detect 5 out of 6 cases when the fault masking scheme fails. Under the rare cases of fault masking failure under S and A types, when extra devices appear at exactly the same locations of the two redundant planes, both comparisons yield conforming results.

Another observation from tables 3.5 and 3.6 is the existence of aliasing when the comparison results are used as signatures for the various types of fault occurrences. For instance, the maskable S type fault (the second row in table 3.5) and the unmaskable G type fault (the first row in table 3.6) have the same signature of comparison results: (PTV-12 $\neq$ PTV-34, OV-56 = OV-78). Due to such ambiguity, two cases cannot be distinguished: 1) a successfully masked S type fault with the correct final result, and 2) unmaskable double faults of G type with the incorrect final result.

In summary, to answer the questions raised earlier in this subsection, the comparisons of the PTV pair and OV pair provide partial resolution, yet they fall short of providing complete information to satisfy the requirements:

- The information provided by the comparison of observable point vectors can help partially the detection of faults. For maskable faults, S and A types of faults can be detected, while G and D types cannot. For unmaskable faults, all the fault types can be detected except for a subgroup of S type and A type of faults.

- The four distinct results of PTV and OV comparisons cannot fully distinguish the 12 faulty scenarios shown in tables 3.5 and 3.6 and a fault free case. Therefore, full diagnostic resolution is not achieved due to such ambiguity.

3.5.2 Achieving a fault secure system with full diagnosis resolution

As is discussed in the previous subsection, depending solely on internal redundancy does not make a system fault secure. Neither does it provide full diagnostic resolution. Consequently, to make the nano PLA system fault secure under the unmaskable faults, extra redundancy needs to be introduced. Basically, an extra fault detection process is necessary for checking the final output from the PLA. A standard approach with insignificant hardware overhead is the use of Berger code for the number of 1's in the output [1]. By doing so, any successfully masked fault will not trigger detection, while an erroneous final result caused by unmaskable faults will be detected due to the flipped bits in the output.

Since the effect of unmaskable faults changes the output unidirectionally, the number of 1's in the final output does not conform with the encoded number. Consequently, the faults are caught by the Berger code based approach. For the faults that are successfully masked, the number of 1's in the final output remains correct. For these cases, the Berger code based detection indicates an error-free output. By introducing the Berger code based detection approach, faults that are successfully masked can be differentiated from the ones that escape the masking approach.

With Berger code, information of whether the number of 1's in the output increases or decreases is available. This extra information essentially opens up the opportunity to achieve full diagnosis resolution, which could not be achieved by relying on the internal redundancy alone, due to the ambiguity.

Table 3.7 illustrates the diagnosis signature with Berger code detection information for both the maskable and unmaskable faults. The column of “Berger code”

Table 3.7: Results of vector comparison and Berger code indication for faults that escape from being masked

fault	type	fault effect example	PTV: 12=34?	OV: 56=78?	Berger Code
AND ↓	G	PTV12=bab; PTV34=$abab$	=	=	$\surd$
AND ↑	S	PTV12=$ababc$; PTV34=$abab$	$\neq$	=	$\surd$
OR ↓	D	OV56=$ab + cd + ab$; OV78=$ab + cd + ab + cd$	=	=	$\surd$
OR ↑	A	OV56=$ab + cd + ab + cd + e$; OV78=$ab + cd + ab + cd$	=	$\neq$	$\surd$
AND ↓↓	G+G	PTV12=b; PTV34=ab	$\neq$	=	+
AND ↑↑	S+S	PTV12=abc; PTV34=abd	$\neq$	=	-
		PTV12=abc; PTV34=abc	$=$	=	-
AND ↑ OR ↓	S+D	PTV12=abe; PTV34=ab; OV56=$abe + cd$; OV78=cd	$\neq$	$\neq$	-
OR ↓↓	D	OV56=cd; OV78=$ab + cd$	=	$\neq$	-
OR ↑↑	A	OV56=$ab + cd + e$; OV78=$ab + cd + f$	=	$\neq$	+
		OV56=$ab + cd + e$; OV78=$ab + cd + e$	=	=	+

indicates whether the number of 1's in the output has been increased (+), decreased (-), or is correct ($\surd$).

As is shown in table 3.7, with the additional information from the Berger code detection approach, the diagnosis signature for each type of fault occurrence consists of a three tuple (PTV-12/PTV-34, OV-56/OV-78, Berger code indicator), The Berger code indicator alone is sufficient to distinguish the maskable faults from the unmaskable one.

Among the maskable faults, G and D fault types cannot be distinguished from a fault-free scenario, due to the identical signature. For the unmaskable faults however, each has a unique diagnosis signature that is different from all the maskable fault types. Therefore, every unmaskable fault type can be successfully identified without ambiguity.

Consequently, with the extra information from the Berger code based detection, both of the following requirements are satisfied:

- Unmaskable faults can be detected, thus making the system fault secure.

- Full diagnosis resolution is achieved for the unmaskable fault types.

3.5.3 Diagnosis resolution on fault location

The previous subsection provides a solution to distinguish various fault occurrence types from each other. In this subsection, we give an analysis on the diagnosis resolution for the detailed location within each plan.

Basically, the input and output vectors of a plane provide a set of candidate positions for a fault occurrence. The comparison between the redundant vector pairs pinpoints the precise location on one dimension of the candidate positions: for the AND plane, the comparison between PTV-34 and PTV-12 pinpoints the precise row of the fault location according to the location of the nonconforming bit; for the OR plane, the disagreeing bit between OV-56 and OV-78 identifies the precise column of the fault location.

We provide a discussion using the AND plane as an example, without loss of generality. For the AND plane faults, the candidate columns have to be obtained through the controlling values in the input vector. Specifically, different fault types render different information in handling the candidate columns:

- A device missing fault of the AND plane changes the output bits from 0 to 1, indicating that none of the controlling values (0) from the input is transferred to the output. Consequently, the candidate column set consisting of all the 0 input bits delivers the following information:

  1. under the device missing fault, all the devices at the positions indicated by the 0 input bits are “off”;
  2. under the fault-free condition, one or more of these devices should be “on”.

  Based upon such information, the repair process needs to access the correct the configurations of the candidate positions. Since the faulty plane(s) among the four copies are identified, a comparison between the faulty ones and the correct ones can be performed to send the reconfiguration signals to the faulty position, according to the correct configurations.

- For the extra device fault of the AND plane, the output changes from 1 to 0, indicating that one (or more) of the controlling values (0) from the input is erroneously transferred to the output. Consequently, the candidate column set consisting of all the 0 input bits delivers the following information:

  1. under the fault-free condition, all the devices at the positions indicated by the 0 input bits should be "off";
  2. under the extra device fault, one (or more) of the devices in the candidate set is erroneously set to "on".

  Based upon such information, the correct configuration of the candidate set is already available. Consequently, the repair process can simply send a reconfiguration signal to all the candidate positions, switching all these devices to the correct state of "off".

3.6 Conclusions

Aggressive fault tolerance approaches are necessitated to overcome the severe unreliability challenge in nano PLAs. Fault masking and online repair based approaches exhibit complementary advantages when dealing with the unreliability challenge. The two-tier approach presented in this chapter addresses the fault tolerance issue in nano PLAs by integrating fault masking and online repair schemes efficiently.

We develop a fault masking scheme particularly suitable for the nano PLAs based on logic tautologies. Such an approach eliminates the necessity for the voting process in a generic NMR based fault masking approach, which turns out to be highly costly in the nano PLA's regular structure. Since fault masking based schemes are intrinsically limited to a certain threshold of fault occurrences, we propose to apply online repair based approaches to enhance the reliability of nano PLAs, when the fault masking approach fails under multiple faults. We identify the crucial challenge of fault diagnosis in an online repair based approach, and propose a solution by exploiting 1) the regular structure of PLA, and 2) the existent redundancy employed in the fault masking approach, to overcome the online diagnosis challenge for nano PLAs. Furthermore, we focus on the fault secure property of the system under the integrated fault tolerance approach, so that the unmaskable faults can be identified and diagnosed for repair.

Overall, the power of the fault tolerance schemes developed in this chapter comes from exploiting: 1) the highly regular structure of nano PLAs, 2) the reconfiguration capability of nano crossbars with molecular devices, and 3) the characteristics of two-level logic functions in a PLA architecture. Most importantly, although redundancy is required in implementing any fault tolerance approaches, the hardware redundancy in these two tiers of fault tolerance approaches are overlapped. The exploitation of existent hardware redundancy makes the overall approach highly efficient yet capable of delivering high fault tolerance capabilities.

Chapter 4

Reliable Parallel Adders in Nano

Future nanoelectronics based systems will enjoy significant increases in device density and performance. Consequently, high performance arithmetic component design that utilizes hardware to maximize parallelism can be foreseen to dominate future nanoelectronic systems. Observing that hardware redundancy is typically employed to achieve high performance in arithmetic component designs, we propose to exploit this inherent redundancy for fault tolerance purposes as well. Thereby, the reliability challenge can be overcome frugally by exploiting the existing redundancy for performance purposes.

In this chapter, we examine the reliability of adders, which are not only the most fundamental component, but also the basic building blocks for arithmetic systems. Particularly, we focus on high performance parallel adders, which are promising for nanoelectronic systems. The high performance adders typically employ extra hardware to calculate carry bits in parallel, so as to eliminate the long delay caused by the serial carry propagation. Such redundancy embedded for performance consideration opens up the possibility of its exploitation for fault tolerance approaches as well, including fault detection, fault masking and reconfiguration based online repair. We first propose the general principle of exploiting redundancy for the dual purposes of performance and reliability in nanoelectronic systems, and then provide the detailed techniques developed for parallel adders. In the end, we provide a comprehensive analysis to assess the proposed technique.

By exploiting the existing redundancy in the high performance arithmetic components, various fault tolerance approaches can be supported very efficiently. The internal redundancy provides direct support for online fault detection, and can be extended to support fault masking as well. Furthermore, since the internal redundancy is embedded with the particular regular structure of parallel adders, online diagnosis schemes can be developed. With the support of high resolution online diagnosis, the reconfigurability of nanoelectronics can be fully exploited for the highly flexible online repair approaches. In contrast to traditional fault tolerance techniques that involve tremendous overhead, the proposed approach opens up a new way of supporting a genre of efficient fault tolerance techniques, particularly targeting at high performance components under the nanoelectronic environment.

4.1 Motivation

Traditionally, the main design optimization tradeoff in electronic system construction has always been between hardware cost and performance. There typically exist multiple tradeoff points in a design optimization space, and the ones with high performance always come at the cost of extra hardware resources. Essentially, the basic principle in achieving a high performance component is through the use of hardware redundancy. These additional hardware resources are used either to construct parallel computation, provide pre-computation, or even perform predictions. The ultimate goal is to shorten the critical path.

The design of adders, the basic arithmetic component, is a classical example of using hardware redundancy to achieve better performance. Serial adders such as the carry ripple adder cost the least amount of hardware, yet their performance suffers from the long delay of carry propagation. A carry select adder utilizes two identical copies of hardware to pre-calculate the higher bits of result, under both cases of one and zero as carry-in. This redundancy in hardware breaks the long carry propagation path and improves performance. Parallel adders, such as the carry lookahead adder, set aside extra hardware to compute carry signals for higher bits in parallel, thus once again avoiding the long carry propagation process through hardware redundancy. In fact, such a principle exists in general for components of various functions and across multiple design hierarchical levels as well. The design of tree multipliers, array multipliers and the use of a branch predictor in a CPU all reflect the same principle of using hardware redundancy for performance.

The fundamental changes at the device level necessitate reconsideration of system design and construction. With the abundant hardware and high device density offered in the nanoelectronic environment, the main concern of hardware and area cost is alleviated. It can be foreseen that the principle of applying hardware redundancy to achieve high performance will be widely applied for the design of nanoelectronic systems. Despite the hardware cost, high performance arithmetic components will become significantly advantageous over their slow peers at the other end of the design optimization tradeoff space.

Typically, *redundant* components in a system mean resources that are unnecessarily spent and can be pruned for cost-efficiency purposes. One needs to examine carefully the precise meaning of *hardware redundancy* employed in high performance arithmetic components, specifically from two different perspectives:

- **Performance wise:** the extra hardware is definitely necessitated.
- **Computation wise:** redundancy exists due to the extra hardware introduced. Therefore, certain computation performed in the system is repeated, or not utilized.

In the example of a high performance adder, extra hardware is employed to generate carry bits in parallel. These extra hardware resources are necessitated to pre-compute carry signals for most significant bits. However, to view it from the computational perspective, redundancy is embedded in that certain computation is repeated, or unused.

For instance, in a carry lookahead adder, since carry signals at position i and $i-1$ are computed independently in parallel, the computation at position i employs a redundant copy of the computation performed at position $i-1$, so that the carry at position i does not need to wait for the serial propagation from position $i-1$. In the example of a carry select adder, one of the two predictive computations for the high bits carry signals is deemed never used. In summary, the *hardware redundancy* in a high performance arithmetic component exists in the sense of computational redundancy, yet is necessitated for good reasons - to ensure parallelism and performance.

To overcome the reliability challenge in nanoelectronic systems, aggressive fault tolerance approaches need to be employed. It is well known that any fault tolerance technique relies on certain forms of computational redundancy to guarantee correctness. In general, duplication of hardware and recomputation are used to form the computational redundancy necessitated for fault tolerance. It has been shown that, under the high fault rates of nanoelectronic environment, tremendous redundancy is required [62]. Since *redundancy* required for fault tolerance purposes is from the computational perspective, the hardware redundancy in high performance arithmetic components fulfills such a requirement. Consequently, the extant hardware redundancy can be exploited for reliability purposes, and opens up the opportunity to achieve fault tolerance efficiently.

Specifically, to examine the possibility and capability of exploiting the existing redundancy in high performance arithmetic components for reliability purposes, one needs to answer a number of questions from the following perspectives.

- **Arithmetic component applicability:** Which components can this idea be applied to? How much will the nanoelectronic systems benefit from this approach based on the applicable components? How does it work? How does functionality and internal structure of the component influence the approach? Does it raise interconnect issues for the nanoelectronic systems?

- **Fault tolerance strategies:** There exist various genres of fault tolerance approaches. For instance, fault masking, fault detection, diagnosis and reconfiguration based repair enhance reliability from multiple angles and at multiple levels. If the existing redundancy for performance can be exploited for reliability purposes, which fault tolerance approaches does it support, and how well does it support them?

- **Evaluation and analysis:** A thorough analysis needs to be made to evaluate the capability of the proposed approach. Specifically:
  - What is the overhead in terms of hardware and performance required by the approach, and how do they compare to the generally applicable classic fault tolerance techniques?
  - From the component perspective, what is the fault coverage? Can the entire component be covered by exploiting the existing redundancy, or, does additional redundancy need to be embedded to ensure coverage of certain parts?
  - From the fault tolerance perspective, what is the capability of the proposed approach, in terms of fault detection, masking capability, diagnosis resolution, and repair overhead?

4.2 Hardware redundancy in high performance parallel adders

The adders constitute the most fundamental arithmetic building block, and various adder designs provide multiple tradeoff points between the optimization criteria of area and delay. In the particular environment of nanoelectronics, the hardware abundance and massive parallelism supported by nano devices diminishes the relative importance of the area constraint. As a result, parallel adders prevail over serial adders by their significant advantage of performance. Naturally, a high performance parallel adder serves as the starting point for us to examine the approach of exploiting redundancy for both performance and reliability purposes, under the nanoelectronic environment.

Specifically, for adders, the critical path for performance is the carry propagation. In high performance parallel adders, the hardware redundancy is devoted to calculate carry bits in advance. Carry propagation and calculation therefore is the crucial point of how hardware redundancy is organized to enhance performance through parallelism. We therefore focus on the extant hardware redundancy for carry generation in parallel adders, to exploit possible resources for fault tolerance purposes.

We examine first the class of Carry Lookahead Adders (CLA), which constitutes a commonly used implementation of parallel adders. We then extend the discussion to more general parallel adders - parallel prefix adders (PPA). We focus on three representative implementations: Kogge-Stone PPA, Brent-Kung PPA, and a hybrid implementation.

4.2.1 Redundant carry generation in CLA

Introduction of CLA

In order to avoid the delay of the rippling carries, a CLA computes block-level generate (*g*) and propagate (*p*) signals so as to calculating carry-out bits in parallel [63]. Since the prediction hardware of carries at the most significant bits becomes exceedingly expensive when the width of a CLA is large, the CLA is typically constructed in multiple hierarchy levels. A CLA consists of multiple basic building blocks called the lookahead carry generators (LCG). Each LCG contains a *g, p* generation block and a carry generation block. The multiple LCGs in a CLA are organized hierarchically and are responsible for generating the carry signal for every bit.

Figure 4.1 shows a 64-bit hierarchical CLA composed of 3 levels. The *g, p* signals are generated hierarchically by the LCGs: initially, at the lowest level, the *g, p* signals of each bit are calculated; then, these one-bit *g, p* signals are used to generate the 4-bit block *g, p* signals at the second level; similarly, the 16-bit block *g, p* signals are

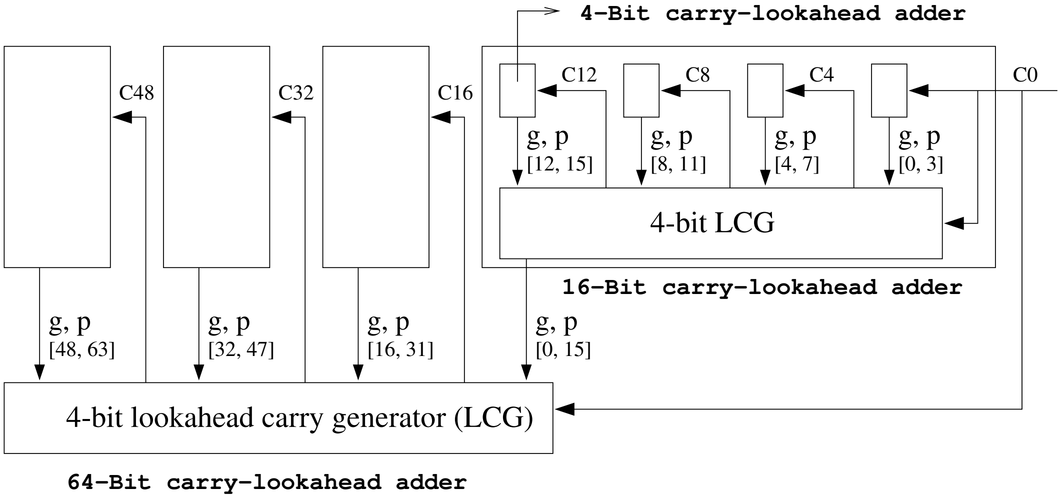

4-Bit carry-lookahead adder
C48
C32
C16
C12
C8
C4
C0
g, p
[12, 15]
g, p
[8, 11]
g, p
[4, 7]
g, p
[0, 3]
4-bit LCG
16-Bit carry-lookahead adder
g, p
[48, 63]
g, p
[32, 47]
g, p
[16, 31]
g, p
[0, 15]
4-bit lookahead carry generator (LCG)
64-Bit carry-lookahead adder

Figure 4.1: A 64-bit hierarchical CLA

generated using the 4-bit block *g, p* signals at the highest level.

The carry bits are also calculated in 3 stages. Initially, the highest level LCG uses carry-in $c[0]$ and the four 16-bit block *g, p* signals to generate the carry-in bits $c[16], c[32]$, and $c[48]$, for the second level LCGs. Then these carry-in bits and $c[0]$ are used by the second level LCGs to further generate the twelve carry-in bits for the next level LCGs. At the third stage, the sixteen LCGs at the lowest level produce the remaining forty eight carry-in signals, thus completing the carry generation for all the bits.

Redundant carry generation in CLA

As is shown in Figure 4.2, a 4-bit LCG takes as inputs the carry-in of the least significant bit and the *g,p* signals generated from the 4 lower-level LCG blocks. The outputs of an LCG include the block *g,p* signals for the current level and 3 carry-in bits for the lower-level LCG blocks.

In a hierarchical CLA, the carry-in to an LCG block is generated by the higher level LCG for performance purposes. We can observe that the same carry-in signal can also be generated through an alternative path at the same level. Consider the 4-

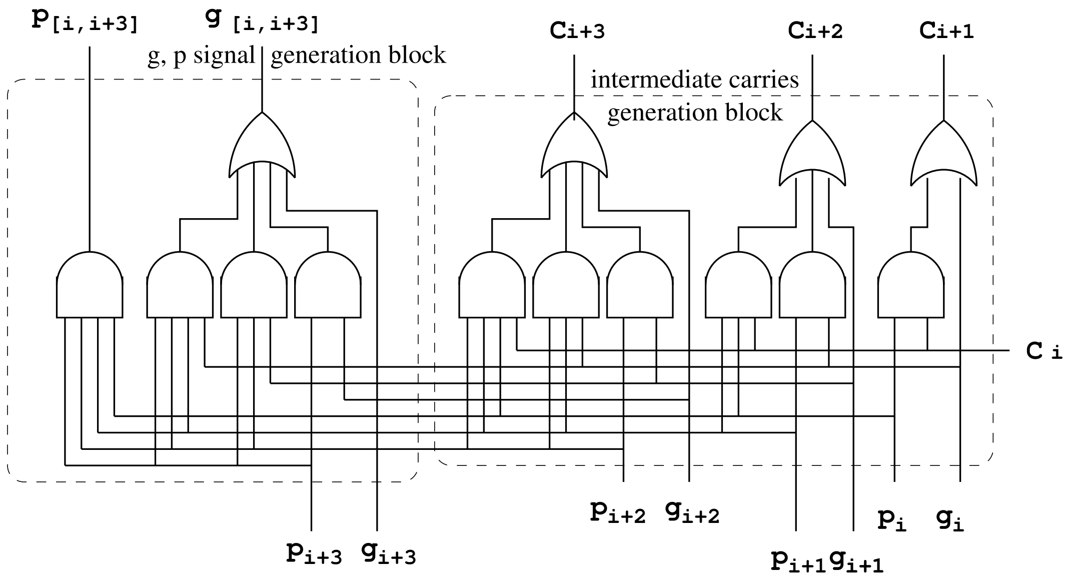
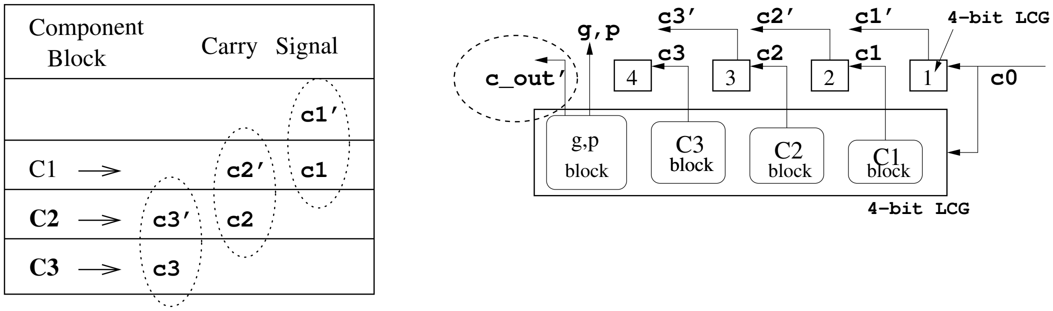


Figure 4.2: A 4-bit Lookahead Carry Generator (LCG) circuit

bit LCG block shown in figure 4.3. The three carry generation blocks ($C3, C2$, and $C1$) independently generate the carry-in bits ($c3, c2$, and $c1$) for the lower level LCGs (numbered 4, 3 and 2) respectively. Alternatively, a redundant copy of the same carry-in bits (indicated as $c3'$, $c2'$, and $c1'$), can be generated in the low level LCGs, based on the signals $c3$, $c2$ and $c1$. Take $c3'$ for example; treating $c2$ as an available carry-in, $c3'$ can be generated as the carry-out signal of LCG block 3, with the *g, p* block signal available in LCG 3, based on the following equation:

$$c\_out = g + p \cdot c\_in$$



Component Block	Carry Signal		
			c1′
C1 —>		c2′	c1
C2 —>	c3′	c2	
C3 —>	c3		

Figure 4.3: Carry generation with internal redundancy

Essentially, a hierarchical CLA enables fast generation of carry-in signals in logarithmic time, since each carry-in signal is provided by the LCG block of a higher level. This opens up the opportunity to generate a redundant copy for each carry signal, by treating the carry-in signal of each LCG block as the carry-out of the adjacent LCG block and calculating it through the alternative path in the lower level LCG block.

In a 64-bit CLA, the single highest level LCG generates $c[16], c[32]$ and $c[48]$. Together with $c[0]$, these signals serve as the carry-in signals for the four second level LCGs. Alternatively, the redundant signals $c[16]', c[32]'$ and $c[48]'$ can be generated at the second level, using $c[0], c[16], c[32]$ and the *g, p* block signals generated by the second level LCGs. Specifically:

$$c[16]' = g[0,15] + p[0,15] \cdot c[0]$$

$$c[32]' = g[16,31] + p[16,31] \cdot c[16]$$

$$c[48]' = g[32,47] + p[32,47] \cdot c[32]$$

In general:

$$c[16i]' = g[16i-16, 16i-1] + p[16i-16, 16i-1] \cdot c[16i-16], \;\; (i = [1,3])$$

Similarly, the twelve carry-in signals generated by the four LCG blocks at the second level can have their redundant copies generated at the third level:

$$c[4i]' = g[4i-4, 4i-1] + p[4i-4, 4i-1] \cdot c[4i-4], \;\; (i = 4j+k, \; j = [0,3], \; k = [1,3])$$

At the lowest level, the redundant copies of the remaining 48 carry signals are generated:

$$c[i]' = g[i] + p[i] \cdot c[i-1], \;\; (i = 4j+k, \; j = [0,15], \; k = [1,3])$$

Redundancy analysis for CLA

In the previous subsection we have described the mechanism of exploiting the existing *hardware redundancy* in a CLA, and how to use it to generate *redundant signals*, namely an extra copy of every carry bit. The conversion from hardware redundancy to redundant copies of the same signal is the key to enhancing reliability, since although faults occur in the computational blocks, the correctness of the computation in an online environment is all about the end results, i.e. signals. Comparing the two copies of the same signal serves as the basis for online fault detection.

However, whether the scheme of redundant signal comparison can be effective in dealing with the faults occurring in the computational hardware to a great extent relies on the amount of overlapping hardware. One needs to examine the common components involved in the computation of two copies of the same signal, since they are to be compared for fault tolerance purposes. For example, if $c2$ and $c2'$ are generated using the same carry generation block, say $C2$, then a fault in $C2$ might result in both erroneous $c2$ and $c2'$, thus providing a conforming comparison of faulty outputs. Figure 4.3 lists the hardware components involved in the generation of each signal, and it can be observed that, for every pair of the same carry signal, the two copies are generated using different carry blocks. For example, the two copies of the same carry signal, $c2$ and $c2'$, are generated using $C2$ and $C1$ respectively.

From figure 4.3, we can also see that each block actually generates two signals, one directly and one indirectly. For example, $C1$ generates $c1$ directly, and $c2'$ indirectly, essentially because $c2'$ depends on $c1$ as an input. Although overlapping hardware is utilized in generating $c1$ and $c2'$, this fault detection capability is not compromised, since the same fault is not going to affect both copies of the same signal that are to be compared. Furthermore overlapping of such a case can be exploited for online diagnosis purposes. A detailed discussion on this is provided in the next section of the paper, regarding the fault tolerance strategies supported by the redundancy.

To summarize, the exploitation of existing redundancy in a CLA for reliability purposes is based on the following observations:

- The carry bits in an adder are correlated in that a higher bit carry signal depends on the propagation of a carry signal from the lower bits.

- A CLA generates high bit carry signals with hardware redundancy, thus eliminating the necessity to wait for carry propagation from lower bits in a serial manner.

- Since a CLA does not rely on such propagation to generate carry bits, and instead calculates carry bits independently within each LCG, the correlation between adjacent carry signals becomes the exploitable redundancy existing in the CLA structure, and can be used for reliability purposes.

In the previous subsection, we have illustrated how a redundant copy of each carry signal can be generated, using the immediate adjacent carry signal (adjacent at block scale at the higher hierarchical levels). In fact, this is not the only possible path for generating a redundant copy for a carry signal. Based on the property of carry unfolding, more redundant copies of a carry signal can be generated:

$$\begin{aligned} c[i] &= g[i-1] + c[i-1] \cdot p[i-1] \\ &= g[i-1] + g[i-2] \cdot p[i-1] + c[i-2] \cdot p[i-2] \cdot p[i-1]. \end{aligned}$$

Obviously, generating the redundant copy from the immediately adjacent one costs the least in terms of both delay, additional hardware and interconnection. The cost increases gradually as the redundant copy of carry relies on the carry bits farther and farther away. However, these multiple paths of generating redundant copies of a single carry bit all utilize disjoint hardware components, thus extending the support to more powerful fault tolerance strategies that necessitate more than two redundant copies.

4.2.2 Redundancy exploitation in Parallel Prefix Adders (PPA)

Introduction to PPA

Based on a number of block *g, p* signals $(g[0,i-1], p[0,i-1])$, the carry signal of every bit $c[i]$ can be calculated directly: $c[i] = g[0,i-1] + p[0,i-1] \cdot c[0]$. This forms the basis of a genre of widely used parallel adders - Parallel Prefix Adders (PPA). For an n-bit addition, a PPA focuses on calculating all the n pairs of $(g[0,i], p[0,i])$ signals, where $i = [0, n-1]$. After all these *g, p* block signals are computed, the desired carry signal and furthermore the sum signal for every bit can be evaluated in the end.

The generation of the *g, p* block signals falls into the framework of prefix computation. Essentially, for two adjacent, or even overlapping blocks B_L and B_R, suppose their associated g(generate) and p(propagate) signal pairs are defined as (g_L, p_L) and (g_R, p_R), respectively; then the *g, p* signals for the merged block $B = B_L B_R$ can be obtained by:

$$g = g_L + g_R \cdot p_L$$

$$p = p_L \cdot p_R$$

In other words, the generation of carry in the large block takes place if 1) the left block generates a carry, or 2) the right block generates a carry and the left block propagates it. The large block will propagate a carry if both the left block and the right block propagate it.

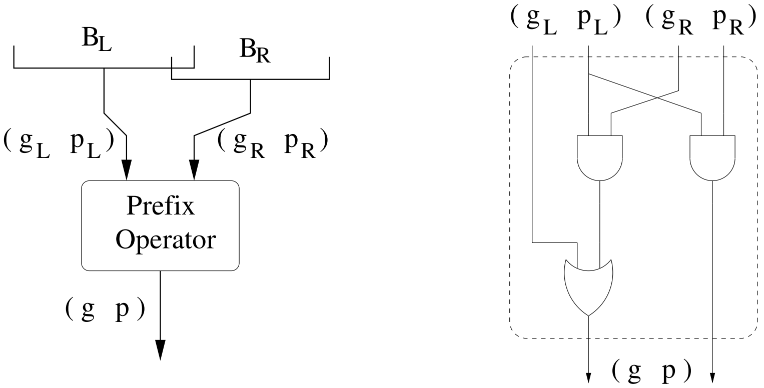
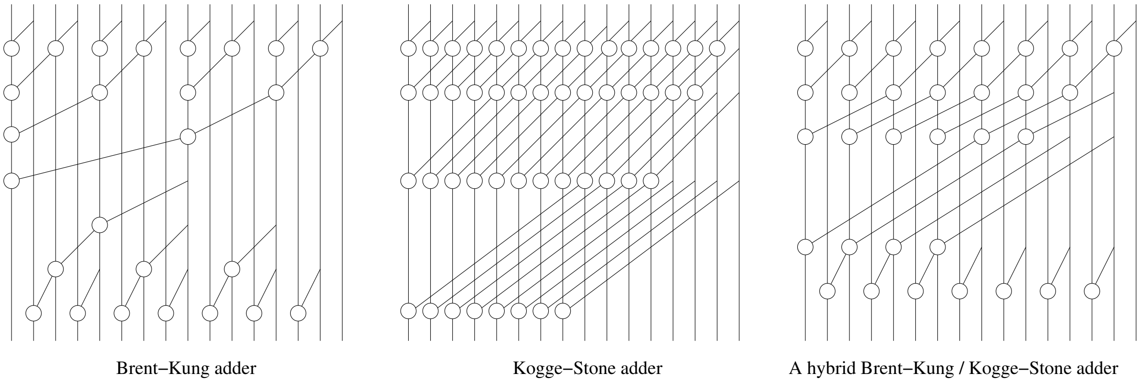


Figure 4.4: Functionality and internal structure of a prefix operator

A *prefix operator* can be defined for the generation of *g, p* pairs. Figure 4.4 illustrates the composition of *g, p* signals from two blocks B_L and B_R by a prefix operator with its gate-level implementation.

In a PPA, a network is constructed to calculate all the pairs of $(g[0,i], p[0,i])$, with each node consisting of a prefix operator. Figure 4.5 illustrates three parallel prefix networks, each implementing a 16-bit PPA. The three examples represent various tradeoff points in terms of hardware, performance, and interconnect overhead in PPA designs. For each of the PPAs shown in figure 4.5, the inputs are *g, p* signals from the 16 individual bits. Through the well-constructed prefix network, *g, p* signals of larger blocks are constructed by applying prefix operators over consecutive smaller blocks. In the end, all the pairs of $(g[0,i], p[0,i])$ are generated at the 16 output positions. Based on these outputs, all the carry and sum signals can be directly calculated.



Brent−Kung adder

Kogge−Stone adder

A hybrid Brent−Kung / Kogge−Stone adder

Figure 4.5: Parallel prefix network examples

4.3 Redundancy in parallel prefix network

In a PPA, the problem of computing carry bits in parallel is converted to the problem of generating the block *g, p* signals for every bit position. For every output bit at position i, the signal pair $(g[0,i], p[0,i])$ is calculated through a tree of prefix operator nodes covering all the inputs from bit 0 to bit i. Overall, for a 16 bit adder, the prefix network consists of 16 trees, with possible sharing of intermediate results, to generate the outputs in parallel.

In PPA, redundancy is embedded in the parallel computation of all the $(g[0,i], p[0,i])$ for performance reasons. Similar to the correlation among the carry bits in a CLA, the outputs of the prefix network bear correlations among them in a PPA. Such correlations are the manifestations of inherent redundancy, since in a PPA all the outputs are generated in parallel, independent of the correlations. Similar to the CLA case, such correlations can be exploited for purposes of reliability.

For instance, between the two adjacent output positions i and $i-1$, we have:

$$g[0,i] = g[i] + g[0,i-1] \cdot p[0,i-1]$$

$$p[0,i] = p[i] \cdot p[0,i-1]$$

Note that $g[i]$ and $p[i]$ are simply bit level g, p signals, i.e., the input of the ith position in the prefix network. In general, we can define the prefix operator as $\circ$ on signal pair (g,p), so that

$$(g,p) = (g_L, p_L) \circ (g_R, p_R)$$

. Therefore, the correlation between adjacent output position $(g[0,i], p[0,i])$ and $(g[0,i-1], p[0,i-1])$ can be represented as:

$$(g[0,i], p[0,i]) = (g[i], p[i]) \circ (g[0,i-1], p[0,i-1])$$

Basically, by using the signals from an adjacent output position ($i-1$), an alternative path of generating $(g[0,i], p[0,i])$ can be formed in the prefix network. Thus, only one extra prefix operator is needed to generate the same output at position i.

Furthermore, the correlation can be extended to output positions that are not immediately adjacent:

$$(g[0,i], p[0,i]) = (g[i], p[i]) \circ (g[i-1], p[i-1]) \circ (g[0,i-2], p[0,i-2])$$

$$(g[0,i], p[0,i]) = (g[i], p[i]) \circ (g[i-1], p[i-1]) \circ (g[i-2], p[i-2]) \circ (g[0,i-3], p[0,i-3])$$

In summary, similarly to the CLA case, the same principle of exploiting inherent hardware redundancy for reliability purposes can be applied to PPAs:

- Through the exploitation of the prefix correlation, for every output position, a redundant copy of the same signal can be generated from the immediately adjacent output position by adding an additional prefix operator.

- Multiple redundant copies of a signal can be generated by extending the prefix correlation to farther apart output positions. However, when the redundant copy is built based on a farther away output position, the costs in terms of delay, hardware and interconnect increase.

In CLA, the correlation between carry signals is exploited. In PPAs, the prefix correlation between *g, p* pairs is exploited instead. Conceptually, the proposed approach exploits the same kind of correlations in parallel adders, since a PPA basically transfers the carry correlation to the one between *g, p* blocks. However, there are two main differences between the case of PPA and CLA, caused by their distinct internal structures used in generating the parallel signals.

- The amounts of hardware overlapping between the redundant signals are different for the two cases.

  In CLA, the three carry generation blocks in each LCG block are completely disjoint and independent. Therefore, a fault from a carry generation block will not be propagated into the redundant signals which are to be compared. In PPA, although all the output *g, p* signals are generated in parallel, there exists an amount of component sharing among the output positions. Note that in each of the PPA networks, a number of intermediate block signals are used to calculate multiple output positions, and this can be easily observed from the fanout number of the nodes in the prefix network. In this case, intuitively, a fault of a prefix operator node might propagate to affect both of the two copies of the same output signal, thus preventing the erroneous output signals from being detected.

  In fact, through a careful analysis based on the superposition of fanout paths, one can find out that such overlapping of hardware components in the redundant signal generation paths do not impact the fault tolerance capability. We will provide a discussion in the following sections addressing this issue.

- The redundancy coverage of the adder varies in these two cases.

For CLA, the redundancy exists in the carry generation process solely, while the *g, p* generation in the LCG blocks is accomplished by a single tree of prefix operation without any redundancy. Due to the existing redundancy, the reliability of carry signal generation can be efficiently enhanced. However, the redundant copies of the carry signals do not help in detecting or tolerating faults occurring in the *g, p* generation process in a CLA. Additional redundancy needs to be introduced to cover the *g, p* part. In the last section, we provide an approach using time redundancy to deal with the *g, p* generation part, such that the entire CLA structure can be fully covered for fault tolerance purposes.

In PPA, since the dominant part consists of the prefix network and the exploitation of the existing redundancy covers the prefix network, no significant part is left uncovered as is in the CLA case.

4.4 Fault tolerance approaches supported by extant redundancy in high performance parallel adders

The existing redundancy in parallel adders, as is discussed in the previous section, can be exploited for multiple fault tolerance approaches. Specifically, the redundancy can provide support directly for online fault detection, and to a certain extent fault masking as well. Moreover, such internal redundancy provides a powerful means to overcome the challenge of fine grained online diagnosis - a crucial stage for reconfiguration based online repair approaches. We provide a detailed description and discussion for these cases in the following subsections.

4.4.1 Online fault detection

Since a redundant copy of each output signal can be generated with an insignificant amount of hardware and a small delay, checking for the conformity of the two copies of the same signal delivers online fault detection capability. Specifically, the redundant copy of output signal at position i can be implemented from the output signal at position $i-1$, as the redundant copy generated through the immediate adjacent one costs the least in terms of hardware and delay, and has the best interconnection locality. These two copies of the same signal are directly compared using an XOR gate.

We can see from figure 4.5 that, for the Kogge-Stone PPA, no adjacent output positions share a common prefix operator. Such a property ensures a fine-grained fault detection capability, indicating at a bit level whether each output signal is faulty or not. In general, for PPAs without such a property, fault detection is achieved at the adder component level. Nonetheless, the *fault-secure* property of the PPA is maintained [1, 51]. In other words, any single fault occurrence can be guaranteed detected. For a fault to evade the redundant copy comparison based detection, the faulty computational unit has to either fan out to *all* the output positions, or to *none* of the output positions. Apparently, neither condition is satisfied for any node in the PPA network. Therefore, a faulty output is always distinguishable from the fault-free one.

The fault detection capability is essentially associated with the way each basic computational unit (in this case, a prefix operator) fans out to multiple output positions. Whether the approach can provide fine-grained fault detection at the bit level depends on the overlapping of computational units between the redundant copies to be compared. If a common prefix operator fans out to the two adjacent output positions i and $i-1$, when the original copy at position i is compared with the redundant copy generated at position $i-1$, both might be infected by the same faulty prefix operator, thus possibly conforming to a faulty bit.

4.4.2 Fault masking

A single fault masking capability achievable by TMR requires at least three copies of each signal. At a first sight, it seems fault masking can be supported easily by extending the fault detection approach to generate multiple copies of each output signal through inherent redundancy. However, the effectiveness of such fault masking schemes hinges on the disjointness of the hardware to generate the three copies, as a fault in one shared unit might cause the duplicated faulty result to outweigh in the majority vote. Consequently, one needs to examine carefully component overlaps in a PPA network, in order to make efficient use of the extant redundancy for fault masking purposes.

The Kogge-Stone PPA poses a unique characteristic in its structure. Notably in the Kogge-Stone PPA, not a single common node is ever shared between adjacent output positions. Although the set of odd numbered output positions has a large set of overlapped prefix operator nodes, as do the even ones, there is no overlap whatsoever between these two sets of components. In other words, the hardware is divided into two disjoint sets with a “clear cut”. The internal redundancy structured into such two disjoint sets naturally supports the generation of two copies of the same signal with no overlapping in hardware. Consequently, the inherent correlation between adjacent output positions can be exploited directly for fault masking approaches.

Traditional TMR approaches utilize triple the amount of original hardware to form three copies of a computation with complete disjointness. For a Kogge-Stone PPA, since the second copy of an output signal can be generated with the internal redundancy disjointly, only the third copy of the computation in a TMR approach needs to be added. In fact, to obtain a disjoint third copy of the output signals for TMR purposes, only half the amount of the original hardware in the Kogge-Stone PPA needs to be added. By duplicating either the hardware involved in the set of odd number output positions, or the even ones, the third copy of each output signal can be constructed. Consequently, a TMR based fault masking can be achieved with 1.5 times the original hardware for a Kogge-Stone PPA, instead of the triplication in a directly implemented TMR approach.

In the Brent-Kung or hybrid PPAs, the overall overlapping of hardware across output positions is so pervasive that the contribution of extant redundancy is hardly of any help in forming redundant copies with fully disjoint hardware. Essentially, a Kogge-Stone PPA utilizes more internal hardware redundancy to achieve higher performance, in comparison to the other implementations of a PPA. In addition, the redundancy in a

Kogge-Stone PPA is organized in a highly regular manner. This regularity, together with the abundance of internal hardware redundancy makes it feasible to exploit the existing redundancy for fault masking purposes without sacrificing the high performance.

4.4.3 Fine grained diagnosis for online repair

Online repair based fault tolerance approaches take advantage of the reconfigurability in nanoelectronic systems and provide the best flexibility in dealing with online occurring permanent faults. In a system with high and variable fault rates as is the case of the nanoelectronic environment, this class of approaches is highly promising. Reconfiguration based online repair is invoked after a fault has been detected and then diagnosed to be within a certain location. The amount of hardware necessitated in the repair phase hinges on the precision of the online diagnosis; therefore, high resolution online diagnosis is crucial to the efficiency of online repair based fault tolerance approaches.

Similar to the case of online fault detection, online diagnosis does not have any controllability over the inputs. In traditional offline diagnosis, all the fault signatures can be stored in a dictionary, which can be referred to when diagnosis is performed. In an online environment, the approach of using a dictionary to store the signature of every fault is not only costly but also infeasible. Because of the lack of controllability, one would have to store all the outputs, under every fault, for every possible input combination, for a dictionary based diagnosis approach.

Due to the aforementioned challenge, traditional offline diagnosis approaches have very limited applicability in an online environment. To achieve online diagnosis capability, one has to rely on a certain invariance, as in the online fault detection case. In distinction to online fault detection however, the invariance for online diagnosis purposes not only has to guarantee the *fault-secure* property, but also the *fault-distinct* capability. Without the *fault-distinct* capability, one faulty unit cannot be distinguished from another. Consequently, such *fault aliasing* will result in tremendous hardware overhead in the repair stage, because all the units in the ambiguity set need to be replaced by spare units. We start the general discussion of online diagnosis for PPA first, and then come back to CLA, as it is essentially a special case in the PPA discussion framework.

We have shown in the fault detection subsection that by generating a redundant copy of each output signal through its adjacent signal, the conformity of the two copies serves as an invariance for online fault detection. Basically, under a fault-free scenario, every pair should agree, and the XOR results of each output position between the two redundant copies should form a zero vector. We will refer to the vector of XOR results as *observable signature* henceforth. When a fault occurs, the observable signature is a non-zero vector. Consider the same invariance for online diagnosis purpose; we can see that if every fault gives out a unique *observable signature*, then fine-grained diagnosis can be achieved with full resolution.

Essentially, the observable positions of a faulty unit are only the output positions that the unit can fan out to. We define the *fanout signature* of a unit by its associated combination of output positions it fans out to. The relationship between a *fanout signature (fs)* , and the correspondent *observable signature (os)*, is as follows: $os[i] = fs[i] \oplus fs[i-1]$. This shows that they have the same distinction capability, since at output position 0 there is no unit involved and $fs[0] = 0$ constantly. Figure 4.6 shows the fanout signature fs, as well as the observable signature os for the faulty black unit in a hybrid PPA.

Since *fanout signature* and *observable signature* are essentially identical in representing the diagnosis resolution, we can examine the diagnosis capability of the proposed approach by looking at the *fanout signature* of each faulty unit, since it is more relevant to the prefix network structure. Apparently, no two units with the same *fanout signature* can be distinguished apart from each other. In such a case, aliasing is inevitable and the units with the signature fall into an *ambiguity group*. Figure 4.7 illustrates all the *ambiguity groups* of the three PPA designs. For each PPA, the prefix operator nodes that are marked by the same number belong to one ambiguity group, and the nodes that are not marked by any number can be diagnosed with full resolution. For instance, the Brent-Kung adder has four ambiguity groups, with group 1 consisting of

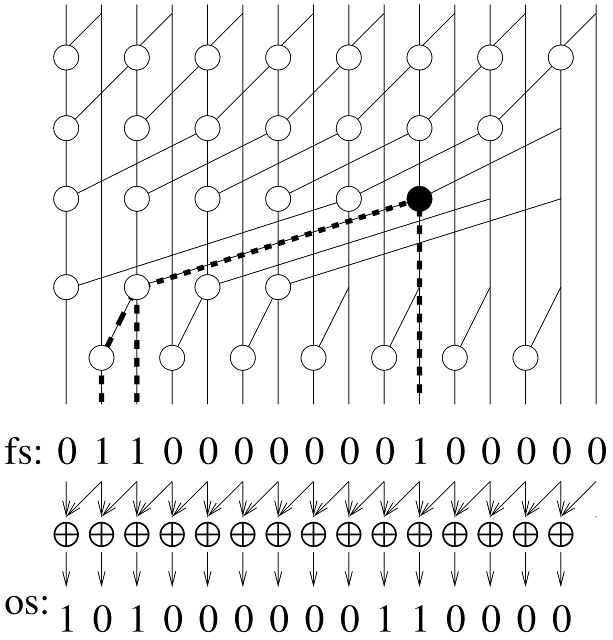

fs: 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0
os: 1 0 1 0 0 0 0 0 0 1 1 0 0 0 0

Figure 4.6: An example of fanout signature (fs) and observable signature (os) for the prefix operator node in black

4 elements, groups 2 and 4 consisting of 2 elements each, and group 3 consisting of 3 elements. Since the associated diagnosis cannot distinguish between the members in an ambiguity group, if the observable signature indicates a fault occurring in an ambiguity group, the subsequent reconfiguration procedure needs to replace all the members for repair purposes.

Online diagnosis for a CLA can be considered as a special case of the discussion above. The carry generation blocks are completely disjoint and each fans out to one output position; for example, block $C2$ generates signal $c2$ only. Consequently, the fanout signatures of the three carry generation block are simply (0, 0, 1), (0, 1, 0) and (1, 0, 0) and full diagnosis resolution can be achieved in CLAs.

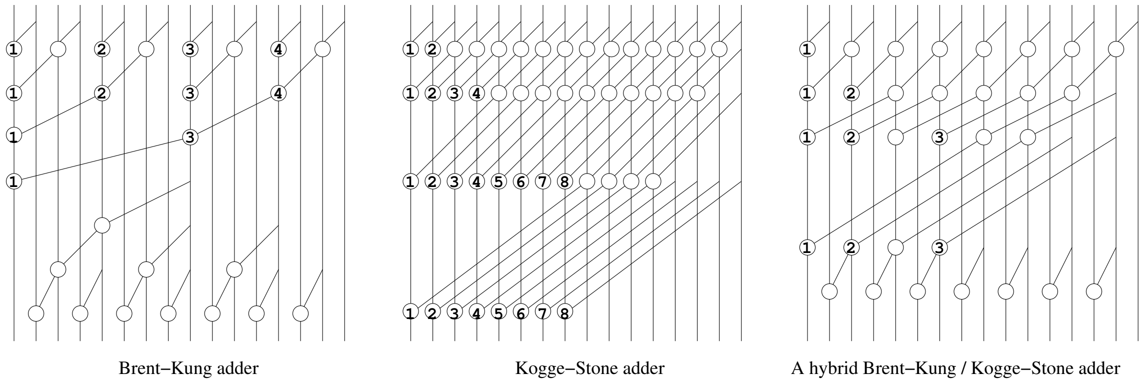


Figure 4.7: Ambiguity group of PPAs

4.5 Fault tolerance capability analysis

In this section, based on the proposed technique of exploiting the existing redundancy in parallel adders, we provide an analysis of the fault tolerance capability from the following perspectives:

- The capability and overhead for fault detection and fault masking.
- The capability of online fault diagnosis and the relevant repair costs.
- The fault coverage of the parallel adders by exploiting the existing redundancy.
- A discussion on the incomplete fault manifestation in an online environment.

4.5.1 Fault detection and fault masking discussion

Table 4.1: Fault detection analysis

Adder	Fault detection capability	extra hw / existing hw
64-bit CLA	bit-level fault secure	6 / 16 (per LCG)
16-bit Kogge-Stone PPA	bit-level fault secure	15 / 49
16-bit hybrid PPA	adder-level fault secure	15 / 32
16-bit Brent-Kung PPA	adder-level fault secure	15 / 26

Table 4.1 summarizes the fault detection capability and hardware overhead for the discussed parallel adders. The third column compares the extra hardware needed to generate the redundant copies of signals as a ratio to the existing amount of hardware in the adder.

In the CLA case, consider the basic unit of a two-input AND gate or a two-input OR gate; the carry generation blocks in an LCG consist of 16 gates in total[1]. To generate the redundant copy of a carry signal from the adjacent one according to the correlation $c_{out} = g + c_{in} \cdot p$, two extra gates are needed. Therefore, 6 gates need to be added altogether for the redundant copies of the 3 carry signals. For each of the PPA cases, a total of 15 prefix operator nodes need to be added at the 15 output positions to generate the redundant signal copies, while the existing hardware consists of the prefix operator nodes within the network.

Comparing to a generally applicable duplication based online fault detection scheme, where the extra hardware equals the existing amount of hardware, the proposed approach utilizes significantly lower hardware overhead. Performance-wise, the proposed approach costs an insignificant two-gate delay to generate the redundant signal copy.

Table 4.2: Fault masking analysis

Adder	Fault masking supported?	extra hw / existing hw	extra hw in TMR
64-bit CLA	yes	18 / 16 (per LCG)	32
16-bit Kogge-Stone PPA	yes	47 / 49	98
16-bit hybrid PPA	no	-	64
16-bit Brent-Kung PPA	no	-	52

Table 4.2 illustrates the support for fault masking by exploiting the existing redundancy and the corresponding overhead. For CLA, the calculation of extra hardware is performed similarly to the fault detection case, based on the carry generation blocks

[1] The hardware of the *g*, *p* generation block in an LCG is not counted because we only consider the components that are covered by the fault detection.

within an LCG. For the Kogge-Stone PPA, the amount of extra hardware consists of three parts: 15 prefix operator nodes at the output end, a redundant copy of the network for the odd output positions with 24 internal nodes, and 8 extra nodes at the output of the extra network. Overall, fault masking can be achieved by adding around the same amount of hardware as the existing one for the proposed approach. This is significantly less than double the amount of extra hardware required in a TMR approach. For the proposed approach, the performance overhead for fault masking is a constant 4-gate delay in the CLA, and 2-gate delay in the Kogge-Stone PPA.

4.5.2 Diagnosis resolution analysis

As is shown in figure 4.7, for a certain number of nodes in each of the parallel prefix networks, aliasing exists for the proposed online diagnosis approach. The overhead introduced by the loss of diagnosis resolution lies in the extra hardware needed to replace the entire ambiguity group, if one of the group members is faulty. Overall, according to the ambiguity group information illustrated in figure 4.7, the expected number of nodes needed in the repair phase, to replace one faulty node, can be computed. Table 4.3 lists the expected number of nodes to replace a faulty one for each PPA designs.

Table 4.3: Expected repair cost considering the ambiguity groups

16-bit Adder	Brent-Kung	Kogge-Stone	hybrid
Repair cost for one faulty node	1.85	1.90	1.63

Observing that the loss of diagnosis resolution is mainly due to the fact that all the members in the same ambiguity group have the same fanout signature, one can improve the diagnosis resolution using additional hardware to make each member's fanout signature distinct. Note that the extra hardware needs to be added within the same framework of the diagnosis scheme. In other words, new output positions need to be added to distinguish each member of an ambiguity group, while the only way to observe any fault at the output positions is through the comparison of redundant copies of the same signal. Again, the only way is to form redundant copies of a signal by exploiting the prefix computation correlation.

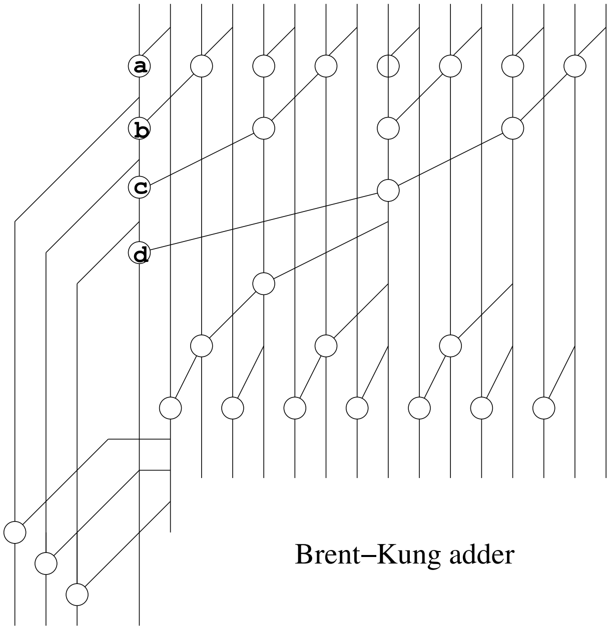


Figure 4.8: An example of using extra hardware to achieve full diagnosis resolution

Figure 4.8 illustrates an example of adding three nodes to distinguish the four nodes, a, b, c, d in the same ambiguity group of a Brent-Kung PPA. Based on the property of prefix operation, the three additional output positions form the same signal as the highest (15th) output position in the original PPA. For example, the fan out line from node c consists of $(g[8, 15], p[8, 15])$ while the 14th output position consists of $(g[0, 14], p[0, 14])$. With one of the additional prefix operator nodes, these two pairs are merged. According to the prefix operation:

$$(g[8, 15], p[8, 15]) \circ (g[0, 14], p[0, 14]) = (g[0, 15], p[0, 15])$$

Therefore, the additional node fans out a redundant copy of the signal from the 15th position. Note that full fault diagnosis resolution can be achieved for all the three additional nodes as well.

Without loss of generality, this approach can be applied to make each member of any ambiguity group distinguishable, thus improving the diagnosis capability to full resolution for all the PPAs. For an ambiguity group of n members, $n-1$ new nodes need to be added. Table 4.4 lists the number of extra nodes needed to achieve full resolution in each of the PPA designs. It also shows the average hardware overhead of repair for one node based on the original PPA. According to tables 4.3 and 4.4, in general, the hardware overhead for repair is very low for the online diagnosis approach. Moreover, with the approach to achieve full diagnosis resolution, hardware repair cost can be cut down significantly.

Table 4.4: Hardware overhead and repair cost for full diagnosis resolution

16-bit Adder	Brent-Kung	Kogge-Stone	Hybrid
Number of extra nodes	7	14	6
Avg repair cost for one faulty node	1.27	1.29	1.19

4.5.3 Fault tolerance coverage on parallel adder subcomponents

As discussed in the previous section, in the PPA cases, the existing redundancy covers the entire adder, yet in the CLA case, the *g, p* generation blocks are not covered by exploiting the redundancy in carry generation blocks. The *g,p* generation block in a CLA does not bear an inherent redundancy as the carry generation blocks. Therefore, redundancy needs to be explicitly added for reliability purposes.

We can observe that, in a CLA, the *g, p* generation by itself is a disjoint data flow from the carry propagation, and is performed in a tree structure with no horizontal level dependencies. In such a case, a time redundancy based approach that recomputes with rotated operands (RERO) can be applied on such a structure for online fault detection and diagnosis purposes.

When the components are performing the same type of computation with independent inputs, online fault detection and diagnosis can be achieved through time redundancy, using a different permutation of the inputs for the recomputation. A simple form of permutation is rotation. In a CLA, the *g, p* generation blocks at the same level use disjoint input signals; therefore, a rotated recomputation does not suffer from the perturbation caused by the correlated input signals. Therefore, RERO can be used not only for fault detection, but also online diagnosis as well. A detailed RERO based approach for the *g, p* generation blocks is proposed in [69].

4.5.4 Inconsistent fault manifestation

For a faulty component, at any time whether the output will be erroneous depends on the input combination. Basically, a fault manifests only when the input combination stimulates the fault and sensitizes its propagation path towards the output. Similarly, if an internal faulty signal is not directly connected to the output, it might lose its observability when it reaches the output positions, after going through some other components. Consequently, in an online environment, depending on the input, even though a faulty component fans out to multiple output positions, it might only manifest at a subset of these positions.

In the example of figure 4.6, theoretically a fault from the black node fans out to three output positions $\{5, 13, 14\}$. However, due to such *inconsistent fault manifestation*, it might not manifest at all on these three positions, depending on whether the faulty signal from the black node is the controlling input when going through other nodes. Out of the three positions, it is certain that the fault will manifest at the 5th position, since the faulty signal goes directly to the output there. Whether the fault will manifest at the 13th output position however, depends on the input to the node as well as the internal logic of the node the faulty signal goes through before reaching the output. Certainly, if the faulty signal is annihilated at the 13th output position, it will not propagate to the 14th one either. Otherwise, whether the fault manifests at the 14th output position depends on the nodes it propagates through at the 14th position.

Inconsistent fault manifestation makes online diagnosis challenging, since each fault might have a number of possible fanout signatures. This increases fault aliasing possibility, thus directly impacting diagnosis resolution. In general, any online diagnosis approach relies on certain fault manifestation signatures to identify the faulty components, thus suffering from the loss of certainty in fault signatures, due to inconsistent fault manifestation. The following observations about inconsistent fault manifestation can serve as a starting point for improving diagnosis resolution.

- Inconsistent fault manifestation has unidirectional impact: the output positions reachable through fan out by a faulty unit might be correct ones, yet the output positions that are not reachable by the faulty block are always correct. This means all the "zero" positions in a fanout signature are determinative.

- Inconsistent fault manifestation occurs when a faulty signal goes through certain computation units before finally reaching the output. Therefore, for faulty signals that can directly reach an output position, it will always consistently manifest. For these positions, the fanout signature bits are determinative. In other words, for the "one" positions in a fanout signature, the subset consisting of direct fanouts from the unit being diagnosed is determinative. For example, in figure 4.6, there is no inconsistent fault manifestation at the 5th output position, if the unit marked as black is faulty.

- Whether a faulty signal propagates through a unit and manifests at the output depends on two issues: the internal logic of the unit, and the other inputs of the unit. For a given arithmetic component such as CLA or PPA, the internal logic of each unit is known. Under the assumption of uniformly distributed input signals at the inputs, the inconsistent fault manifestation at the output side can be quantified and evaluated in a probabilistic sense. Therefore, even for the output positions in a fanout signature that are impacted by insignificant fault manifestation, one can obtain a relatively precise probability of fault manifestation. This probabilistic information can be utilized for online diagnosis techniques that are based on fault manifestation accumulation over time.

Although inconsistent fault manifestation introduces challenges to online diagnosis resolution, it does not have any impact on fault detection or fault masking. Since inconsistent fault manifestation unidirectionally changes faulty output positions to fault-free ones, neither fault detection nor fault masking capability will be compromised, since the only thing that matters in these two schemes is the correctness of the results, i.e., signals on the output side.

4.6 Conclusions

This chapter presents a novel way to enhance reliability in the arithmetic components of nanoelectronic systems without adding tremendous overhead. The proposed approach relies on the key observation that computational redundancy is typically embedded in high performance arithmetic components to deliver parallelism. Specifically, we identify the carry propagation correlation among adjacent bits for parallel adders and exploit such computational redundancy for fault tolerance purposes. We examine the cases of carry lookahead adders and general parallel prefix adders, and develop a number of techniques to construct fault detection, fault masking and online fault diagnosis approaches utilizing the extant hardware redundancy.

The proposed approach for parallel adders provides an efficient means of enhancing the reliability for the basic arithmetic building blocks in nanoelectronic environment. Moreover, the general principle behind the proposed approach, of exploiting the extant redundancy for the dual purposes of parallelism and reliability, can be extended and applied to a genre of high performance arithmetic components, which are crucial and dominant in the nanoelectronic systems. We show that the computational redundancy embedded in the high performance arithmetic components can be exploited to support various fault tolerance approaches. The proposed principle therefore opens up a new way to develop with very low overhead powerful fault tolerance schemes and thus overcome the reliability challenge in nanoelectronic systems efficiently.

Chapter 5

Locality Aware Redundancy Allocation for Defect / Fault Tolerance

A high level of redundancy is required to deal with the challenge of high defect and fault rates in the nano environment. The reconfigurability of nano devices and the regular structure of nano fabrics make reconfiguration based repair an essential approach for both defect and fault tolerance. Ideally, repair based approaches have best hardware efficiency when full sharing of redundancy is achievable. However, nanoelectronic systems are subject to the strict constraint of localized interconnections, which limit the sharing of redundant resources to within a small neighborhood.

In this chapter, we focus on this challenge and develop the corresponding model for the redundant resource sharing issue under locality constraint. Such a model captures the redundancy sharing essence of a system, and is applicable to any specific topology or layout of a nanoelectronic system. Based on the established model, defect tolerance is a well defined problem and can be addressed by existing algorithms. A case study is presented in the end to examine how such a problem can be addressed with the detailed topology, configuration and setup given in a nanoelectronic environment.

5.1 Motivation

Online reconfigurability opens up the opportunities for developing defect and fault tolerance approaches efficiently. After identifying the defects through manufacture test, reconfiguration can be performed to bypass the defective units with spare ones [20, 32, 59, 71, 82]. For online repair based fault tolerance approaches to be efficient in utilizing hardware redundancy, the system need to have two attributes:

- Online reconfigurability is a necessity, since the commonly shared redundancy needs to be dynamically configured through an online repair process upon fault occurrence.

  Online reconfiguration can be implemented at multiple design hierarchical levels. Logic level reconfiguration is naturally supported by the nanoelectronic devices [13, 34, 53]. At higher design hierarchical levels, reconfigurable nano switch blocks are embedded among the nano computational blocks, thus enabling the reconfiguration of interconnections among the computational blocks [26, 27, 57].

- The system consists of multiple identical components; therefore redundancy can be shared among multiple units for replacement upon faults.

  In fact, high regularity exists widely at various system hierarchical levels. At the logic level, PLAs with regular structure can have spare wires and devices for repair purposes; at the arithmetic level, most components such as parallel adders and multipliers are composed of multiple identical units; FPGA and memory systems are both representative structures with high regularities and identical CLB units; at the processor architecture level, multiple homogeneous ALU and computational units are utilized for to achieve parallelism; in a multi-processor systems, faulty processor elements can be replaced by a pool of spare ones.

For reference convenience, we denote in this paper the original multiple identical components in the system as *functional units*, while the redundant resources that can be shared by multiple functional units in an online repair process are denoted as *spare units*.

When every spare unit can be shared among all the functional units for replacement purposes, optimal reliability is achieved due to the maximum sharing of redundancy. However, such a maximum redundancy sharing demands complete interconnections between every pair of functional unit and spare unit. When defect and fault rates are exceedingly low, as in the case of CMOS based systems, a small pool of spare units is enough to fulfill the reliability requirement. The interconnections to support full sharing are then affordable.

When a large number of redundant resources are required, as in the case of nanoelectronic environment, not only are the interconnection complexity necessitated to implement a full redundancy sharing scheme boosting tremendously, but also interconnection is strictly limited to be within a local range. As a result, maximum redundancy sharing is no longer affordable, as spare units can only be shared in a localized manner.

A number of parameters are related to the locality constraint. First of all, the topology and layout of a system varies depending on the underlying nanofabric and the design hierarchical level. Secondly, the interconnection range and complexity rely on the specific nanoelectronic device, or the implementation of the switch blocks. As a first cut on approaching such a problem, a model needs to be established for the redundancy sharing under locality constraint. Based on the formulated model, multiple perspectives including topology, interconnection, hardware overhead and reliability can be analysed, thus providing guidance for developing efficient defect / fault tolerance schemes.

Specifically, the following issues need to be addressed:

- How to model the localized resource sharing and isolate it from the topology details of a system?

- What are the defect, fault, reliability and failure models of a system under such locality constraint?

- How to perform reconfiguration for defect tolerance?

- For online faults, how to perform reconfiguration for revocable / irrevocable online repair?

5.2 Mathematical model and framework

5.2.1 Bipartite graph model for redundancy sharing

Essentially, a reconfiguration based repair replaces a defective / faulty functional unit by a local spare unit. The “localized sharing” of spare units includes the following two perspectives. On the one hand, each functional unit can be repaired by a set of spare units that are accessible according to the locality constraint. On the other hand, each spare unit can be used to replace a number of functional units, thus representing the sharing of redundancy and fault tolerance efficiency. Similarly, the accessible functional units are limited within the neighborhood of the spare unit, under the locality constraint.

Mathematically, the relationship between the functional unit set and the spare unit set can be represented by a *Bipartite Graph* model.

Definition: A *bipartite graph* $B(N1, N2, E)$ is a 3-tuple where $N1$ and $N2$ are two disjoint sets of nodes and E is a set of edges. Every edge $e \in E$ connects a node $n1 \in N1$ with a node $n2 \in N2$.

For a system consisting of a functional unit set, a spare unit set, and the corresponding locality constraint defining the neighborhood range, it can be uniquely represented by a bipartite graph $B(N_f, N_s, L)$. Node set N_f is the functional unit set, N_s is the spare unit set, and each edge $e = (n1, n2) \in L$ indicates that the corresponding functional unit $n1$ and spare unit $n2$ are within a neighborhood defined by the locality constraint, so that $n2$ can be used to replace $n1$.

Various perspectives of the redundancy sharing problem can be represented by the attributes in the bipartite graph model.

- The amount of redundancy embedded in the system is depicted by $|S|$, i.e., the number of spare unit.

- The edges between the two node sets indicate the accessibility between the functional units and the spare units defined by the locality constraint. Essentially, the edges represent the potential repair possibilities.

- The overall number of edges in the bipartite graph, $|L|$, depicts the interconnection complexity of the system.

- The locality constraint of the system is represented by the fanout degree of each node. Specifically, the fanout degree of a spare unit node illustrates how many functional unit can be repaired by the spare unit; the fanout degree of a functional unit represents the number of spare units accessible.

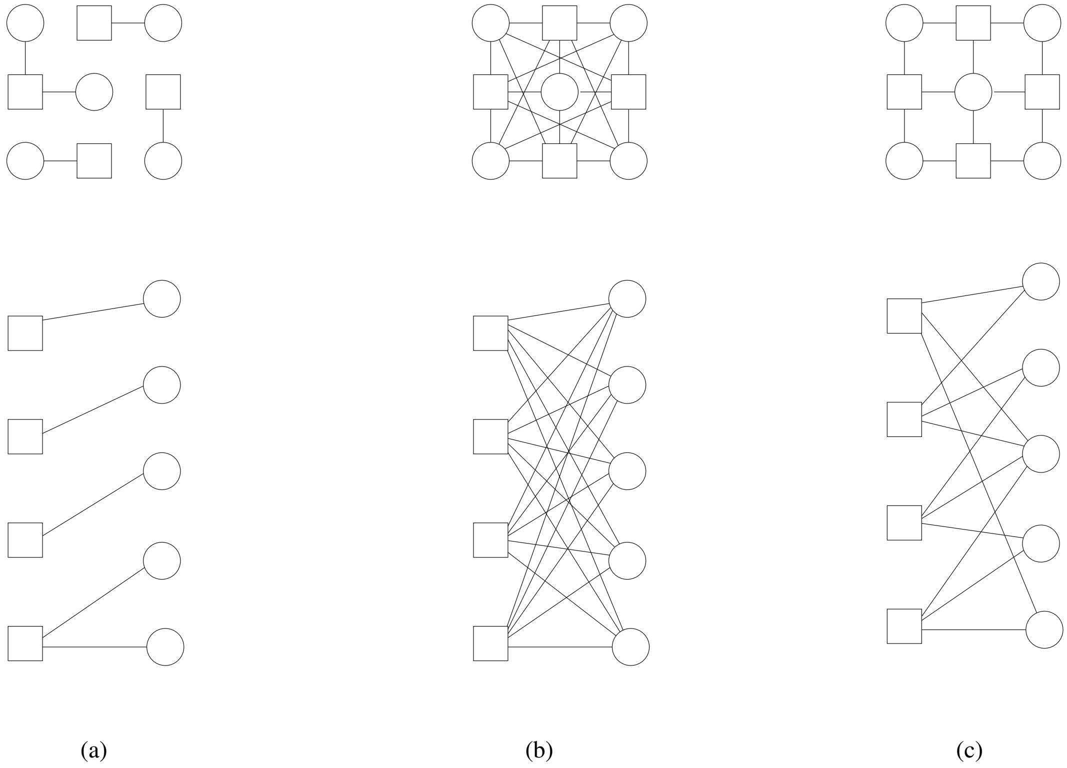


Figure 5.1: Examples of topology layout with their bipartite graph representations: (a) no sharing - each functional unit has 2 exclusive spare units (b) full sharing - each spare unit can replace any functional units (c) localized sharing - each spare unit can replace the neighboring functional units

Figure 5.1 shows three examples, each with 4 functional units (illustrated by squares) and 5 spare units (illustrated by circles). The upper part of each example shows the 2-D topological layout of the units. Each line connecting a functional unit and a spare unit represents that interconnections are available in the systems, possibly through multiple switching blocks, to replace the functional unit with the corresponding spare unit. The bottom part shows the bipartite graph representation for the topology above.

Figure 5.1(a) shows an example of dedicated redundancy without sharing. The bipartite graph of the example configuration is divided into 4 disjointed sub-graphs, each contains one functional unit and its dedicated spare unit(s). Since each spare unit is used exclusively to repair one specific functional unit, the degree of each spare unit node in the bipartite graph is always one. Eliminating the sharing of redundancy makes the interconnect complexity low. However, the system is very inflexible to deal with high occurrence of faults.

Figure 5.1(b) shows a full sharing example, which demands high complexity of interconnections. The corresponding bipartite graph is a *complete bipartite graph*.

Figure 5.1(c) shows an example of sharing under the locality constraint. Each functional unit has 3 accessible spare units within the neighborhood. For the 5 spare units, the ones at the corners each can be used to replace two neighboring functional units. The spare unit at the center can be used to replace either of the 4 functional units in the neighborhood. This is depicted in the bipartite graph, where each functional unit node has a degree of 3, four of the spare unit nodes have degrees of 2, while one spare unit node has a degree of 4.

Overall, the bipartite graph representation provides a form to model the redundancy sharing relationship independent of the specific layout topology. Given any topology with the associated specification in terms of functional / spare unit layout and sharing attribute, a unique bipartite graph can be constructed accordingly to characterize the localized redundancy sharing relationship.

5.2.2 Repair under bipartite graph model

A reconfiguration based repair is performed by replacing a defective / faulty functional unit with a locally accessible spare unit. In the bipartite graph, after performing a repair, the spare unit node and its associated edges are deleted from the graph. When a defective / faulty functional unit has no accessible spare units in its neighborhood, no repair can be performed. Such an unrepairable case is typically caused by the exhaustion of local redundant resources, A suboptimal reconfiguration performed for some other defect / fault occurrences might lead to the quick exhaustion of local redundant resources. Consequently, the reliability of the system not only depends on the amount of redundant resources and the amount of sharing, but also heavily hinges on the algorithm spare unit selection when repair is performed. We consider the system as having failed when at least one of the defective / faulty functional unit becomes unrepairable.

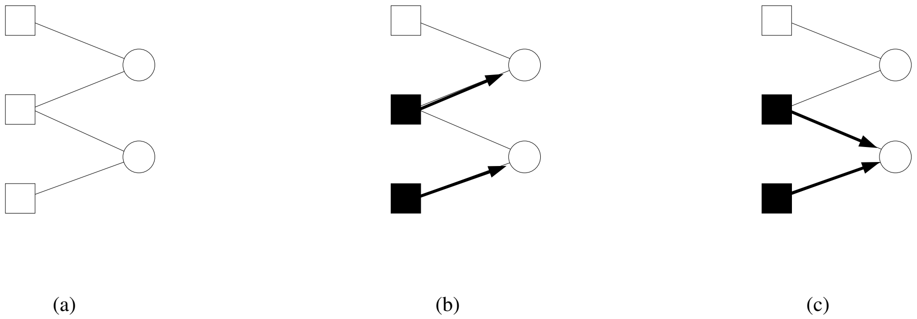


Figure 5.2: Examples of successful and failing repair

Figure 5.2 illustrates a simple example with two different allocation schemes for repair. The bipartite graph representation of a system is given in figure 5.2(a), where three functional units share two spare units under locality constraints.

Figure 5.2(b) and (c) shows two different spare unit allocation possibilities, with two defective functional units. The allocation of spare units for replacement is shown by the bold edges with arrows, indicating a spare unit that is chosen to replace the specific defective / faulty functional unit. The spare allocation in (b) illustrates a successful repair, with both defective / faulty units replaced by the two spare units. The allocation in (c) is an unsuccessful repair, due to the contention of one spare unit shared by the two defective / faulty functional units.

In an online environment, if the first fault occurs at the middle node, the allocation in (c) is possible to take place due to the lack of knowledge for the upcoming faults. Intuitively, this simple example shows the difference between an off-line and an on-line algorithm for the cases of defect versus fault tolerance:

- In an off-line environment, all the information regarding the defective functional units are available for the repair process, thus making successful repairs more easily achievable.

- When faults are occurring dynamically at run-time, the spare unit allocation to be made without the knowledge of future fault occurrences. Consequently, successful repair over a sequence of faults is much harder.

5.2.3 Defect tolerance as bipartite graph matching problem

After manufacturing test, a set of defective functional units are identified, thus providing full information to perform reconfiguration based repair. Based on the bipartite graph model, the defect tolerance problem can be mapped to a *Bipartite Graph Matching Problem*, so that a set of spare units can be identified to form an one-to-one match to the defective functional units.

Definition: *bipartite graph matching for defect tolerance:* Given a bipartite graph $B(N_f, N_s, L)$ representation of a nanofabric configuration and the set of defective functional units N'_f, where $N'_f \subset N_f$, find a repair allocation as a match function $M : N'_f \rightarrow N'_s$ such that: $\forall n \in N'_f$, $(n, M(n)) \in L$, and $\forall n1, n2 \in N'_f$ where $n1 \neq n2, M(n1) \neq M(n2)$.

Such a definition of the defect tolerance problem is based on a “one-for-one” model between spare units and functional units. The assumption of the model is that the granularities of functional units and spare units are the same. In other words, exactly one spare unit is used to replace one defective functional unit.

The same bipartite graph model can be set up based on a different model of "multi-for-one", where spare units are the sub-components of a functional unit. For instance, a functional unit may represent a parallel prefix adder, while a spare unit is one prefix operation node. Such an assumption makes the bipartite graph more compact. On the other hand, a defective functional unit might require multiple spare units for repair purposes. Nevertheless, a "multi-for-one" model can be converted straightforwardly to the "one-for-one" model. Consequently, without loss of generality, we provide our discussion based on the one-for-one assumption in this paper.

Overall, the problem of performing spare unit allocation for defect tolerance is mapped to a well-defined bipartite graph matching problem, which can be solved through a max-flow algorithm [15]. The complexity of the algorithm is $O(|L'|\sqrt{|N'_f|})$, where L' is the set of edges connected to N'_f. The repair is successful when a match is found to replace every defective functional unit with a localized spare unit. When the system is unrepairable, a quantitative evaluation can be performed by capturing the percentage of repaired functional units over the total number of defective functional units.

5.2.4 Fault tolerance based on the bipartite graph model

Dealing with dynamically occurring faults at run time means that the fault tolerance approach has to adopt an online algorithm for spare allocation. This is different from the defect tolerance phase, where the algorithm is run offline, with all the defect information available. For fault tolerance, the system reacts to dynamic fault occurrences by performing reconfiguration based online repairs. In the bipartite graph representation, a repair process is modeled by deleting the corresponding spare unit node from the graph and marking the functional unit node as fault-free again. Each repair process changes the bipartite graph into a new one, and the multiple choices of allocating a spare unit indicates the multiple variations of the current bipartite graph. Figure 5.3 exhibits an example of two different resultant bipartite graphs according to the different choices of spare unit node for replacement.

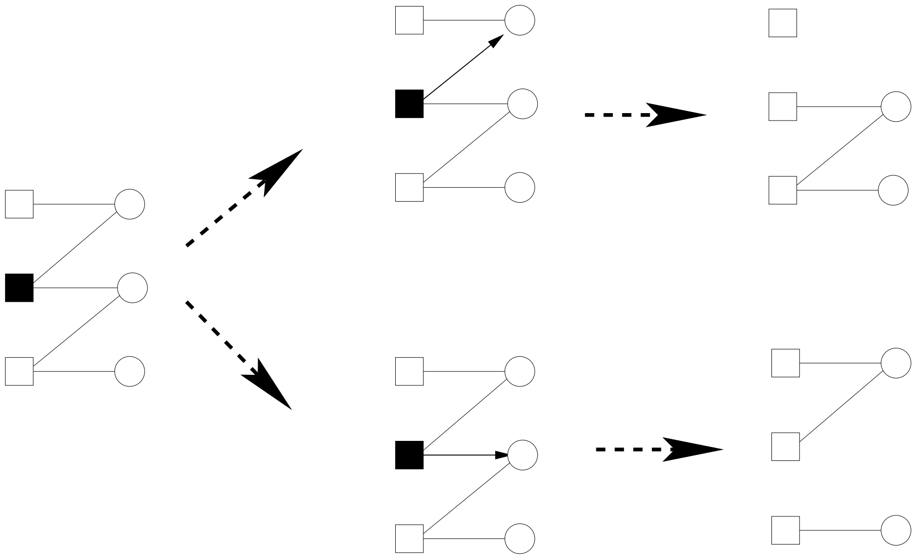


Figure 5.3: Examples of bipartite graph change according to spare allocation

The fault tolerance algorithm repairs a faulty functional unit by selecting an accessible spare unit for replacement. Apparently, when there are multiple spare units to select from, each choice affects the system in a different neighborhood, thus resulting in different new bipartite graphs. An optimal spare allocation algorithm therefore needs to guarantee the remaining system to be the most robust one among all the possible variations.

Reliability model

Before approaching the problem of finding an optimal algorithm, optimality needs to be defined first. For the specific fault tolerance problem, we need to compare the possible bipartite graph variations for the system's reliability. As a simple example, the two variations of the bipartite graph in figure 5.3 both have 2 spare unit nodes and 3 edges. Which one represents a system that is more robust for the upcoming faults, and why?

The answer to this question hinges on the reliability model of a system. First of all, we assume independent fault occurrence on each functional unit with equal probability. Furthermore, we assume that the faults occur sequentially, and one repair is performed after every fault occurrence. For f faults occurring sequentially on a set of m functional units, there are altogether m^f equally possible occurrence patterns. The reliability of the system under the next f faults is therefore defined by the percentage of repairable cases over all the m^f patterns.

We need to examine more carefully about the "repairable cases", because for each of the m^f fault occurrence pattern, the multiple selection choices at each repair step result in a repair decision tree. Apparently, not all the repair selections can lead to successful repair for the given fault sequence. Counting all the repair possibilities is computationally impractical. Furthermore, it is not even a good way to capture the system's reliability, since a large number of poor decisions lead to overall repair failure. Therefore, the "repairable cases" need to be modeled in a deterministic manner. For any one of the m^f fault occurrence patterns, it is defined as *repairable* if at least one successful repair sequence exists for it.

Definition: For a given system's bipartite graph $BG(N_f, N_s, L)$ with m functional unit nodes and n spare unit nodes ($|N_f| = m, |N_s| = n$), a sequence of f fault occurrences on the functional unit set N_f is defined as a vector $< u_1, u_2, ..., u_f >$ where $u_i \in N_f$. A *successful repair sequence* for such a fault sequence is a vector $< v_1, v_2, ..., v_f >$ where $v_i \in S$, such that $\forall 0 < i < j \leq f, v_i \neq v_j$, and $(u_i, v_i) \in L$.

A fault occurrence pattern is *repairable*, if there exists a successful repair sequence for it. To decide whether a fault occurrence pattern is repairable or not, one needs to search for a successful repair sequence. Such a problem bears high similarity to the defect tolerance model, because the fault occurrence sequence is given, and an offline algorithm can be applied.

In fact, this is the bipartite graph matching problem under the multi-for-one model, because multiple faults can occur upon the same functional unit node, thus demanding multiple spare units for repair purposes. As is discussed in the defect tolerance subsection, the multi-for-one model can be converted to the one-for-one model and solved with the bipartite graph matching algorithm. For denotation convenience, we define a Φ function for the process of determining whether a fault occurrence pattern is repairable.

Definition: Given a bipartite graph representation $BG(N_f, N_s, L)$ and a fault occurrence sequence s, a $\Phi(BG, s)$ is a function that: $\Phi(BG, s) = 1$ if the fault sequence s is repairable on BG, and $\Phi(BG, s) = 0$ if no successful repair sequence exists for s on BG.

Given a bipartite graph, every fault occurrence pattern can be determined repairable or not by solving the Φ function. To capture the reliability of a given system, an overall ratio can be therefore calculated for repairable patterns over the total number of fault occurrence patterns.

The system reliability model under the next f faults for any bipartite graph representation is defined as follows:

Definition: For a bipartite graph $BG(N_f, N_s, L)$ where $|N_f| = m$, let P be the set of all possible occurrence patterns of f faults:

$P = \{< u_1, u_2, ..., u_f >: u_i \in F, 0 \leq i \leq f\}$, then $|P| = m^f$

let R be the set of all repairable patterns in P:

$R = \{r : r \in P, \Phi(r) = 1\}$, then $|R| = \sum_{\forall p \in P} \Phi(p)$

let U be the set of all unrepairable patterns in P:

$U = P - R$, then $|U| = |P| - |R|$

The reliability $\mathcal{RE}$ of the system under f faults is defined as:

$$\mathcal{RE}(BG, f) = \frac{|R|}{|P|} = \frac{\sum_{\forall p \in P} \Phi(p)}{m^f}$$

The failure rate $\mathcal{FA}$ of the system under f faults is defined as:

$$\mathcal{FA}(BG,f) = \frac{|U|}{|P|} = 1 - \mathcal{RE}(BG,f) = \frac{\sum_{\forall p \in P} \Phi(p)}{m^f}$$

Since a fault occurrence pattern is identified as repairable whenever a successful repair sequence exists, the reliability model gives the rate of successful repair assuming all the allocation choices are made with the right decision. This gives an upper bound for the online repair, no matter how well the allocation algorithm can perform because no knowledge of the future faults can be used to direct the decision making for each repair step, and a number of repairable fault sequences may actually end up unrepairable.

Reliability function characteristics

Plotting the entire reliability curve $\mathcal{RE}(BG,f)$ for a given bipartite graph BG over the upcoming f faults is computationally impractical, since exponential number of Φ functions need to be calculated for each point. Nevertheless, based on the bipartite graph model, we can identify a number of characteristics of the reliability function $\mathcal{RE}$. Figure 5.4(a) provides an illustrative plot for the reliability function $\mathcal{RE}(f)$ of a given bipartite graph BG as a monotonous decreasing function over f.

Intuitively, the reliability of a system always decreases as the number of fault occurrences increases. For the specific fault tolerance problem, the repair process under a sequence of faults leaves fewer and fewer spare units available. Eventually, the system becomes unrepairable due to the exhaustion of spare units in a local area. We provide a formal proof below that the reliability function $\mathcal{RE}$ defined is a monotonic decreasing function over the number of fault occurrences.

Since $\mathcal{RE}(BG,f)$ is a discrete function over $f \in Z^+$, one only needs to prove $\forall f \geq 1, \mathcal{RE}(BG,f) \geq \mathcal{RE}(BG,f+1$ to show that $\mathcal{RE}$ is a monotonically decreasing function over f. As in the definition of $\mathcal{RE}$ and $\mathcal{FA}$, suppose the bipartite graph has m functional unit nodes ($BG(N_f, N_s, L)$ with $|N_f| = m$), let P be the set of all fault sequence patterns for f faults, R be the set of all repairable fault sequence patterns and U be the set of all unrepairable fault sequence patterns, then $\mathcal{FA}(BG,f) = |U|/|P| =$

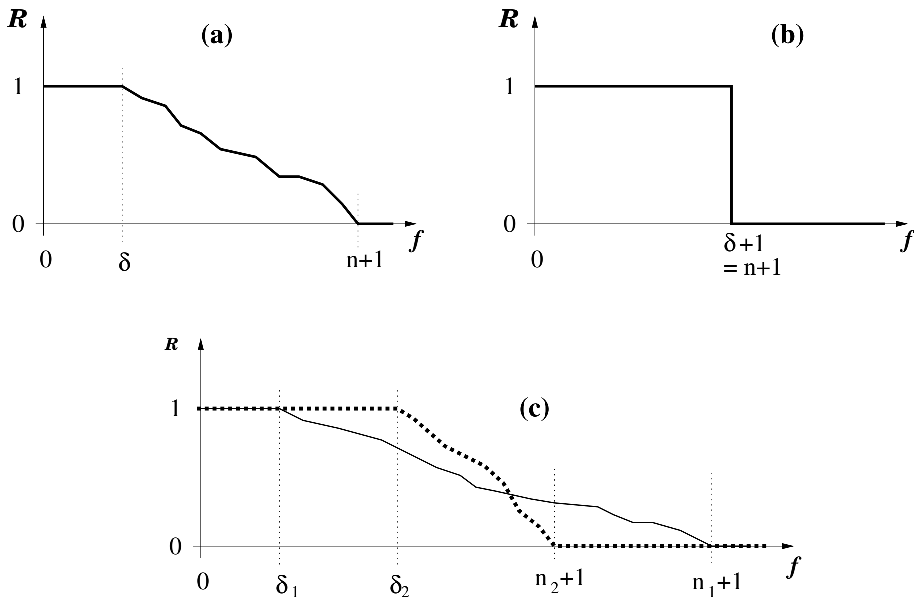


Figure 5.4: (a) Reliability plot of a general bipartite graph; (b) Reliability plot for a system with full interconnection for maximum sharing; (c) Reliability plots for two systems with different parameters

$|U|/m^f$.

For $f+1$ faults, there are altogether m^{f+1} fault sequence patterns. If the sequence pattern of the first f faults is unrepairable, then obviously the system is unrepairable for the $f+1$ fault sequence as well, regardless of the location of the last fault. Therefore, for every $u \in U$, there are m distinct unrepairable sequences for $(f+1)$ faults, and they are all distinct. On the other hand, a repairable sequence of f faults might turn out to be unrepairable due to the $(f+1)$th fault. Let N be the set of such new unrepairable fault sequences ($|N| \geq 0$), then $\mathcal{FA}(BG, f) = \frac{|U|}{|P|} \leq \mathcal{FA}(BG, f+1) = \frac{|U| \times m + |N|}{|P| \times m}$. Since $\mathcal{RE} = 1 - \mathcal{FA}$, $\mathcal{RE}(BG, f) \geq \mathcal{RE}(BG, f+1)$, and thus proving the monotonical decreasing of function $\mathcal{RE}$.

Beyond the monotonous decreasing attribute that we have proved, there are two characteristic points for $\mathcal{RE}(f)$: the “1-point”, beyond which the reliability starts to fall lower than 1, and the “0-point” where it reaches 0.

Apparently, a system is always repairable if the number of fault occurrences is less than the minimum fanout degree in the functional unit node set. For a given system's bipartite graph $BG(N_f, N_s, L)$ with m functional unit nodes and n spare unit nodes ($|N_f| = m, |N_s| = n$). Let $d(u)$ denote the fanout degree of a node u, and $\delta(N)$ be the minimum fanout degree of a node set N: $\delta(N_f) = MIN\{d(u)\}, \forall u \in N_f$. For $f \leq \delta(N_f), \mathcal{RE}(BG, f)$ = 1. With more than δ faults ($f > \delta$), the reliability starts to fall below 1, since the system is unrepairable if the f faults all occur on the functional unit node with the minimum fanout degree. Consequently, the "1-point" of the reliability function $\mathcal{RE}$ is at $\delta(N_f)$ of the bipartite graph.

The "0-point" of the reliability function is determined by the number of spare units n. Obviously, when $f > n$, there exists no successful repair sequence for any fault occurrence pattern. On the other hand, when $f = n$, there certainly exists at least one fault sequence pattern for which the system is repairable. We can construct such a fault sequence pattern by traversing through the spare unit node set S in any order, and having the i'th fault occur on any one of the functional units connected to the i'th spare unit. Apparently, the sequence of spare units we traverse through is a successful repair allocation. Since function $\mathcal{RE}(f)$ reaches 0 at $f = n+1$, and $\mathcal{RE}(f)$ is a monotonically decreasing function, $\mathcal{R}(f) = 0$ for $f > n$.

For a system with maximum sharing, where every spare unit is accessible by any functional unit, the system is represented as a complete bipartite graph with $\delta(N_f) = n$. As is shown in figure 5.4(b), the reliability curve for such a system is simple: $\mathcal{RE}(f) = 1$ for $f \in [0, n]$, and $\mathcal{RE}(f) = 0$ for $f > n$. Such a system is fully reliable for up to n faults, and the system's reliability becomes zero for more than n faults. With the support of full connectivity, reliability relies solely on the amount of redundancy. Among all the systems with the same amount of redundancy, one with maximum sharing has the best reliability.

Two bipartite graphs of different interconnection patterns present distinct reliability curves over fault occurrences. Nevertheless, the system with a higher $\delta(N_f)$ can maintain a full reliability for more faults, while the reliability of a system with more spare units (m) reaches its zero point under higher fault occurrences. Figure 5.4(c) gives the illustrative plots of two systems with different parameters.

Fault tolerance algorithm and optimality

The fault tolerance approach needs to make a selection among the several choices of spare unit nodes when repairing a faulty functional unit. The various possible choices lead to new variations of the bipartite graph, depending on which spare unit node and its corresponding edges to be deleted. Each repair allocation choice's impact to the reliability of the system lies in the neighborhood of the selected spare unit node.

In the bipartite graph representation, the adjacent node set $adj(u)$ denotes the neighborhood of a node u: $adj(u) = \{v : (v, u) \in L\}$ for bipartite graph $BG(N_f, N_s, L)$. When a spare unit node v is selected to replace a faulty functional unit u, v is to be deleted from the graph with all the edges connected to it. Consequently, all the functional unit nodes in the neighborhood of v will have one fewer spare unit accessible. In the bipartite graph representation, the degree of each node in $adj(v)$ is decreased by one. The multiple choices of spare unit nodes $v_1, v_2, ..., v_i$ therefore affect the corresponding sets of functional unit nodes in $adj(v_1), adj(v_2), ..., adj(v_i)$.

Lemma: Recall that $d(v)$ denotes the degree of node v, and $\delta(V)$ denotes the minimum degree among a node set V. To repair a faulty functional unit node u, among the $d(u)$ choices of spare unit nodes $v_1, v_2, ..., v_{d(u)}$, choosing the spare unit node v such that $\delta(adj(v)) \geq= \delta(adj(v_i)), \forall 1 \leq i \leq d(u)$ results in a bipartite graph with the best reliability for the next upcoming $\delta(adj(v))$ faults.

Intuitively, if a functional unit has a very small number of accessible spare units, then one should avoid allocating these spare units for repair as much as possible, since they constitute the scarce repair resource for this particular functional unit. The algorithm suggested by the lemma essentially selects a spare unit v, such that the minimum fanout degree of its neighboring functional units $(\delta(adj(v)))$ is the greatest among all the choices. The lemma states that such a selection gives a resultant system with the best reliability for a certain number ($\delta(adj(v))$) of upcoming faults.

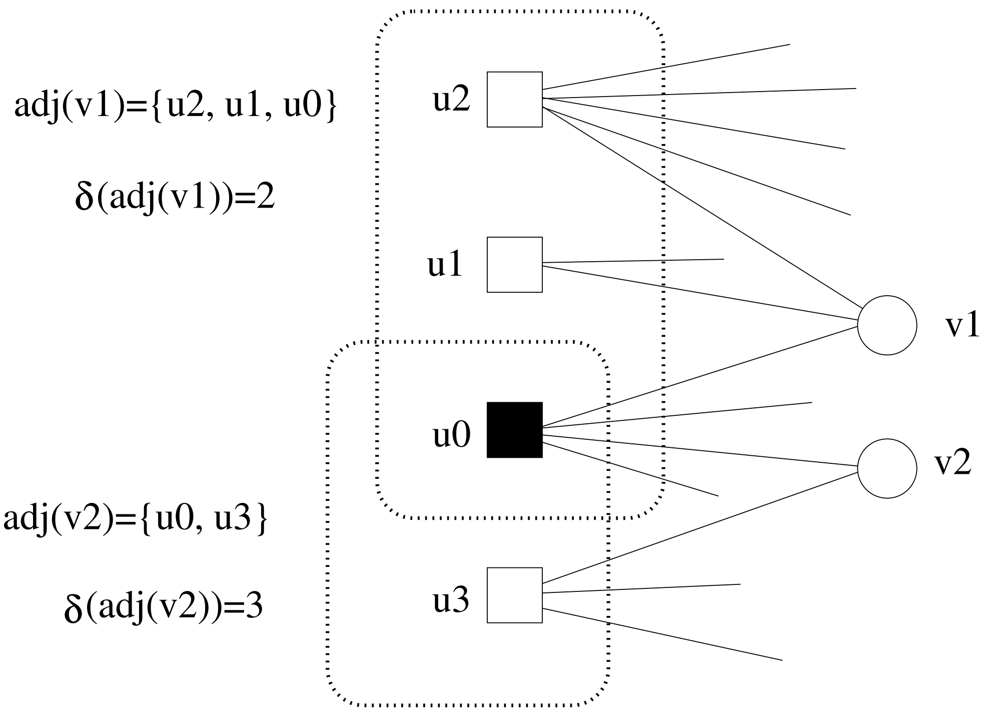


Figure 5.5: Spare unit selection example

Figure 5.5 provides an example for such a selection directed by the lemma with a partially shown bipartite graph. Particularly, a comparison is made between the two accessible spare unit nodes $v1$ and $v2$ for the faulty functional unit $u0$. The neighboring functional unit set of $v1$ includes the upper three functional units, $u2, u1$, and $u0$, with their corresponding fanout degrees of 5, 2, and 4. Therefore, the minimum fanout degree in this node set is $d(u1) = 2$, as is shown in the figure by $\delta(adj(v1)) = 2$. For the neighboring functional unit set of $v2$, which includes $u0$ and $u3$, the minimum degree is $d(u3) = 3$, therefore $\delta(adj(v2)) = 3$.

Between $v1$ and $v2$, the lemma indicates selecting $v2$ over $v3$ for repairing the faulty functional unit $u0$, since $v2$ has a larger minimum fanout degree in its neighboring node set. The intuition behind such a selection is that, selecting $v1$ makes $u1$ the most vulnerable node within the shown graph for upcoming faults, while selecting $v2$ makes $u3$ the most vulnerable node. Comparing these two selections, selecting $v2$ has a resultant system with the most vulnerable node stronger than the choice of $v1$.

The formal proof for the lemma is given below.

Proof:

To prove the lemma, we only need to show that, between any two choices of spare units $v1$ and $v2$, the one that has a larger $\delta(adj(v))$ results in better reliability for the next $\delta(adj(v))$ faults.

Apparently, the difference between the two resultant bipartite graphs of selecting $v1$ versus $v2$ resides only in the set of nodes $adj(v1) \cup adj(v2)$. Consequently, to compare the reliability of the two resultant systems, we only need to focus on the subgraph $BG'(N'_f, N'_s, L')$, where $N'_f = adj(v1) \cup adj(v2), N'_s = adj(N'_f)$, and L' includes all the edges between the node sets.

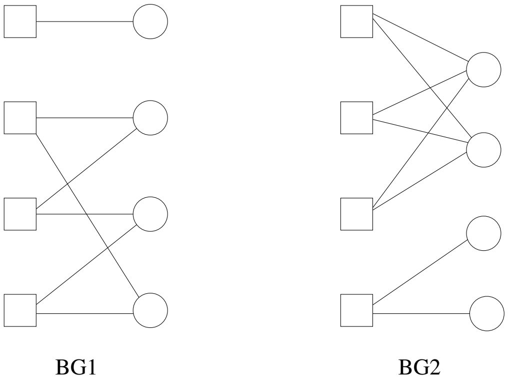
Let $\delta1 = \delta(adj(v1))$, $\delta2 = \delta(adj(v2))$, and suppose $\delta1 < \delta2$. Therefore, when $v1$ is selected, the resultant bipartite graph from BG' has $\delta1$ as the minimum degree in its functional unit node set. The reliability function of the resultant system thus has $\delta1$ as the 1-point. When $v2$ is selected, the resultant bipartite graph's reliability function has $\delta2$ as the 1-point. Since $\delta1 < \delta2$, the system after selecting $v2$ can maintain 100% reliability for the next $\delta2$ fault occurrences, which is more than the system after choosing $v1$ for the repair selection.

In general, selecting the spare node v among the choices of $(v_1, v_2, ..., v_{d(u)})$ where $\delta(adj(v))$ is the largest among all the $\delta(adj(v_i)), 1 \leq i \leq d(u)$, the resultant system has the best reliability for the next $\delta(adj(v))$ fault occurrences.

Ideally, an optimal algorithm selects the allocation of spare unit such that the resultant system is the most reliable one among all the possible decisions. With the definition of reliability of a system, we can see that the system's reliability is indeed a function of fault occurrences. It is worth noticing that the provided algorithm guarantees the selection to be optimal in reliability for a certain number of upcoming faults.

When comparing the reliability of two systems, one that is more robust for a small number of upcoming faults is not necessarily more reliable for high fault occurrences. The simple example in figure 5.4(c) shows that reliability for a low level of fault occurrences depends largely on $\delta(N_f)$, while for a high level of fault occurrences depends more on the number of spare units (n).

Figure 5.6 presents a concrete example, where two bipartite graphs are shown with the same number of edges and the number of spare unit nodes. For two fault occurrences, $BG1$ has a failure rate of 1/16, since the system becomes unrepairable when both faults occur on the one node with degree 1. For three fault occurrences, $BG1$ becomes unrepairable only when: 1) two of the three faults occur on the single node with degree 1, or 2) all the 3 faults occur on any single node. For $BG2$ however, when the three faults occur among the upper three nodes that share two spare unit nodes, the system becomes unrepairable. Such patterns outnumber the unrepairable patterns for $BG1$ under three faults. Consequently, the reliability of $BG2$ is worse than $BG1$ under three faults, even though with two faults $BG2$ is more reliable.



Fault #	1	2	3
R(BG1)	1	15/16	50/64
R(BG2)	1	1	37/64

Figure 5.6: Reliability comparison for two bipartite graphs under different fault occurrences

Nevertheless, when a decision has to be made without any knowledge of the future fault occurrences, the proposed fault tolerance algorithm is able to make a selection that is optimal for the “near future” for upcoming faults. In practice, concentrating on the reliability of the system against the “near future” upcoming faults is certainly more fundamental and tangible than making the system more reliable in the probability sense for the “far future”, after a long sequence of fault occurrences.

Furthermore, such a fault tolerance algorithm makes a localized decision, since the information required is limited to the neighborhood of the candidate spare unit nodes. For a faulty functional unit u, the algorithm compares and selects among the $|d(u)|$ accessible spare units. Since the selection needs the information of $\delta(adj(v_i))$ for each candidate spare unit v_i, each spare unit (v_i) can store the degrees for all its neighboring functional unit nodes ($d(adj(v_i))$), as well as the minimum degree among them ($\delta(adj(v_i))$). The decision of spare unit allocation can therefore be made quickly based on polling and comparing the stored information among all the candidate spare units.

After performing the repair with the selected spare unit node, updated degree information needs to be propagated to the neighborhood for consistency. For a selected spare unit node v, all its neighboring functional units ($adj(v)$) need to be updated by decreasing the degree of each by one. Then, the change of degree information of each functional unit node ($u_i \in adj(v)$) needs to be propagated to its neighboring spare units ($adj(u_i)$), to keep the corresponding information stored in these spare units uptodate. Consequently, communication required to perform a repair is limited to the candidate spare units, i.e., the adjacent nodes of the faulty functional unit ($adj(u)$). The information update after the repair process is propagated to a wider range neighborhood: first to the adjacent functional unit nodes of the selected spare unit node ($adj(v)$), then to the neighborhood of this set ($adj(u_i)$, where $u_i \in adj(v)$).

5.2.5 Discussion

Overall, the proposed model provides a general framework for the reliability of systems equipped with shared redundancy, with locality constraints. The online fault tolerance algorithm provides an optimal solution for a limited number of upcoming faults, and can be achieved by using localized communication and decentralized control only.

In practice, beside the framework established by the reliability model and the defect / fault tolerance algorithms, a number of issues need to be considered:

1. Whether the spare units are kept standby or involved in computation as redundancy?

2. When the spare units are loaded with redundant computations, the boundary between functional units and spare units becomes blurred.

3. For a system that supports preemption, i.e., reversible reconfiguration, what is the algorithm, and what is the tradeoff between reliability benefit versus performance overhead?

4. For defect tolerance, the bipartite graph matching provides one solution, possibly among multiple ones. How to direct the defect tolerance process so as to guarantee that the resulting system is more robust towards run time faults?

5. The bipartite graph representation is powerful in modeling any arbitrary topology layout. However, when the system is rather large yet laid out with high regularity, heuristic solutions involving less communication overhead need to be developed.

6. For a system at a low design hierarchy level, a simpler algorithm without optimality might be preferred; communication overhead might need to be decreased furthermore.

These are all highly relevant perspectives of a system with its specifications. In the following section, we provide a case study involving the defect / fault tolerance in nanoelectronic environment with shared redundancies, namely an online repairable NMR system. Rather than providing a straightforward example with the solution provided by the general framework discussed in the previous section, we try to look into the specific issues disclosed above for the given case study.

Applying the generalized bipartite graph based framework would provide a solution for both defect and fault tolerance. However, it would certainly demand much more significant overhead due to the size of the bipartite graph. In addition, the many detailed parameters provided by the specific case is not addressed in the general bipartite graph based model.

Instead, we provide heuristic algorithms for defect and fault tolerance for the given specific system, taking into consideration some particular issues mentioned above. We can see that, when the specific parameters are given for a system of high regularity, efficient heuristic approaches can be developed to achieve powerful defect / fault tolerance capabilities.

Overall, the bipartite graph based model is highly beneficial in terms of providing a general framework in performing evaluation and research on the novel emerging reliability challenge, namely the redundancy sharing under locality constraints. Under such a framework, a genre of defect and fault tolerance approaches can be developed. Particularly, the algorithms can be directly applied to systems at high design hierarchy levels, where optimality in terms of repair is crucial while the control and communication overhead for carrying out the optimal repair algorithm can be amortized by the complex computation performed by the functional units.

5.3 Case study: Flexible NMR Supported by Reconfiguration

The traditional NMR technique is rigid for the number of redundant copies. When N is preset to a fixed number, the system either can no longer guarantee correctness when the redundancy is below the fault occurrences, or becomes wasteful when the redundancy is set to aim at the worst case scenario. More importantly, the locality issue is not addressed in traditional NMR approaches. These challenges make the well-known fault tolerance approach hardly applicable to the nanoelectronic systems, which are highly unreliable with strict locality constraints.

To solve these problems, localized sharable redundancy can be added to introduce flexibility to an NMR approach. A voter with multiple failing computational units

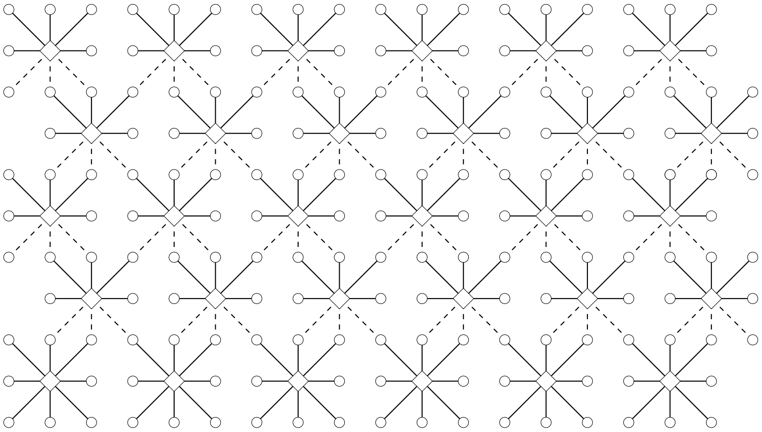


Figure 5.7: An example flexible NMR grid structure with five-modular redundancy initialized

can then allocate the spare units (denoted as “backup units” in this case study) to maintain the NMR structure with desirable fault tolerance. We present a structure denoted as *flexible NMR* henceforth that deals with the defects and faults in the computation units, which occupy the majority of components and constitute possibly the main concern from the reliability perspective.

5.3.1 Nanofabric topology for flexible NMR

We use a grid structure since it is representative of a number of nanofabrics [18, 26]. Nanoelectronic computational units, which perform ALU operations are placed in the grid structure. Due to the interconnection constraint, fast communication can be attained only among neighboring components in the grid. In the example shown in figure 5.7, we assume voters (shown as diamonds) and units (shown as circles) are placed at the crosspoints of the grid structure.

As an example each voter shown on figure 5.7 is initialized with five-modular redundancy[1]. A voter uses five out of the eight accessible units. These five units are

[1] A level of five modular redundancy is utilized solely for illustrative purposes in this discussion, while the actual quantity in a particular implementation would need to be determined based on a more precise understanding of the exact fault rates in nanotechnology.

denoted as *in-use* units while the remaining three units are denoted as *backup* units. In-use units are connected to the voter by solid lines and backup units are connected by dotted lines. These backup units are utilized as in-use units by the neighboring voters, representing the flexible additional redundancy for the voter on an *as needed* basis. With this configuration, three out of the five in-use units of a voter serve as backup units for other neighboring voters. All voters except those in the bottom row are initialized with five modular redundancy. Voters in the bottom row are initialized with extra redundancy of eight in-use units each.

Figure 5.8 shows a number of failing units on the example grid where faulty computation units are denoted as black dots. A voter needs to maintain a certain pre-defined fault tolerance capability threshold. In this example, we have defined the threshold to be three fault-free in-use units. The clustering of failing units results in the loss of fault tolerance capability for a number of voters. Particularly, two voters indicated in black diamonds in the figure have fault tolerance capability fall below the pre-defined threshold.

In fact, when the number of fault-free in-use units of a voter falls below the threshold, a voter can acquire the backup units from its neighbors to recover from this loss in fault tolerance capability. When a voter tries to utilize a backup unit, the neighboring voter that is currently using the unit releases it. Then, the newly acquired unit is reconfigured to perform the computation required by the acquiring voter. Figure 5.9 shows a reconfiguration of backup units and in-use units to withstand the failing units of figure 5.8, such that all the voters can retain a fault tolerance capability that is above the threshold.

Due to the localized communication constraint, communication between voters and units, including the comparison of data and the transfer of reconfiguration information, is limited to a voter and its surrounding units. In order to flexibly adjust the redundant backup units in the presence of a possibly unbalanced distribution of faults, redundancy needs that cannot be met locally should be propagated and transferred to distant areas. Essentially, when a voter fails to acquire enough backup units from its

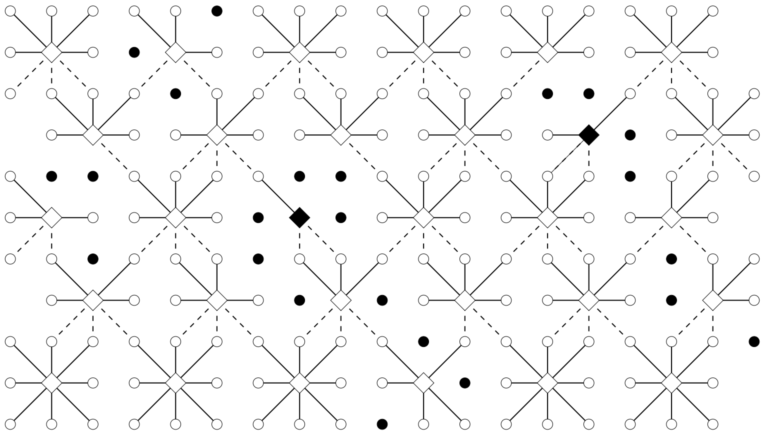


Figure 5.8: A number of failing units in the grid example result in loss of fault tolerance capability for two voters

local neighborhood, this redundancy requirement should be propagated until it reaches a region of the nanofabric with available backup units.

There are two issues that need to be considered when constructing an underlying nanofabric topology to support propagation of redundancy needs that cannot be satisfied locally. The first one is to avoid cycles in a resource request propagation path. The second issue is to direct propagation of the backup unit request towards nanofabric regions rich in available redundancy.

A cycle of resource request propagation represents a deadlock situation. One way to avoid cycles is by properly designing the reconfiguration algorithm. However, this requires both global control and complex checking mechanisms in the algorithm, thus making its online implementation hard on a simple voter. An alternative way to avoid cycles in the propagation path is to enforce properties that preclude cycle formation in the underlying nanofabric topology. In the example of figures 5.7 to 5.9, the underlying nanofabric topology enforces a unidirectional propagation of information about backup units. The nanofabric grid is initialized so that every voter has three units below it as backup units and the other five as in-use units. When a voter needs to acquire a backup unit, it always acquires redundancy from the neighboring voters in a lower row. In turn, every voter's three in-use units on the upper row can be requisitioned by the

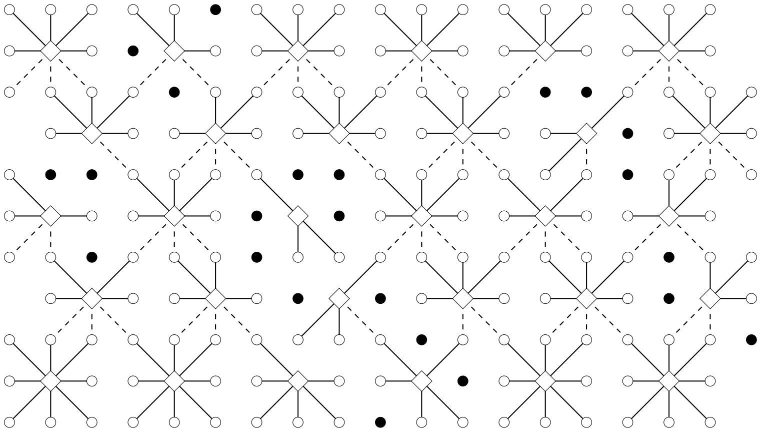


Figure 5.9: A possible reconfiguration for the flexible NMR grid example to withstand faulty units

neighboring voters in the row above it and so on. The rows of voters therefore represent different levels in the propagation of requests for backup units and the propagation of reconfiguration information always flows downwards.

Although the second issue of directing the unmet redundancy requests towards resource-abundant voters relies largely on the reconfiguration algorithms, it imposes certain properties on the underlying nanofabric topology to help attain algorithmic optimality. In the example of figures 5.7 to 5.9, by enforcing unidirectional propagation of resource information, such information propagation terminates either at a voter with available resources, or at a voter at the edges of the nanofabric. In the example, the voters at the edges, especially those at the bottom edge, are initialized with relatively more redundant resources. This extra redundancy at the edges maximizes the probability that such resource requests for redundant resources succeed.

Essentially, any topological structure that satisfies the following properties can be used to support flexible NMR.

- Each voter is initialized with an above-threshold number of in-use units and can access some backup units locally.
- Nanofabric topology enforces unidirectional propagation of requests for backup

units.

- Extra redundancy is embedded at points where such information propagation terminates.

These properties specify a genre of regular structures with the example shown and analyzed being one possible instance. Various nanofabric topologies can be constructed by varying the number of units connected to each voter, the number of neighbors per voter, the number of voters that can access a shared backup unit etc. Figure 5.10 shows another possible structure with four-modular redundancy embedded in each voter.

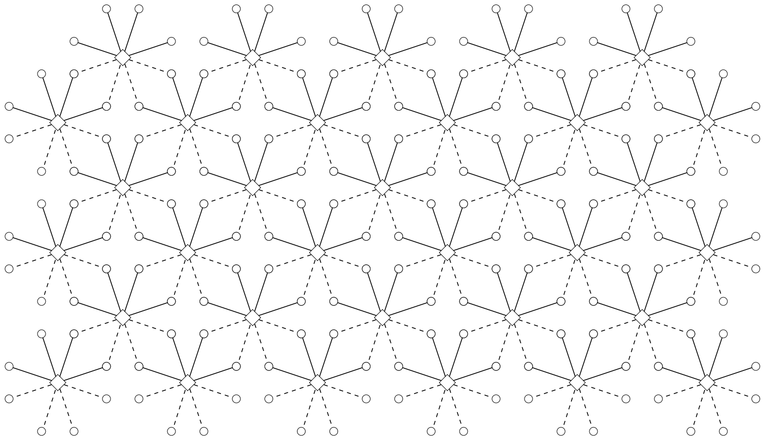


Figure 5.10: An alternate topology of flexible NMR structure

5.3.2 Reconfiguration Algorithms

The proposed regular nanofabric topology enables flexible reconfiguration of voter-unit connections. Although redundancy is embedded around every voter with the initialized NMR, an effective reconfiguration algorithm is needed to re-allocate units to overcome the impact of faulty units on the fault tolerance capability.

We consider two cases:

- When the system is running, dynamically occurring faults might degrade the number of fault-free units for a voter below its pre-defined threshold. For this case, we propose a *dynamic reconfiguration algorithm* with simple and localized control in section 5.3.3 to maintain the fault tolerance capability of the victim voters.

- After the manufacturing process, a large number of defects are typically identified. Clustering of some of these defects might result in foregoing even the initial fault tolerance capability in a number of voters. We propose a *static reconfiguration algorithm* in section 5.3.4 that reconfigures the resources according to a manufacture-time defect map. Since this algorithm can be carried out offline, it can be complex.

5.3.3 Dynamic Reconfiguration

The dynamic reconfiguration algorithm is applied on a per-fault basis. It is invoked whenever a voter compares the results of its in-use units and identifies a faulty unit.

```
• Dynamic_reconfig(Voter $v$, Unit $u$)
  //invoked when voter $v$ identifies a faulty unit $u$; return success if $v$ is above FT threshold after reconfiguration, return failure otherwise

    1. release $u$ as an in-use unit, mark $u$ as faulty
    2. if $v$'s fault-free in-use units exceed threshold, return success
       //still enough fault tolerance capability, no need to repair
    3. if $v$ has insufficient fault-free backup units, return failure
       //there is not enough redundancy, so no possibility of repair
    4. while ($v$'s fault-free in-use units below threshold)
    (a) if all the fault-free backup units have been visited, return failure
        //when all the backup units are critical to their voters; then no one is available to repair $v$
    (b) pick a fault-free backup unit $u_0$, which is an in-use unit of a neighboring voter $v_0$ on the lower row
    (c) if $v_0$'s surrounding fault-free units exceeds threshold
        //otherwise releasing $u_0$ would sacrifice $v_0$
          i. configure $u_0$ to be an in-use unit of $v$
         ii. inform $v_0$ to perform Dynamic_reconfig($v_0$, $u_0$)

5. return success
```

In this algorithm, the voter first deals with the two simple cases: no need to repair, or not repairable. If fault tolerance capability is above the threshold, no reconfiguration is needed. If voter v cannot gather sufficient fault-free backup units to maintain the threshold fault tolerance capability, then the reconfiguration process signals a failure.

Except for the two simple cases, the voter needs to allocate backup units. In this case, the voter allocates the backup units based on the situation of the neighboring voters that are using these units for their own NMR computations. Suppose a fault-free backup unit u_0 is used by a neighboring voter v_0 as an in-use units, then the reallocation of u_0 for v should not result in an unacceptable fault tolerance capability for voter v_0. In other words, if u_0 is one of the crucial fault-free in-use units of v_0 and there is no fault-free backup units for v_0, then u_0 should not be allocated from v_0. Otherwise, u_0 can be allocated for v, and v_0 is notified for an update and possibly needs to utilize its own backup units.

In the flexible NMR approach, a confirmed computation consists of two phases: the computation phase and the confirmation phase. In the confirmation phase the voter performs a majority vote for the multiple computation results and applies the dynamic reconfiguration algorithm if a faulty unit is identified. The reconfiguration of a unit starts to take effect at the subsequent computation phase; therefore the computation prior to the voting remains valid and is not cancelled by the reconfiguration of any related units.

In this algorithm, a voter accesses available resources locally and the communication consists of quite simple messages. Therefore, this algorithm can be applied online with localized control of low complexity. The reconfiguration process of a voter might propagate through the neighboring voters according to the availability of redundant resources in the local area:

- If there are no available resources in the neighboring area, then the voter inevitably fails and no propagation path is necessarily formed.
- If the redundant resources in the neighboring area are abundant, then the reconfiguration of the starting voter can be easily achieved without invoking the reconfiguration of other voters.
- If the resources in the local area are available yet limited, then the voter successfully acquires the resource, and initiates a chain of reconfiguration processes for other voters along the unidirectional propagation path.

In the worst case, the dynamic reconfiguration results in one failure voter, regardless of the formation of a propagation path. Figure 5.9 is essentially the resulting configuration of applying the dynamic algorithm on the example grid, regardless of the fault occurrence sequence.

When multiple faults occur on a single voter, the algorithm can proceed as long as the faulty units remain the minority in the voting process. When faults occur on multiple voters, the algorithm can be executed by the voters in parallel. If two neighboring voters are trying to do the reconfiguration at the same time and the reconfiguration involves a common unit, then according to the flexible NMR topology and the dynamic reconfiguration algorithm, the voter at the higher level in the unidirectional propagation path allocates the unit while the other voter is forced to invoke another reconfiguration process at the next confirmation phase.

5.3.4 Static Reconfiguration Algorithm

After manufacturing test in nanoelectronic fabrication, a defect map can be generated with the location of each defective unit. Applying the dynamic reconfiguration algorithm on the given defect map does not necessarily yield the best solution for balancing and recovering the most number of defective units. Since the algorithm needs only to be carried out once and can be performed off-line, it can utilize the global defect map and incorporate more complex control.

```
• Static_reconfig(flexible NMR structure $v[N][M]$)
  //applied to a structure with $N \times M$ voters connected to units that are marked
  correspondingly by a defect map; redistributing redundancies so as to recover
  the FT capability for the maximum number of voters; return the reconfigured
  structure

    1. for $i = 1$ to $N - 1$ //from the bottom row up
        (a) for $j = 1$ to $M$ //from left to right
              – if voter $v[i][j]$'s defect-free surrounding unit below threshold, mark
                $v[i][j]$ as fail
                //insufficient redundancy to recover FT capability
        (b) for $j = 1$ to $M$
            //from left to right deal with voters in row $[i + 1]$, first with the ones
            that necessitate reconfiguration
              – if $v[i+1][j]$'s defect-free surrounding units below threshold, mark
                $v[i + 1][j]$ as fail
                //insufficient redundancy to recover FT capability
              – if $v[i+1][j]$'s defect-free in-use units below threshold, reconfigure
                available backup units of $v[i + 1][j]$ as in-use units until threshold
                FT capability achieved
                //perform reconfiguration
        (c) for $j = 1$ to $M$ *//from left to right*
            *//deal with other voters in row* $[i + 1]$
                for every backup unit $u$ of $v[i + 1][j]$
                    if $u$'s voter in row $[i]$ has more than the threshold number of
                    in-use units, assign $u$ as in-use unit for $v[i + 1][j]$
                    *//push the redundant resources upwards*
    2. return the reconfigured $v[N][M]$
```

The static reconfiguration algorithm tries to minimize the number of failing voters through a global resource reorganization. It starts from the bottom row of voters, which are initialized with the most redundant resources. According to the topologic order of processing voters from the bottom row upwards, the algorithm tries to push the extra redundancy in the last row upwards and utilizes it to recover the fault tolerance capabilities of voters with defective units.

When processing each row, the algorithm first updates the voters in the current row, i, identifies the defective units of voters in row i and marks out the unrepairable voters. After row i has been updated, the algorithm tries to push the maximum possible available resources in row i to its upper row $i + 1$ as long as the non-failing voters in row i retain above threshold fault tolerance capability.

However, the distribution of resources from row i towards row $i+1$ needs to satisfy firstly the crucial voters in row $i + 1$. A voter in row i might have limited redundant resources to distribute upwards; therefore distribution needs to make the highest priority to the voters that necessitate repair. The algorithm goes through the voters in row $i + 1$ from left to right, assigning the redundant resources to the crucial voters first.

After performing the reconfigurations for the crucial voters, the remaining redundant resources of voters in row i are assigned to the remaining voters in row $i + 1$. Since the static reconfiguration algorithm starts from the resource abundant bottom row of voters and pushes redundant resources upwards, it can redistribute the limited redundant resources more effectively in a global manner.

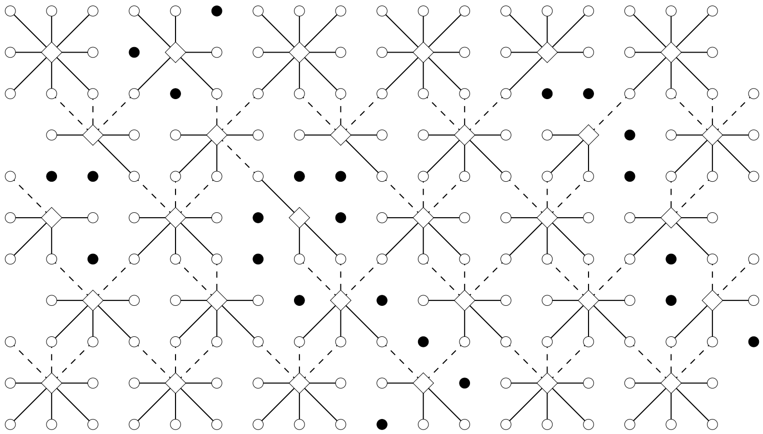


Figure 5.11: The resulting example grid configuration after applying the static algorithm on a given defect map

In the case of no defects, after performing the static reconfiguration algorithm, the new configuration map becomes the upside down version of the original configuration. The voters in the uppermost row are assigned with the extra resources and become the terminating points of the unidirectional propagation paths. Figure 5.11 shows the configuration of the grid example after performing the static algorithm on a given defect map of figure 5.8. The static algorithm starts from the resource abundant bottom row up, resulting in a configuration with the top row having the most abundant resource. The resulting topology of a static algorithm still satisfies the criteria of the flexible NMR structure, thus supporting the application of the dynamic reconfiguration algorithm at the operation-time for the fault tolerance of the nanofabric.

5.3.5 Simulation Results

To verify the proposed strategy, we have simulated the proposed flexible NMR approach in C++. Our main target is to investigate how much fault tolerance capability can be achieved under various fault rates by the proposed flexible NMR approach. We compare the proposed strategy to the traditional NMR approach that utilizes exactly the same amount of hardware resources. We investigate different grid sizes varying from

10×10 to 30×30 and measure the voter failure rate under multiple fault rates. For the static reconfiguration algorithm, a randomly generated defect map is setup for every fault rate point. The same defect map is used to test the effectiveness of the traditional NMR approach and the dynamic reconfiguration algorithm, for the purpose of a fair comparison. To test the dynamic reconfiguration algorithm, the faulty units on the defect map are ordered randomly in a sequence and the algorithm is invoked by the sequence of the occurring faults. The result is exhibited through the ratio of failing voters, i.e., the number of voters that cannot maintain the threshold fault tolerance capability of three fault-free in-use units divided by the total number of voters in the grid. Consistent results are observed across different sizes of grid structures.

Figure 5.12 shows the comparison of the proposed algorithms to the traditional NMR approach. The data are obtained on a 30×30 grid structure. The x axis indicates the fault rates of units in the nanoelectronic environment; the y axis shows the ratio of failing voters, with the smaller percentage indicating the higher fault tolerance capability. For each fault rate of the units, the algorithms are repeated 50 times and the average results are shown.

The dotted curve shows the results of traditional NMR. Basically a voter can only utilize its initially assigned in-use voters and fails when its fault-free in-use unit number falls below three. The middle curve indicated by the "+" sign shows the results of performing the dynamic reconfiguration algorithm whenever a unit becomes faulty. The lowest curve with the "*" sign represents the result of performing the static reconfiguration algorithm with a predefined rate of injected defects.

It can be observed that both the dynamic and the static reconfiguration algorithm perform significantly better than a typical FMR approach without any flexibility. As is to be expected, the static algorithm outperforms the dynamic algorithm when the unit fault rate is high, which starts from around 0.4 as is shown in figure 5.12.

Figure 5.13 presents a zoom-in view of the comparison in the unit fault rate range [0, 0.2]. It can be observed that the proposed static and dynamic algorithms perform equally well and can almost guarantee the fault tolerance of all the voters within this

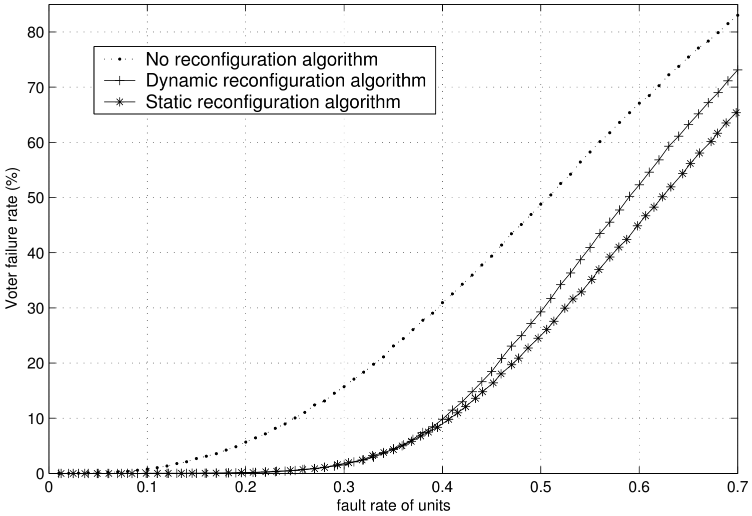


Figure 5.12: Comparison of Flexible five modular redundancy using static and dynamic reconfiguration vis-a-vis traditional five modular redundancy.

fault rate range.

5.4 Conclusions

The severe reliability challenge of future nanoelectronic systems imposes fault tolerance as a fundamental requirement, and furthermore requires considerations on: 1) high fault rate, 2) regular structured topology, and 3) strictly localized interconnect.

In this chapter, a model that captures the locality particularities in fault tolerance approaches, particularly in terms of redundancy sharing, is proposed. The model is applicable to any specific topology or layout of a nanoelectronic system, and serves as a basis to formulate both the defect and fault tolerance problems under locality constraints. We provide the associated algorithms for both defect and fault tolerance approaches on various system models with a detailed case study for a given topology on a grid topology. Particularly, the case study illustrates the approach of a flexible N modular redundancy scheme, with enhanced fault tolerance capability by introducing flexibility and adaptability exploiting the regular structure of nanofabrics.

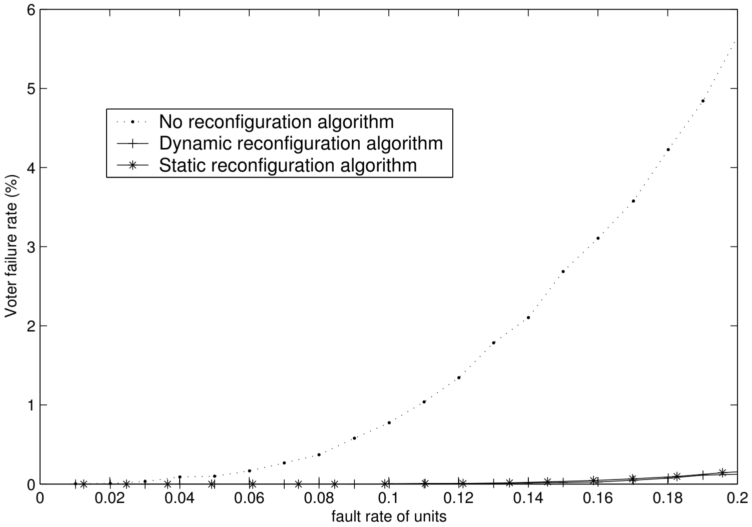


Figure 5.13: Comparison of Flexible five modular redundancy using static and dynamic reconfiguration vis-a-vis traditional five modular redundancy in 0-0.2 fault rate range.

Chapter 6

Defect Aware Logic Mapping in Nano PLAs

The bottom-up self-assembly fabrication process of nanoelectronics imposes high regularity on the nano fabric, indicating crossbar-based regular structured logic in nanoelectronic systems. While programmable logic arrays (PLAs) are naturally supported by such a regular structure, new challenges associated with the defects and connectivity emerge for the logic synthesis on nanocrossbar based PLAs.

Specifically, the strict topological constraints in the length and connectivity of nanowires, in addition to the massive defects in the nanofabric, lead to a fundamental difference between the nanoelectronic crossbar logic and the classic PLA structure. Mapping a logic function to the underlying nano crossbar can no longer be guaranteed successful, unless performed in a particular way that matches the specific topological constraints.

In this chapter we present the mathematical model for the problem of logic mapping in the nanoelectronic environment. Based on the model, we develop the corresponding algorithm to address the emerging mapping problem in nano PLA logic synthesis process. We further provide a number of enhancement techniques to improve algorithm runtime by significantly cutting down on the unnecessary backtracking pro-

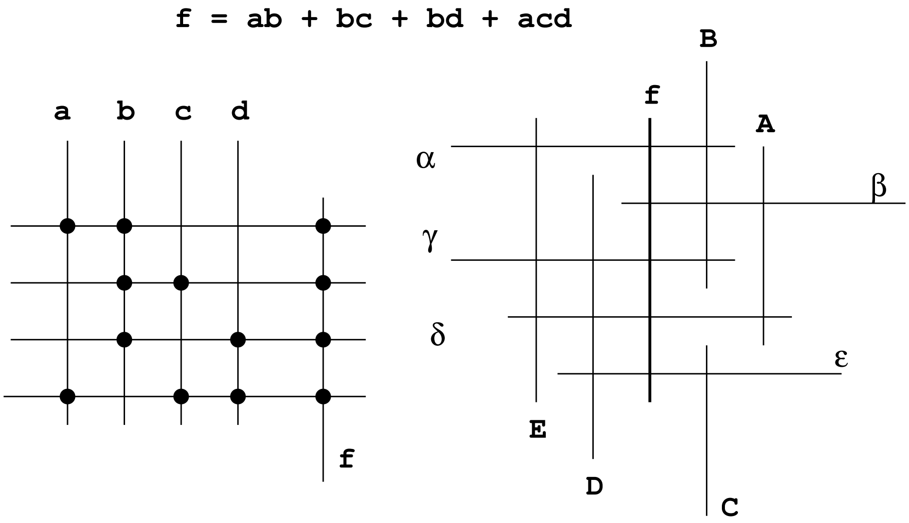


Figure 6.1: Logic function mapping in traditional PLA versus nanowire crossbars

cesses. Overall, a new topological constraint imposed by the nanoelectronic characteristics is identified and formulated. The established model and algorithm provide an important initial step for logic synthesis in the nanoelectronic environment.

6.1 Motivation

Evidently, mapping a logic function onto a regular PLA is quite straightforward since the PLA structure provides the full flexibility of mapping a logic variable to any vertical wire and mapping a product term to any horizontal wire. However, for a nanoelectronic crossbar structure, this mapping has to obey certain constraints, due to the existence of massive breaks. On the one hand, the nanowires are broken periodically due to the limited wire length and connectivity. The locations of these regular breaks are defined according to the particular crossbar design, and knowledge about them is thus available at the fabric time. On the other hand, a large number of random breaks are introduced by the manufacturing defects. These rather arbitrary breaks demand a post-fabric testing process to identify the locations.

Figure 6.1 provides an example of a logic function implemented in a traditional

PLA and a nanowire-based crossbar with a number of breaks in wires. In this example, if in the first step we map variable a to the vertical wire A, then the selection for the product terms ab and acd is immediately constrained to the horizontal wires $\{\beta, \delta\}$ since these are the only two wires crossing wire A.

It can be seen from this example that mapping a variable or a product term onto a specific nanowire imposes constraints on the subsequent mapping. Therefore, (i) determining whether a logic function can be successfully mapped to a specific crossbar with topology constraints and (ii) finding a mapping, if one exists, constitute important new challenges for logic synthesis on nanelectronic crossbars.

6.2 Logic mapping in nano crossbar

6.2.1 Bipartite graph model

The process of mapping a logic function onto a topologically constrained crossbar based nanofabric can be essentially modeled mathematically by mapping it to the matching problem of two *bipartite graphs*.

Definition: A *bipartite graph* $B(N, N', E)$ is a 3-tuple where N and N' are two disjoint sets of nodes and E is a set of edges. Every edge $e \in E$ connects a node $n \in N$ with a node $n' \in N'$.

Consider a two-level logic function in a sum-of-products form. The relationship between the logic variable set and the product term set can be represented by a bipartite graph, with node set N_{var} representing the logic variables and node set N_{prod} representing the product terms. In the logic function bipartite graph, there exists an edge between a node in N_{var} and a node in N_{prod} if and only if the corresponding product term contains the variable. Figure 6.2(a) illustrates the bipartite graph of the logic function shown in figure 6.1.

A crossbar architecture can also be represented by a bipartite graph, with one set of nodes representing the horizontal nanowires and the other set of nodes representing the vertical nanowires. A crosspoint connection between two orthogonal nanowires can then be represented as an edge in the bipartite graph. A nanowire crossbar with breaks in the regular structure (either periodic due to the topology constraints or arbitrary due to fabrication time defects) is shown in figure 6.1 and its bipartite graph representation is shown in figure 6.2(b).

When using a crossbar structure to implement a two-level logic function, the relationships between the product term set and the variable set in the logic function are represented by the connections between the horizontal wires and the vertical wires in the crossbar. The problem of mapping a two-level logic function onto a target nanowire crossbar entails matching each variable and each product term with specific nanowires in the crossbar structure, such that the relationship between a variable and the product term pair can be represented by activating the device at a crosspoint in the crossbar. This logic function to crossbar mapping problem can be formulated as embedding the logic function bipartite graph into the nanowire crossbar bipartite graph.

Definition: *bipartite graph embedding:* Given a logic function bipartite graph $B1(N_{var}, N_{prod}, E1)$ and a crossbar nanofabric bipartite graph $B2(N_{vert}, N_{hor}, E2)$, find a node mapping $(M : N_{var} \rightarrow N_{vert}; N_{prod} \rightarrow N_{hor})$ such that $\forall (n, n') \in E1, (n \in N_{var}, n' \in N_{prod}), (M(n), M(n')) \in E2$ holds.

The dotted lines on Figure 6.2(c) illustrate a valid mapping from the logic function bipartite graph in figure 6.2(a) into the crossbar bipartite graph shown in figure 6.2(b).

A CMOS PLA with full connectivity essentially represents a *complete bipartite graph* with an edge between every pair of nodes in the horizontal wire and vertical wire sets. Embedding an arbitrary bipartite graph onto a complete bipartite graph is straightforward, due to the full connectivity of the target complete bipartite graph. Consequently, mapping a specific product term or variable of a logic function onto a wire in the CMOS PLA does not impose any further constraints on the subsequent embedding steps. The process of mapping a logic function onto the CMOS PLA structure is thus trivial. On the contrary, the bipartite graphs corresponding to nano crossbars are not

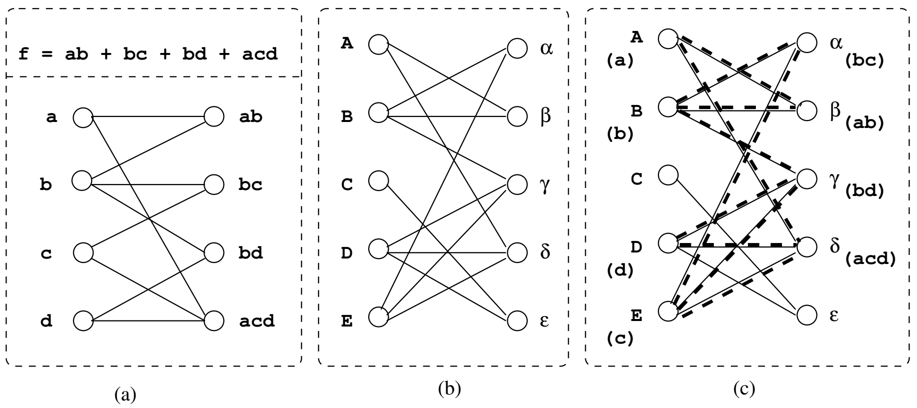


Figure 6.2: Bipartite graph representations of (a) a sum-of-product logic function and (b) a nanowire crossbar with (c) a successful embedding

complete bipartite graphs, due to breaks in the wires as discussed. Hence, the mapping of a specific node in the logic function bipartite graph onto a node in the nanowire crossbar bipartite graph imposes a number of constraints on the subsequent embedding steps. In general, the bipartite graph embedding problem can be reduced to the NP-complete problem of *Balanced Complete Bipartite Subgraph*, indicating the fact that this problem has exactly the same complexity as the problem of searching for the largest possible perfect sub-crossbar.

6.2.2 Embedding algorithm

We develop a recursive algorithm to embed the logic function bipartite graph $B1(N_{var}, N_{prod}, E1)$ into the nanowire crossbar bipartite graph $B2(N_{vert}, N_{hor}, E2)$, such that all the edges in $B1$ can be mapped to the existing edges in $B2$.

```
EMBED ($N1 \subseteq (N_{var} \cup N_{prod})$, $N2 \subseteq (N_{vert} \cup N_{hor})$)
//taking as input $B1$'s unmapped node set N1 and $B2$'s unmapped node set N2

    1. if ($N1 = \oslash$): //no more unmapped nodes in $B1$
            return success
    2. if ($N2 = \oslash$): //no more unmapped nodes left in $B2$ to map $N1$, thus fail
```

```
        **return fail**
3. select one unmapped node $n1 \in N1$
4. if ($n1 \in N_{var}$): *//variable → unmapped vertical wires*
        **repeat** steps (a) - (d) for every node $n2 \in (N_{vert} \cap N2)$
   else ($n1 \in N_{prod}$): *//product term → unmapped horizontal wires*
        **repeat** steps (a) - (d) for every node $n2 \in (N_{hor} \cap N2)$
           (a) justify $n1 \rightarrow n2$: if $n1$ can be mapped to $n2$, mark $n1 \rightarrow n2$;
               else repeat with a different $n2$
           (b) ***EMBED*** *($N1 - n1$, $N2 - n2$)*
           (c) if (success):
                   **return success**
           (d) else (fail): *//backtrack, try a different n2*
                   cancel the mapping from $n1$ to $n2$
5. **return fail** *//tried every $n2$ in the set, failed to map n1 to any possible node,
   therefore impossible to map $N1 \rightarrow N2$*
```

The above algorithm embeds $B1$ to $B2$ in two steps. First, it maps one node from $B1$ to $B2$. Then it recursively solves the reduced embedding problem with the remaining unmapped nodes. The critical part of the ***EMBED*** algorithm is the justification step in 4(a), which is responsible for pruning unsuccessful trials when traversing the solution space. The sufficient condition required to be checked at step 4(a) to perform the justification is as follows: assume $E1$ is the edge set of $B1$ and $E2$ is the edge set of $B2$, then a node $n1$ in $B1$ *cannot* be mapped to a node $n2$ in $B2$ if

> $\exists(\tilde{n1} \in B1$ and $\tilde{n2} \in B2)$, $\tilde{n1}$ already mapped to $\tilde{n2}$, $(n1, \tilde{n1}) \in E1$ while $(n2, \tilde{n2}) \notin E2$.

For an already mapped node $\tilde{n1}$ that is connected to $n1$ in $B1$, if the edge $(n1, \tilde{n1})$ cannot be embedded in $B2$, then such a mapping of node $n1$ does not lead to a valid solution. If all the unmapped nodes in $B2$ are justified as unmappable to $n1$, then the algorithm fails.

6.2.3 Heuristics to prune impossible mappings

For a particular $n1$ in $B1$, the recursive ***EMBED*** algorithm explores a number of solution subspaces, each determined by a possible selection of $n2$ from $B2$. The algorithm is capable of exploring all these subspaces for a solution by performing a sequence of trial mappings from $n1$ to each possible $n2$. To expedite the process of searching for a valid solution, it is crucial to identify and prune subspaces which have no solutions at an early stage. We propose three heuristic justification criteria on top of the basic sufficient condition in step 4(a), so as to enhance the pruning capability, and improve the algorithm runtime.

Fanout embedding heuristic

According to the constraint imposed in the mapping process, when $n1 \in B1$ is mapped to $n2 \in B2$, the fanout nodes of $n1$ are deemed to be mapped within the fanout nodes of $n2$. Therefore, if the number of edges connected to $n1$, i.e., the degree of $n1$, exceeds the degree of $n2$, then such a mapping is deemed to fail as it cannot lead to any valid solution. In the example of figure 6.2, node c in 6.2(a) can be mapped to node A, B, D, E, but not C in 6.2(b) according to this criterion, since node c in 6.2(a) has a degree of 2 while node C in 6.2(b) has a degree of 1.

Furthermore, during the mapping process, it is possible that a subset of the fanout nodes of $n1$ as well as a subset of the fanout nodes of $n2$ have been mapped already. Therefore, a more precise justification should consider the fanout nodes that have not yet been mapped. Basically, if the number of unmapped fanout nodes in $n1$ is larger than the number of unmapped fanout nodes of $n2$, then $n1$ cannot be mapped to $n2$. Otherwise, we denote that $n1$ can be *fanout embedded* to $n2$. For m the total number of nodes in $B1$ and $B2$, the justification process of $n1$ to $n2$ is of $O(m)$ time.

Fanout-fanout embedding heuristic

When $n1$ in $B1$ is mapped to $n2$ in $B2$, then the unmapped fanout nodes of $n1$, denoted as $\Phi(n1)$, are to be mapped within $n2$'s unmapped fanout nodes, denoted as $\Phi(n2)$. If $n1 \rightarrow n2$ leads to a valid solution, then there should exist a valid mapping $\Phi(n1) \rightarrow \Phi(n2)$. If there is no mapping to guarantee a *fanout embedding* from every node in $\Phi(n1)$ to $\Phi(n2)$, then $n1 \rightarrow n2$ does not lead to any valid solution.

Consider a simple example in figure 6.2. Node ab in figure 6.2(a) has 2 unmapped fanout nodes a and b with out degrees 2 and 3. Node ε in figure 6.2(b) has 2 unmapped fanout nodes C and D with out degrees 1 and 3, respectively. Although ab and ε have the same number of unmapped fanout nodes, mapping ab to ε does not lead to any valid solution, since there is no way to map $\Phi(ab)$ to $\Phi(\varepsilon)$.

The justification process for $n1$ with $n2$ has a time complexity of $O(m^2)$. Going through sorted $\Phi(n1)$ and $\Phi(n2)$ takes $O(m \log m)$, which is dominated by the calculation for the degree numbers in $\Phi(n1)$ and $\Phi(n2)$, which takes $O(m^2)$.

Fanout chain embedding heuristic

Suppose we have a bipartite graph $B1(N1, N1', E1)$ with node $n1 \in N1$; then the fanout node set of $n1$, denoted as $\Psi(n1)$, is a subset of $N1'$. We can observe that the set $\Psi(n1)$ has a fanout set that contains nodes that are connected to any member of $\Psi(n1)$. The fanout set of a node set N is defined as $\widehat{\Psi}(N) = \bigcup_{n \in N} \Psi(n)$.

Assume that $n1$ is mapped to $n2$ in $B2(N2, N2', E2)$; then not only is $\Psi(n1)$ deemed to be mapped within $\Psi(n2)$, but also the fanout set of $\Psi(n1)$ is to be mapped within the fanout set of $\Psi(n2)$. Therefore, $n1 \rightarrow n2$ implies $\Psi(n1) \rightarrow \Psi(n2)$, which further implies $\widehat{\Psi}(\Psi(n1)) \rightarrow \widehat{\Psi}(\Psi(n2))$, $\widehat{\Psi}(\widehat{\Psi}(\Psi(n1))) \rightarrow \widehat{\Psi}(\widehat{\Psi}(\Psi(n2)))$, and so on. For ease of representation, we define the following:
$\widehat{\Psi}^1(N) = \widehat{\Psi}(N)$; $\widehat{\Psi}^{i+1}(N) = \widehat{\Psi}(\widehat{\Psi}^i(N))$.

The sequence of $\widehat{\Psi}^i(N)$ is denoted as a *fanout chain* and an example is shown in figure 6.3. Figure 6.3(a) shows the two bipartite graphs and figure 6.3(b) shows the

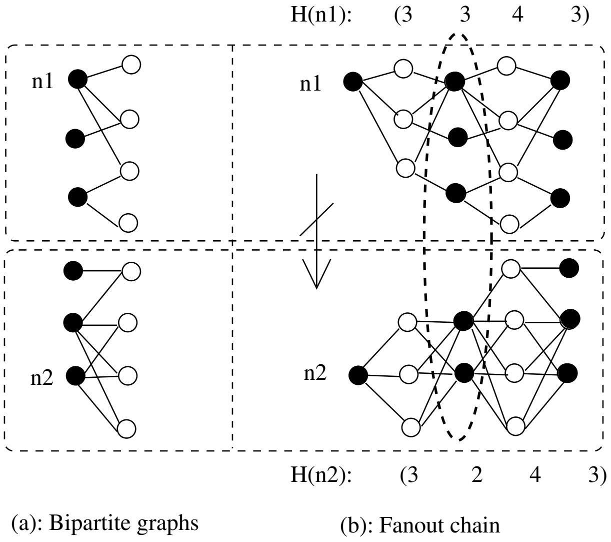


Figure 6.3: Fanout chain embedding example

fanout chains of node $n1$ and $n2$ for the two bipartite graphs. We define *height chain* as is shown in figure 6.3(b):

$$H(n1) = (|\widehat{\Psi}^1(n1)|, |\widehat{\Psi}^2(n1)|, |\widehat{\Psi}^3(n1)|, ...)$$

A necessary condition for the mapping of a node $n1$ to a node $n2$ to lead to a valid solution is that the fanout chain of $n1$ must be mapped into the fanout chain of $n2$. The height chain of $n1$ consequently needs to be smaller at every point than the height chain of $n2$. As is shown in the example of figure 6.3, $n1$ cannot be mapped to $n2$ since $H_2(n1) = 3$, which is larger than $H_2(n2) = 2$.

Based on the information of *height chain*, the following condition is sufficient to determine that node $n1 \in B1$ *cannot* be mapped to a node $n2 \in B2$: $\exists i$ such that $H_i(n1) > H_i(n2)$. Basically, if at any point the height chain of $n1$ is greater than the height chain of $n2$, then the fanout chain of $n1$ cannot be embedded into the fanout chain of $n2$. Otherwise, there is a *fanout chain embedding* from $n1$ to $n2$.

For any node $n1 \in N1$ in $B1(N1, N1', E1)$, it can be shown that $H_{2i}(n1)$ and $H_{2i-1}(n1)$ are two monotonically increasing functions, with the upper bound for $H_{2i}(n1)$ being $|N1|$ and the upper bound for $H_{2i-1}(n1)$ being $|N1'|$. Furthermore, these two functions reach their maximum values within $MAX(2|N1|, 2|N1'|)$ steps. This implies that the height chain for any node in a bipartite graph reaches a maximum value in a limited length bounded by $MAX(2|N1|, 2|N2|)$ and remains constant afterwards. Therefore, calculating the height chain of any node takes time $O(m)$ where m is the number of nodes in the two bipartite graphs $B1$ and $B2$. In fact, since the degree of each node is constant, the Ψ height chain needs to be calculated only once before the algorithm starts.

6.2.4 Heuristic justification summary

Figure 6.4 shows an overall illustration of the three heuristic conditions described in the previous subsection. Basically, as is shown in figure 6.4a, the *fanout embedding* heuristic prunes illegal mappings according to the fanout number of $n1$ and $n2$. The *fanout-fanout embedding* heuristic prunes illegal mappings according to the fanout number of all the fanout nodes of $n1$ and $n2$, thus performing more aggressively by gathering information one step further beyond $n1$ and $n2$, as is shown in figure 6.4b.

Invoking advance justification can help aggressively reduce the number of subspaces to explore by processing larger amounts of information ahead. However, the more aggressive, the more time overhead the justification process itself requires. It is certainly possible to go beyond the *fanout-fanout embedding*, to implement more aggressive justification criteria. However, the number of cases to investigate for such an advancement in justification grows exponentially. Essentially, a complete justification process that detects all the invalid searching subspaces has exactly the same complexity as the bipartite graph embedding problem itself. In the application of the proposed algorithm, the level of justification that achieves the best run-time depends on various issues such as the size of the logic function and the connectivity of the nano crossbar; thus the decision should be made according to the specific trade-off points.

Figure 6.4c shows an example of the fanout chain embedding justification, which instead of performing aggressive precise justification targeting individual nodes, keeps

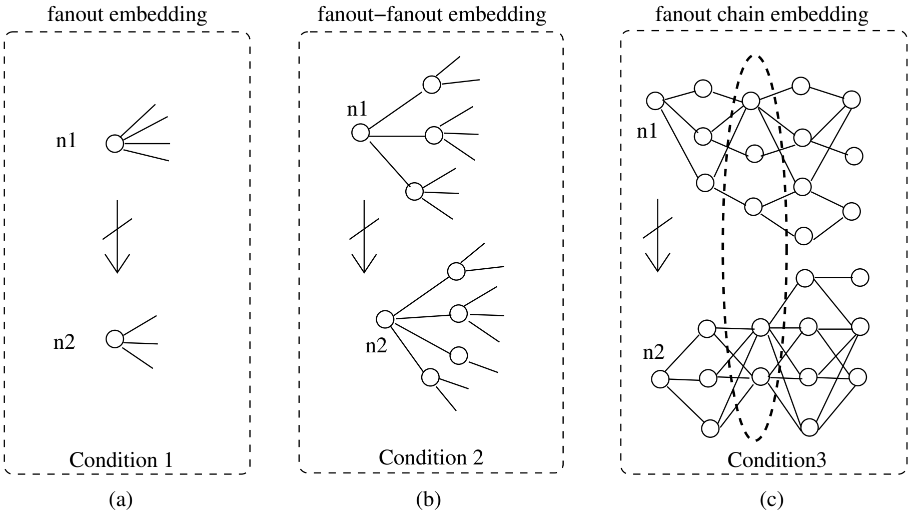


Figure 6.4: Examples for all the heuristic justification conditions

track of the quantitative variation regarding a set. Therefore, it avoids the tracking of exponentially increasing the number of possibilities when targeting individual nodes, thus enabling it to monitor the entire chain of the fanout set for a particular node with very low computational overhead.

The proposed fanout chain embedding justification essentially performs a fanout embedding justification on each point of the chain to form a height chain. It is certainly possible, however, to extend this approach, by applying a more aggressive justification such as the fanout-fanout embedding on each chain point, forming a more complex structure than the proposed height chain. Doing so, again, represents a tradeoff between the computational complexity of performing the justification and the reduction in the backtracking processes in the algorithm.

6.3 Experimental results

The proposed algorithm is implemented in C++. While the *basic* version solely uses the *fanout embedding heuristic*, the *advanced* version uses all the three heuristics to cut down on backtracking significantly. The algorithm is run on a computer with 2.8GHz

CPU and 1GB memory, under the Linux operating system. Table 6.1 summarizes the experimental results of embedding the logic function bipartite graph onto a nanowire-based crossbar bipartite graph for four logic synthesis benchmarks from the LGSynth93 benchmark set [50].

For each of the benchmarks, we use multiple nanowire-based crossbars (each representing a different defect map or topology constraints) by using the corresponding bipartite graph. The size of the crossbar (*size* column in table) is determined according to the number of variables and product terms in the logic function. The connectivity of the crossbar (*degree* column in table) is defined as the fixed number of nodes that each node is connected to. The crossbar size increases with the crossbar degree. The *find mapping* column lists whether the algorithm has successfully found the mapping and the *run-time* column illustrates the number of seconds spent in running the algorithm for the particular benchmark configuration. We set a runtime limit of 1200 seconds for both versions of the algorithm. When this limit is exceeded, the algorithm returns a failure indication and is denoted as "> 1200" in the table.

It can be seen from table 6.1 that the success of the mapping of a logic function onto a nanowire-based crossbar and the running time of the embedding algorithm depend heavily on the connectivity of the crossbar. This is true for both the basic and the advanced versions.

- When the connectivity is exceedingly low, the search space for exploration is limited. Therefore, it is easy to justify at the decision point that none of the search spaces leads to a solution. Obviously, the algorithm fails to return a successful mapping within a very low run-time under this case.
- As connectivity increases, the search space grows exponentially. Backtracking is required to explore the solution space for a possible solution. The runtime of the algorithm then sharply increases with the increased connectivity. Under this case, where the search space is huge with only a limited number of solutions, the search process takes an exceedingly long time. When connectivity further

increases, so do the number of valid solutions in the search space. Therefore, the identification of solutions while exploring the solution space is straightforward.

Although both versions show similar behavior, increasing connectivity levels for the underlying crossbar favor the *advance* algorithm, which utilizes all three heuristics, cuts off backtracking processes and is thus significantly faster than the *basic* algorithm. In fact, in a number of cases, the *basic* algorithm runs for more than the preset time limit (1200s) without returning any successful mapping while the *advance* algorithm returns a valid solution within a few hundred seconds.

6.4 Conclusions

With nanoelectronic technologies evolving at a rapid clip towards practical realization of nanoelectronic computing systems, it is important to identify and propose research on the new challenges emerging during their design process. In this chapter, a new challenge of mapping a logic function onto a nanowire crossbar is identified, under the inherent constraints of limited connectivity and unreliability. The research work presented in this chapter sets up an important initial contribution in the logic synthesis for nanoelectronics, opening up multiple future research directions, including the decomposition of large logic functions onto nanoelectronic crossbar structures and an incremental bipartite graph creation integrated with the defect testing procedure of the nanofabrics.

Table 6.1: Experimental results for selected logic synthesis benchmarks

PLA	# of	Crossbar	Crossbar	find mapping?		run-time (s)	
circuit	inputs	size	degree	basic	adv	basic	adv
rd53	10	11	4	×	×	17	1
		13	5	×	×	149	16
		15	6	×	×	909	114
		17	7	√	√	1	1
		21	9	√	√	3	1
		25	11	√	√	6	0
misex1	16	16	4	×	×	581	81
		18	5	×	×	> 1200	> 1200
		20	6	√	√	1072	4
		22	7	√	√	768	80
		24	8	×	√	> 1200	203
		26	9	×	√	> 1200	441
5xp1	14	18	3	×	×	1	1
		20	4	×	×	103	1
		22	5	√	√	19	13
		24	6	√	√	13	12
		26	7	√	√	6	2
		28	8	√	√	7	2
bw	10	13	3	×	×	1	1
		15	4	×	×	1	12
		17	5	×	×	> 1200	> 1200
		19	6	×	×	> 1200	> 1200
		21	7	×	×	> 1200	> 1200
		23	8	×	√	> 1200	66
		25	9	×	√	> 1200	32
		27	10	×	√	> 1200	52
		29	11	×	√	> 1200	79
		31	12	×	√	> 1200	115

Chapter 7

Applying Error Checking Code on Carry-Save Arithmetic

This chapter focuses on a new approach for fault tolerant arithmetic computations for nanoelectronic systems, by exploiting the emerging new characteristic of Negative Differential Resistance (NDR), which has been displayed in multiple nanoelectronic device candidates. Basically, the NDR characteristic can be used to support representational redundancy in the form of multi-valued logic (MVL), based on which Carry Save Arithmetic (CSA) can be implemented, not only to provide significant speedup in computations, but also to enable efficient fault tolerance approaches to be applied. In this chapter, we show how a linear block code based information redundancy fault tolerance scheme can be applied to the CSA operations in the NDR based nanoelectronic systems. We further examine the design space in implementing the proposed scheme and illustrate that an overall optimal configuration can be achieved in the representational redundancy and hardware redundancy implemented by using multiple bits.

It is widely known that linear block code based schemes can provide an optimal fault tolerance capability with the amount of redundancy given, yet their application is typically restricted to domain of data transfer and storage subsystems. Therefore, by proposing the linear block code based fault tolerance scheme in the arithmetic components, we envision a unified fault tolerance approach across the multiple subsystems of arithmetic, storage and communication in a nanoelectronic system.

7.1 Preliminaries

Hardware redundancy is one commonly used approach to enhance reliability. Dynamic faults occurring during the runtime of the system demand much more aggressive fault tolerance schemes than the static manufacturing defects. Hardware duplication based approaches applied for the fault tolerance requirement typically use a majority vote among multiple redundant computations. Illustrative fault tolerance strategies of this type include R-fold module redundancy and NAND multiplexing [85]. Since large amounts of computational resources are available in nanotechnology based systems, hardware redundancy has been perceived as an attractive approach. Circuit-level fault tolerance for signal processing [25], probabilistic construction of logic gates [60], NAND multiplexing [30, 31, 66], the flexible NMR approach [70] and the hardware/time hybrid redundancy approach [67, 68] constitute approaches in this direction at various hierarchical levels. However, research has shown that in order to achieve an acceptable level of reliability enormous amounts of hardware redundancy (in the order of 10^3 to 10^5) might be necessary to cope with failure rates in the order of 10^{-2} to 10^{-1} [62].

Recomputation in time is an alternative approach to concurrent error detection and correction to guard against transient faults [64, 67, 68, 84]. Since the computation is performed a second time with the same unit, this approach requires less hardware overhead. However, time redundancy based fault tolerance delays the computation, thereby compromising system performance. Furthermore, the application of a time redundancy based approach on permanent faults demands subsequent online diagnosis and reconfiguration phases, which are time and hardware consuming.

Information redundancy is widely used to implement fault tolerance in communication and storage systems. Error detection and correction are performed by organizing the redundant information in the data according to a strict algebraic structure.

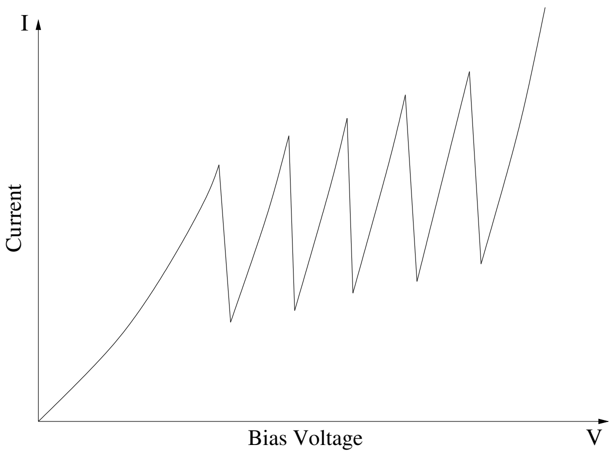


Figure 7.1: Multipeak characteristic of I-V curves of negative differential resistance nanotechnologies

In CMOS technology, several information redundancy based arithmetic fault-tolerance techniques have been developed [61, 65, 83]. Coding based fault tolerance approaches have been proposed on highly regular logic blocks such as PLA-like crossbar structures in nanoelectronic systems [77].

7.1.1 NDR based Multi-valued logic

Despite the fact that the dominant systems implemented under CMOS technology are binary based, various research on the topic of multi-valued logic has been proposed due to its capability of reducing interconnection complexities and the enhanced storage of information per unit area [9, 17, 35, 37, 40]. Nanotechnologies such as resonant tunneling diodes (RTD), resonant tunneling transistors (RTT) and molecular electronics have been investigated for their potential to enable ultra-high speed and extremely dense circuits [10, 55]. The current-voltage curves in these nanotechnologies display multiple negative differential resistance (NDR) areas. Figure 7.1 shows an example of the multipeak current-voltage curve with five peaks.

The multiple-folded nature of the current-voltage curves in these NDR nanotechnologies facilitates compact multi-valued logic circuit implementations as an N-peak

NDR device can support $N+1$ logic states. [3, 56, 86]. Due to the high switching speed and the folded nature of the I-V curves, multi-valued logic with a large number of logic states has been shown to be of practical importance [90]. Fifteen state SRAM cells [90], six state counters [49] and multiple-state logic gates [73, 87] have been designed. Multi-valued logic based arithmetic component designs have also been proposed [28]. It can be therefore projected that future nanoelectronic systems based on these NDR devices are highly likely to be constructed with multi-valued logic systems, which provide high representational capability and with less interconnection congestion.

7.1.2 Carry Save Arithmetic

The carry save technique has been developed as a classic strategy to cope with the carry propagation delay problem in arithmetic unit design [45, 63, 88]. Carry save arithmetic uses a redundant number system to absorb the carry bits and further eliminate the carry propagation entirely. Carry save arithmetic operations can be performed with a constant delay regardless of the number of bits in the operands, making this technique extremely appealing in enhancing performance. The elimination of carry propagation also provides potentials in applying coding based fault tolerance techniques [65]. Recent research has explored the design of binary logic based carry save addition using 1-out-of-3 codes [83].

To clarify the general carry save principle, an example of carry save addition is shown in figure 7.2. It can be observed from the example that the sum of the two operands is initially represented as a *position sum*, where the carry bits are absorbed using the representational redundancy. Each digit of the position sum is then decomposed into two digits, the *interim sum* and the *transfer digit*, based on the radix. In the last stage, the final sum digits are obtained by adding up the interim sums and the transfer digits. It can be seen from the example that, due to the carefully selected operand digit range [0, 11] and the radix 10, the final addition of an interim sum digit (within the range [0, 9]) and a transfer digit (within the range [0, 2]) always falls in the operand

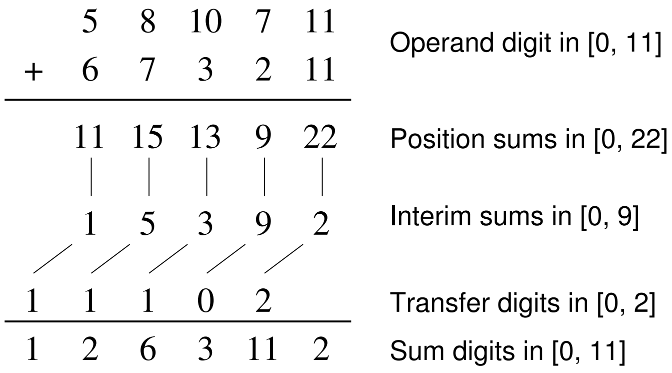


Figure 7.2: Carry save addition supported by a 23-valued NDR nanotechnology.

digit range [0, 11], thus introducing no carry propagation. Overall, carry propagation is entirely eliminated throughout the whole carry save addition process.

In general, consider a carry save addition of two operands in radix r on a redundant number system where each digit can be represented in a range larger than r. The operand digits are set to be in the range [0, α] and the sum of any two operand digits falls in the range [0, 2α]. This sum vector with the digits in the range [0, 2α] is typically represented without any carry digit and denoted as the *position sum*. The position sum needs to be translated back into the range [0, α], and is decomposed into an *interim sum* and a *transfer digit* according to the radix r, where:

$$\text{position sum} = \text{transfer digit} \times \text{radix} + \text{interim sum}$$

The interim sum digits are therefore in the range [0, $r - 1$] and the transfer digits are in the range [0, $\lfloor (2\alpha)/r \rfloor$]. If the following equation can be satisfied:

$$\lfloor (2\alpha)/r \rfloor + r - 1 \leq \alpha \tag{7.1}$$

then adding an interim sum digit to a transfer digit yields a final sum digit in the range of [0, α] without generating any carry. Therefore, the sum of the two operands is computed without generating any carry propagation with the sum digits remaining in the operand digit range [0, α]. In general, to perform a single carry save addition for n operands without introducing any carry propagation, equation 7.1 needs to be modified to:

$$\lfloor (n\alpha)/r \rfloor + r - 1 \leq \alpha \tag{7.2}$$

In binary logic (where $r = 2$) inequalities 7.1 and 7.2 do not have any positive solutions for α, hence making it infeasible to perform carry save addition. Instead, the carry save technique is only used to reduce the number of operands (typically from 3 to 2) in multi-operand additions. Thus, carry save addition is typically used as an important building block to speed up the multi-operand additions in binary multipliers.

On the other hand, since NDR nanotechnologies can implement multi-valued logic with a sizable number of logic states, they can support redundant number systems that satisfy inequality 7.1 and even further inequality 7.2 with a certain n. General addition can then be performed without carry propagation. When applied in the multiplication operation, the carry save technique can speed up the intermediate multi-operand addition as well as the final sum calculation.

7.1.3 Linear Block Codes

A linear block code $\mathcal{C}(n, k)$ encodes a k-digit[1] dataword into an n-digit codeword ($n > k$). The codewords are constructed with a strict algebraic structure enforced, by exploiting the redundancy in the coding space. The $(n - k)$ redundant digits are used to enforce the algebraic structure for each valid codeword [7]. Specifically, a linear block code is constructed based on a *finite field* $GF(q)$ of q numbers. The field operations of addition, subtraction, multiplication and division can be performed on the field elements. A well constructed linear block code can achieve the maximum fault tolerance capability provided by the redundant information. For example, a (7, 4) Hamming code is a linear block code on $GF(2)$ with $n = 7$ and $k = 4$ using three digits of redundant information to provide 2-digit error checking and 1-digit error-correction capability for any 4-digit dataword. Linear block codes have been widely applied in communication systems due to their well-defined structure and scalability: the level of error detection/correction capability can be precisely determined by a number of systematic ways to form the algebraic structure [7].

[1] We use *digit* instead of *bit* so as to indicate that the number system is based on multi-valued logic, rather than limited to binary logic.

The algebraic property of a finite field $GF(q)$ requires that $q = p^m$, where p is a prime number and m is a positive integer. The number of field elements is either a prime number p (i.e. $m = 1$) or a power of a prime number p (i.e. $m > 1$). When $m = 1$, the field operations addition and multiplication are identical to the modulo p operations of arithmetic addition and multiplication. Therefore, when a finite field is constructed from a prime number of elements, the field operations within the range of p are identical to the corresponding arithmetic operations.

A linear block code $\mathcal{C}$ is a subspace of the linear vector space $GF(q)^n$. Each element of $\mathcal{C}$ (i.e., an n-digit valid codeword) is a vector on the finite field $GF(q)$. According to the properties of linear block codes, the linear combination of any two codewords is always a valid codeword. The zero vector, since it is a member of any linear vector space, is always a valid code. A codeword can be generated by multiplying a dataword with a *generator matrix* G. The generator matrix G can be represented in a *systematic form* of $G = [I|P]$, where I is the identity matrix and P is a $k \times (n-k)$ matrix. When generated by a systematic matrix form, the resulting codeword has the original dataword as its leftmost k digits.

The Hamming distance of two codewords v_1 and v_2 is the number of different digits between v_1 and v_2. The Hamming weight of a codeword v is the number of non-zero digits in it. A basic lemma shows that, if d is the minimum Hamming weight over all non-zero codewords in $\mathcal{C}$, then the Hamming distance between any two codewords must be a multiple of d. The minimum Hamming distance of any two valid codewords is also d.

The set of vectors that span the vector space orthogonal to $\mathcal{C}$ constitutes the check matrix H for $\mathcal{C}$. Since H is the orthogonal complement of G, multiplying a valid codeword by H always generates a zero vector. A non-zero result, defined as a *syndrome vector*, indicates an invalid codeword and can be used to identify the corresponding valid codeword from a syndrome table. Suppose an error occurs and changes e digits of a codeword c into c': if $e < d$, then the error is detectable since the error vector c' cannot be a valid codeword. If $e \leq \lfloor (d-1)/2 \rfloor$, then the erroneous codeword can be corrected to the codeword with the minimum Hamming distance to c'.

7.2 Motivation

To guarantee online fault tolerance for the nanoelectronic arithmetic components, we propose an information redundancy based carry save arithmetic approach as a practical technique for the emerging nanotechnologies based on the following observations.

- In emerging systems based on nanotechnologies, fault tolerance schemes are required at various design hierarchical levels and in multiple subsystems. Fault tolerance strategies are required to ensure reliable data transfer and storage. Linear block code based approaches have been successfully applied in the data communication and storage area for online fault tolerance as mature techniques due to their optimality in checking/correcting errors and the existing systematic way to construct them. It can be envisioned that in a future nanoelectronic system, linear block codes are inevitably required to support reliable data transfers over unreliable interconnections and to guarantee the correctness of the data stored in the memory blocks. Therefore, if the same technique can be extended to the arithmetic components, then the existing encoding/decoding hardware for the data transfer and storage systems can be utilized without adding extra overhead. Furthermore, with a unified linear block code interface, arithmetic components can directly utilize the codewords transferred or stored and perform the fault tolerant arithmetic operations without delay. Overall, a unified linear block code based fault tolerance approach to interconnection, storage and arithmetic subsystems can eliminate the intermediate encoding/decoding process at the interfaces of the multiple subsystems and effectively minimize encoding/decoding time and the associated hardware overhead in the entire system.

- Negative Differential Resistance (NDR) is an important characteristic of a number of emerging nanoelectronic devices, thus making them naturally support multi-valued logic with large numbers of logic states. In a multi-valued logic based system, a native digit-level redundant number system can be easily implemented. Carry save arithmetic is then ideal for providing a significant performance boost by eliminating the carry propagation and achieving constant delay in the basic addition operation.

Traditionally, despite its significant advantage in the speedup for computation, carry save arithmetic is not popular among CMOS based binary systems due to its requirement of being processed based on a high radix of three or more. With the emerging nanoelectronic devices exhibiting the NDR characteristics, carry save arithmetic can be constructed on a multi-valued logic system which naturally supports the high radix computations.

- With carry save arithmetic, carry propagation is eliminated. This provides significant potential for applying information redundancy based fault tolerance schemes. Online faults are not as prominent a problem in the CMOS systems; thus the benefit of eliminating carry propagation for fault tolerance purposes is not as significant in the CMOS systems. However, as nanoelectronic devices are highly unreliable and online faults are projected to be significantly high, aggressive fault tolerance schemes become the fundamental requirement of constructing the future nanoelectronic systems.

  In general carry save arithmetic, the basic addition process is split into three stages, each without any propagation of carries. Within each stage the addition operation can be mapped to an identical field addition operation when a proper finite field is carefully chosen. The arithmetic operation therefore can be checked with the algebraic structure of a finite field and a linear block code can be therefore applied for fault tolerance purposes.

The main challenges in applying linear block code based fault tolerance in carry save arithmetic under the nanoelectronic environment essentially consist of the following three aspects.

1. The elimination of carry propagation in carry save arithmetic makes it possible to maintain the strict algebraic structure necessitated for linear block codes. However, the basic addition process in a CSA computation is split into three stages, which include not only addition, but also a decomposition phase. Consequently, challenges remain as of how to apply linear block codes on all the stages to guarantee the fault tolerance throughout the computation process.

2. The construction of a fault tolerant CSA on multi-valued logic involves a number of parameters, including the number of logic states for the multi-valued logic, the range of radix and operands in the CSA configuration, as well as the finite field construction parameters in the linear block code. Not only do these parameters need to be carefully considered together to make the system work, but also there exist multiple possibilities in the design space, depending on the selection of the various parameter combinations. Therefore, achieving optimality in constructing such a fault tolerant system remains challenging.

3. The fault tolerant capability of such an approach needs to be evaluated and verified. This is particularly important for the nanoelectronic environment, as reliability becomes a fundamental concern.

In the following sections, we address these challenges by proposing a linear block code based fault tolerance scheme on nanoelectronic CSA components. We further provide a discussion on parameter selection to attain system optimality, as well as a proof for the fault tolerance capability of the proposed approach.

7.3 Fault tolerant carry save arithmetic

In this section we present the basic theory of the proposed approach by first describing the mathematical underpinnings with an example, followed by the tradeoff

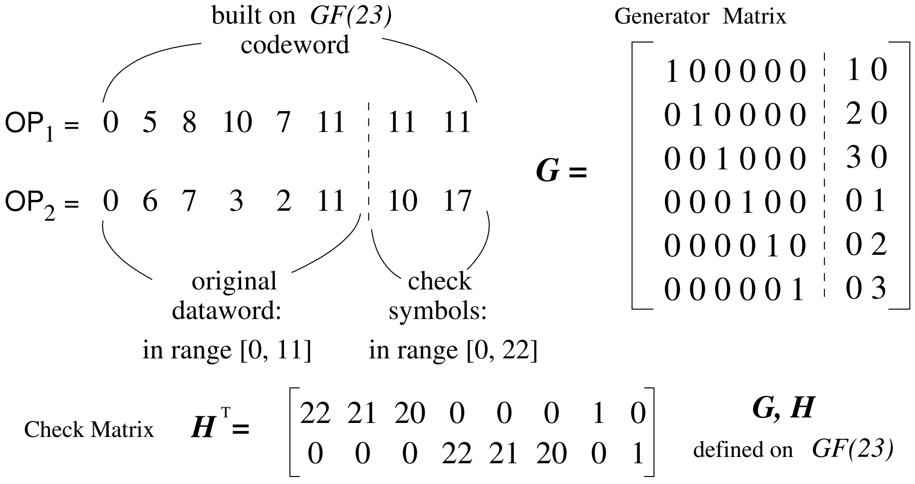


Figure 7.3: An example linear block code on $GF(23)$. The codeword is composed of four data digits and two check symbols.

discussion on the implementation aspect.

7.3.1 Linear Block Code based Error Checking in Carry Save Addition

We use the running example of figure 7.2 that is based on the configuration of $(r = 10, \alpha = 11)$. In order to construct an algebraic structure where the field operation is identical to the arithmetic operation for error checking, we need to choose a finite field with a prime number of elements to construct a linear block code. In this example, we use the prime number 23 to construct the finite field $GF(23)$, i.e., $q = 23$, such that the position sum with the largest range of $[0, 22]$ can be covered. Since the final sum might be one digit longer than the operands, we extend the operands by adding a zero to the left.

The generator matrix of a linear block code on $GF(q)$ can be constructed in a systematic form according to a given specification of how many faults need to be checked/corrected at the same time. Figure 7.3 shows an example generator matrix G, check matrix H and two codewords for the operands. The code symbols are generated

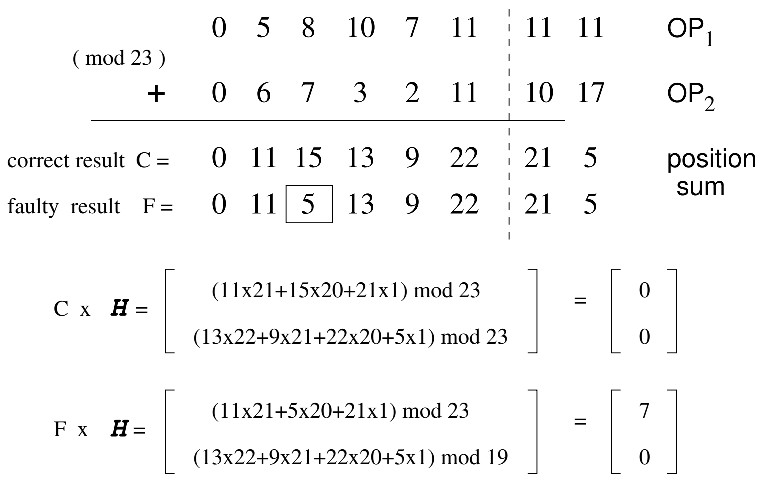

0 5 8 10 7 11 11 11 OP1
(mod 23)
+ 0 6 7 3 2 11 10 17 OP2
correct result C = 0 11 15 13 9 22 21 5
faulty result F = 0 11 5 13 9 22 21 5
position sum
C x H = (11x21+15x20+21x1) mod 23
(13x22+9x21+22x20+5x1) mod 23
= 0 0
F x H = (11x21+5x20+21x1) mod 23
(13x22+9x21+22x20+5x1) mod 19
= 7 0

Figure 7.4: Error checking during position sum calculation entails multiplying the check matrix H with the position sum.

by multiplying the six data digits with G. For example, for the data word OP_2, the first check symbol is obtained as $10 = (6 \times 2 + 7 \times 3) \bmod 23$ by multiplying it with the second to last column of G. Since the generator matrix G is in a systematic form, the left six digits of the codeword are identical with the dataword and the rightmost two digits serve as the check symbols. According to the linear block code theory, the validity of a codeword can be checked by multiplying it with the check matrix H and checking if the result is zero.

As can be seen from the example in figure 7.2, a carry save addition is performed in three stages:

Stage 1: The position sum is calculated by adding the two operands digit by digit.

Stage 2: The position sum is decomposed into an interim sum and the transfer digits.

Stage 3: The interim sum digits and the shifted transfer digits are added to generate the final sum.

Error checking during the position sum calculation

Figure 7.4 explains the error checking during position sum calculation using the running example. The two 8-digit operand codewords are added to generate an 8-digit position sum. Since each position sum digit is within the range of $GF(23)$, this addition process is identical to the field addition, i.e., modulo 23 addition. The correctness of the position sum calculation can be verified by checking whether the result codeword is valid using the check matrix H. Basically, a valid codeword, when multiplied with the check matrix H under a field operation, should yield a zero vector. Figure 7.4 also shows the situation when an error occurs in the third digit of the position sum, thus changing the correct value 15 to the faulty value 5. When this faulty position sum is multiplied by the check matrix H, it results in a non-zero syndrome vector (7, 0), indicating an error. This syndrome vector can also be used to determine the correct result using a syndrome table [7].

Error checking the position sum decomposition

In the second stage of a carry save addition, the position sum is decomposed into the interim sum and transfer digits according to the radix r. Unfortunately, such a 1-to-2 decomposition differs from the addition process, which is inherently a linear combination process. Decomposing the check symbols according to this transformation does not yield correspondingly two valid codes. Therefore, we cannot directly apply the validity checking of linear block code. This can be observed from the example shown in figure 7.5 that the check symbols of the interim sum, (17, 4), are not the decomposition results of the position sum check symbols (21, 5) based on radix 10.

Nevertheless, applying a linear block code for fault tolerance necessitates the exploitation of the linearity in the operation. We make the observation that such a decomposition process is basically a reverse computation of the addition process, since the position sum is essentially a linear combination of the interim sum and the transfer digits. Based on this observation, fault tolerance in this stage can be approached in the opposite direction of the actual computation flow; Figure 7.5 shows the error checking procedure. Consider a right-shift of the transfer digit vector by one digit and magnified by the radix as a *radix vector*, and the position sum is exactly the sum of the radix vector and the interim sum vector. Error checking and error correction can be then achieved by performing the following checking: adding the code symbols of the interim sum with the code symbols of the radix vector should generate the corresponding code symbols of the position sum. The check symbols of the radix vector can be constructed by obtaining the check symbols for the transfer digits first and then performing a field multiplication by the radix.

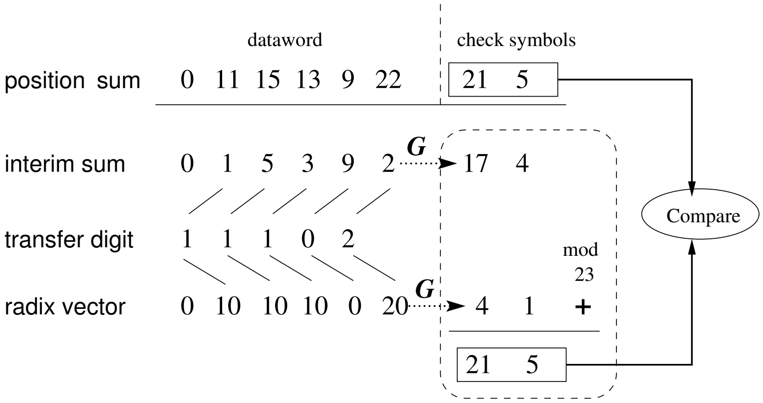


Figure 7.5: Error checking for the position sum decomposition stage entails checking if codewords for the transfer digit vector and the radix vector add up to the check symbols of the position sum.

Error checking the final sum

The final sum is obtained by adding the interim sum with the transfer digits. Since each interim sum digit is in the range [0, 9] and each transfer digit is in the range [0, 2], every digit in the final sum is in the range [0, 11]. Since this addition is identical to the field addition on $GF(23)$, the algebraic structure of the linear block code is preserved. In a manner identical to Stage 1, faults in this stage can be detected by checking the validity of the final sum codewords. This error checking strategy is summarized in

Figure 7.6.

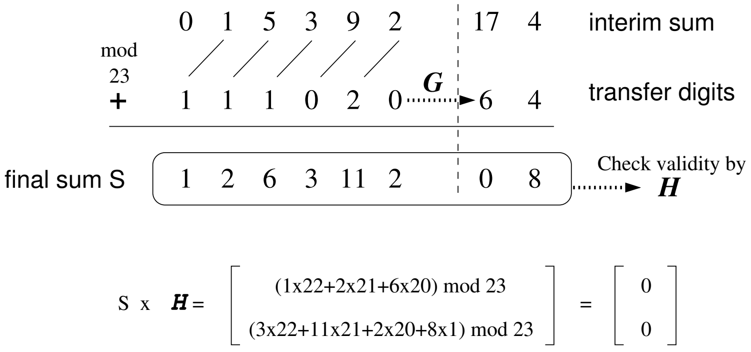


Figure 7.6: Error checking for the final sum entails multiplying the check matrix H with the final sum.

Overall Flow

Figure 7.7 shows this overall flow of incorporating fault tolerance into the carry save addition. In Stage 1, the two operands are encoded using G. The resulting operand codewords are added to form a position sum codeword. This stage is checked by validating the position sum codeword using H. In Stage 2, the position sum is decomposed into the interim sum and the transfer digits. The radix vector is obtained by right shifting the transfer digits first, and then multiplying by the radix r. The check symbols for the radix vector are added to the check symbols of the interim sum modulo q and compared with the check symbols of the position sum. In Stage 3, the check symbols for the transfer digit vector are calculated using G and the final sum codeword is generated by adding the interim sum codewords and the transfer digit codewords modulo q. Error checking in this stage is performed by testing the validity of the final sum codeword using H.

Concurrent Error Correction

The error correction capability in the linear block codes is determined by the minimum Hamming distance d between any two codewords. When an error occurs with e digits changed from a valid codeword c, if $e \leq \lfloor (d-1)/2 \rfloor$, then the resultant error

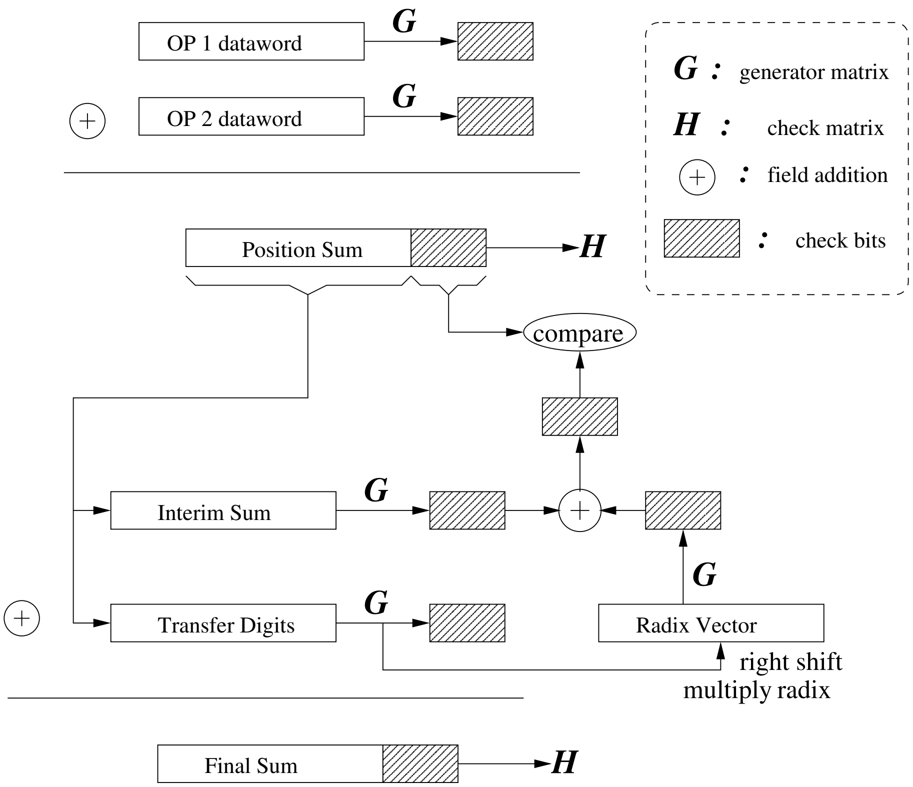


Figure 7.7: Fault tolerant carry save addition

vector c' can be corrected to its nearest valid codeword. Typically, the implementation of error correction in the linear block code approach is performed by building up a syndrome table, which stores the valid codeword for each group of erroneous codewords. By looking up the resultant syndrome vector in the syndrome table, errors changing e digits with $e \leq \lfloor (d-1)/2 \rfloor$ can be corrected [7].

7.3.2 Implementation Overview

The carry save arithmetic system needs to be constructed based on a redundant number system with radix r and operand digit range α satisfying the condition of inequality 7.1. The algebraic field structure of finite field $GF(q)$ is used to support the application of a linear block code on error detection and correction. Various levels of fault tolerance can be embedded into a system based on the expected defect rate, by using an appropriate generator matrix G. Research in algebraic coding has provided numerous methods of constructing G to check and correct a given rate of errors [7]. The check matrix H is generated as the orthogonal complement of G. With a systematic generator matrix, the resulting codeword always contains the original dataword as its prefix. To encode, a dataword vector is multiplied by G under the field operation, i.e. modulo q multiplication. Similarly, a codeword is checked by multiplying it with H modulo q and checking if it yields a zero syndrome vector. A non-zero syndrome vector can also be compared to a syndrome table for error correction.

Encoding / Decoding Process

While the encoding operation of a linear block code consists of multiplying a dataword vector with the generator matrix G, the error checking operation entails multiplying a codeword with the check matrix H. The hardware for encoding and error checking needs to implement a field multiplication of a vector with a matrix using the basic field addition and field multiplication. Each field operation, such as addition or multiplication, is essentially a function mapping from q^2 to q. Consequently, there are only $q \times q$ possibilities for each field operation on $GF(q)$. The field addition and field multiplication can thus be easily implemented with very small hardware overhead, possibly by storing all the $q \times q$ results in a multi-valued ROM-based look-up table. Such encoding and error checking hardware is common across communication, storage and arithmetic subsystems; thus the hardware overhead is amortized across all these subsystems.

The major operation of error checking in every stage consists of the multiplication of a vector with the matrix G or H. In terms of computation latency, multiplying an n digit vector with a matrix is the bottleneck. Multiplying an n digit vector with each column of a matrix consists of: n 1-bit field multiplications followed by a field addition of n elements. The n multiplications can be performed in parallel in one cycle. These n results can be then added using a tree structure. The latency of the error checking process is thus dominated by the addition in the tree structure, which is $O(\log n)$. The process of error checking does not interfere in the critical path of the arithmetic operations. Therefore, although the checking result might be delayed by a few clock cycles, the checking operation in itself does not impact the overall system performance.

Finite field construction: trade-off between using MVL states and multiple digits in hardware

Equation 7.1 essentially implies that $r < \alpha$. Furthermore, it can be easily observed that the radix r needs to be equal to or greater than three to have a positive integer solution for α. A number of examples of (r, α) combinations that can satisfy equation 7.1 include $(r = 3, \alpha \geq 4)$, $(r \in [4, 6], \alpha \geq 7)$ and $(r = 7, \alpha \geq 8)$.

To process the proposed fault tolerance on carry save arithmetic, a finite field $GF(q)$ needs first to be identified as a basis for the linear block code. The number of elements in the field, q, should be a prime number and satisfy $q \geq 2\alpha$; i.e., q should be a prime that is greater than the maximum possible value for the position sum digits. This is to ensure that the arithmetic operations are then identical to the field operations, which are modulo q computations.

Table 7.1: Hardware & representation redundancy tradeoff for CSA based on various multi-valued logic states

Configurations	MVL state # v	OP digit #	PS digit #
radix r = 3	2	3 ×	4×
operand range α = 4	4	2 ×	3×
operand #: 2	5	1 ×	2×
position sum range $2\alpha = 8$	9	1 ×	1×
radix r = 4	2	3 ×	4×
operand range α = 5	4	1 ×	3×
operand #: 2	8	1 ×	2×
position sum range $2\alpha = 10$	16	1 ×	1×
radix r = 5	2	4 ×	6×
operand range α = 12	4	2 ×	3×
operand #: 3	8	2 ×	2×
position sum range $3\alpha = 36$	16	1 ×	2×

We now examine the various tradeoff points in the design space related to the representation of q elements by the number of logic states supported in the multi-valued logic system. Suppose the underlying nanoelectronic technology can support multi-valued logic with v states. Table 7.1 exhibits the tradeoff between representational power and the number of digits required in the proposed scheme for the multi-valued logic systems with various logic state numbers v.

The two configurations are set up according to inequalities 7.1 and 7.2 as the necessary conditions for performing the 2-operand and 3-operand carry save additions. The "MVL state # v" column indicates the number of logic states v supported by the underlying electronics. The "OP digit #" column shows the number of digits necessary to represent each number in the operands under the radix shown in the corresponding configuration. The "PS digit #" column shows the number of digits required to represent each number in the position sum vector.

From the table we can see that, with 2-operand CSA based on radix 4, when implemented in a binary system which only supports 2 logic states, 3 bits are necessary to represent each operand digit. In other words, since the operand digits range from zero to seven, we need at least three binary bits to represent each operand number. Since position sum numbers have a larger range of $2\alpha = 10$, four bits are necessary in binary. Essentially, a binary system needs to rely solely on extended hardware support by multiple bits. It also can be observed from examining this case that, although four bits are necessary, the representational capability of 16 states provided by four bits is not fully utilized since the position sum can be at most 10 in this case. Similarly, under the configuration of 3-operand addition in the lower half of table 7.1, a binary system requires six bits to represent the range of 36 in the position sum.

With a multi-valued logic system, high radix numbers can be approached from two directions: using the supported multiple logic states or using multiple bits. These two dimensions provide higher flexibility in the multi-valued logic based carry save arithmetic designs. For instance, with a multi-valued logic system supporting 8 states, one digit is enough to cover the whole operand range ($\alpha = 5$), while each position sum number still requires two digits. With a 16-valued logic system, the position sum is covered by the representational capability, thus necessitating no additional digits for either the operands or the position sum. However, under this situation half of the representational range is reserved solely for the position sum addition process and is of no use in the later stages of the carry save addition process. Since multi-valued logic systems naturally support much more powerful representational capabilities, the implementation of carry save addition can exploit the tradeoff between such a representational capability and the number of digits required to represent each number.

It is worth noticing that the size of the field addition / multiplication table is fixed by q; thus using a multi-digit representation does not increase the size of field operation table. For the range of the position sum vector represented by multiple digits, the corresponding table entries in the field operation table are listed in multiple digits as well.

As is shown in table 7.1, multiple tradeoff points exist in a multi-valued logic system when implementing the proposed scheme. For particular devices that support a large number of states, multi-valued logic can afford an abundant representational redundancy so as to implement compact carry save arithmetic designs with fewer number of digits. When the underlying nanoelectronic device cannot provide enough logic states to cover all the ranges of the operands and the position sum, tradeoffs can be made by using the multiple digits to compensate for the lack of representational capability.

7.3.3 Extension to Other Arithmetic Operations

The proposed technique can be extended to other carry save arithmetic operations as follows:

- Subtraction is equivalent to adding the complement of a number; therefore, the proposed scheme can be directly applied.

- Multiplication with constants is typically implemented as a collection of shifts-and-adds. Since the error detection and correction for the shift operation by a linear block code is easy to implement, the proposed scheme can be applied too.

- A general multiplication process uses multi-operand additions, and is typically implemented by organizing several carry save addition blocks into a tree structure. Within each tree node, the carry save addition block can accomplish a constant delay for the multi-operand addition; thus the total delay of the multiplication is proportional to the depth of the tree, which equals the logarithm of the operand length.

  The carry save addition block in a multi-operand addition process performs a reduction on operand numbers by converting multiple operands into a single position sum vector, which is further split into an interim sum vector and the transfer digits. It is easy to observe that inequality 7.2 will be violated if the radix r is less than or equal to n. This observation not only explains the reason why carry save addition is not supported by binary system but nonetheless can be applied in multi-valued logic, but also shows that the application of carry save addition can be easily extended to multiple-operand addition, thus enabling a fast multiplication process. The proposed linear block code based approach can be applied similarly to the multi-operand carry save addition process. Therefore, the proposed fault tolerance scheme can be extended to benefit the general multiplication operation.

7.4 Error Checking Capability

It is difficult to justify and verify the error detection/correction capability of any fault-tolerance technique proposed for nanotechnologies, since convincing experimental data are not available at this current research stage. However, with well-structured information redundancy approaches such as linear block codes, it is possible to provide a formal proof, not only to validate, but also thus to precisely quantify its fault tolerance capability. Therefore, even despite the lack of knowledge regarding the implementation details of the underlying nanotechnologies, the fault tolerance level of the proposed scheme can be securely guaranteed.

In this section, we provide such a formal proof to show that the proposed technique can guarantee a predefined $(d-1)$-fault detection capability in the carry save addition operation. Since a carry save addition can be divided into three stages as described in the previous section, the proof is constructed by validating the error checking capability at each stage. We first outline some preliminaries and setups for the proof; subsequently, the validation of the three stages is shown in three separate subsections.

Assuming we apply a linear block code $\mathcal{C}$ with the minimum Hamming distance d between any two codewords, then

$$\forall c_1, c_2 \in \mathcal{C}, |c_1 - c_2| \geq d.$$

The information redundancy in $\mathcal{C}$ should provide a $(d-1)$-error checking capability. We will show that, in each of the three stages of the carry save addition, any error that changes fewer than d digits of the result can be detected by the proposed scheme.

Assume in the carry save addition, the operand digits to be within the range [0, α]; then the position sum digits are within the range [0, 2α]. Suppose the linear block code is constructed on $GF(q)$, where q is a prime number $\geq 2\alpha$. Let us denote the dataword, codeword and check symbols for a vector v as $D(v)$, $C(v)$ and $K(v)$, respectively. The encoding process using a generator matrix $G = [I|P]$ can be expressed by $K(v) = D(v) \times P$ and $C(v) = D(v) \times G$. Since G is constructed in a systematic form, each codeword has its dataword as a prefix. Therefore, $C(v) = [D(v)|K(v)]$.

7.4.1 Validation of Error Checking for the Position Sum Calculation

A position sum codeword $C(ps) = C(op_1) + C(op_2)$ is calculated by adding the codewords of the two operands. Since arithmetic addition is identical to field addition, the resultant vector remains a valid codeword, i.e., $C(ps) \in \mathcal{C}$. When an error occurs and changes e digits of $C(ps)$, where $e < d$, we claim that the faulty position sum codeword is invalid, i.e.,

$$C'(ps) \notin \mathcal{C}.$$

This is because $|C'(ps) - C(ps)| = e < d, C(ps) \in \mathcal{C}$. Since the minimum Hamming distance between any two codewords in $\mathcal{C}$ is d, if $C'(ps) \in \mathcal{C}$, we would have

$$|C'(ps) - C(ps)| \geq d > e,$$

a contradiction.

Therefore, $C'(ps)$ cannot be a codeword in $\mathcal{C}$. The check in this stage performed with H will justify $C'(ps) \notin \mathcal{C}$ with a nonzero syndrome vector.

7.4.2 Validation of Error Checking for the Position Sum Decomposition

The linear block code based fault tolerance in this stage basically guards the correctness of the position sum splitting by guaranteeing that the radix vector and the interim sum add up to the position sum. The only case that a position sum digit is split wrongly yet passes the check is under the rare situation that the position sum digit is split into an incorrect interim sum digit and an incorrect transfer digit, such that the incorrect radix digit and the incorrect interim sum digit add up to the exact position sum digit. Under this situation it is easy to see that either the transfer digit or the interim sum digit must fall out of the corresponding range illustrated in figure 7.2. Therefore, this single rare fault can be prevented by performing a simple range checking on the interim sums and the transfer digits, which is equivalent to a voltage range checking in the NDR based devices.

We now provide the formal proof for the part that the linear block code can guarantee the correctness of the radix vector and the interim sum adding up to the position sum.

The dataword part of the position sum, $D(ps)$, is split into two parts: the interim sum dataword $D(is)$ and the transfer digits dataword $D(td)$.

$D(td)$ is used to deliver the radix vector dataword $D(rv)$ and the relationship $D(ps) = D(is) + D(rv)$ holds.

$$\begin{aligned} K(is) + K(rv) &= D(is) \times P + D(rv) \times P \\ &= (D(is) + D(rv)) \times P \\ &= D(ps) \times P \\ &= K(ps) \end{aligned}$$

Error checking in this stage is performed by checking the equality relationship:

$$K(ps) = K(is) + K(rv). \tag{7.3}$$

Suppose when a fault occurs, the position sum dataword $D(ps)$ is split into the erroneous $D'(is)$ and $D'(td)$. When

$$D'(is) - D(is) = e_1, D'(td) - D(td) = e_2, e_1 + e_2 < d$$

holds, we want to show that

$$K(ps) \neq K'(is) + K'(rv) \tag{7.4}$$

thus implying that the check in equation 7.3 does detect the fault.

Since the check symbols $K'(is)$ and $K'(rv)$ are generated by $D'(is) \times P$ and $D'(rv) \times P$, we have

$$[D'(is)|K'(is)] \in \mathcal{C}$$

and

$$[D'(rv)|K'(rv)] \in \mathcal{C}.$$

$[D'(ps)|K'(ps)]$ is the linear combination of the above two codewords under field addition; therefore, $[D'(ps)|K'(ps)] \in \mathcal{C}$.

Since $|D'(td) - D(td)| = e_2$ and $D'(rv)$ is derived from $D'(td)$ digit by digit, $|D'(rv) - D(rv)| = e_2$.

Since

$$|D'(is) - D(is)| = e_1$$

$$|D'(rv) - D(rv)| = e_2$$

$$D'(is) + D'(rv) = D'(ps)$$

$$D(is) + D(rv) = D(ps)$$

We have:

$$0 \leq |D'(ps) - D(ps)| \leq e_1 + e_2 \leq d.$$

Since the error actually occurs in the transfer digits, and $D'(rv)$ is derived by multiplying $D'(td)$ with the radix r, then

$$|D'(ps) - D(ps)| \neq 0$$

$$D'(rv) + D'(is) = D'(ps) \neq D(ps)$$

Suppose $K'(is) + K'(rv) = K'(ps)$; if we can show

$$|K'(ps) - K(ps)| > 0$$

then inequality 7.4 holds since

$$K'(ps) \neq K(ps)$$

In fact,

$$|[D'(ps)|K'(ps)] - [D(ps)|K(ps)]| \geq d.$$

since we have $[D'(ps)|K'(ps)] \in \mathcal{C}$ and $[D(ps)|K(ps)] \in \mathcal{C}$, from the above proof, we have:

$$0 < |D'(ps) - D(ps)| < d,$$

thus

$$|K'(ps) - K(ps)| > 0$$

and the check in equation 7.3 can detect the fault.

7.4.3 Validation of Error Checking for the Final Sum

In the third stage, the final sum vector is calculated by adding the codewords of the interim sum and the transfer digit vector. Since the error checking mechanism in this stage is identical to the one in the first stage, the proof of stage 1 can be applied to this stage directly.

7.5 Conclusions

The proposed scheme achieves fault tolerance in arithmetic operations via information redundancy. The same strategy has been widely used for fault tolerance in interconnection and memory subsystems. Within such a unified fault tolerance system, data are stored, transmitted and calculated using linear block codewords. The same error checking/correction unit can be used to provide fault tolerance for the different components and during distinct operations of the system. The powerful fault tolerance capability provided by information redundancy can be thus applied to the overall system with highly eliminated hardware / performance overhead because of the sharing of encoding / decoding hardware and process. For the future systems based on the unreliable nanoelectronic devices, such a unified fault tolerance scheme across multiple components is particularly important, because aggressive yet efficient fault tolerance approaches are essential due to the high rates of dynamically occurring faults in the nanoelectronic systems.

By using a linear block code generated by a systematic generator matrix, the proposed scheme achieves concurrent error checking without introducing any delays in the critical path of the arithmetic operation. Error checking is performed on the check symbol part of a codeword and does not interfere with the calculation of the datawords. Therefore, the advantage of fast addition with constant delay provided by carry save arithmetic can be fully preserved.

The proposed approach provides flexibility to adjust the fault tolerance and redundancy levels in two ways. First, different levels of fault tolerance can be achieved by constructing linear block codes with a desired Hamming distance. Second, tradeoffs can be made between the amount of decode/encode hardware and error checking delay.

The proposed technique utilizes the multi-valued logic support in nanotechnology, yet does not depend on the specific underlying technology in the implementation of the multi-valued logic. As a rather general approach, it can be applied to any nanotechnology that supports multi-valued logic, possibly based on the negative differential resistance (NDR) characteristic displayed by multiple nanoelectronic device candidates.

Chapter 8

Fault Tolerant Computational Model for Nanoelectronic Processors

For a processor architecture based on unreliable nanoelectronic devices, fault tolerance is required to ensure the basic correctness of any computation. Since any fault tolerance approach demands redundancy either in the form of time or hardware, reliability needs to be considered in conjunction with the performance and hardware tradeoffs. In this chapter, a new computational model for nanoelectronics processor architectures is introduced, which provides flexible fault tolerance to deal with the high and time varying faults. The model guarantees the correctness of instruction execution, while dynamically balancing hardware overhead and performance penalty. The correctness of every instruction is confirmed by multiple execution instances through a hybrid hardware-time redundancy approach. To achieve high system performance, multiple speculative computation branches are executed in parallel. Hardware resource growth that these speculative computations entail is controlled so that the utilization of hardware is balanced between the two competing goals of performance and fault tolerance. Simulation confirms that the proposed computational model achieves flexible fault tolerance under a wide range of failure rates, while at the same time guaranteeing high system performance and efficient utilization of hardware resources. In addition, the impact on the proposed computational model of other nanoelectronic characteristics are discussed, including the necessity for localizing of interconnections and the regularity of nanofabric structures.

8.1 Motivation

According to the above analysis, applying hardware or time redundancy based fault tolerance schemes for the arithmetic/logic computation subsystem is the only choice. However, this is challenging due to the high cost in either hardware resource or performance. We motivate the proposed approach in this section by investigating a set of conflicts among hardware resources, system performance and fault tolerance in a nanoelectronic architecture under high and time varying rates of faults.

Triple-Modular Redundancy (TMR) and in its generalized form, N-Modular Redundancy (NMR), have been some of the most commonly applied hardware redundancy based approaches for fault tolerance. To apply this straightforward strategy, an instruction can be computed by N distinct units in parallel and the result confirmed by a majority/plurality vote. This strategy is supported by the emerging nanotechnologies due to the abundant hardware resources.

A careful analysis reveals however that this approach is practical only if the fault rates are steady and the faults are evenly distributed, since the amount of redundancy is predefined and fixed. With high occurrence of time varying faults, however, a low predetermined number of computation units might generate distinct results and fail to confirm the computation. On the other hand, setting the redundant computation unit number high to match the worst case scenario consumes unnecessary hardware overhead. Therefore, the rigidity of the NMR fault tolerance strategy makes it extremely hard to match a predefined amount of redundancy with the high and varying fault occurrence in the nanoelectronic environment.

Consider a straightforward time redundancy strategy instead. Multiple clock cycles may elapse before the result of an instruction can be confirmed. For time redundancy based schemes, recompute with shifted operand (RESO) can be used to deal with dynamic permanent faults [64, 92]. However, the application of RESO is strictly limited to a small subset of functions, such as the addition operation. For general arithmetic/logic computation, a time-redundancy based scheme that reuses the same computation unit over multiple time slots is only effective for transient faults. If the component becomes permanently faulty or the transient fault lasts across multiple cycles, distinct computation units need to be allocated. Basically, both hardware and time redundancy are required in a complementary manner, so as to provide high flexibility. With time redundancy, an instruction is always confirmable despite the dynamically varying fault rate, since it can always allocate new computation units at the next cycle when the current results do not conform.

Severe compromises in system performance can be introduced by the time redundancy based approach if subsequent instructions need to wait and contend for a common centralized control unit, which performs both the instruction issue and the fault tolerance control task. Such a centralized control becomes the performance bottleneck in the system with time-redundancy based fault tolerance approaches. This problem can be resolved by introducing more parallelism. In fact, the control for fault tolerance purposes can be separated from the main architecture control unit, forming a second level distributed control for fault tolerance. Supported by the abundant hardware resource in nanoelectronic environment, multiple dedicated control units can be used in parallel, each performing a decentralized fault tolerance scheme for an instruction being executed. Consequently, the latency caused by the time redundancy approach on one instruction does not block the next instruction from being issued. By adding a new layer of decentralized control units for fault tolerance purposes, instructions can be issued and dispatched without stalls.

Computations at the processor architecture level are not isolated. The existence of large number of dependencies among the instructions further causes complications in the performance overhead of time redundancy based approaches. The main performance bottleneck thus resides among the instructions with data dependencies. Basically, applying a time redundancy approach inevitably costs prolonged latency in the confirmation process. It might take an unpredictable number of cycles, as determined by the time varying fault occurrence, before an instruction can be confirmed. Therefore, any successor instructions that rely on the unconfirmed result of a current instruction have to be delayed, resulting in a domino effect on the subsequent dependent instructions and a tremendous number of stalls in an instruction pipeline. Consequently, the latency introduced by time redundancy becomes a severe problem when data dependency exists among instructions, especially in a pipelined environment.

To solve this problem, it can be observed that a dependent instruction need not wait for the confirmation of its predecessor results; additional units can be used to speculatively execute an instruction. In other words, a dependent instruction can use the unconfirmed results in a speculative manner. Multiple speculative branches may be formed for a dependent instruction. As results are confirmed, the correct branches of the dependent instruction are retained and the remaining branches are pruned. While speculation can speed up instruction execution in the presence of data dependencies, one has to carefully manage the amount of speculation. Speculative branches can grow exponentially and quickly exhaust the available hardware, furthermore compromising parallelism and performance in a processor system. We will present an allocation algorithm that carefully manages the speculative instruction execution by allocating hardware frugally.

8.2 Nanoelectronic Processor Computational Model

In a nutshell, the key features of a high-performance fault-tolerant computational model for the nanoelectronic processor architectures are:

- decentralized fault tolerance control units (denoted as *voters*) with a large number of complex computation units (denoted as *C-units*)
- fault tolerance scheme that utilizes hardware and time redundancy to guarantee the correctness of computations

- support for speculative instruction execution of dependent instructions
- hardware allocation algorithm that dynamically balances hardware resources and system performance when dealing with speculation branches for dependent instructions.

8.2.1 Voter/C-unit structure

In the nanoelectronic environment where the device densities can be 1 to 3 orders of magnitude higher than the current CMOS systems, a nanoelectronic processor can support a large number of computation units. Under such a scenario, parallel instruction improve system performance. However, the parallel instruction execution demands not only multiple execution units, but also decentralized control units to support fault tolerance to guarantee correctness of each instruction.

In the proposed architecture for a nanoelectronic processor, the system includes a pool of decentralized control units, denoted as *voters*. The control of the instruction is handled over to the voter once such a connection is established with the allocated voter. The centralized control performs the fetching, decoding and dispatching of the subsequent instructions in the instruction queue without delay. The voters support the parallel execution of the instruction and guarantee the correctness of the computations. Specifically, when an instruction is issued, a voter is allocated to it. The voter manages the reliable execution of the instruction by applying hybrid redundancy based fault tolerance approaches.

Computation units (*C-units*), which carry out the actual arithmetic and logic computations doing instruction execution, are managed by the voters. C-units are dynamically allocated by a voter, and receive from the voter the input values to perform a specific computation. Upon returning result back to the voter, the C-unit is released by the voter and is free for future allocations. In order to confirm the result of an instruction, a voter assigns the same computation to multiple C-units, and compares their results. Figure 8.1 shows a functional view of the instruction issue process. It can be seen that the Issue Machine constitutes the highest centralized control while the voters constitute the second level of decentralized control.

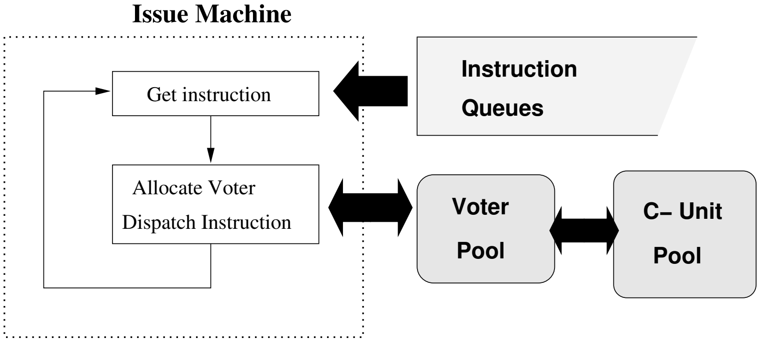


Figure 8.1: Instruction issue process with voter/C-unit structure

8.2.2 Fault tolerant computation

In order to guarantee the correctness, an instruction should be confirmed by at least two results in agreement.[1] The fault tolerance computation is composed of two processes:

1. an initial NMR hardware redundancy approach, followed by
2. a time redundancy approach, which continues invoking new computation instances until two of the results conform.

The main idea behind such a hybrid approach is two-folded: 1) the hardware redundancy based NMR provides an initial trial to confirm the instruction, such that under the situation of low fault rate the computation can be quickly confirmed with minimum hardware resource required; 2) the time redundancy based approach provides full flexibility by trading off performance for reliability, so that high fault rates can be

[1] Note that in the computation of a processor architecture level the results typically consist of multiple bits, and it is presumed that faults in computation units exhibit themselves in distinct ways. Therefore, the faults occurring in multiple computation units tend to have negligible chances of generating a conforming yet faulty result. Further insurance against letting faulty computations slip through can be attained by increasing the threshold of agreement.

handled in a hardware-efficient way. In this section, we provide an exposition of the proposed scheme based on the minimum redundancy of 2 C-units.

Specifically, a voter accomplishes fault tolerance computation for each instruction by combining hardware and time redundancy in the following way. At the beginning, the voter allocates two C-units for an instruction for the initial hardware redundancy based NMR. Triple modular redundancy (TMR) is not necessary for this initial allocation, since the follow-on time redundancy can be invoked if an instruction cannot be confirmed with the initial C-units. After the execution, the unconfirmed values are stored by the voter while the C-units are released so as to support the computation requirement for other instructions. If the initial two results conform, then the instruction is deemed confirmed. Otherwise, the voter incrementally applies time redundancy by allocating one C-unit at a time until two of the results agree and the instruction can be confirmed. Similarly, each C-unit allocated during the time redundancy process is released once the result is stored back to the voter.

The voter performs a comparison each time a new result is returned. Since the previous stored results in a voter are all deemed distinct, the only possible agreement is between one of the previously stored results and the newly returned result. Therefore, only two-input comparators are needed in the voters and the number of comparisons performed inside a voter equals the number of existing results stored in the voter. Figure 8.2 depicts the comparisons needed to be performed upon the nth result being available.

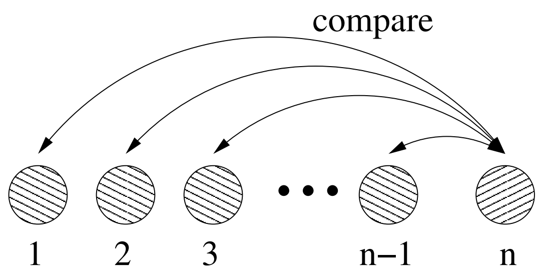


Figure 8.2: Comparison performed in a voter when the nth result is returned

Due to the involvement of time redundancy approach, the confirmation of an instruction might demand unpredictable latency. For a multi-cycle pipelined processor architecture, such a confirmation process can stride across multiple cycles. In implementing high performance processor architectures, pipelines of various depths have been utilized, varying from the simple five-stage pipelines in RISC architectures to the complex twenty-stage pipelines in Pentium 4 [33]. However, for the purpose of illustration, we make considerable simplification and divide an instruction confirmation process into three pipeline stages:

a: Instruction decode and initial allocation of C-units by the voters.

b: Instruction execution carried out by C-units.

c: Result comparison and new C-unit allocation (if needed).

According to the above functionality division, the confirmation process of an instruction has a pattern of {a, b, c, (b, c)*}, which consists of a first initial allocation cycle (a), and followed by a number of recurrent (b, c) steps, due to possibly non-conforming comparisons.

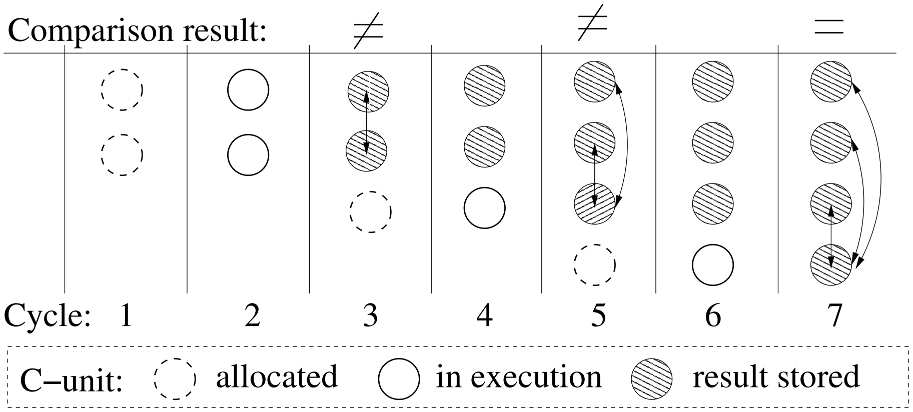


Figure 8.3: An example of the instruction confirmation process

Figure 8.3 shows an example of an instruction being confirmed in seven cycles with four C-units allocated throughout the process. In cycle 1 the voter initially allocates two C-units and the results are available in the third cycle. The two C-units are released at this point and the results stored in the voter. Since the results are not conforming, a new C-unit is allocated at the third cycle. After the execution stage in the 4th cycle, the new result is available at the 5th cycle and is compared with the two stored results. Again the comparisons cannot achieve an confirmation, thereby a new C-unit is allocated in cycle 5. The instruction is finally confirmed at cycle 7 when one of the three previous stored results conforms with the newly returned result.

To carry out the fault tolerance approach, a voter needs a number of storage elements and comparators. Although multiple values need to be compared to check for possible conformity between two results, the comparisons are always performed pairwise, since they are only required between the returning result and the existing ones. Therefore, these comparisons can be performed in parallel with a number of two-input comparators, avoiding the necessity of multi-input comparators, which are expensive both in terms of hardware and latency. Furthermore, a tradeoff needs to be examined when designing the voters: a voter can utilize a large number of comparators for fast parallel comparisons; alternatively, a voter can perform the comparisons in serial with shared comparators, which might result in a prolonged multi-cycle comparison process in the pipelined environment.

The storage elements in a voter is for the purpose of keeping the non-conformable results. The number of storage elements set in a voter depends on the tradeoff between hardware and fault tolerance requirement. At the rare occasions where fault rate is extremely high and all the storage elements are used up in a voter, a simple strategy of discarding a fraction of the existing results can be used. Since all the results stored are unconfirmed and distinct, the chance of discarding a correct result is very low.

8.2.3 Speculative computation to improve performance

We have discussed the fault tolerance computation scheme, which confirms each instruction through a hybrid redundancy of hardware and time. The main concern for system performance, consequently, hinges on the data dependencies on the instructions yet to confirm. Consider the situation where an instruction B takes as input the result of a yet unconfirmed instruction A (we denoted B as a *child instruction*, and A as a *parent instruction*), since the execution of B needs to be based on a correct input, B's execution is delayed until the confirmation of A's result.

Alternatively, in order to improve system performance, dependent instructions do not need to stall for the confirmation of their operands. A child instruction can speculatively use the results of the yet unconfirmed parent instruction. As a confirmed result of the parent instruction is obtained at a later cycle, it can be used to confirm or prune the corresponding speculative branches of the child instruction.

As an example, suppose instruction B uses the result of instruction A as an input, while A takes an exceedingly long time to confirm. B can start executing speculatively without delay, based on the multiple unconfirmed results of A. Multiple *speculative branches* for computing B can thus be formed. As the result of A is finally confirmed, the correct branches of B are retained and the incorrect speculative branches are pruned. The conforming results within a correct branch of B's execution can further confirm the instruction B.

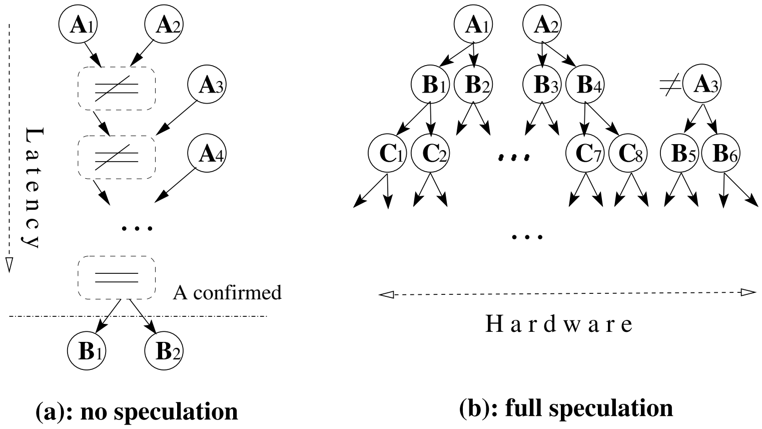
According to the above analysis, two extremal positions in terms of hardware vs. latency tradeoff can be envisioned.

- In a *no speculation* approach, if the input data of a child instruction depends on a parent instruction that is not yet confirmed, the execution of the child instruction is delayed and waits for the confirmed input from the parent instruction, thus necessitating no extra computation units for speculations.

  The *no speculation* approach is a simple scheme that uses a small constant number of C-units, yet inevitably results in significant delay among the dependent instructions. Particularly, in the case of a sequence of dependent instructions, the delay caused by the time redundancy approach is transferred to all the descendent instructions, resulting in a domino effect of latency accumulation. Consequently, the *no speculation* approach represents a mechanism with low hardware, yet high latency overhead.

- In a *full speculation* approach, an instruction can start execution by generating multiple speculative branches for every unconfirmed input. Since the initial NMR needs to be applied in each speculative branch for the purpose of local confirmation[2], the formation of full speculative branches necessitates at least twice the number of the hardware resources to compute all the branches of a child instruction in parallel. Consequently, the *full speculation* approach represents a mechanism with low latency, yet high hardware overhead.

Figure 8.4 illustrates the *no speculation* and the *full speculation* approaches. When executing dependent instructions with the *no speculation* case as shown in figure 8.4(a), the latency can be arbitrarily long; in the case of *full speculation* as shown in figure 8.4(b), an exponential growth in the hardware requirements is encountered.



(a): no speculation

(b): full speculation

Figure 8.4: An example for the cases of (a) no speculation and (b) full speculation

Basically, while speculation can speed up instruction execution in the presence of data dependencies, speculative branches can grow exponentially and exhaust rapidly even the abundant hardware available in a nanoelectronic environment. To avoid the

[2] We assume each speculative branch always allocates the minimum number, i.e., two, redundant computations for local confirmation purposes.

severe latency problem in the *no speculation* approach and the exponential growth of hardware allocation in the *full speculation* approach, a hardware allocation framework needs to be developed to achieve both frugal hardware resource allocation and short overall latency. The essence of such a resource allocation mechanism is to control the generation of speculative branches, such that hardware resources are allocated on an as-necessary basis. Specifically, extra hardware is only allocated when an instruction is known definitely not confirmable.

8.2.4 Dynamic hardware allocation algorithm

For a child instruction that depends on the unconfirmed input data from its parent instruction, the correctness of the result relies on:

1. the correctness of the input data, and
2. the confirmation of the computation process within the child instruction.

The challenge of performing speculative execution essentially comes from these two points.

First of all, a parent instruction might have multiple unconfirmed results, thus forming multiple speculation branches for a child instruction. These speculation branches are dynamically changing. According to the time redundancy approach of the parent instruction, new unconfirmed results might emerge and form additional branches. Furthermore, the confirmation of the parent instruction will cause merging of correct branches and pruning of incorrect ones.

Secondly, within the child instruction itself, the hardware and time redundancy approach applied to confirm the computation for each speculation branch is dynamically changing: two C-units are allocated initially for each branch and additional ones are added in later cycles if necessary.

Overall, these two means of guaranteeing the correctness of a child instruction, when combined, result in an exponential hardware growth in the full speculation approach. The underlying reason for the exponential growth of hardware resources needed in the full speculation approach is that it does not differentiate the hardware allocation policy in a parent instruction and a child instruction. In a sequence of dependent instructions, the exponential growth in hardware resources occurs when a speculation tree is formed with the branch number doubling with depth. The most significant part of the hardware is therefore spent on extensive speculations, a large portion of which might turn out to be based on faulty input data.

From the above analysis, we can draw the conclusion that, hardware allocation should be concentrated on the root-level parent instruction, where the confirmation will accelerate the further confirmation of the descendent instructions. In other words, the C-unit allocation in the child instructions should be controlled so as to reduce the hardware resources spent in the speculative execution. This is the core idea that enables reductions in the exponential hardware growth to be otherwise expected.

An ideal C-unit allocation algorithm for the child instructions needs to achieve the goals of frugal hardware resource allocation, quick confirmation with low latency and fault tolerance for a computation. Specifically, the following aspects should be addressed:

- Parent instructions should be provided with higher priority for obtaining hardware resources in order to quickly prune out wrong speculative branches.

- Child instructions should be updated with the states of the parent instructions during the fault tolerant computation, so as to effectively control the speculation branches.

- The initial C-unit allocation for a child instruction should try to preserve the confirmation possibility of the instruction, yet with minimum hardware resources.

- For a child instruction based on unconfirmed input, hardware allocation should be frugal yet maintain the possibilities for confirmation. In other words, for fault tolerance of the speculative branches, hardware is only added when the

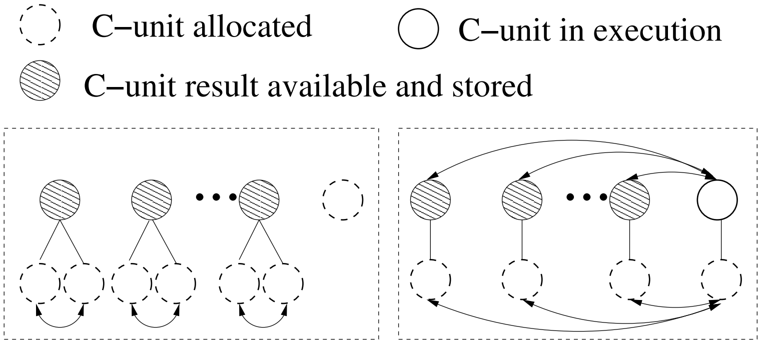


Figure 8.5: Initial allocation of C-unit in a child instruction

instruction becomes impossible to confirm.

Essentially, when the fault rate is low, the initial speculative branches suffice for the child instruction to quickly confirm, without consuming large amounts of hardware resources. When the fault rate is high, the speculation branches should be controlled to a limited number to avoid an exponential growth of hardware requirement. Hardware resources need to be highly prioritized, in this case, on the root level of the speculation tree, thus guaranteeing the quick confirmation of the parent instructions.

We explore the hardware allocation algorithm in depth in the following three subsections. Basically, we discuss the initial C-unit allocation of the dependent instructions, how the parent instruction level information is utilized for the child instructions and how the hardware growth in the dependent instructions is managed. The description of the proposed technique is followed by an example and a discussion subsection.

Initial C-unit allocation of dependent instructions

The minimum hardware resources needed to confirm an instruction in the shortest time are two C-units. For a child instruction based on an unconfirmed input, the full speculation approach allocates likewise two initial C-units for each unconfirmed input. This eventually leads to an exponential growth in hardware requirement for a sequence of dependent instructions. In fact, two of the results in the parent instruction will turn out to conform, yet the initial allocation of the child instruction in the full speculation approach presumes every unconfirmed result to be distinct, thus generating a number of redundant speculative branches.

When a child instruction is issued, information from the previous comparisons in the parent instruction should be utilized. Obviously, an input with the *full resolution* of information, i.e., known to be *correct* or *incorrect*, is easy to deal with. However, for most of the unconfirmed inputs, at a particular cycle, such full resolution is not achievable. However, even if an input is not under full resolution, information can be extracted by distinguishing between the following two states:

- **distinct:** the unconfirmed result is known to be distinct among all the other unconfirmed results.

- **conformable:** the unconfirmed result has a *comparison companion*, i.e., is to be compared with another unconfirmed result. In this case there is a possibility that the unconfirmed result might conform with the comparison companion's result.

In the example shown in figure 8.3, at the 3rd cycle and the 5th cycle, the stored results, represented by the shadowed circles, are all in the *distinct* state, since the newest comparison indicates they differ from each other. On the other hand, at the fourth and sixth cycle, the stored results are all in the *conformable* state, since they are all to be compared with the new result in execution.

Figure 8.5 shows the two specific situations as well as the corresponding initial C-unit allocation cases for a child instruction. In the case shown on the left side, all the results in the parent instruction are known to be *distinct* since a comparison has just completed and it turns out no conformity is achieved. For the child instruction, two C-units are initialized for each distinct result, forming multiple speculation branches that can be locally confirmed. There is no need to allocate any C-units in the child instruction to take the result of the newly issued C-unit in the parent instruction, since this result will not be available until two cycles later.

However, in the case shown on the right side of figure 8.5, all the available results in the parent instruction are to be compared with the last C-unit, the result of which will be available in the next cycle. Therefore, every result has its comparison counter part and is *conformable* with the new result in the parent instruction. In this situation, the child instruction only needs to allocate one C-unit for each unconfirmed input, with the same comparison companion relationship constructed as the parent instruction. A C-unit is also allocated in the child instruction for the C-unit currently being executed in the parent instruction, the result of which will be available in the next cycle. With this initial C-unit allocation, the child instruction can confirm in minimum latency when the fault rate is low.

To summarize, for every unconfirmed result in the parent instruction, the initial C-unit allocation for a child instruction is described below:

- for inputs known to be incorrect, no C-unit is allocated.
- if the input is known to be *distinct*, allocate two C-units initially.
- if the input is *conformable*, allocate only one C-unit initially and assign the comparison companion according to the parent instruction.

Information propagation

In a child instruction, based on 1) the state of the input data, and 2) the computation within a specific branch, the result of a speculative branch at any cycle can fall into one of the following categories:

+ **Full resolution:**

    - ***Correct:*** both the input data and the computation are confirmed to be correct; thus the instruction can be confirmed with this result.
    - ***Wrong:*** either the input data or the computation within the branch itself is confirmed to be wrong; thus this result is known to be incorrect.

+ **Half resolution:**

- ***Global:*** the input data is confirmed to be correct but the correctness of the computation within the branch is not confirmed yet.
- ***Locally confirmed:*** the input data is not confirmed but the computation is locally confirmed within the branch.

+ **Zero resolution:**

- ***Unknown:*** No information is available for the correctness of the input data, and the result is not confirmed within the branch either.

Table 8.1 illustrates how a specific result falls into one of the above categories according to the information from the input data as well as the computation within the child instruction branch.

Table 8.1: Speculation branch result categories

Computation within speculative branch	Input data from parent instruction		
	correct	wrong	unconfirmed
confirmed with conformation	*correct*	*wrong*	*locally confirmed*
confirmed to be incorrect	*wrong*	*wrong*	*wrong*
unconfirmed	*global*	*wrong*	*unknown*

It can be observed that in order for a result within a speculative branch to be *correct*, it has to satisfy two conditions: 1) the input data has to be confirmed as correct, and 2) the result conforms with another one locally within the branch. When any of the two conditions turns out to be false, the result is deemed *wrong*. When one of the conditions is true while the other not available yet, the result is either *locally confirmed* or *global*.

If a pair of results in a branch are marked as *global*[3] and they conform, then the instruction can be confirmed with this pair of results marked as *correct*. Otherwise, if the result pair within a branch agrees but is *unknown*, then the pair of results becomes *locally confirmed*.

[3] If an instruction does not depend on the results of any unconfirmed instructions, i.e., it has no parent instruction, then all its unconfirmed results are marked as *global*.

During the process of the fault tolerance computation, the information obtained from a parent instruction comparison needs to be dynamically propagated to all the child instructions, so as to direct the branch pruning and hardware allocation in the child instruction. Basically, according to the comparison performed in the parent instruction, two types of information can be propagated.

- Information propagation on non-conforming comparison results

  If a pair of results in a parent instruction is compared, the non-conforming comparison result affects only the child instructions that had initially allocated one C-unit for each of the results, as is shown in the right hand part of figure 8.5. When the two C-units in the child instruction are set as comparison companions according to the parent instruction, this is based on the presumption that the two inputs are conforming. When the resulting C-unit pair in the parent instruction does not comform, the information should be propagated to cancel the comparison between the two child C-units, since they are deemed non-conforming due to their distinct data inputs. Through the propagation of the non-conforming comparison results, the *distinct* property is propagated from the parent instruction results to the corresponding results in the child instructions.

- Information propagation on conforming comparison results

  When a parent instruction is confirmed to be *correct*, information is passed to preserve the correct and prune the wrong speculation branches. The speculative branches that take the *correct* results as inputs are marked as *global*, indicating that the input inherited from the parent instruction is confirmed. If the results of the children are already *locally confirmed*, then they become *correct* and the child instruction can be confirmed too. The speculative branches taking the *wrong* results as input will inherit the *wrong* attribute and propagate the information to all the descendants, thus pruning the speculation branches.

C-unit update

The speculative execution of a child instruction is based on the unconfirmed results of the parent instruction. These unconfirmed results of the parent instruction, however, may be dynamically generated after the initial C-unit allocation of the child instruction. A full speculation scheme allocates new C-units and forms a new speculation branch in the child instruction for every newly generated result in the parent instruction.

To control the growth of hardware resources in the speculation of child instructions, the allocation of new C-units for a child instruction should be strictly limited. A voter only allocates new C-units in two cases: 1) the initial C-unit allocation, and 2) when the child instruction becomes *impossible to confirm*. The first case has been discussed in the previous section, and we focus on the second case in this section. There are two situations when a child instruction can be identified as *impossible to confirm*:

- All the results in the child instruction turn out to be *wrong*.

  This occurs when every speculation branch of the child instruction turns out to have a wrong input. In other words, the results which the child instruction inherited from the parent instruction all turn out to be *wrong*. Obviously the confirmation of the child instruction is impossible under this situation.

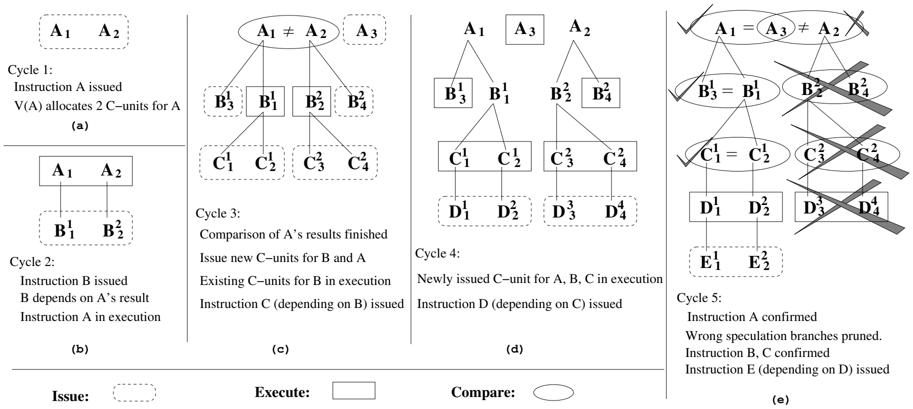
  The new C-units are allocated by taking the results which are not identified as *wrong* in the parent instruction. Similar to the initial C-unit allocation process, two C-units are allocated for each *distinct* result while one C-unit is allocated for each *conformable* result.

- There exist some unconfirmed results in the child instruction which are not *wrong*, but all known to be *distinct*.

  This is either due to the propagation of the *distinct* attribute, or because the computation within the branch fails to conform. Under this situation the child instruction becomes impossible to confirm, due to the lack of local confirmation capability within each speculative branch.

To make the child instruction possible to confirm, new C-units are allocated by duplicating the computation of the *distinct* results.

8.2.5 An example


A_1 A_2
A_1 A_2
B^1_1 B^2_2
$A_1 \neq A_2$ A_3
B^1_3 B^1_1 B^2_2 B^2_4
C^1_1 C^1_2 C^2_3 C^2_4
A_1 A_3 A_2
B^1_3 B^1_1 B^2_2 B^2_4
C^1_1 C^1_2 C^2_3 C^2_4
D^1_1 D^2_2 D^3_3 D^4_4
$A_1 = A_3 \neq A_2$
$B^1_3 = B^1_1$ B^2_2 B^2_4
$C^1_1 = C^1_2$ C^2_3 C^2_4
D^1_1 D^2_2 D^3_3 D^4_4
E^1_1 E^2_2
Issue:
Execute:
Compare:

Cycle 1:
Instruction A issued
V(A) allocates 2 C–units for A
(a)

Cycle 2:
Instruction B issued
B depends on A's result
Instruction A in execution
(b)

Cycle 3:
Comparison of A's results finished
Issue new C–units for B and A
Existing C–units for B in execution
Instruction C (depending on B) issued
(c)

Cycle 4:
Newly issued C–unit for A, B, C in execution
Instruction D (depending on C) issued
(d)

Cycle 5:
Instruction A confirmed
Wrong speculation branches pruned.
Instruction B, C confirmed
Instruction E (depending on D) issued
(e)

Figure 8.6: An example of five cycles for a sequence of instructions using the proposed allocation algorithm

We show an example of dynamic allocation of C-units in figure 8.6, where a sequence of instructions, A, B, C, D, E, are executed and confirmed, each depending on the result of the previous one. A C-unit of any instruction can be in one of the three main states: issue, execute and compare. For representational convenience, we use the superscript of a C-unit to indicate the index of the parent-level C-unit from which the input is taken; we use the subscript of a C-unit for its own index.

Figure 8.6(a) shows the first cycle, during which instruction A is issued; the voter of A initially allocates two C-units, A_1 and A_2, in this cycle. In the second cycle shown in the (b) part, A_1 and A_2 are in the execution cycle, while at the same time instruction B is issued. B depends on A's result. At this stage A_1 and A_2 are set as comparison companions. According to the algorithm, the voter of B allocates two C-units B^1_1 and

B_2^2, taking the results from A_1 and A_2 correspondingly.

In the third cycle, as is shown in figure 8.6(c), the comparison of A_1 and A_2 is finished and the results do not conform. A_1 and A_2 are thus known to be *distinct*, and the information is passed further to split B_1^1 and B_2^2 to be *distinct* as well.

To continue the fault tolerance computation of A, the voter of A allocates and issues a new C-unit A_3 in this cycle. Now that B is known to be *impossible to confirm*, since all its C-units are *distinct*, the voter of B also needs to allocate two new C-units by duplicating B_1^1 with B_3^1 and B_2^2 with B_4^2. These newly allocated C-units are issued for instruction B in the third cycle.

Instruction C is also issued in the third cycle. Since B_1^1 and B_2^2 are in the execution stage and their results will be available in the next cycle, C-unit allocation for instruction C is only considered for them. Since both B_1^1 and B_2^2 are *distinct*, two C-units are allocated for each: C_1^1 and C_2^1 are set to take the result from B_1^1, while C_3^2 and C_4^2 are set to take the result from B_2^2. On the other hand, B_3^1 and B_4^2 are still in the issue stage, so no C-units are allocated to await their result at this cycle.

The (d) part of figure 8.6 shows the fourth cycle of the example. A_3, B_3^1 and B_4^2 are now in the execution stage while the results of B_1^1 and B_2^2 are available and have been passed to $\{C_1^1, C_2^1\}$, $\{C_3^2, C_4^2\}$ correspondingly. No comparison is made between B_1^1 and B_2^2 because they are already identified to be distinct. All four C-units of instruction C are in the execution stage. Instruction D is issued with four C-units allocated, inheriting the comparison companion relationship of instruction C.

The (e) part in figure 8.6 shows the propagation of confirmation as well as the branch pruning process in the fifth cycle. In this cycle, instruction A is confirmed with $A_1 = A_3$, while the result of A_2 is identified to be *wrong* and the corresponding speculative branch is pruned. Also in this cycle, instructions B and C are locally confirmed with $B_3^1 = B_1^1$ and $C_1^1 = C_2^1$. Since the speculative branch of A_1 is confirmed to be correct, instruction B is confirmed, which further confirms instruction C.

The example shows that the proposed hardware frugal allocation algorithm can be used to achieve a fault tolerance scheme with high flexibility for performance. In this example, although A_2 is faulty and the confirmation of A is delayed for two cycles, with the proper control of speculative branches, the delay is not propagated to the subsequent dependent instructions and the whole sequence of instructions is confirmed in time.

8.2.6 Discussion

The fault tolerance computational model for the nanoelectronic processor architecture basically consists of a voter/C-unit architecture to perform a hybrid hardware and time redundancy based fault tolerance approach, and a novel controlled speculation mechanism for data dependent instructions.

With the new computational model, among data dependent instructions, the speculated execution of the instructions is performed out of order while the confirmed results are always generated in order. In general, such a computational model supports out-of-order execution among independent instructions so as to exploit the parallelism offered by the nanoelectronic environment. The related issues of out-of-order execution, including the imprecise exception problem, are essentially similar to the same issues existing in traditional CMOS based architectures, and can be approached accordingly with a number of available techniques being utilized currently.

The speculative execution in the computational model can be further extended beyond the arithmetic instructions to the branch instructions. Essentially, multiple speculations can be formed on the branch targets and the wrong speculation can be pruned according to the address calculation.

Overall, the fault tolerant computational model targets the main reliability challenge of the emerging nanoelectronic environment, and provides a novel approach that integrates fault tolerance, performance and hardware overhead considerations. These conflicting optimization criteria are effectively balanced under such a computational model at the processor architecture level.

8.2.7 Simulation Results

In the above described computational model, hybrid hardware and time redundancy is used to guarantee the correctness of instruction executions. For the instructions with dependencies, the hardware allocation algorithm is used to balance the performance and hardware utilization according to the occurrence of faults. A simulation framework is developed to evaluate the effectiveness of the computational model. Detailed simulation results are available in [67, 68, 72].

Basically, two sets of simulations are provided. First, the new fault tolerance computational model is compared with the *no speculation* and *full speculation* approaches, which represent the two extreme points in the performance/latency tradeoff. Both hardware requirement and latency are compared for the three strategies. Through this set of experiments, the tradeoff between latency and hardware is examined. Secondly, various replication quantities, N, for the initial NMR hardware redundancy approach are compared. The results show the multiple tradeoff points existing within the computational model and they have corresponding applicability under various fault rates.

It turns out that, as expected, the *no speculation* approach suffers from significant delays in comparison to the other two models. Since no speculation branch is ever formed, the data dependencies among the instruction sequences result in the delayed issuing of child instructions. The tremendous latency of the *no speculation* approach grows and makes the gap even larger as fault rates increase. The *full speculation* approach, as expected, achieves the minimum latency since it utilizes hardware resources without any constraints to provide the same redundancy for every speculation branch. The new computational model, in terms of latency, exhibits behavior similar to that of the *full speculation* approach. Overall, simulation results confirm that the new computational model can achieve a near-optimum performance that is comparable to the *full speculation*, which is the best case in terms of performance [67, 68, 72].

The hardware resource consumption in the fault tolerance computation models is considered from two aspects. First, the number of C-units occupied at each cycle shows the amount of computational unit hardware requirement in the system. Notice that a C-unit is released once its computation is finished and the result is returned to the voter. The number of C-units occupied therefore indicates the amount of parallelism existing when multiple speculation branches are executed. Second, since the unconfirmed results are stored by the voter, the number of unconfirmed results during the computation depicts the storage hardware requirement in a voter. From both aspects of hardware consumption, the new computational model displays efficient hardware allocation, that significantly outperforms the *full speculation* approach and is close to the *no speculation* approach in hardware requirement.

Essentially, through the simulation results it can be observed that the new computational model shows significant results both in performance and hardware, thus achieving the goal of fault tolerance with ideal balance of hardware and latency. In comparison to the other two models, the new computation model can be seen to compete with the best aspects of both, i.e., the short delay of the *full speculation* and the frugal hardware allocation in the *no speculation* model.

As the emerging nanoelectronics are expected to offer a density boost in the order of 10^3 to 10^6 compared to today's CMOS technology, a crucial determinant factor for the amount of corresponding boost of parallel computation power at the architectural level is the amount of hardware that is demanded for fault tolerance purposes. According to the simulation results, the new computational model shows significant potential by using an order of magnitude less hardware for fault tolerance purposes while sacrificing neither reliability nor performance, thus supporting eventually the boost of computation power in the nanoprocessor architectures.

The minimum initial NMR of C-unit allocation for the proposed computational model is two; however, the manner in which various initial C-unit settings influence the behavior of the algorithm bears further scrutiny. In this experiment set, we compare the minimum case of 2 initial C-units with the cases of initially allocating 3 and 4 C-units. The simulation results are expected to provide knowledge of various tradeoff points within the proposed scheme.

With more initially allocated C-units, an instruction can be confirmed more quickly when the fault rate is high. On the other hand, when the fault rate is low, some initially allocated C-units are essentially redundant, thus consuming comparatively more hardware resources. The simulation results confirm this syllogism. Essentially, when the fault rate is low, the configuration of minimum initial C-unit allocation provides a highly attenuated loss of latency, while exhibiting a significantly reduced amount of hardware consumption. When the fault rate is high, allocating more C-units in the initial cycle helps reduce the overall latency by and large, while consuming almost the same hardware resources as the minimum initial C-unit allocation configuration. Therefore, the initial C-unit allocation number in the proposed computational model provides multiple optimal points under various fault rate ranges.

8.3 Topological Structure Design of Nanoprocessor Architectures

The proposed computational model aims at providing fault tolerance for the nanoelectronics based processor architecture, with the tradeoff consideration for performance and hardware issues. In addition to the unreliability challenge, a number of other characteristics, particularly, the localized communication constraint imposed by nanodevices, need to be investigated. Due to these new characteristics, multiple issues are raised when the proposed behavioral computational model is mapped to a number of structural components in a processor, consisting of both CMOS and nano devices. In this section, we discuss a coarse distribution of architecture level components in a CMOS/nano hybrid system, based on a number of existing research approaches for the interface design among the CMOS and nano systems. We investigate the message passing mechanisms that are crucial in the communication of the proposed fault tolerance computational model, with a further topological constraint imposed by the limitation of localized communication in nanoelectronic environment.

8.3.1 Research approaches supporting the addressing mechanism at the CMOS/nano interface

A number of research approaches have shown various interface designs that integrate a crossbar based nanoelectronic system with a CMOS based system [18, 19, 21, 80, 81]. Based on these techniques, each nanowire/device can be addressed individually through a combination of CMOS devices.

In the CMOL approach [80, 81], a layer of nano crossbar is posed above a layer of CMOS cells. The interface between the nano wire and the CMOS cells is formed through a limited number of pins, such that each CMOS cell is connected to exactly one nanowire. A particular angle is formed between the CMOS cells and the nano wires; therefore, the addressing of each nanodevice located at the cross point in the nano crossbar structure can be approached through the combination of two CMOS cells.

An alternative approach has been proposed in [18, 21], which utilizes a decoder, possibly formed stochastically to perform the mapping from CMOS wires to nano wires. A number of microwires of CMOS scale are initially connected on a one-to-one base with the same number of nanowires, then expanded to an exponential number of nanowires through a decoder. A detailed description of the nanowire based architecture can be found in [18, 19, 21]

These techniques facilitate the communication between a CMOS system and a nano crossbar based system, thus exhibiting multiple possibilities of implementing a CMOS/nano hybrid architecture system. These approaches essentially provide the basis for the proposed computational model, which exploits the potentiality of both CMOS and nanoelectronic devices to overcome the challenges for the processor architectures in the nanoelectronic environment.

8.3.2 Processor architecture components across CMOS/nano layers

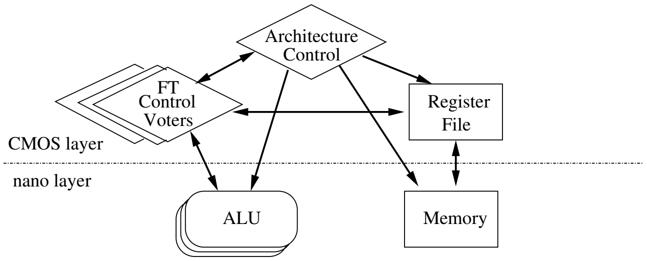


Figure 8.7: Decomposition of the implementation layers across a CMOS/nano hybrid system for the components in the proposed nanoprocessor architecture

We envision the construction of a nanoprocessor architecture in a hybrid approach, consisting of a reliable CMOS layer and an unreliable nano layer, each containing correspondingly the components of logic devices, buses/wires and storage elements. The main modules of the proposed nanoprocessor architecture consist of the main control system for the processor, the additional fault tolerance control of voters, the register file, memory and ALU components. Figure 8.7 exhibits the information connection among the modules and our vision of their implementation in a CMOS-nano hybrid approach.

Basically, the main concern of implementing a component in the CMOS or the nano layer is related to the following differences:

- **Performance:** consisting of the following two aspects:
  - *Computation:* nano devices outperform CMOS devices significantly.
  - *Communication:* nano signal transfer is expensive and suffers from significant wire delay, thus making it in general worse than the CMOS level communications.
- **Hardware:** nano devices offer significantly more abundant hardware resources than CMOS.

- **Reliability:** CMOS devices are significantly more reliable than nanodevices.
- **Locality:** communication among nanodevices is strictly constrained to within a nearby locality, in contrast to CMOS which has a comparatively much wider communication range.
- **Power:** nanoelectronic devices consume significantly lower power than the CMOS counterparts.

According to the above analysis, a component suffers from the reliability problem when implemented in the nano layer, while tending to be much more reliable when implemented in the CMOS layer. On the other hand, a component is more expensive in hardware if implemented in CMOS, in comparison to its implementation in nanoelectronics.

Based on the the consideration of the reliability/hardware cost tradeoff, ALU components and memory blocks, occupying the majority of hardware requirement in a processor architecture, should be implemented with nanoelectronic devices, so as to benefit from the abundant hardware resources. Furthermore, since ALU is heavily computational oriented, a nanodevice based ALU also has the advantage of computational performance boosting. However, from the aspect of locality, implementing these components in the nano layer strictly limits the communication range among the components within the nano layer. Global communication in the nano layer is prohibitively expensive, while only localized communication among the ALU components (C-units) is supported. Consequently, the communication among the ALU components should be limited within a local area.

The control units, including the processor architecture controller and the voters for fault tolerance purposes, impose higher demands of reliability and massive communications with other modules. Therefore, the implementation of these control components in the CMOS layer is advantageous in terms of reliability, locality and communication performance.

Due to the large loads of communication between the register file and the voters, it is more beneficial in terms of communication performance to implement the register file in the CMOS level.

8.3.3 Message passing mechanism

During the speculative computations, the information regarding an instruction being confirmed or speculation branches being confirmed/pruned is transferred among the voters of dependent instructions. Unconfirmed results are transferred directly among the C-units. Voters and C-units also communicate during the C-unit allocation process and the result transmission process.

Essentially, three types of messages are transferred in the proposed fault tolerant computational model:

- **voter/voter:** messages transferred among the voters contain information about the speculative computation: confirmation of instructions and confirmation / pruning / splitting of speculation branches. These messages are always transferred unidirectionally from the voters of the parent instructions to the voters of the dependent child instructions.

- **voter/C-unit:** messages transferred among the voters and the C-units contain the C-unit allocation and release information, parameters of the instructions sent from a voter to the C-units, and the transmission of unconfirmed results from a C-unit to its voter, thus consisting of bidirectional transfer of messages.

- **C-unit/C-unit:** messages transferred among the C-units contain the direct passing of unconfirmed results among the C-units of dependent instructions. The messages are always transferred unidirectionally from the C-units of the parent instructions to the C-units of the dependent child instructions.

The three types of messages described above can be transferred by two genres of communication mechanisms: one with the sender in control, while the other with the receiver in control.

The first and the third types of messages, the voter/voter and the C-unit/C-unit, are transferred unidirectionally from the parent instructions to the dependent child instructions. Since the parent instructions are always dispatched ahead of the child instructions, the sender of the messages in this case cannot be aware of the receivers. The receivers, on the other hand, can establish a connection with the sender easily at the initial stage, both in the cases of voter/voter messages and the C-unit/C-unit messages. Therefore, the messages need to be passed in a *receiver-in-control* mechanism, where a sender simply broadcasts the message while each receiver is responsible for identifying the messages relevant to it.

The second type of messages, which are transferred between a voter and multiple C-units, consist of information passing in both directions: from the voter to the C-units during C-unit allocation, and from the C-units back to the voter when returning the results. In this case, the voter and the C-units are of the same instruction. The C-units are allocated and initialized by the voter, during which process the communication bond can be established on both sides. Therefore, either the voter or the C-unit, when acting as the sender of a message can clearly identify the receivers, supporting a *sender-in-control* message passing mechanism to address the specific receivers without the overhead of broadcasting.

The receiver-in-control message passing mechanism can be easily implemented on a bus topology, while the sender-in-control mechanism fits best to a star topology with the voter connecting multiple C-units. Specifically, the messages within the voters are passed through a common bus, supporting the receiver-in-control, sender-broadcasting mechanism. A similar bus can be used for message passing among the C-units. Among a voter and its related C-units, a star topology is used to support the sender-in-control message passing.

In a CMOS-nano hybrid system, since the voters are in the CMOS layer while the C-units are implemented by the nanoelectronics, the bus among the voters is implemented by CMOS-level wires while the bus among the C-units is implemented by nanowires. The connections among the voters and the C-units can be implemented through the access mechanism developed for the CMOS/nano interface [18, 21, 80, 81].

Error checking code (ECC) can be applied to the message transfer process to enhance reliability. It is worth noticing that, although the reliability of information transfer can be enhanced by applying well developed coding techniques, as in the memory case, the proposed fault tolerant computational model neither presumes, nor relies on the reliability in the message transfer process involving the nanodevices. Any fault or failure in the message transfer of voter/C-unit and C-unit/C-unit can be dealt with in the same manner as the failure of C-unit computations, thus covered by the proposed fault tolerance scheme.

8.3.4 Locality constraint aware network topology

The simple network topology model described above inherently bears a number of constraints. The receiver-in-control mechanism requires a broadcasting on the common bus; however, such a common bus is always subject to the contention problem. One solution to the contention problem is to introduce multiple buses. When broadcasting, a random subset of buses is selected so as to minimize the collision probability. Furthermore, the star topology with the voter at the center is not flexible in the run-time environment where the number of C-units requested by a voter is dynamically changing. Allowing voters to share accessibility to all the C-units provides maximum flexibility, however, at the price of a highly complex mesh network topology.

More importantly, when the characteristics of a nanoelectronic environment are included in the consideration, a number of new challenges are raised. Neither the global common bus nor the complete mesh network fits the locality constraint severely imposed in a nanoelectronic environment.

Taking into consideration the localized interconnection constraints, the accessibility of any component in the nanoprocessor is limited to a localized neighborhood. Figure 8.8 exhibits a revised logic topology of the voters and the C-units adjusted to the localized communication constraint. As is shown in figure 8.8, the voters are connected by a number of buses that cover merely a local neighboring area. A similar localized bus structure is shown among the C-units. The communication among the voters, as well as among the C-units, is therefore limited to the ones that share at least one common local bus. The interconnection among the voters and the C-units is also organized to support the sharing of C-units among the voters in a localized manner. Essentially, neighboring voters have accessibility to a common set of C-units, thus being flexible when unbalanced C-unit allocation occurs dynamically among multiple neighboring voters.

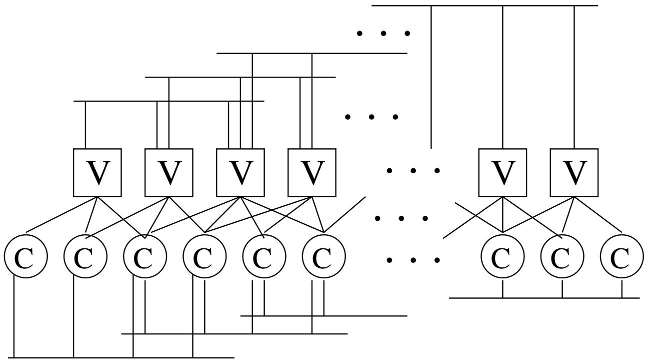


Figure 8.8: The logic message passing channels for voters and C-units considering the locality constraint of nanoelectronic environment

Figure 8.8 shows the logic topology of the voters and C-units, where these components are placed on a number of one-dimensional lines. When the specific regular structure constraint is considered, these components need to be physically placed on a two-dimensional regular structure based nanofabric. It can be envisioned that the voter and the C-units need to be placed with an interleaved manner, as is shown in the example of figure 8.9, where each voter has a number of locally accessible C-units. The accessibility among the voters and among the C-units is similarly limited to within a local range, with an underlying network implementing the logic topology of figure 8.8.

The localization constraint further influences the schedule and dispatch of instructions. Basically, according to the localized communication among the voters, dependent instructions need to be dispatched into a neighboring area where communi-

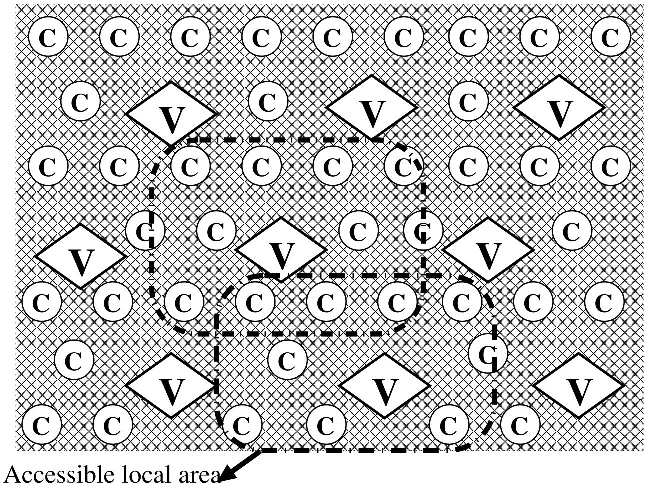


Figure 8.9: The topology of voters and C-units on a regular structured nanofabric considering the locality constraint

cation is feasible for the voter/voter messages. When allocating C-units for the fault tolerance computation, the same locality principle needs to be applied for the C-units so as to enable the fast transfer of unconfirmed results for dependent instructions among the neighboring C-units.

It is certainly true that the locality principle of most programs supports the dispatch of dependent instructions to the neighboring voters and C-units. However, how to optimally perform the mapping of a program with arbitrary dependency structure onto the regular fabric based nanoprocessor remains challenging.

8.4 Conclusion

Multiple challenges and opportunities are raised by the new characteristics of nanoelectronics. Particularly, for the construction of a nanoelectronics based processor architecture, unreliability, localized communication constraints, performance and hardware resource tradeoffs all need to be considered. A new architectural level computational model for the future nanoelectronic processors is developed, addressing the above issues.

The main techniques in this architecture level computational model include: 1) a fault tolerance scheme exploiting hardware and time redundancy dynamically to guarantee the correctness of each instruction, 2) the idea of computation through multiple speculative branches to improve system performance in data dependent instructions, 3) the algorithm of dynamically allocating computation units to avoid the exponential growth in hardware consumption, and 4) localized communication among the processor components in the computational model considering the interconnect overhead in the nanoelectronic environment.

Such a computational model exploits the abundant hardware resources provided by the nanoelectronic technology and makes tradeoffs between hardware and computation performance. Fault tolerance in instruction execution can be guaranteed and system performance is boosted as well; yet communications among the nanoelectronics based components are designed to be within a localized area. Through the simulation with multiple parameters, several tradeoff points are identified for the computational model, thus providing an insight in selecting the proper parameter for a certain fault rate range to achieve the best hardware/latency tradeoff. The overall simulation results confirm the effectiveness of the computational model from both a performance and a hardware overhead perspective, thus evincing that a strong solution is provided to the vital challenges in architecture level fault tolerant computation in nanoelectronic processors. Overall, the computational model sets up a starting point of designing nanoelectronic based processors, and based on the emerging nanoelectronic characteristics provides a framework in investigating multiple nanoelectronic characteristics related issues including reliability, performance, hardware and localized communication at the architecture level for the future processors based on nanoelectronics.

Chapter 9

Conclusions and Future Work

In the thesis work, we target the severe reliability challenge of the future nanoelectronics based systems. Particularly, we develop efficient fault tolerance schemes that are based on the particular characteristics of emerging nanoelectronic devices. These approaches address the fault occurrences projected to be significantly high, variable, and with clustered behavior in the nanoelectronic environment.

The work in the thesis consists of multi-fold approaches: 1) exploiting the new characteristics of the emerging nano devices, 2) alleviating the tremendous cost of fault tolerance schemes by examining the particular structure of the components and exploiting the existing redundancy, 3) a hierarchical fault tolerance strategy, taking advantage of a large set of fault tolerance approaches on the corresponding applicable design abstraction levels. In addition to providing a set of fault tolerance schemes, we present a set of design methodologies that are novel for the nanoelectronic system, as a result of considering reliability together with traditional design optimization goals such as performance, topology and hardware costs.

The thesis work opens up multiple directions for future explorations for reliable nanoelectronic system constructions, including: 1) extending the fault tolerance approaches and system design methodologies as the research on nanoelectronic devices evolves; 2) a simulation framework that evaluates the overall reliability, performance and hardware costs of the system, under various probabilistic fault behavior models; 3) an overall organic integration of the multiple fault tolerance schemes across the multiple system design hierarchical levels, and the design of an online reliability monitoring mechanism.

References

[1] Abramovici, M., Breuer, M. A., and Friedman, A. D., 1990: *Digital Systems Testing and Testable Design*. IEEE Press.

[2] Avouris, P., Appenzeller, J., Martel, R., and Wind, S., 2003: Carbon nanotube electronics. *Proceedings of the IEEE*, **91**(11), 1772–1784.

[3] Baba, T., 1999: Development of quantum functional devices for multiple-valued logic circuits. In *International Symposium on Multiple-Valued Logic*, 2–9.

[4] Bachtold, A., Hadley, P., Nakanishi, T., and Dekker, C., 2001: Logic circuits with carbon nanotube transistors. *Science*, **294**, 1317–1319.

[5] Bahar, R. I., Hammerstrom, D., Harlow, J., Joyner, W. H., Lau, C., Marculescu, D., Orailoglu, A., and Pedram, M., 2007: Architectures for silicon nanoelectronics and beyond. *IEEE Computer*, **40**(1), 25–33.

[6] Beckett, P., and Jennings, A., 2002: Towards nanocomputer architecture. In *Asia-Pacific Computer System Architecture Conference*, 141–150.

[7] Blahut, R. B., 2002: *Algebraic Codes for Data Transmission*. Cambridge University Press.

[8] Bose, P., and Abraham, J. A., 1982: Test generation for programmable logic arrays. In *DAC*, 572–580.

[9] Butler, J. T., 1995: Multiple-valued logic. *IEEE Potentials*, **14**(2), 11–14.

[10] Chen, J., Reed, M. A., Rawlett, A. M., and Tour, J. M., 1999: Observation of a large on-off ratio and negative differential resistance in an electronic molecular switch. *Science*, **286**, 1550–1552.

[11] Chen, R. H., Korotkov, A. N., and Likharev, K. K., 1996: Single-electron transistor logic. *Appl. Phys. Lett.*, **68**(14).

[12] Chen, Y., Jung, G. Y., Ohlberg, D. A. A., Li, X., Stewart, D. R., Jeppesen, J. O., Nielsen, K. A., Stoddart, J. F., and Williams, R. S., 2003: Nanoscale molecular-switch crossbar circuits. *Nanotechnology*, **14**, 462–468.

[13] Collier, C. P., Wong, E. W., Belohradsky, M., Raymo, F. M., Stoddart, J. F., Kuekes, P. J., Williams, R. S., and Heath, J. R., 1999: Electronically configurable molecular-based logic gates. *Science*, **285**, 391–394.

[14] Commission, E., 2001: *Technology Roadmap for Nanoelectronics*. European Commission.

[15] Cormen, T. H., Leiserson, C. E., and Rivest, R. L., 1990: *Introduction to Algorithms*. McGraw-Hill.

[16] Cui, Y., Zhong, Z., Wang, D., Wang, W. U., and Lieber, C. M., 2003: High performance silicon nanowire field effect transistors. *Nano. Lett.*, **3**, 149–152.

[17] Current, K. W., 1994: Current-mode cmos multiple-valued logic circuits. *IEEE Journal of Solid-State Circuits*, **29**(2), 95–107.

[18] DeHon, A., 2003: Array-based architecture for fet-based, nanoscale electronics. *IEEE Transactions on Nanotechnology*, **2**(1), 23–32.

[19] DeHon, A., 2005: Nanowire-based programmable architectures. *ACM JETC*, **1**(2), 109–162.

[20] DeHon, A., and Naeimi, H., 2005: Seven strategies for tolerating highly defective fabrication. *IEEE Design & Test of Computers*, **22**(4), 306–315.

[21] DeHon, A., and Wilson, M. J., 2004: Nanowire-based sublithographic programmable logic arrays. In *FPGA*, 123–132.

[22] Ellenbogen, J. C., and Love, J. C., 2000: Architectures for molecular electronic computers: 1. logic structures and an adder designed from molecular electronic diodes. *Proceedings of the IEEE*, **88**(3), 386–425.

[23] Emmert, J., Strout, C., Skaggs, B., and Abramovici, M., 2000: Dynamic fault tolerance in fpgas via partial reconfiguration. In *FCCM*, 165–174.

[24] Forshaw, M., Stadler, R., Crawley, D., and Nikolic, K., 2004: A short review of nanoelectronic architectures. In *Nanotechnology*, volume 15, 220–223.

[25] Fujisaka, H., Hamano, D., Kamio, T., and Sakamoto, M., 2004: A fault-tolerant architecture for nanoelectronic signal processing. In *IEEE Conf. on Nanotechnology*, 586–588.

[26] Goldstein, S. C., and Budiu, M., 2001: Nanofabrics: Spatial computing using molecular electronics. In *ISCA*, 178–191.

[27] Goldstein, S. C., Budiu, M., Mishra, M., and Venkataramani, G., 2003: Reconfigurable computing and electronic nanotechnology. In *ASAP*, 132–143.

[28] Gonzalez, A. F., and Mazumder, P., 1997: Compact signed-digit adder using multiple-valued logic. In *Conference on Advanced Research in VLSI*, 96–113.

[29] Han, J., 2004: *Fault-tolerant Architectures for Nanoelectronic and Quantum Devices*. Ph.D. thesis, Delft University of Technology.

[30] Han, J., Gao, J., Qi, Y., Jonker, P., and Fortes, J. A. B., 2005: Toward hardware-redundant, fault-tolerant logic for nanoelectronics. *IEEE Design and Test of Computers*, **22**(4), 328–339.

[31] Han, J., and Jonker, P., 2002: A system architecture solution for unreliable nanoelectronic devices. *IEEE Transactions on Nanotechnology*, **1**(4), 201–208.

[32] Heath, J. R., Kuekes, P. J., Snider, G. S., and Williams, S., 1998: A defect-tolerant computer architecture: Opportunities for nanotechnology. *Science*, **280**, 1716–1721.

[33] Hennessy, J. L., and Patterson, D. A., 2002: *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, Third edition.

[34] Huang, Y., Duan, X., Cui, Y., Jauhon, L. J., Kim, K., and Lieber, C. M., 2001: Logic gates and computation from assembled nanowire building blocks. *Science*, **294**, 1313–1317.

[35] Hurst, S. L., 1984: Multiple-valued logic - its status and its future. *IEEE Transactions on Computers*, **33**, 1160–1179.

[36] ITRS, 2006: *International Technology Roadmap for Semiconductors Emerging Research Devices*. ITRS.

[37] Jain, A. K., Bolton, R. J., and Abd-El-Barr, M. H., 1993: Cmos multiple-valued logic design. *IEEE Transactions on Circuits System.*, **40**(8), 503–514.

[38] Javey, A., Wang, Q., Ural, A., Li, Y. M., and Dai, H. J., 2002: Carbon nanotube transistor arrays for multistage complementary logic and ring oscillators. *Nano. Lett.*, **2**, 929–932.

[39] Juhnke, T., and Klar, H., 1995: Calculation of the soft error rate of submicron cmos logic circuits. *IEEE Journal of Solid-State Circuits*, **30**(7), 830–834.

[40] Kameyama, M., 1990: Toward the age of beyond-binary electronics and systems. In *International Symposium on Multiple-Valued Logic*, 162–166.

[41] Karnik, T., Hazucha, P., and Patel, J., 2004: Characterization of soft errors caused by single event upsets in cmos processes. *IEEE Transactions on Dependable and Secure Computing*, **1**, 128–143.

[42] Kastner, M. A., 1992: The single-electron transistor. *Review of Modern Physics*, **64**, 849–858.

[43] Keyes, R. W., 1985: What makes a good computer device? *Science*, **230**, 138–144.

[44] Kim, I., Zorian, Y., Komoriya, G., Pham, H., Higgins, F. P., and Lewandowski, J. L., 1998: Built in self repair for embedded high density sram. In *ITC*, 1112–1119.

[45] Kim, T., Jao, W., and Tjiang, S., 1998: Arithmetic optimization using carry-save-adders. In *DAC*, 433–438.

[46] Krieger, Y. G., 1993: Molecular electronics: Current state and future trends. *J. Structural Chem*, **34**, 896–904.

[47] Kuekes, P. J., Stewart, D. R., and Williams, R. S., 2005: The crossbar latch: Logic value storage, restoration, and inversion in crossbar circuits. *Journal of Applied Physics*, **97**(3), 034301.

[48] Kumar, B. K., and Lala, P. K., 2002: An architecture for self-healing digital systems. In *IOLTW*, 3–7.

[49] Kuo, T., Lin, H. C., Potter, R. C., and Schupe, D., 1993: Multiple-valued counter. *IEEE Transactions on Computers*, **42**(1), 106–109.

[50] Laboratory, C. B., 1993: *1993 LGSynth Benchmarks*. North Carolina State University, Department of Computer Science.

[51] Lala, P. K., 2000: *Self-Checking and Fault-Tolerant Digital Design*. Morgan Kaufmann.

[52] Lent, C. S., Tougaw, P. D., Porod, W., and Bernstein, G. H., 1993: Quantum cellular automata. *Nanotechnology*, **4**, 49–57.

[53] Luo, Y., Collier, C. P., Jeppesen, J. O., Nielsen, K. A., DeIonno, E., Ho, G., Perkins, J., Tseng, H., Yamamoto, T., Stoddart, J. F., and Heath, J. R., 2002: Two-dimensional molecular electronics circuits. *ChemPhysChem*, **3**, 519–525.

[54] Mange, D., Sipper, M., Stauffer, A., and Tempesti, G., 2000: Toward robust integrated circuits: The embryonics approach. *Proceedings of the IEEE*, **88**(4), 516–541.

[55] Mazumder, P., Kulkarni, S., Bhattacharya, M., Sun, J. P., and Haddad, G. I., 1998: Digital circuit applications of resonant tunneling devices. *Proceedings of the IEEE*, **86**(4), 664–686.

[56] Micheel, L. J., Taddiken, A. H., and Seabaugh, A. C., 1993: Multiple-valued logic computation circuits using micro- and nanoelectronic devices. In *International Symposium on Multiple-Valued Logic*, 164–169.

[57] Mishra, M., and Goldstein, S. C., 2003: Defect tolerance at the end of the roadmap. In *ITC*, 1201–1210.

[58] Montemerlo, M. S., Love, J. C., Opitech, G. J., Gordon, D. G., and Ellenbogen, J. C., 1996: *Technologies and Designs for Electronic Nanocomputers*. MITRE.

[59] Naeimi, H., and DeHon, A., 2004: A greedy algorithm for tolerating defective crosspoints in nanopla design. In *International Conference on Field-Programmable Technology*, 49–56.

[60] Nepal, K., Bahar, R. I., Mundy, J., Patterson, W. R., and Zaslavsky, A., 2005: Designing logic circuits for probabilistic computation in the presence of noise. In *DAC*, 485–490.

[61] Nicolaidis, M., Duarte, R. O., Manich, S., and Figueras, J., 1997: Fault-secure parity prediction arithmetic operators. *IEEE Design and Test of Computers*, **14**(2), 60–71.

[62] Nikolic, K., Sadek, A., and Forshaw, M., 2001: Architectures for reliable computing with unreliable nanodevices. In *IEEE-NANO*, 254–259.

[63] Parhami, B., 2000: *Computer Arithmetic Algorithms and Hardware Designs*. Oxford University Press.

[64] Patel, J. H., and Fung, L. Y., 1982: Concurrent error detection in alus by recomputing with shifted operands. *IEEE Transactions on Computers*, **31**, 589–592.

[65] Pradhan, D. K., 1974: Fault-tolerance carry-save adders. *IEEE Transactions on Computers*, **23**, 1320–1322.

[66] Qi, Y., Gao, J., and Fortes, J. A. B., 2005: Markov chains and probabilistic computation - a general framework for multiplexed nanoelectronic systems. *IEEE Transactions on Nanotechnology*, **4**(2), 194–205.

[67] Rao, W., Orailoglu, A., and Karri, R., 2005: Architectural-level fault tolerant computation in nanoelectronic processors. In *ICCD*, 533–542.

[68] Rao, W., Orailoglu, A., and Karri, R., 2005: Fault tolerant nanoelectronic processor architectures. In *ASPDAC*, 311–316.

[69] Rao, W., Orailoglu, A., and Karri, R., 2006: Fault identification in reconfigurable carry lookahead adders targeting nanoelectronic fabrics. In *ETS*, 63–68.

[70] Rao, W., Orailoglu, A., and Karri, R., 2006: Nanofabric topologies and reconfiguration algorithms to support dynamically adaptive fault tolerance. In *VTS*, 214–221.

[71] Rao, W., Orailoglu, A., and Karri, R., 2006: Topology aware mapping of logic functions onto nanowire-based crossbar architectures. In *DAC*, 723–726.

[72] Rao, W., Orailoglu, A., and Karri, R., 2007: Towards nanoelectronics processor architectures. *JETTA Special Issue on Test, Defect Tolerance, and Reliability of Nanoscale Devices*, **23**, 235–254.

[73] Raychowdhury, A., and Roy, K., 2005: Carbon-nanotube-based voltage-mode multiple-valued logic design. *IEEE Transactions on Nanotechnology*, **4**(2), 168–179.

[74] Rueckes, T., Kim, K., Joselevich, E., Tseng, G. Y., Cheung, C. L., and Lieber, C. M., 2000: Carbon nanotube-based nonvolatile random access memory for molecular computing. *Science*, **289**, 94–97.

[75] Shivakumar, P., Kistler, M., Keckler, S. W., Burger, D., and Alvisi, L., 2002: Modeling the effect of technology trends on the soft error rate of combinational logic. In *DSN*, 1112–1119.

[76] Snider, G., Kuekes, P. J., and Williams, R. S., 2004: Cmos-like logic in defective, nanoscale crossbars. *Nanotechnology*, **15**, 881–891.

[77] Snider, G., and Robinett, W., 2005: Crossbar demultiplexers for nanoelectronics based on n-hot codes. *IEEE Transactions on Nanotechnology*, **4**, 249–254.

[78] Somenzi, F., and Gai, S., 1986: Fault detection in programmable logic arrays. *Proceedings of the IEEE*, **74**(5), 655–668.

[79] Stan, M. R., Franzon, P. D., Goldstein, S. C., Lach, J. C., and Ziegler, M. M., 2003: Molecular electronics: From devices and interconnect to circuits and architecture. *Proceedings of the IEEE*, **91**(11), 1940–1957.

[80] Strukov, D. B., and Likharev, K. K., 2005: Cmol fpga: A reconfigurable architecture for hybrid digital circuits with two-terminal nanodevices. *Nanotechnology*, **16**, 888–900.

[81] Strukov, D. B., and Likharev, K. K., 2006: A reconfigurable architecture for hybrid cmos/nanodevice circuits. In *ACM FPGA*, 131–140.

[82] Tahoori, M. B., Momenzadeh, M., Huang, J., and Lombardi, F., 2004: Defects and faults in quantum cellular automata at nano scale. In *VTS*, 291–296.

[83] Townsend, W. J., Abraham, J. A., and Lala, P. K., 2003: On-line error detecting constant delay adder. In *IOLTS*, 17–22.

[84] Townsend, W. J., Abraham, J. A., and Swartzlander, E. E., 2003: Quadruple time redundancy adders. In *DFT*, 250–256.

[85] von Neumann, J., 1956: Probabilistic logics and the synthesis of reliable organisms from unreliable components. In *Automata Studies*, editors C. Shannon, and J. McCarthy. Princeton University Press.

[86] Waho, T., 1995: Resonant tunneling transistor and its application to multiple-valued logic circuits. In *International Symposium on Multiple-Valued Logic*, 194–199.

[87] Waho, T., Chen, K. J., and Yamamoto, M., 1996: A novel multiple-valued logic gate using resonant tunneling devices. *IEEE Electron Device Letters*, **17**(5), 223–225.

[88] Wallace, C. S., 1964: A suggestion for a fast multiplier. *IEEE Transactions on Electronic Computers*, **13**, 14–17.

[89] Wasser, R., 2003: *Nanoelectronics and Information Technology*. Wiley-VCH.

[90] Wei, S., and Lin, H. C., 1992: Multivalued sram cell using resonant tunneling diodes. *IEEE Journal of Solid-State Circuits*, **27**(2), 212–216.

[91] Wolf, S. A., Awschalom, D. D., Buhrman, R. A., Daughton, J. M., von Molnar, S., Roukes, M. L., Chtchelkanova, A. Y., and Treger, D. M., 2001: Spintronics: A spin based electronics vision for the future. *Science*, **294**, 1488–1495.

[92] Wu, K., and Karri, R., 2000: Algorithm level re-computing with shifted operands - a register transfer level concurrent error detection technique. In *ITC*, 971–978.