1-1-2012

A Fault-Tolerant Alternative to Lockstep Triple Modular Redundancy

Andrew Lockett Baldwin
Portland State University

A Fault-Tolerant Alternative to Lockstep Triple Modular Redundancy

by

Andrew Lockett Baldwin

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science
in
Electrical and Computer Engineering

Thesis Committee:
W. Robert Daasch, Chair
Douglas V. Hall
Mark G. Faust

Portland State University

Abstract

Semiconductor manufacturing defects adversely affect yield and reliability. Manufacturers expend vast resources to reduce defects within their processes. As the minimum feature size get smaller, defects become increasingly difficult to prevent. Defects can change the behavior of a logic circuit resulting in a fault. Manufacturers and designers may improve yield, reliability, and profitability by using design techniques that make products robust even in the presence of faults. Triple modular redundancy (TMR) is a fault tolerant technique commonly used to mask faults using voting outcomes from three processing elements (PE). TMR is effective at masking errors as long as no more than a single processing element is faulty.

Time distributed voting (TDV) is proposed as an active fault tolerant technique. TDV addresses the shortcomings of triple modular redundancy (TMR) in the presence of multiple faulty processing elements. A faulty PE may not be incorrect 100% of the time. When a faulty element generates correct results, a majority is formed with the healthy PE. TDV observes voting outcomes over time to make a statistical decision whether a PE is healthy or faulty. In simulation, fault coverage is extended to 98.6% of multiple faulty PE cases. As an active fault tolerant technique, TDV identifies faulty PE's so that actions may be taken to replace or disable them in the system. TDV may provide a positive impact to semiconductor manufacturers by improving yield and reliability even as fault frequency increases.

Table of Contents

List of Tables

List of Figures

Chapter 1 Introduction

The integrated circuit has become commonplace in nearly every facet of modern civilization. Innovations to the integrated circuit, over several decades, have led to advances in computing, communication, education, entertainment, health care, travel, and weaponry. Semiconductor manufacturers relentlessly pursue smaller feature sizes and lower power consumption to reduce costs and increase the capability of products they produce. The feature size refers to the dimensions used when printing circuitry on a semiconductor part using a photolithography process. Transistor gate length, metal-wire width and the placement pitch are semiconductor features that can be minimized in order to fit more circuitry into a given area. Smaller feature size allows for greater transistor density and component integration. For example, CPU manufacturers like Intel and AMD are now integrating essential computer components from chipsets and expansion cards directly into the CPU. Products exist today in which the memory controller hub, PCI-Express interface, graphics processor, caches, and multimedia encoder/decoder are fabricated directly on the CPU die. Integration is possible and cost effective because a smaller feature size allows for greater circuit density with a fixed manufacturing cost.

Semiconductor manufacturers must be mindful of random manufacturing defects as they attempt to shrink the feature size. A random defect is an imperfection introduced during the fabrication process. Defects can be attributed to impurities in the crystalline structure, particles, equipment malfunction, variations in temperature or pressure, or variations in the process. Random defects appear in different shapes and sizes. Some defects are too small to adversely affect the circuit. Larger defects may cause shorts or

opens in the circuitry, resulting in failures or faults. Defects larger than one-tenth the minimum feature size may cause rare faults. Typical manufacturing processes guard against defects that are larger than one-third of the minimum feature size [2] as they are most likely to cause faults. Defect density (λ) is specified in units of defects per unit area and impacts the frequency of defects large enough to cause a fault.

Typically, each new process generation reduces the minimum feature size. As the minimum feature size gets smaller, the impact of random defect size and density increases. The manufacturing process must continually improve in order to keep defect density at an acceptable level [2]. Figure 1 shows the decrease in allowable defect density and size with respect to the minimum feature size [2]. Figure 1 shows that, as minimum the feature size (x-axis) decreases, the allowable defect density (left axis) decreases. The size of defect included in defect density (right axis) also decreases, but at a faster rate than defect density. The implication of Figure 1 is that designs using a smaller feature size are susceptible to failure due to smaller defects. The problem is compounded by the reality that smaller defects occur at a higher relative frequency than large defects as shown in Figure 2 [2]. Acceptable defect density will become increasingly difficult to achieve with each successive process generation.

Manufacturers employ techniques to improve manufacturing yield and product reliability in the presence of faults. Lockstep triple modular redundancy (TMR) is one such technique. Lockstep TMR utilizes triplicated processing elements (PE) and majority voting to mask faults. As long as a majority of PE's are fault-free, majority voting will mask erroneous results and only propagate correct results [5]. Lockstep TMR may preserve manufacturing yield or provide fault tolerance against online faults [4]. When

one of the PEs contains a fault, whether from a manufacturing defect or an online failure, the remaining two PE's form a majority and evict or mask the erroneous result.

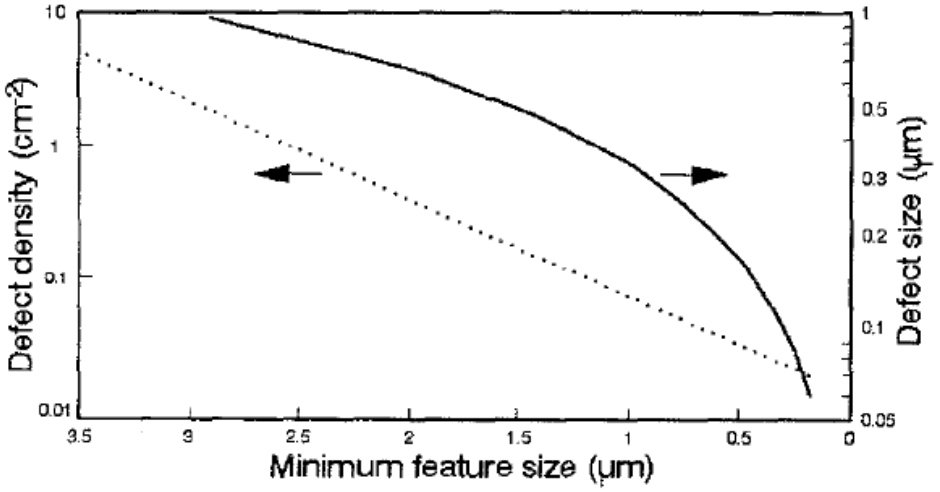


Figure 1: Defect density and size with respect to the minimum feature size [2]. As the minimum feature size (X-axis) gets smaller, defect density (left-axis) decreases, and the size of defect included in defect density (right-axis) also decreases.

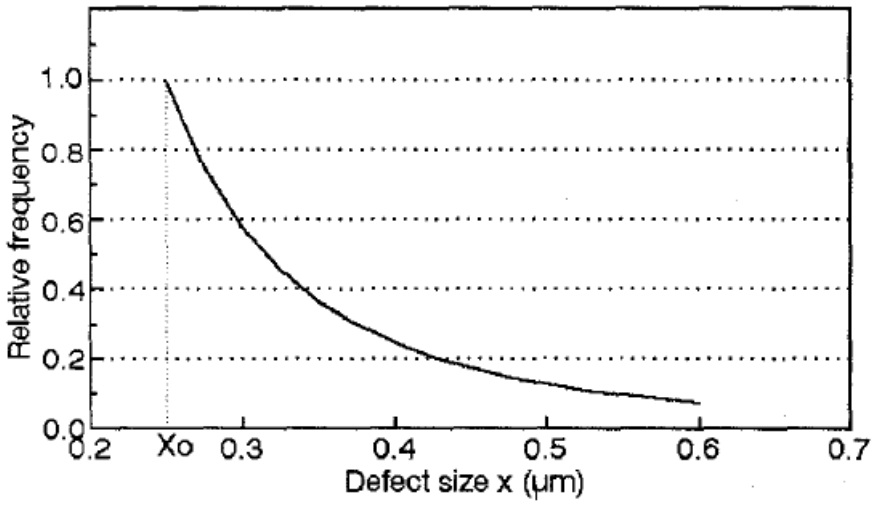
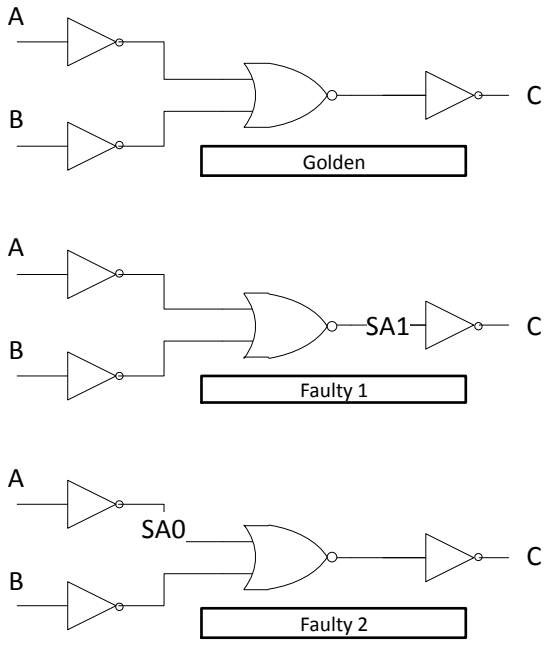


Figure 2: Relative frequency of defects with respect to size [2].

Designers that utilize TMR assume that, at most, only a single PE ever contains a fault [6]. If multiple faulty elements are present, aliasing may occur. Aliasing is when

two faulty results form a majority and evict the correct result. Figure 3 shows a simple example of how aliasing occurs. Figure 3 shows three PE's performing the function of a two-input NAND gate. The *Golden* element contains no fault, while *Faulty1* and *Faulty2* elements each contain different faults. In this work, faults are modeled as stuck-at-0 (SA0) or stuck-at-1 (SA1). The node where the fault occurs is constrained to the stuck-at value, independent of the logic around it. The truth table shows that for the input pattern (A=0, B=1), *Faulty1* and *Faulty2* elements both generate faulty results that form a voting majority. In this simple example, the correct result would be evicted and the faulty result would be propagated because of its majority status.

Some fault pairs are expected to generate aliased results. Faults that modify circuit behavior in the same way are called, equivalent, dominant, or collapsible faults. For example, if one PE contained a SA1 on the input of an inverter and another PE contained a SA0 on the output of the same inverter, both PE's would exhibit the same syndrome and the faults would be equivalent. When activated by an input pattern, equivalent faults generate aliased results because they are, in effect, the same faulty circuit. The faults displayed in Figure 3, are not equivalent, dominant, or collapsible, yet they generate aliased results. Aliasing extends beyond equivalent fault pairs and presents a challenge to hardware redundancy in the presence of multiple faults. Future technologies may produce devices so small that achievable defect density cannot protect against the likelihood of multiple faulty PE's in a TMR system. Additionally, if TMR is used to protect yield, any online fault may constitute the second faulty PE [4].



Truth Table

A	B	C [Golden]	C [Faulty 1]	C [Faulty 2]
0	0	1	0	1
0	1	1	0	0
1	0	1	0	1
1	1	0	0	0

Figure 3: Example of Aliasing when multiple processing elements are faulty. The faults shown in Faulty 1 and Faulty 2 circuits are not equivalent or collapsible, but they still generate identical faulty results.

In most cases, even a faulty PE is correct part of the time. When a faulty PE generates correct results, it votes in favor of the non-faulty PE. If two PE's are faulty, but not generating aliased results, a statistical, time-distributed voting model may be able to a correctly identify the non-faulty PE by observing which element(s) are in the majority most of the time.

This thesis will explore the impact of multiple faulty PE's in a TMR system and propose a time-distributed voting model to extend the usefulness of TMR to systems with multiple faulty PE's. The proposed design is effective for multiple faulty PE's as long as

they do not contain aliasing fault pairs. The design will also enable each PE instance to operate on its own independent data stream. This thesis will summarize the potential for improved throughput and efficiency, while remaining fault tolerant against single and multiple faults using modified TMR methods.

Chapter 2 Background

2.1 Semiconductor Defects and Faults

In the semiconductor manufacturing process, an imperfection that occurs during processing is called a defect. When a defect modifies the behavior of a given circuit to the point of failure, it becomes a fault. There are a large variety of defects that can cause faults. Figure 4 from [2] shows examples of two defects that commonly cause faults during the manufacturing process. These include a defect shorting two metal lines together (Figure 4a) and a void in a metal line resulting in an open (Figure 4b).

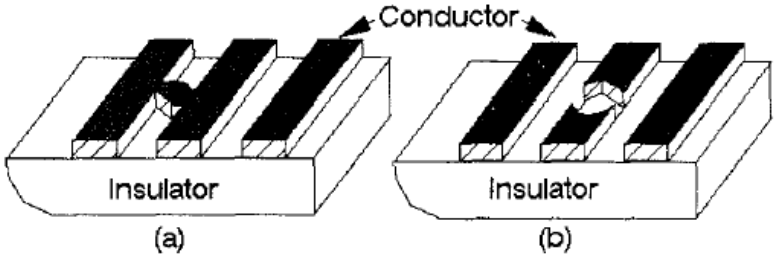


Figure 4: Conductor Defects [2].
(a) Short between parallel metal lines, material deposited where it should not be.
(b) Open metal line, material not deposited where it should be.

Defects do not always result in faults. There is a close relationship between defect size and fault occurrence. This relationship is largely dependent on the minimum feature size of the process. If a defect is much smaller than the minimum feature size defined within a process, then the likelihood of it causing a fault is small. The small defect may be permanently contained within the part and never cause any adverse effect. However, if a defect is close to or larger than the minimum feature size, then the probability of it causing a fault increases. As the manufacturing process continues to evolve, smaller

transistors, and denser designs will become more sensitive to smaller defects that are increasingly difficult to prevent.

A defect may induce a fault that renders the part defective at the completion of the manufacturing process. When the fault is observed during test, the part is rejectedas a yield loss. Conversely, a part may pass initial testing and then fail during its useful lifetime due to latent faults. Examples of common defects that cause latent faults are shown in Figure 5 [2]. In Figure 5a, a metal line with a partial non-conductive void may behave as expected during test, but the reduced cross-sectional area of the line results in higher current density. Over time, the higher current density stresses the line causing it to fail due to electro migration resulting in an open circuit. In Figure 5b, metal lines with a conductive particle partially bridging the gap between them increases the electric field through the insulator when differential voltages are present. Over time this increased electric field breaks down the insulator separating the metal lines and the lines become shorted together. In both of these cases as with a myriad of failure models, the part may not actually fail until a significant amount of time has passed in normal operation.

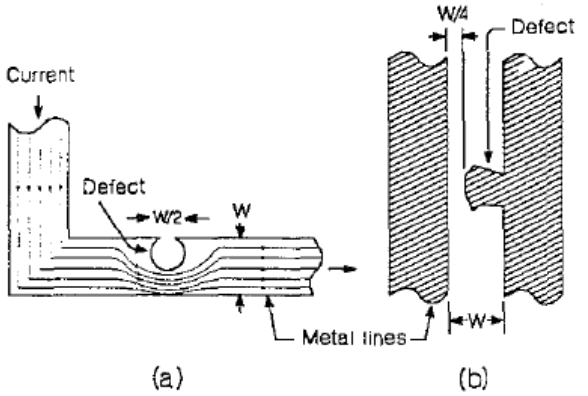


Figure 5: Illustration of defects that cause latent faults [2].
(a) Defect causing a clear field in the metal line, resulting in current crowding.
(b) Defect leaving material between two metal lines causing high electric field.

Manufacturers must guard against defects that may cause manufacturing and latent faults. Semiconductor defect density describes the average number of random defects per unit area that are likely to cause faults. Defect density is a critical indicator of process health and yield [2][4]. Poisson statistics are used to predict yield fallout from defects. If circuit area and the defect density for a manufacturing process are known, the probability of observing exactly *n* defects is calculated by the equation shown in Figure 6e. The Poisson equation in Figure 6e assumes random defects to be uniformly distributed.

$$A = \ area \quad \text{(a)}$$

$$n = number\ of\ defects \quad \text{(b)}$$

$$d = \frac{defects}{Unit\ Area} = defect\ density \quad \text{(c)}$$

$$\lambda = \ dA = average\ defects\ per\ unit \quad \text{(d)}$$

$$P(n, \lambda) = \frac{(\lambda)^n}{n!} \exp(-\lambda) \quad \text{(e)}$$

Figure 6: Poisson statistics
P(n,λ) computes the probability that the part contains exactly *n* defects when average number of defects per unit (*λ*) is known.

Slight increases in defect density may have a large impact on yield. Designs that are salvageable in the presence of defects and faults help to mitigate the otherwise costly yield fallout. Several strategies, including hardware redundancy, exist which allow a circuit to continue to operate or gracefully degrade when faults occur. The next two sections will review fault tolerance and hardware redundancy as viable design methodologies to manage circuit operation in the presence of faults.

2.2 Redundancy and Fault Tolerance

Semiconductor manufacturers go to great lengths to reduce defects within their process, but defects cannot be eliminated completely. Manufacturers can minimize the effect defects have on yield and reliability by utilizing hardware redundancy in their designs. Redundancy refers to the use of multiple processing elements (PE) to handle failures. If one instance fails, a redundant element may be used to accomplish the task. Redundancy is utilized in systems across many disciplines to improve reliability. A suspension bridge utilizes redundant support cables to distribute the load evenly and ensure the bridge does not collapse if a single cable fails. Hospitals use backup power generators to supply power during a utility outage. Hard drives can be configured in a RAID array such that no data is lost if a single drive fails. In all these cases the goal of the designers is to avoid undesired behavior due to a single point of failure.

Semiconductor manufacturers use redundancy to make designs more robust. When fabrication of a design is complete the part is tested. If individual components in the part are faulty, the part may be repaired by activating a redundant component or utilizing an alternate data path. When repair is not possible, faulty components may be disabled and the part sold at a lower price or used for an alternate purpose. For example, the part may have less cache available, or may have a graphics or audio module disabled; however, it need not be wholly scrapped.

Hardware redundancy is used by semiconductor manufacturers to account for latent and online faults. Latent faults are undetectable when the part is initially tested, but fail during the part's useful lifetime. Fault tolerance is a design methodology which enables a part to continue to operate after a fault has occurred. Active and passive

techniques exist for achieving fault tolerance. The passive technique attempts to mask faulty results as they occur. The active technique attempts to identify faulty hardware and remove it from the system [6]. N-Modular Redundancy (NMR) is an example of a passive technique. NMR uses multiple PE's and majority voting to mask erroneous results and prevent them from getting propagated. As long as a majority of PE's is healthy, majority voting can effectively mask errors [6].

2.3 Lockstep TMR

Lockstep Triple modular redundancy (TMR) is a passive NMR approach in which only three PE's are used with majority voting to select the correct result [6]. Figure 7 graphically shows data flow through a (TMR) fault tolerant system containing a single faulty PE (dotted-line). Since two of the PE's are healthy, the faulty result gets discarded when the vote is executed. The voting algorithm is executed continuously. Each output is verified by majority to ensure it is correct. Majority voting may be conducted at the bit level or the word level depending on the application. Word level voting compares PE results in their entirety. An indeterminate state may be reached if all three PE's compute different results. Bit level voting generates the system output using a majority determination from each bitwise comparison of the PE results. Bit-level voting does not allow for indeterminate states when multiple PE's are faulty. When only a single PE is faulty, Lockstep TMR will obtain the correct output value using bit level or word level voting. Lockstep TMR becomes unreliable or indeterminate when multiple PE's are faulty. TMR may improve yield fallout by masking faults caused by manufacturing defects. When one element in a TMR system is already faulty due to a manufacturing

defect and a second element incurs an online or latent fault during normal operation, majority voting is no longer able to identify the correct result [6]. Majority voting in a TMR system is only guaranteed to be correct when a single element is faulty. Multiple faulty PE's may produce identical yet incorrect results. Aliasing occurs when two identical, yet incorrect results form a majority and evict or mask the correct result. When aliasing occurs, the TMR system propagates erroneous data without detection.

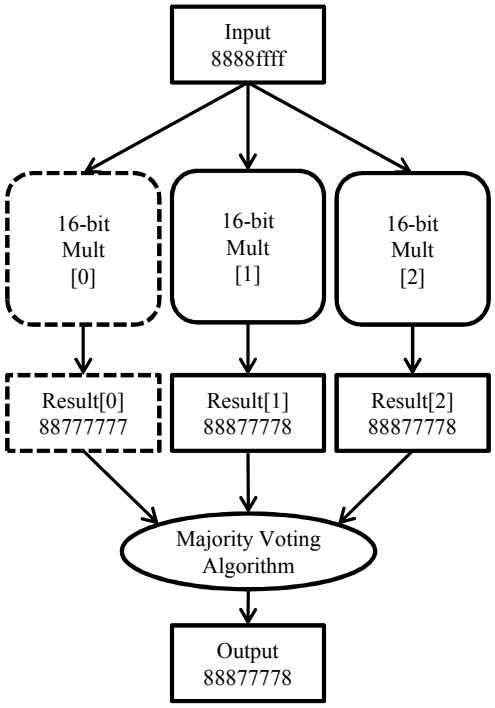


Figure 7: A 16-bit multiplier TMR implementation.
Three 16-bit multiplier modules operate on the same input data. Module 0 is faulty. Modules 1 & 2 are not faulty and form a majority. The faulty result from module 0 is masked at the output.

The yield and reliability benefits of TMR come at a cost. Additional area is required for the redundant PE's and voting algorithm. Additional power is required to compute triplicated results and execute the voting algorithm for every input cycle. TMR does not conduct a thorough check of all possible faults in the PE. The only faults that are

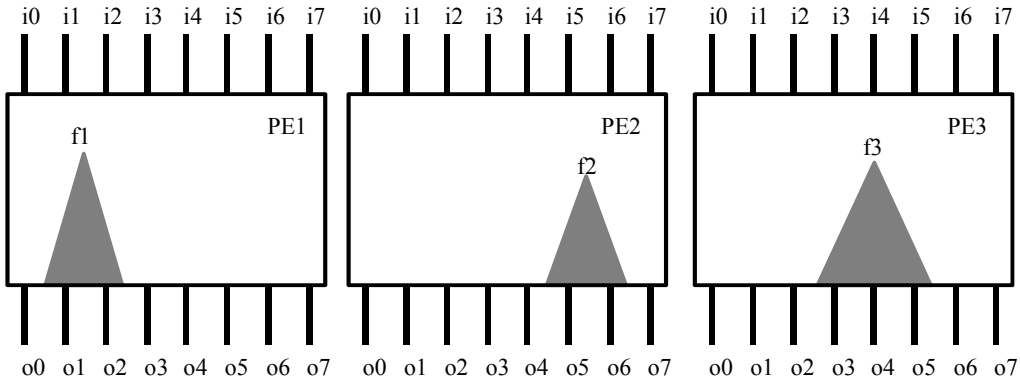
checked are those that are activated by the input patterns. Faults existing within one or more PE's may remain dormant and unobserved by the voting algorithm if no input is presented to activate the fault.

Space compaction is a compression technique which attempts to propagate any errors in a circuit from input to output [1]. Compaction generates a signature key to uniquely identify a distinct fault state in the circuit. In a triple redundant system with signature keys generated for each PE, faults are detectable if keys form a majority. However, when multiple PE's are faulty, a different signature key will be generated for each PE. Space compaction does not provide reliable detection of multiple faulty PE's because a vote on three different signature keys would return an ambiguous outcome.

In safety critical devices, an emergency is the worst time to discover a previously undetected fault. Some input patterns that would activate faults within critical circuitry occur rarely online. For example, the defibrillation circuitry in an implantable cardioverter defibrillator (ICD) is activated only when a dangerous rhythm is detected in the heart. This circuitry delivers an electric shock to return the heart to a normal sinus rhythm. Even though this defibrillation circuitry is rarely used, if it malfunctions at the onset of a heart attack or other cardiac event, the results could be catastrophic. TMR operates only on the input patterns received while online. A failure in the defibrillation circuitry due to multiple faulty elements would be undetectable until a cardiac event actually occurs. A preferable fault tolerant approach would be to continually test for faults in the entire system and identify failures, while there is time and opportunity to respond to them.

2.4 Fault Cones and Aliasing

Designers of TMR systems assume that, at most, only one processing element is faulty. When this assumption fails, aliasing may occur. Aliasing at either the bit level or the word level occurs when two faulty results agree forming a majority, and the correct result is voted out by the voting algorithm. Aliasing risk is determined by the fault cone associated with each potential fault in a system. A fault cone refers to the propagation of a fault through a processing element to the output bits where the fault becomes observable. Figure 8 illustrates graphically the effect of fault cones within a processing element. The fault *f1* affects output bits 1 and 2, *f2* affects output bits 5 and 6, and *f3* affects output bits 3, 4, and 5. The issue of primary concern is the activation of *f2* and *f3* faults simultaneously. Since *f2* and *f3* fault cones overlap across output bit 5, a scenario exists in which *PE2* and *PE3* may agree and create a majority. The fault cone merely provides a description of which output bits may be affected by a given fault. Some output bits in the fault cone may be correct when the associated fault is activated. When *f2* flips only bit 5 and *f3* flips only bit 5, then their outputs will mutually agree on the incorrect result. Equal, yet erroneous results are the source of aliasing and underscore the limitations of TMR.



f1 activated	f2 activated	f3 activated	Bit-level voter	Word-level Voter	Comment
0	0	0	no fault observed	no fault observed	
0	0	1	f3 observed	f3 observed	
0	1	0	f2 observed	f2 observed	
0	1	1	possible aliasing: on o5	possible word aliasing	f2 and f3 cones overlap
1	0	0	f1 observed	f1 observed	
1	0	1	no bit level aliasing	indeterminate	f1 and f3 do not overlap
1	1	0	no bit level aliasing	indeterminate	fl and f2 do not overlap
1	1	1	possible aliasing on o5	indeterminate	

Figure 8: Fault cone illustration
The fault f1 affects output bits 1 and 2; f2 affects output bits 5 and 6; f3 affects output bits 3, 4 and 5. Overlap of f2 and f3 at output bit 5 indicates potential aliasing.

Table 1 demonstrates how aliasing may adversely affect the outcome of the voting algorithm by casting out the correct processing element result. The correct result is denoted by X. The Y and Z values are different, incorrect results. In cases 1-5, TMR works as expected by identifying X as the passing result. However, in cases 6-8, the healthy PE is determined to be faulty due to aliasing.

Table 1: Aliasing in TMR

Identity A x B = X

Case	Input	PE1 Result	PE2 Result	PE3 Result	PE1 Vote	PE2 Vote	PE3 Vote	Voting Result	Disposition
1	A x B	X	X	X	Pass	Pass	Pass	No Fault	Correct
2	A x B	Y	X	X	Fail	Pass	Pass	PE1 Faulty	Correct
3	A x B	X	Y	X	Pass	Fail	Pass	PE2 Faulty	Correct
4	A x B	X	X	Y	Pass	Pass	Fail	PE3 Faulty	Correct
5	A x B	X	Y	Z	Fail	Fail	Fail	Indeterminate	Correct
6	A x B	X	Y	Y	Fail	Pass	Pass	PE1 Faulty	Incorrect
7	A x B	Y	X	Y	Pass	Fail	Pass	PE2 Faulty	Incorrect
8	A x B	Y	Y	X	Pass	Pass	Fail	PE3 Faulty	Incorrect

Table 1 demonstrates outcomes using word-level voting. Bit-level and word-level voting methodologies are not equivalent. Bit-level voting never reaches an indeterminate outcome since there are only two possible values (logic 0 and logic 1) that may be contained in each bit. Since there are an odd number of PE's, a majority will always be observed for each bit. Word-level voting may reach an indeterminate outcome when all PE's compute different results. An indeterminate voting result is a useful indicator that at least two processing elements contain faults.

In this thesis, time-distributed voting (TDV) is proposed. TDV is an alternative methodology to lockstep TMR that is developed to address cases when one or two PE's are faulty. The TDV method is to observe majority voting outcomes over time. Faulty PE's may not be wrong all the time. When a fault is not activated by the input pattern, the result will be correct and the faulty PE will vote in agreement with the healthy PE. A statistical opportunity exists for faulty PE(s) to help identify the healthy PE(s). Fault cones provide insight to cases where TDV may be successful. If the fault cones for two faulty PE's do not overlap, then aliasing cannot occur, and all observed majority voting

outcomes will favor the healthy PE. Effectively, the PE's are in competition with each other and the highest scoring PE(s) is the winner.

TDV is an active fault tolerant method which attempts to identify faulty PE's, rather than mask erroneous results. Conceding that multiple PE's may be faulty, removes the benefit of voting in lockstep since doing so would propagate faulty results. The proposed design removes the lockstep constraint and enables each PE to operate on its own independent data stream. In order to create voting opportunities, pseudorandom input patterns are interleaved into the independent data streams for each PE at a fixed rate. The pseudorandom patterns are capable of providing high fault coverage of all possible faults in the PE. Since the lockstep constraint is removed, a mechanism is required that can identify patterns common to all data streams, capture the PE results when they become available and execute the majority voting algorithm. In this thesis, the content addressable memory (CAM) has been chosen to handle commonality detection and capture the appropriate PE results. The next section provides an overview of the CAM used in the proposed design.

2.5 Content Addressable Memory (CAM)

The methodology proposed in this thesis allows multiple processing elements (PE) to operate on independent data streams. Each PE may operate on a different input pattern in each computation cycle. Majority voting is no longer viable for every computation cycle. Instead, voting occurs when patterns are observed to be common to all input data streams. Due to the continuous nature of streaming data, the entire data stream cannot be searched simultaneously. In order to observe commonalities, a window

must be drawn around the portion of each data stream closest to the processing element. The window is implemented as a first-in-first-out (FIFO) buffer. The input pattern getting processed during a computation cycle is referred to as the active input. Commonalities are observed by searching each FIFO buffer for the active inputs from the mutually exclusive data streams. Figure 9 illustrates how commonalities are detected. The arrows indicate the active input for each data stream. In the FIFO, data remains stationary while a pointer cycles through each location within the buffer. After the data is used, it is replaced with the next input pattern from the data stream.

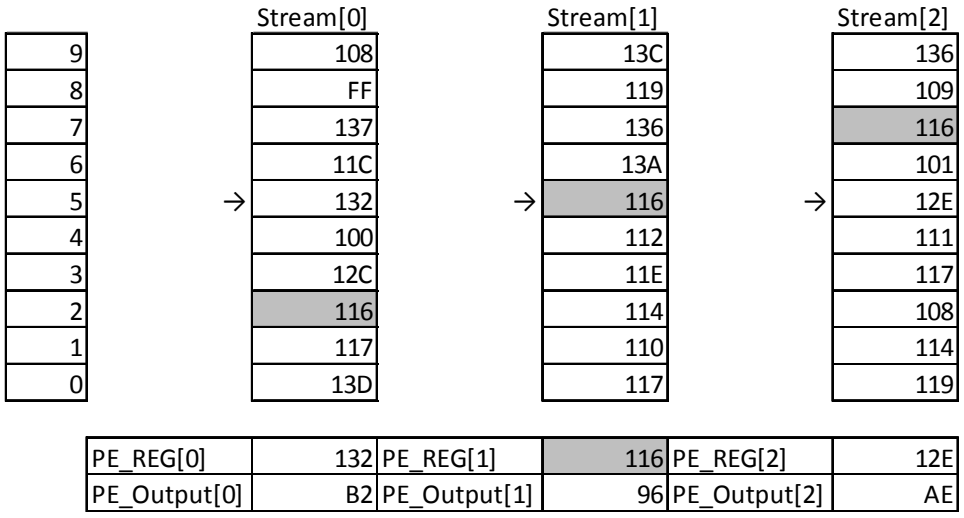


Figure 9: Data stream commonality detection
The active input (116) in Stream[1] is common to Stream[0] and Stream[2]. When this commonality is detected and the results aligned, majority voting is executed.

The active input (116) from *Stream[1]* exists in both *Stream[0]* and *Stream[2]*. If the results are captured and aligned, a majority voting opportunity exists. The voting opportunity is realized by storing the *PE_Output[1]* value obtained in the current clock cycle as well as *PE_Output[0]* and *PE_Output[2]* when they become available in future clock cycles. When all PE_Output results become available, the majority voting

algorithm will be executed to determine if any faults are observed. In the proposed design, content addressable memory (CAM) cells have been used to construct the FIFO buffers for each processing element. CAM cells enable a fully associative search of the FIFO buffers for the active input patterns from the other two data streams. The search is completed in a single clock cycle.

The CAM cell design requires a memory element, search lines, match lines, and comparison circuitry. Figure 10 from [7] shows cascaded NOR-type CAM cells with a timing diagram. During *SL precharge*, *slpre* is asserted to precharge low all search lines (*SL,* $\overline{SL}$). During *ML precharge*, $\overline{mlpre}$ is de-asserted to precharge *ML* high. Once *ML* is high, *slpre* and *mlpre* are de-asserted and *ML evaluate* begins. For each stored bit, either SL or $\overline{\text{SL}}$ is asserted high to assert the search value. If the search value does not match the stored value, a discharge path is established for *ML* to reflect the mismatch. NOR-type CAM cells discharge the *ML* whenever any stored bit is mismatched to its corresponding search value [7]. If the stored value matches the search value for all bits in the word, the ML value remains high to indicate the match.

Figure 11 from [10] shows a cascaded NAND-type CAM. The NAND-type CAM differs from the NOR-type in that the *ML* discharge path is established serially through each bitcell's pass transistor. NAND-type CAM cells use the same timing diagram shown in Figure 10, except the *ML* is discharged only when the stored value and the search value match.

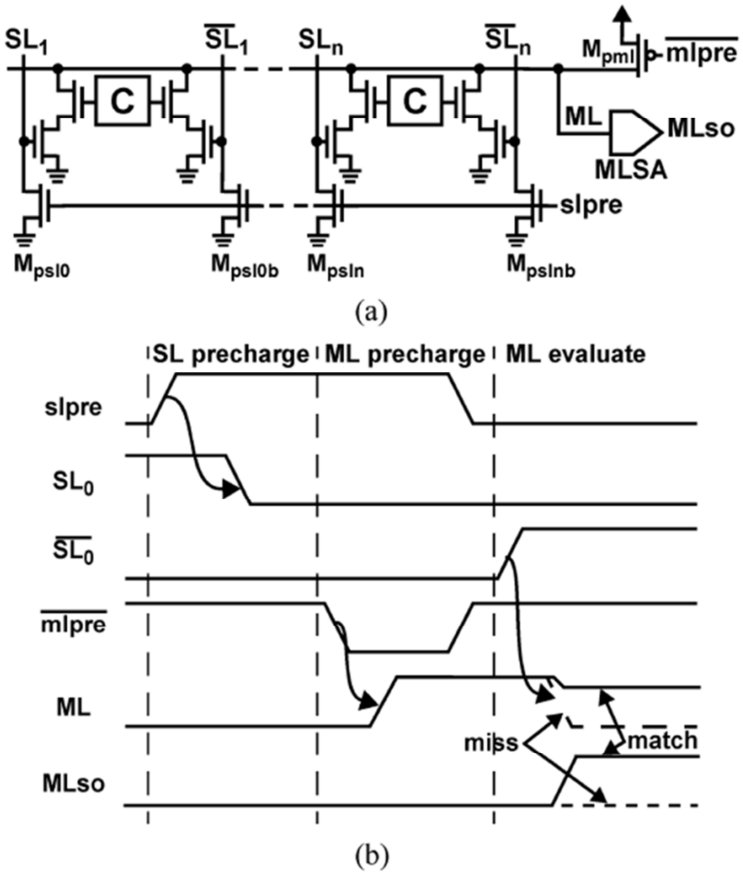


Figure 10: NOR-type CAM with timing diagram[10]

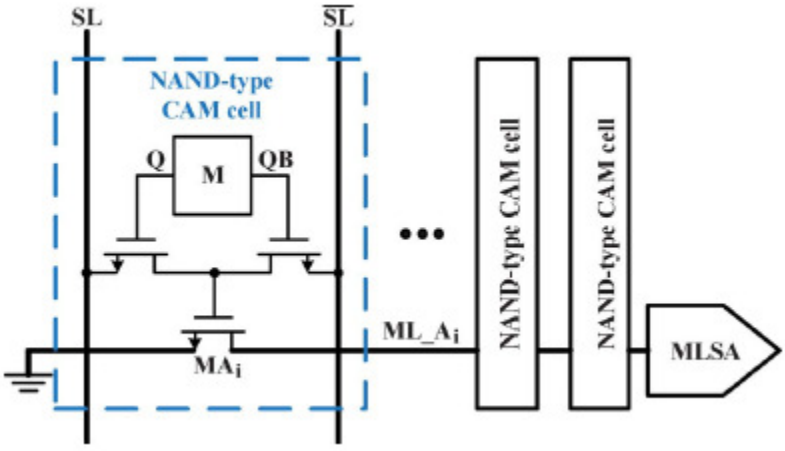


Figure 11: NAND-type CAM cell[10]

When any bit-cell value Q matches its associated search line value SL, a current path is created to discharge the match line ML_A through the matching bit. A mismatch is indicated by the ML_A signal maintaining an active high value.

Circuit designers can optimize power consumption and performance by selecting an appropriate CAM topology. Power is consumed each time the *ML* in a CAM cell is pre-charged from low or discharged due to a search event. If *ML* is rarely discharged, the CAM will consume less power. In the proposed design, data stream commonalities are expected to be rare events compared to cycles where no commonality is detected. The NAND-type CAM is ideal because *ML* will only be discharged in the rare cases that a commonality is detected. If a NOR-type CAM were used for this design, the match line would be discharged for every input cycle where a commonality is not observed, consuming more power.

The CAM topology affects performance and requires additional area for storage and match detection circuitry. Area is determined by the size of memory array needed and the number of search fields implemented. Performance is determined by the delay associated with pre-charging and discharging *ML*. A NOR-type CAM is typically faster than a NAND-type CAM because the NOR-type CAM can discharge the match line through any one or more mismatched bitcells. The NAND-type CAM discharges the *ML* serially through all bitcells when a match is observed.

2.6 Processing Element: The ISCAS 85 C6288 Benchmark

The ISCAS ‘85 C6288 benchmark circuit was used to evaluate the proposed design. The C6288 benchmark is a matrix implementation of a 16-bit multiplier utilizing 32 input bits, and 32 output bits. The Verilog implementation of the benchmark contains 2448 discrete nodes. Each node may be simulated as a stuck-at-0 (SA0) or a stuck-at-1 (SA1) fault. The benchmark provides 4,896 possible faults to evaluate the design.

Articles for the C6288 benchmark provide a minimum set of 12 engineered test patterns for fault testing [3]. The C6288 multiplier consists of a 15x16 matrix of full and half adders as shown in Figure 12. Each half or full adder is implemented as shown in Figure 13 with interconnectivity described in Figure 14.

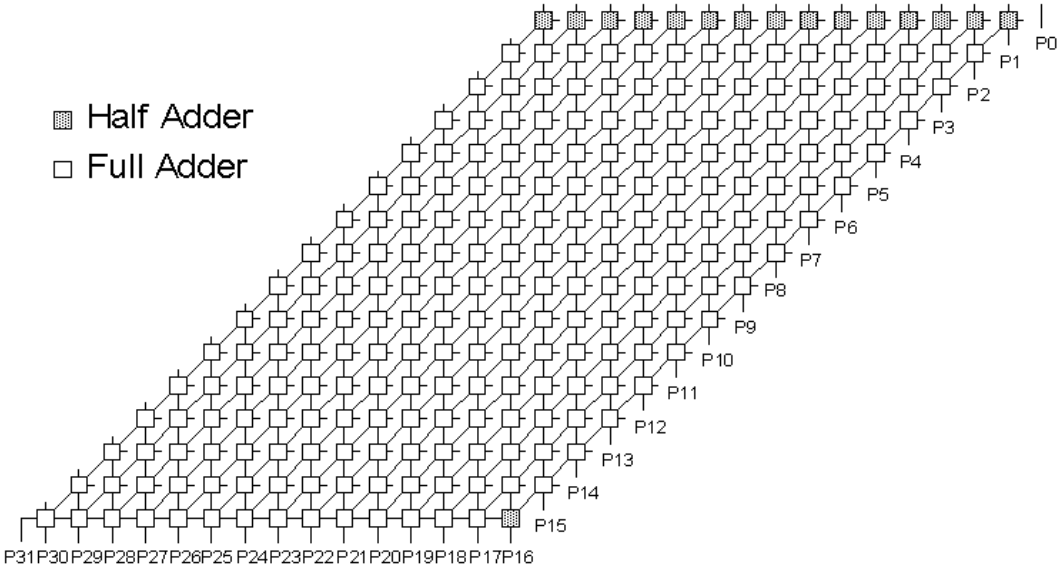


Figure 12: ISCAS-85 C6288 16x16 Multiplier [3]

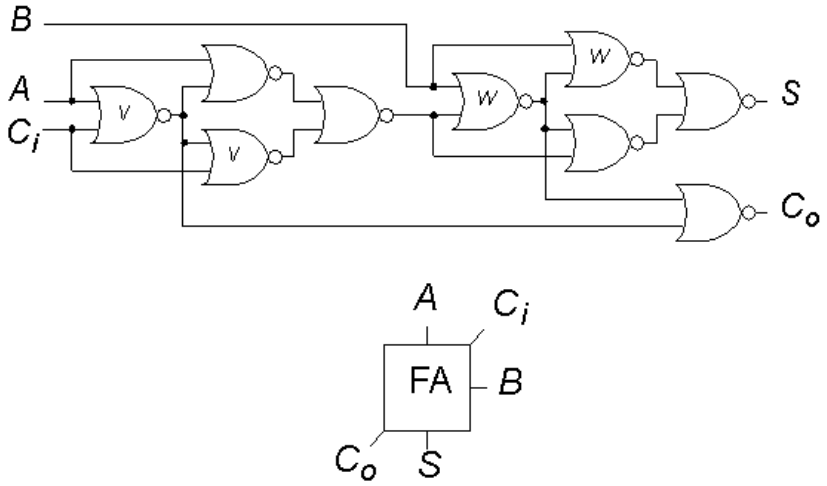


Figure 13: Adder module used in C6288 benchmark [3]
The 15 top-row half adders lack the C_i input; each has two inverters at locations V. The single half adder in the bottom row lacks the B input, thereby acquiring two

inverters at locations W.

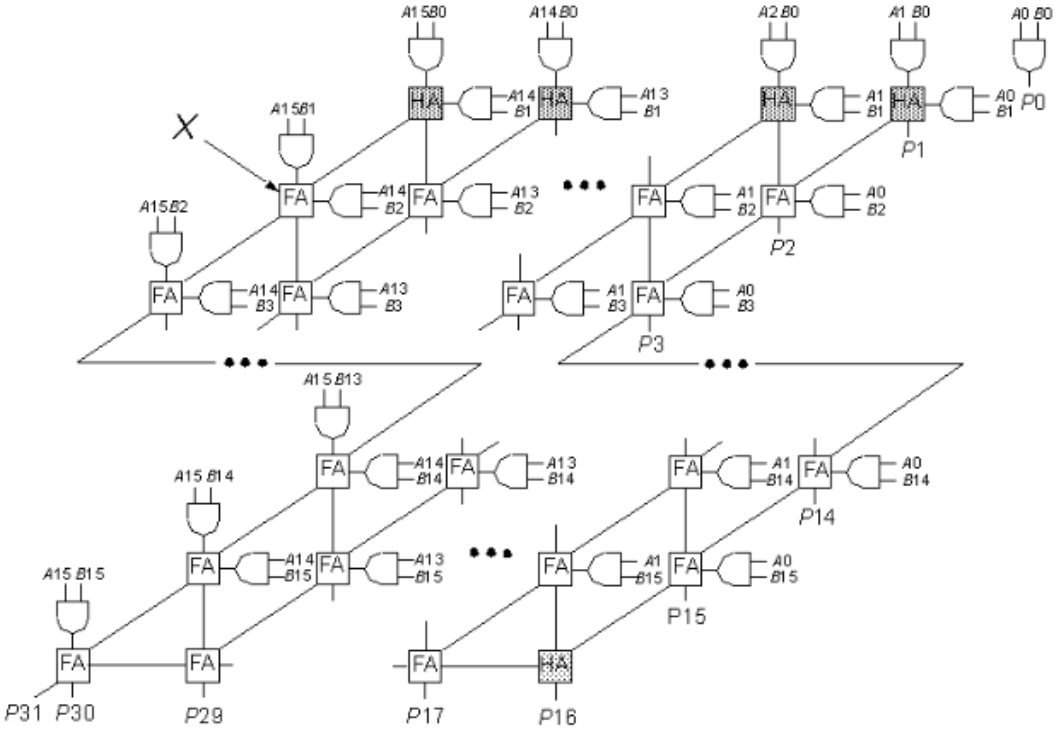


Figure 14: Alternate depiction of C6288 Multiplier showing connectivity [3] The input combinations are AND'ed together to form input to the half and full adders.

Each row in the C6288 matrix computes a partial product used in the multiplication. The output bits are computed by summing the skewed partial products for each column in the matrix and propagating carry bits to the left adjacent column. The Half adders used in the C6288 are modified full adders with one input removed.

Chapter 3 Design

3.1 Methods

Verilog HDL is used to facilitate evaluation of the time distributed voting algorithm. The C6288 benchmark is a structural Verilog implementation of a 16-bit multiplier. In order to evaluate the faulty behavior of the C6288, faults need to be injected. The faults that need to be injected are *stuck-at-0* (SA0) and *stuck-at-1* (SA1) faults for every node in the C6288 circuit. Fault injection in accomplished using Verilog *force* and *release* procedural statements. The *force* statement overrides a node's logic value with the specified force value. Once the *force* statement is executed, the affected node holds the forced value until the release statement is executed. The release statement returns the node to its nominal behavior.

Pseudorandom input patterns are used to activate the injected faults. The pseudorandom patterns were obtained using a linear feedback shift register (LFSR). The patterns generated in an LFSR have some uniformity as each pattern is identical to the previous pattern with the exception of one bit. To overcome this uniformity, LFSR patterns were only collected after completely shifting the previous pattern out of the LFSR. This method was used to collect 1,200 pseudorandom patterns for fault excitation.

The faulty behavior of the C6288 is characterized, using single stuck at faults. A simple Verilog test-bench was used to force a fault in the C6288 PE. All 1,200 pseudorandom patterns were then cycled through the faulty PE while collecting the results. This process was repeated for all 4,896 single stuck-at faults in the PE to generate a table with 4,896 fault rows by 1,200 input patterns. The data table is used as the basis

for simulation of fault combinations.

The analysis in this thesis includes a characterization of fault pairs. A fault pair or fault combination is defined as two discrete single stuck-at faults injected into two distinct processing elements. The characterization is completed by exhaustively simulating all possible combinations of two single stuck-at faults in the C6288. A Verilog HDL prototype has been developed to demonstrate an implementation of TDV. The prototype, however, is not optimal for obtaining the volume data needed for fault pair analysis. Fault pair analysis was done by hashing fault rows in the data table to observe voting outcome's when one PE is healthy, and two PE's are afflicted by different faults. A Microsoft VBA script was used to perform the data hashing. The next four sections will provide a more detailed description of the processes uses to accomplish the analysis.

3.2 Proposed TDV Design Overview

Lockstep TMR is a passive fault tolerant technique which uses majority voting to mask erroneous data from a single faulty processing element (PE). Lockstep TMR is not effective when more than one PE is faulty. The primary objective of the proposed design is to improve fault tolerance in the presence of multiple faulty PE's. Time distributed voting (TDV) is proposed in this thesis as an alternative to lockstep TMR. TDV is an active fault tolerant technique designed to identify which PE(s) are faulty and which ones are not. TDV achieves the same coverage as lockstep TMR for single faulty PE's, while extending coverage to non-aliasing cases of multiple faulty PE's. Conveniently, the proposed design may also lead to improvements in performance and efficiency by allowing redundant PE's to operate on separate and independent data streams.

When a PE contains a fault, it may not generate erroneous results 100% of the time. A faulty PE only generates erroneous results for a subset of input patterns that activate the fault. The input patterns that do not activate the fault will generate correct results matching those from a healthy PE. This observation creates a statistical opportunity to identify non-faulty elements using word-level majority voting. In order to exploit this statistical opportunity, each processing element has an associated weight. Weights are updated every time a majority vote is executed. The rules for updating these weights are detailed in Table 2. In the case where all results match, no fault is observed and the weights remain unchanged. When a single result is mismatched, the majority weights are incremented and the minority weights are decremented by one. When all PE results are mismatched, multiple PE's are faulty, but the majority vote provides no insight about which element is faulty or if all elements are faulty. In the case where all results are mismatched, the weights remain unchanged.

Table 2: Method for updating PE weights based on majority voting results.

Input Pattern	Result[0]	Result[1]	Result[2]	Weight[0]	Weight[1]	Weight[2]
A	X	X	X	+0	+0	+0
A	Y	X	X	-1	+1	+1
A	X	Y	X	+1	-1	+1
A	X	X	Y	+1	+1	-1
A	X	Y	Z	+0	+0	+0

As majority voting occurs over time, the updated weights form a prediction about which element(s) if any are faulty. The weights are interpreted as shown in Table 3. The X, Y, and Z values shown in the table represent final PE weights after TDV and are related as X>Y>Z. In all cases, the PE with the greatest positive value (+X) is identified as healthy. When two PE's tie for first place (+X), both are identified as healthy. When

two PE's contain values of smaller magnitude that are equal and opposite, (+/-Y,-/+Y), both are identified as faulty. When two PE's contain values of smaller magnitude that are not equal and opposite, (+/-Y,+/-Z), both are identified as faulty with aliasing observed for some input patterns.

Table 3: Interpretation of final PE weights after TDV.

Weight PE[0]	Weight PE[1]	Weight PE[2]	Interpretation (X > Y > Z)
0	0	0	No Fault Observed in any element
-X	+X	+X	PE[0] is faulty; PE[1] and PE[2] are not faulty
+X	-X	+X	PE[1] is faulty; PE[0] and PE[2] are not faulty
+X	+X	-X	PE[2] is faulty; PE[0] and PE[1] are not faulty
+X	+Y	-Y	PE[0] is not faulty; PE[1] and PE[2] are faulty
-Y	+X	+Y	PE[1] is not faulty; PE[2] and PE[0] are faulty
+Y	-Y	+X	PE[2] is not faulty; PE[0] and PE[1] are faulty
+X	+/-Y	+/-Z	PE[0] is not faulty; PE[1] and PE[2] are faulty (aliasing observed)
+/-Z	+X	+/-Y	PE[1] is not faulty; PE[2] and PE[0] are faulty (aliasing observed)
+/-Y	+/-Z	+X	PE[2] is not faulty; PE[0] and PE[1] are faulty (aliasing observed)

3.3 Prototype Implementation

The proposed design has been implemented in Verilog HDL. Verilog provides an efficient, logical environment for observing data flow and fault simulation. Figure 15 contains the block diagram of the proposed design. Dotted lines indicate contributions from this thesis. The FIFO buffers deliver stream data to the processing elements. The CAM cells used in the FIFO buffers contain two search fields and two match line outputs, and are referred to as double-ported. Using double-ported CAM cells, the FIFO contents may be searched for matches to the active input patterns from the other two data streams. Each cycle, the active input for each stream is forwarded to a search field in the other two buffers. Table 4 contains the search field inputs for each buffer and the conditions that would identify a buffer's active input as a commonality. When a commonality is detected, the input pattern is stored in the common symbol array and the corresponding

PE result is stored in the results array. The common symbol array holds the common input patterns until results from other processing elements become available and majority voting occurs.

Table 4: Commonality detection using CAM cell FIFO buffers.

Buffer	Active Input	Search Field 1	Search Field 2	Commonality Condition
FIFO[0]	A	B	C	MW1[2] && MW2[1]
FIFO[1]	B	C	A	MW1[0] && MW2[2]
FIFO[2]	C	A	B	MW1[1] && MW2[0]

The common symbol array is implemented using triple-ported CAM cells. Three search fields correspond to the active inputs for the 3 FIFO buffers. When a commonality is detected, the common symbol array is searched to determine if the pattern is already present and add it to the array if needed. The common symbol array indexes into the result arrays to store latent results when common patterns finally make their way to the processing elements. Finite states in the PE are not considered when commonalities are detected in the FIFO. It is assumed that the PE's contain only combinational logic since stated logic may change as input patterns progress to the PE.

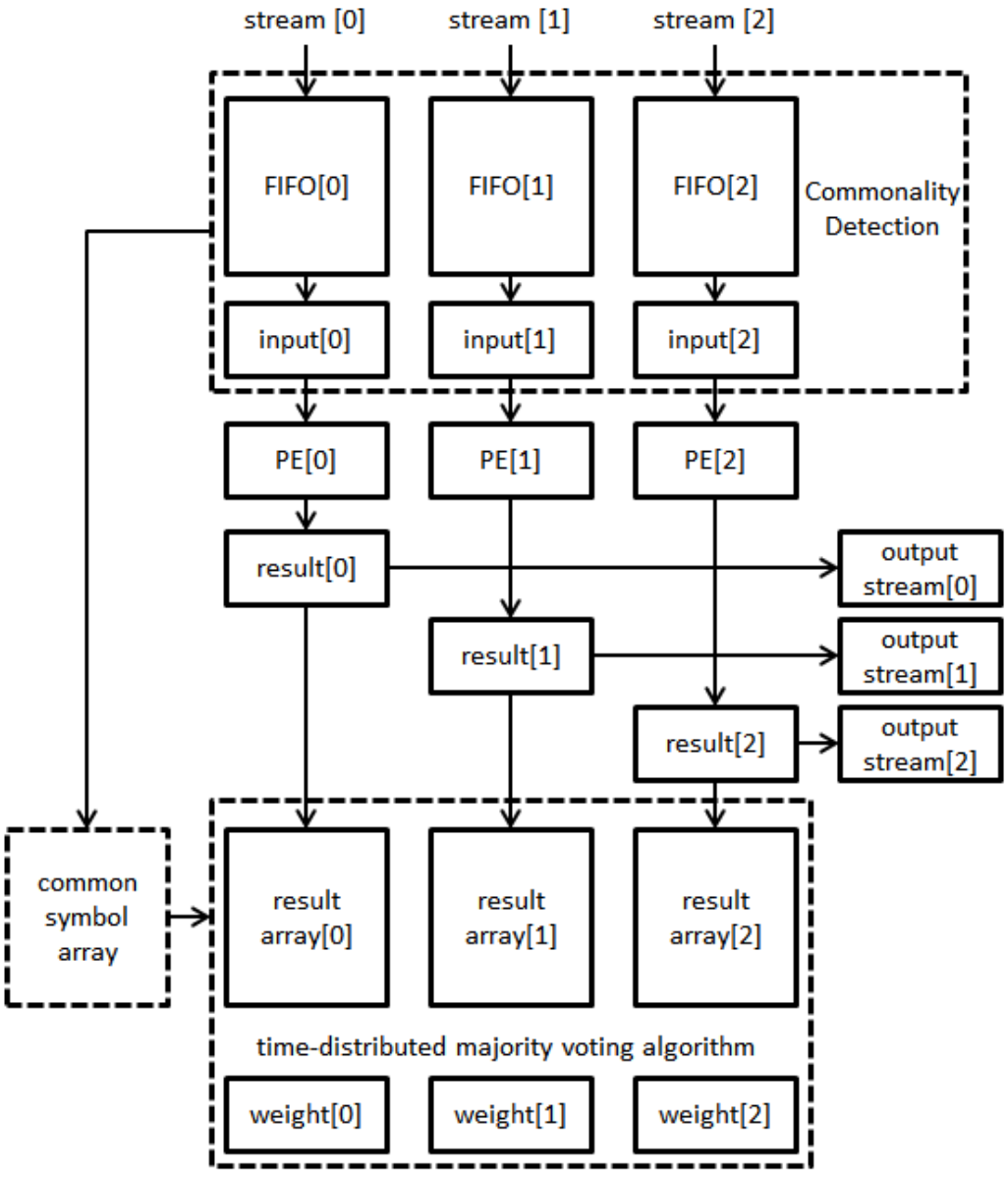


Figure 15: Block Diagram of Proposed Design
Data flows through the FIFO buffer, into the processing element, and to the output data stream. Surrounding circuitry (dotted lines) monitors for stream commonalities and performs majority voting when commonality results become available.

3.4 Implementing the Multi-Ported CAM in Verilog

The FIFO buffers and the common symbol array in the proposed design are implemented using CAM cells in Verilog HDL. CAM cells enable a fully associative search of the buffer contents. Each buffer needs to be searched for the active input patterns from the other two data streams. The common symbol array needs to be searched for the active input patterns from all three data streams. The conventional NAND-Type CAM cell only contains one search field. Additional search fields, match lines, and comparison logic have been added to the CAM cells used in this design so that multiple searches may occur concurrently. The modified CAM cells are described as multi-ported.

Two variations of multi-ported CAM cells are needed. A double-ported version is used in the FIFO buffers, and a triple-ported version is used in the common symbol array. The number of ports that a CAM cell has refers to the number of input search patterns a CAM cell can operate on. Figure 16 contains the Verilog behavioral implementation of the multi-ported CAM cells used in this design. The first snippet (lines 27-36) is the double-ported CAM cell which has inputs for the stored memory value (*q*), and two search lines (*sl1* and *sl2*). The outputs (*node1* and *node2*) are asserted high whenever the value of their respective search line input matches the stored memory value. The second snippet (lines 38-49) is the triple-ported CAM cell. It is identical to the double-ported CAM cell except that it also has a third search line input (*sl3*) and a third output (*node3*).

```
27  module nandcamx1_2port(q, sl1, sl2, node1, node2);
28     input q, sl1, sl2;
29     output node1, node2;
30     wire qbar, sl1bar, sl2bar;
31     inv invqbar(q,qbar);
32     inv invsl1bar(sl1,sl1bar);
33     inv invsl2bar(sl2,sl2bar);
34     assign node1 = ((q && sl1) || (qbar && sl1bar)) ? 1'b1 : 1'b0;
35     assign node2 = ((q && sl2) || (qbar && sl2bar)) ? 1'b1 : 1'b0;
36  endmodule
37
38  module nandcamx1_3port(q, sl1, sl2, sl3, node1, node2, node3);
39     input q, sl1, sl2, sl3;
40     output node1, node2, node3;
41     wire qbar, sl1bar, sl2bar, sl3bar;
42     inv invqbar(q,qbar);
43     inv invsl1bar(sl1,sl1bar);
44     inv invsl2bar(sl2,sl2bar);
45     inv invsl3bar(sl3,sl3bar);
46     assign node1 = ((q && sl1) || (qbar && sl1bar)) ? 1'b1 : 1'b0;
47     assign node2 = ((q && sl2) || (qbar && sl2bar)) ? 1'b1 : 1'b0;
48     assign node3 = ((q && sl3) || (qbar && sl3bar)) ? 1'b1 : 1'b0;
49  endmodule
```

Figure 16: Multi-Port CAM modules in Verilog.
The double-ported CAM contains a storage value (q), 2 search lines (sl1 and sl2) and 2 match indicators (node1 and node2). The triple-ported CAM contains a storage value (q), search lines (sl1, sl2 and sl3) and 3 match indicators (node1, node2, and node3.

A multi-ported CAM cell is instantiated for each stored bit in the FIFO buffers and common symbol array. All CAM bitcells for a word are connected serially using the same match line. When a match occurs, all node outputs for a corresponding search field will be asserted high. A match creates a discharge path to drop the precharged match line value low. In the proposed design, registers are used as the memory element while the CAM logic performs the search functions.

A de-asserted match line, for any word in the FIFO buffer, indicates that the corresponding search pattern has been found. A commonality is observed when the active input word for any single buffer is observed to exist anywhere in both of the other two buffers. When commonalities are observed in the FIFO buffers, the common input pattern

is stored in the common symbol array.

The common symbol array uses the active input patterns from the data streams as search fields. When a match is detected in the common symbol array, it indicates that a common pattern has arrived at the processing element and the result needs to be stored in the results array so it is available for majority voting. The common symbol array aligns the result arrays to their associated input pattern. Match lines are used to index into the correct position in the results array and capture the PE result when it becomes available. Valid flags are set to indicate the results are available. When results are available from all data streams for a given input pattern, the majority voting algorithm is initiated.

3.5 Evaluating faults in the C6288 Benchmark

Fault simulation is completed iteratively using Verilog force and release statements. Each possible fault in the C6288 is indexed in a table and associated with an integer value referred to as its fault node. There are 4,896 fault nodes in the C6288 Verilog benchmark used in this design. On the first fault node iteration, no fault is injected, and input test patterns are cycled through the processing element to produce correct results. In subsequent fault node iterations, a single fault is injected into the processing element and the input test patterns are processed with the fault present. The Verilog code snippet in Figure 17 shows the proper use of the Verilog force and release statements to cycle through all the possible faults.

```
119     always @ (posedge node_toggle) begin
120        case (node_word_int)
121           1: begin force pe1.A[0]=1'b0; end
122           2: begin release pe1.A[0]; force pe1.A[0]=1'b1; end
123           ...
124           ...
125           ...
126           4896: begin release pe1.C14_15; force pe1.C14_15=1'b1; end
127           default: begin release pe1.C14_15; end
128        endcase
129     end
```

Figure 17: Code snippet for injecting stuck-at faults during simulation
The force statement turns on a specific fault, release statement turns off the previous fault.

The simulation generates a table with 4,897 rows and 1,200 columns. The first row (fault node 0) contains the fault-free or "Golden" results for all input patterns. Each subsequent row contains results for the same input patterns simulated with a different fault node. The resultant data table may then be hashed to observe the behavior of the system for any singular fault or combination of faults.

Chapter 4 Results

4.1 Operational Verification of the Design

A voting opportunity exists when an input pattern is common to all FIFO buffers. CAMs search the buffers continuously to find commonalities. Figure 18 shows the observation of a commonality during simulation. Each *cam_reg* column contains buffer contents for one data stream with a depth of 32 words. Each row contains a 32-bit input pattern en route to a processing element. A buffer's asserted *hit* bit indicates that it's active input (*pe_reg*) has been found in all other buffers. In Figure 18 the active word from the middle buffer is common to the first and third buffers. When a commonality is detected, it gets stored in the common symbol array.

The asserted *hit* bit initiates a routine that stores the common input pattern to the common symbol array (*cam_comsym*), captures the common pattern's result in the results array (*result_reg*), and sets flags to indicate that the result is valid. Figure 19 shows the common pattern observed in Figure 18 added to the *cam_comsym* array and the PE result stored in the *results* array. When PE results for the common pattern become available from other buffers, they are stored and aligned in the results array with *valid* bits set. When all *valid* bits are asserted, the voting algorithm is executed as shown in Figure 20.

Cycle Count	2148		
hit	0	**1**	0
pe_reg	32'hb31d1556	**32'hffffd555**	32'hba098745
result	32'h0eed8cbe	32'hd5542aab	32'h624ce36d

Index	cam_reg[0]	cam_reg[1]	cam_reg[2]
[0]	32'h4dd52192	32'hfaca8cd9	32'h23c17f7a
[1]	32'h26ea90c9	32'h7d65466c	32'h11e0bfbd
[2]	32'hb31d1556	**32'hffffd555**	32'hba098745
[3]	32'h598e8aab	32'h237f4ac5	32'h5d04c3a2
[4]	32'h2cc74555	32'h91bfa562	32'hae8261d1
[5]	32'h9663a2aa	32'hc8dfd2b1	32'hd74130e8
[6]	32'h4b31d155	32'h646fe958	32'heba09874
[7]	32'h2598e8aa	32'hb237f4ac	32'hf5d04c3a
[8]	32'h92cc7455	32'hd91bfa56	32'h7ae8261d
[9]	32'hc9663a2a	32'h6c8dfd2b	32'hbd74130e
[10]	**32'hffffd555**	32'h3646fe95	32'hdeba0987
[11]	32'h32598e8a	32'h9b237f4a	32'hef5d04c3
[12]	32'h192cc745	32'hcd91bfa5	32'hf7ae8261
[13]	32'h0c9663a2	32'h66c8dfd2	32'hfbd74130
[14]	32'h864b31d1	32'h33646fe9	32'hfdeba098
[15]	32'h432598e8	32'h19b237f4	32'hfef5d04c
[16]	32'h2192cc74	32'h8cd91bfa	32'h7f7ae826
[17]	32'h90c9663a	32'h466c8dfd	**32'hffffd555**
[18]	32'h4864b31d	32'ha33646fe	32'h5fdeba09
[19]	32'ha432598e	32'h519b237f	32'h2fef5d04
[20]	32'h52192cc7	32'ha8cd91bf	32'h17f7ae82
[21]	32'ha90c9663	32'h5466c8df	32'h0bfbd741
[22]	32'h54864b31	32'h2a33646f	32'h05fdeba0
[23]	32'haa432598	32'h9519b237	32'h82fef5d0
[24]	32'hd52192cc	32'hca8cd91b	32'hc17f7ae8
[25]	32'hea90c966	32'h65466c8d	32'he0bfbd74
[26]	32'h754864b3	32'hb2a33646	32'hf05fdeba
[27]	32'hbaa43259	32'h59519b23	32'h782fef5d
[28]	32'hdd52192c	32'haca8cd91	32'h3c17f7ae
[29]	32'h6ea90c96	32'hd65466c8	32'h1e0bfbd7
[30]	32'h3754864b	32'heb2a3364	32'h8f05fdeb
[31]	32'h9baa4325	32'hf59519b2	32'h4782fef5

Figure 18: A commonality is observed
A buffer's hit signal is asserted to indicate the buffer's active input (pe_reg) is found in all other buffers.

	cam_comsym	Results[0]	Results[1]	Results[2]	Valid	Adj
[0]	32'h00000000	32'h00000000	32'h00000000	32'h00000000	4'b0000	4'b0111
[1]	32'h00000000	32'h00000000	32'h00000000	32'h00000000	4'b0000	4'b0111
[2]	**32'hffffd555**	32'h00000000	**32'hd5542aab**	32'h00000000	**4'b1010**	4'b1101
[3]	32'h00000000	32'h00000000	32'h00000000	32'h00000000	4'b0000	4'b0111
[4]	32'h00000000	32'h00000000	32'h00000000	32'h00000000	4'b0000	4'b0111
[5]	32'h00000000	32'h00000000	32'h00000000	32'h00000000	4'b0000	4'b0111
[6]	32'h00000000	32'h00000000	32'h00000000	32'h00000000	4'b0000	4'b0111
[7]	32'h00000000	32'h00000000	32'h00000000	32'h00000000	4'b0000	4'b0111
[8]	32'h00000000	32'h00000000	32'h00000000	32'h00000000	4'b0000	4'b0111
[9]	32'h00000000	32'h00000000	32'h00000000	32'h00000000	4'b0000	4'b0111
[10]	32'h00000000	32'h00000000	32'h00000000	32'h00000000	4'b0000	4'b0111
[11]	32'h00000000	32'h00000000	32'h00000000	32'h00000000	4'b0000	4'b0111
[12]	32'h00000000	32'h00000000	32'h00000000	32'h00000000	4'b0000	4'b0111
[13]	32'h00000000	32'h00000000	32'h00000000	32'h00000000	4'b0000	4'b0111
[14]	32'h00000000	32'h00000000	32'h00000000	32'h00000000	4'b0000	4'b0111
[15]	32'h00000000	32'h00000000	32'h00000000	32'h00000000	4'b0000	4'b0111
tally		-3	3	3		

Figure 19: Common patterns are stored in the common symbol array
When hit is asserted, the common pattern is stored in the common symbol array (cam_comsym) and the active input result is captured in the results array.

The *tally* values shown in Figure 19 and Figure 20 contain voting outcomes summed over time. The *adj* bits determine how the *tally* value for each PE is adjusted each time voting is executed. The *adj[0]* bit determines if the weights are to be adjusted. The other three bits, *adj[1]*, *adj[2]*, and *adj[3],* indicate whether the *tally* for PE1, PE2, and PE3, respectively is to be incremented or decremented. If a PE result belongs to the majority, its *tally* value is incremented. If the result is in the minority, *tally* is decremented. If all PE results are matched or all are mismatched, the *tally* values are not adjusted. After voting has completed for all test patterns, the PE(s) with the highest *tally* is interpreted as healthy. The PE(s) with the lowest *tally* is deemed faulty. The *tally* values are then reset and the process begins again.

	cam_comsym	Results[0]	Results[1]	Results[2]	Valid	Adj
[0]	32'h00000000	32'h00000000	32'h00000000	32'h00000000	4'b0000	4'b0111
[1]	32'h00000000	32'h00000000	32'h00000000	32'h00000000	4'b0000	4'b0111
[2]	**32'hffffd555**	**32'hd5542aad**	**32'hd5542aab**	**32'hd5542aab**	**4'b1111**	**4'b1011**
[3]	32'h00000000	32'h00000000	32'h00000000	32'h00000000	4'b0000	4'b0111
[4]	32'h00000000	32'h00000000	32'h00000000	32'h00000000	4'b0000	4'b0111
[5]	32'h00000000	32'h00000000	32'h00000000	32'h00000000	4'b0000	4'b0111
[6]	32'h00000000	32'h00000000	32'h00000000	32'h00000000	4'b0000	4'b0111
[7]	32'h00000000	32'h00000000	32'h00000000	32'h00000000	4'b0000	4'b0111
[8]	32'h00000000	32'h00000000	32'h00000000	32'h00000000	4'b0000	4'b0111
[9]	32'h00000000	32'h00000000	32'h00000000	32'h00000000	4'b0000	4'b0111
[10]	32'h00000000	32'h00000000	32'h00000000	32'h00000000	4'b0000	4'b0111
[11]	32'h00000000	32'h00000000	32'h00000000	32'h00000000	4'b0000	4'b0111
[12]	32'h00000000	32'h00000000	32'h00000000	32'h00000000	4'b0000	4'b0111
[13]	32'h00000000	32'h00000000	32'h00000000	32'h00000000	4'b0000	4'b0111
[14]	32'h00000000	32'h00000000	32'h00000000	32'h00000000	4'b0000	4'b0111
[15]	32'h00000000	32'h00000000	32'h00000000	32'h00000000	4'b0000	4'b0111
Tally		**-4**	**4**	**4**		

Figure 20: Voting is initiated
When all results for common pattern are available, voting is initiated. *Tally* is updated to reflect voting outcome.

4.2 Fault Coverage with 1,200 Pseudorandom Input Patterns

Pseudorandom data may be used to test logic circuits. The number of pseudorandom patterns required to provide 100% fault coverage is heavily dependent on the circuit used as a processing element. The C6288 benchmark was selected as the PE. A pseudorandom sample of 1,200 32-bit patterns was collected. A sample size of 1,200 patterns was chosen somewhat arbitrarily to be 100 times larger than the 12 patterns provided in the C6288 documentation.

The 1,200 input test patterns were collected using a linear feedback shift register (LFSR) and only saving every 32nd output pattern. The appearance of randomization is

improved by completely shifting one pattern out of the LFSR before the next pattern is taken. All 1,200 pseudorandom patterns were simulated with an ideal processing element as well as all 4,896 possible faults. The result of this simulation is a data table with 4,897 rows and 1,200 columns. Each row represents a different fault state, and each column contains output results for one of the input patterns. From this table, all analysis and evaluations can be performed or derived to assess the design.

Fault coverage was evaluated iteratively by cycling through each fault node in the data table to check for incorrect results. There were 17 SA0 fault nodes in which no erroneous results were observed for any of the 1,200 pseudorandom input patterns. The un-activated faults were mapped to their location in the C6288, which is shown in Figure 21. Sixteen un-activated faults belong to the half-adder modules and one belongs to a full adder module. Upon examination, these 17 faults are not observable because there is no input combination that would ever cause these nodes to compute to logic 1. The fault nodes are therefore perpetually SA0 by virtue of the C6288 design. The remaining 4,879 fault nodes are all activated by some subset of the 1,200 pseudorandom input patterns.

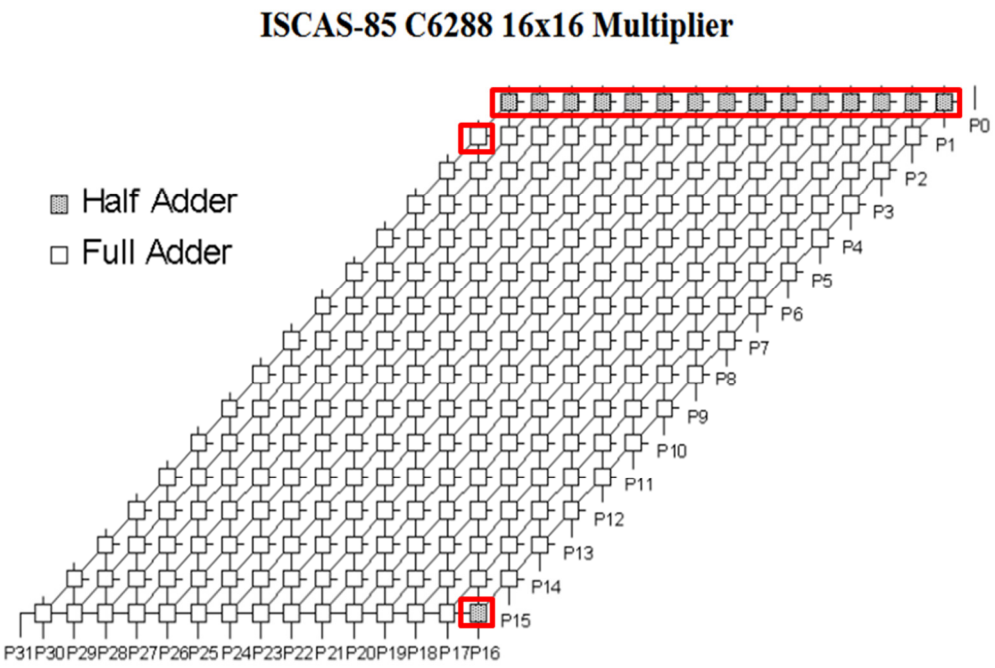


Figure 21: Half and Full Adders containing un-observable faults [3]

Fault coverage for the C6288 benchmark has been calculated at several interval set sizes between 1 and 1,200 pseudorandom input patterns. Figure 22 shows the escapes (non-activated faults) and fault coverage with respect to the pseudorandom input set size. Escapes drop to zero and 100% fault coverage of the observable 4,879 possible faults is achieved with 150 pseudorandom input patterns.

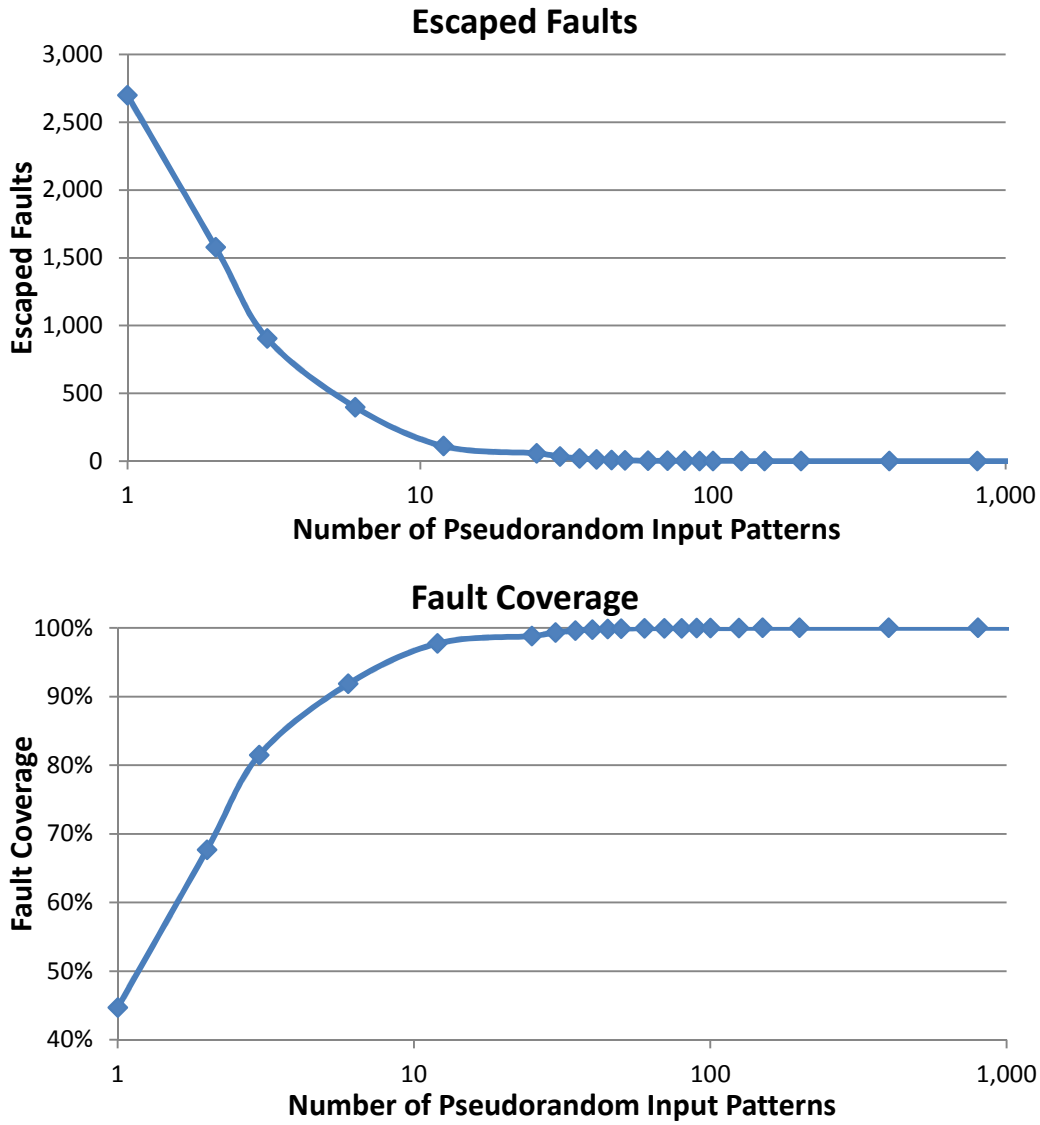


Figure 22: Fault Coverage profile for C6288 Benchmark
The top plot shows the number of faults that are not activated by the input pattern set. The bottom plot shows the fault coverage as percentage of observable faults that are activated by the pseudorandom input patterns. Both plots are shown with respect to the size of the pseudorandom input data set.

4.3 Fault Coverage with Multiple Faulty Elements

A lockstep TMR system effectively masks erroneous data if only a single

redundant PE is faulty. This thesis considers the case in which multiple PE's may be faulty. The structural Verilog implementation of the C6288 benchmark contains 2,448 discrete nodes. Each node may be modeled as a SA0 or a SA1 fault. In total, there are 4,896 possible single stuck-at faults. A simulation was conducted in which pseudorandom input patterns were used as inputs to three processing elements. The first processing element contains no faults and its result is designated as G (Golden). The 2nd and 3rd processing elements each contain a single stuck-at fault from the 4,896 possible faults, and their outputs are designated F1 and F2 respectively. For this simulation, the faulty elements are associative and each fault combination is only simulated once (Fault1+Fault2 = Fault2+Fault1). The case in which Fault1 = Fault2 is discarded in the simulation because aliased results would be generated anytime the fault is activated. A total of 11,982,960 fault pairs have been simulated.

$$fault\ pairs\ to\ simulate = \frac{(N)(N-1)}{2} = \frac{(4{,}896)(4{,}896-1)}{2} = 11{,}982{,}960$$

The simulation used the data table from section 4.2 to hash together all possible fault combinations and capture the voting results for all 1,200 pseudorandom test patterns. The simulation output was written to text files with outcome statistics tabulated for several pseudorandom input pattern set sizes. Exhaustive fault-pair analysis was completed for a single point estimate with set sizes of 2, 3, 6, 12, 25, 50, 100, 200, 400, 800, and 1,200 pseudorandom input patterns. Table 5 shows definitions for the tabulated statistics obtained during the simulation for each fault-pair. In the explanation column, G, F1, and F2 refer to the output results of the Golden, Faulty1, and Faulty2 elements

respectively.

Table 5: Definitions for statistics obtained during simulation for each fault pair.

Column Name	Example	Explanation
Fault Nodes	G_N12_N13	1st element is Golden 2nd element has fault index 12 3rd element has fault index 13
G_Tally	606	Final *tally* vote count (unbiased) for 1st Element after input set has been applied
F1_Tally	22	Final *tally* vote count (unbiased) for 2nd Element after input set has been applied
F2_Tally	-22	Final *tally* vote count (unbiased) for 3rd Element after input set has been applied
ALIASES	282	Number of occurrences where F1=F2
G_0_Tally	594	Number of occurrences where there is no minority; No change to G_Tally
G_1_Tally	606	Number of occurrences where G is in the majority; Increment G_Tally
G_-1_Tally	0	Number of occurrences where G is in the minority; Decrement G_Tally
F1_0_Tally	594	Number of occurrences where there is no minority; No change to F1_Tally
F1_1_Tally	314	Number of occurrences where F1 is in the majority; Increment F1_Tally
F1_-1_Tally	292	Number of occurrences where F1 is in the minority; Decrement F1_Tally
F2_0_Tally	594	Number of occurrences where ther is no minority; No change to F2_Tally
F2_1_Tally	292	Number of occurrences where F2 is in the majority; Increment F2_Tally
F2_-1_Tally	314	Number of occurrences where F2 is in the minority; Decrement F2_Tally
XYZ[derived]	312	Number of occurrences where G≠F1≠F2
XXX[derived]	282	Number of occurrences where G=F1=F2

The statistic from Table 5 that is of most interest is *G_-1_Tally*. The *G_-1_Tally* counts the number of input patterns that generate aliased results for a given fault

combination. This count is incremented for each pattern in which *F1* and *F2* agree, and vote out *G*. Table 6 shows a summary of aliasing occurrences for different input pattern set sizes. Aliasing occurs with over 99% of the activated observable faults in the C6288 benchmark. Only 28 activated observable faults never generate aliased results when combined with any other fault during this simulation. In a lockstep TMR system with multiple faulty C6288 elements, a substantial subset of possible fault combinations (approximately 3.2%) exist that will produce aliased results and evict the correct result for some input patterns.

Table 6: Aliasing Summary

Input Pattern Set Size	Unactivated Observable Faults	Activated Observable Faults	Activated Observable Fault Combinations	Number of Faults that Alias	Number of Faults that do not Alias	Percentage of Activated Observable Faults that Alias
2	1577	3302	5449951	3288	14	99.58%
3	905	3974	7894351	3946	28	99.30%
6	398	4481	10037440	4447	34	99.24%
12	111	4768	11364528	4734	34	99.29%
25	58	4821	11618610	4793	28	99.42%
50	5	4874	11875501	4846	28	99.43%
100	2	4877	11890126	4849	28	99.43%
200	0	4879	11899881	4851	28	99.43%
400	0	4879	11899881	4851	28	99.43%
800	0	4879	11899881	4851	28	99.43%
1200	0	4879	11899881	4851	28	99.43%

Table 7 shows the number and percentage of fault combinations that alias with respect to the pseudorandom input pattern set size. The fault combinations in the table are comprised only of faults that are activated by the input pattern set. Figure 23 and Figure 24 graphically show the relationship between pseudorandom input pattern set size and the frequency of aliasing fault combinations. With fewer input patterns, some faults may not

be activated and therefore cannot contribute to aliasing. As the input set size increases, fault coverage improves and more aliasing fault combinations are observed. Figure 22 showed that in this single point estimate, 100% fault coverage is achieved with approximately 150 pseudorandom input patterns. A larger set of 1,200 patterns increases fault activation and the frequency of aliasing in fault pairs. Figure 23 shows that as the input set size increases, the percentage of fault combinations that alias approaches an upper bound of around 3.2%. Figure 24 indicates this upper bound is around 380,000 fault pairs.

Table 7: Aliasing Fault Combinations.

Pattern set size	Total Activated Observable Fault Combinations	Aliasing Fault Combinations	Percentage of fault combinations that alias
2	5449951	96361	1.77%
3	7894351	145437	1.84%
6	10037440	205827	2.05%
12	11364528	276495	2.43%
25	11618610	317298	2.73%
50	11875501	353489	2.98%
100	11890126	369431	3.11%
200	11899881	377027	3.17%
400	11899881	378817	3.18%
800	11899881	379375	3.19%
1200	11899881	379437	3.19%

The simulation results have been filtered to include only the 379,437 aliasing fault combinations from the 1,200 pattern set. Over 99% of activated observable faults are found to alias with at least one other fault. The frequency that aliasing occurs for all faults is shown in Figure 25. The distribution shows the percentage of possible fault combinations that alias for all singular aliasing faults. For example, there are 797 singular faults that alias with 3% to 3.5% of other faults. On average each fault aliases with

approximately 3.2% of other possible faults.

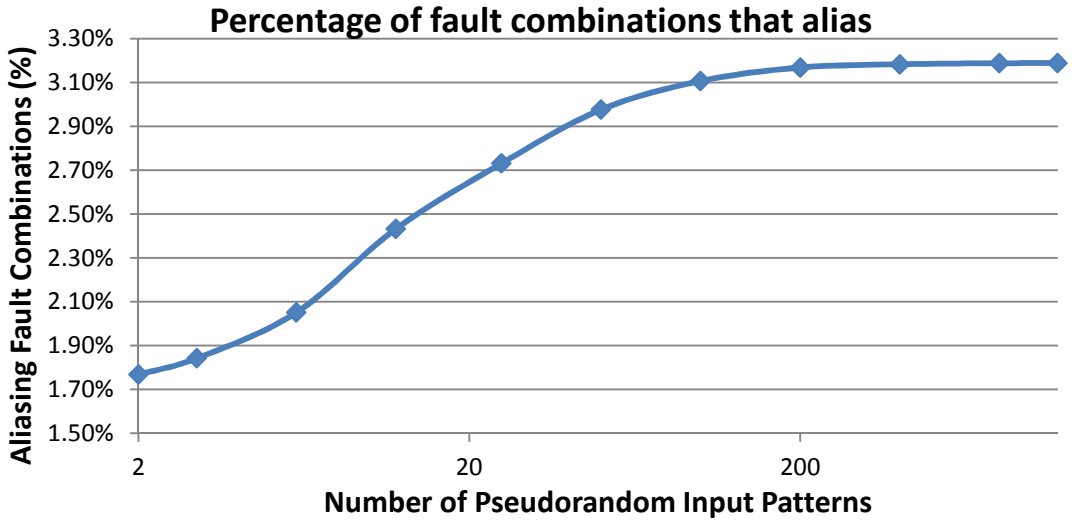


Figure 23: Percentage of fault combinations that alias
Aliasing increases as coverage improves because more faults are activated. Only activated and observable faults can contribute to aliasing. Therefore, the percentages in this plot are given with respect to the total number of activated observable fault combinations for each input set size.

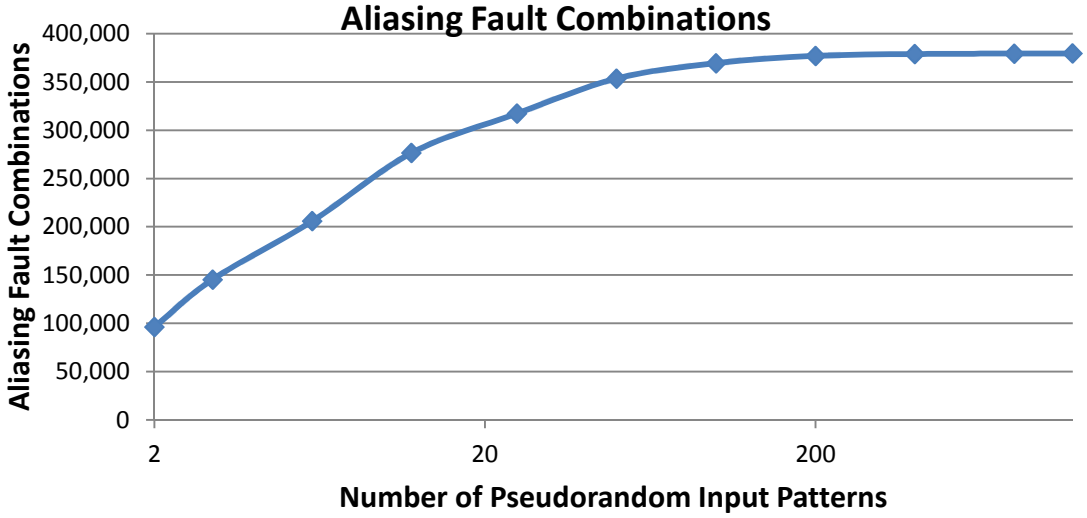
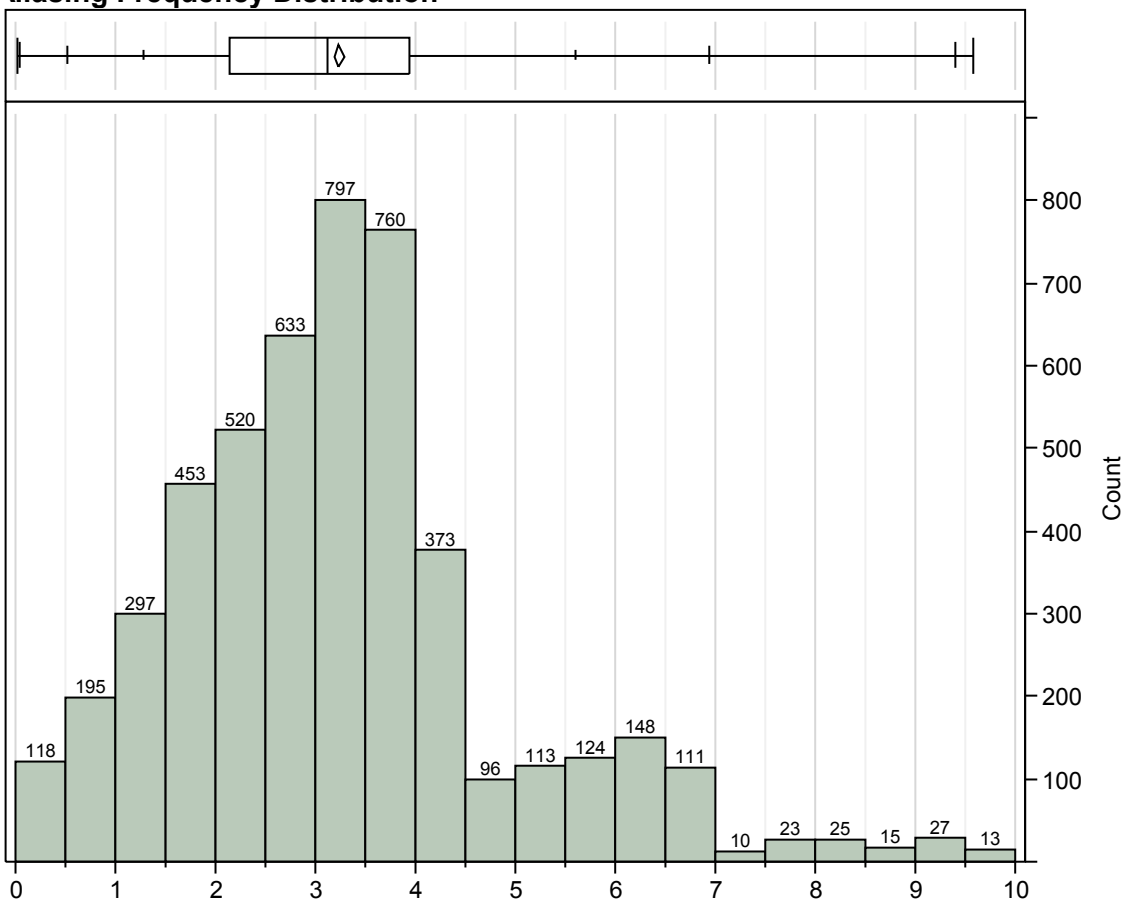


Figure 24: Aliasing Fault Combinations
As input pattern set size increases, observation of aliasing fault combinations increases to a finite upper bound.

Aliasing Frequency Distribution



Quantiles		
100.00%	maximum	9.5857
99.50%		9.4001
97.50%		6.947
90.00%		5.6071
75.00%	quartile	3.9373
50.00%	median	3.1128
25.00%	quartile	2.1439
10.00%		1.2781
2.50%		0.5154
0.50%		0.0466
0.00%	minimum	0.0206

Moments	
Mean	3.224832
Std Dev	1.637596
Std Err Mean	0.023512
Upper 95% Mean	3.270927
Lower 95% Mean	3.178738
N	4851

Figure 25: Frequency distribution of aliased fault pairs for all faults.
Shows the percentage of possible fault combinations that alias for all singular faults.

The distribution in Figure 25 shows three distinct modes. The modes are confirmed in the cumulative distribution plot shown in Figure 26. From Figure 26, 84.3% of singular faults fall into Mode 1. 11.7% of singular faults fall into Mode 2, and 4% of singular faults fall into Mode 3. Mode 1 indicates singular faults where aliasing was observed for less than 211 fault combinations. Mode 2 indicates singular faults where aliasing was observed for between 211 and 317 possible fault combinations. Mode 3 indicates singular faults where aliasing was observed for more than 318 possible fault combinations.

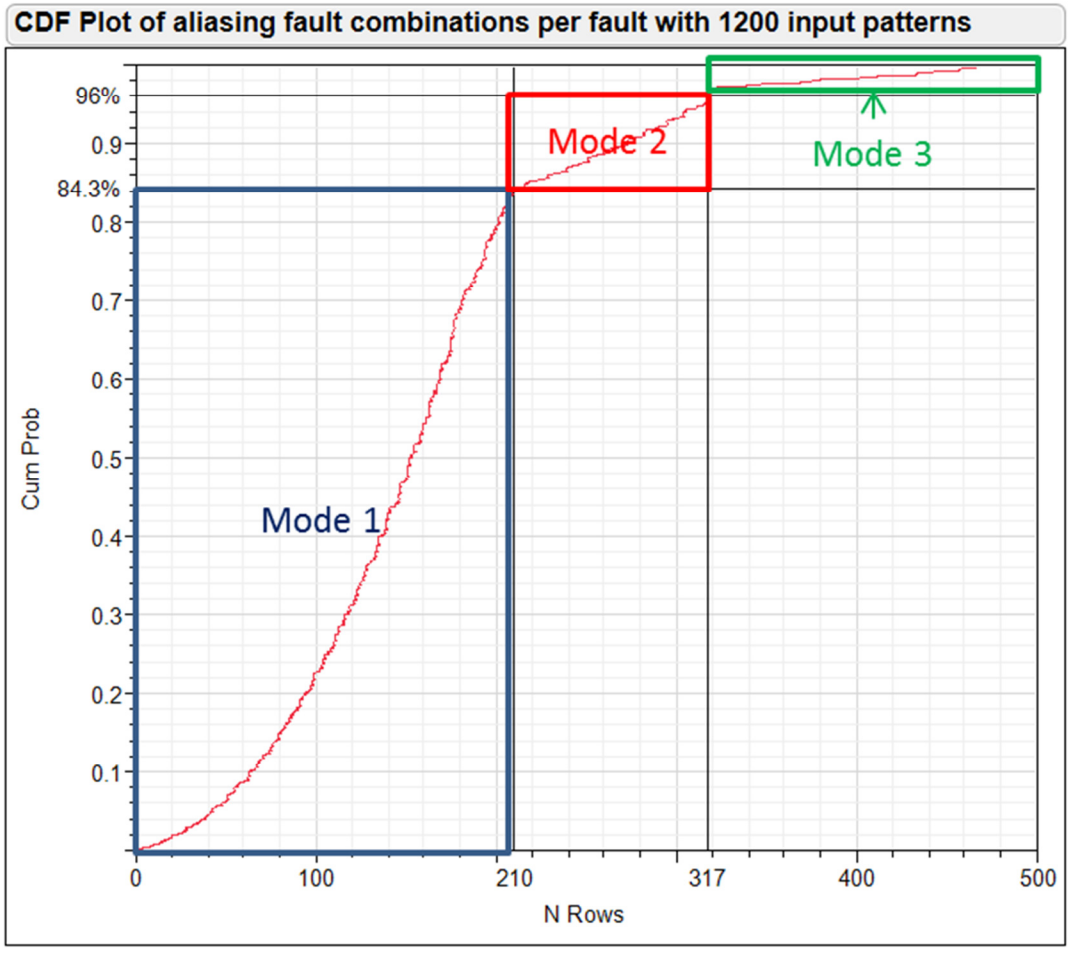


Figure 26: Cumulative Distribution Plot of aliasing fault combinations

There are two factors that contribute to the multiple distribution modes shown in Figure 26. These factors are proximity and propagation paths. Proximity refers to the spatial columnar distance between aliasing fault locations in the C6288 PE. 100% of observed aliasing fault pairs are either in the same column or in an immediately adjacent column in the PE. Figure 27 shows an example of fault proximity. A processing element that contains a fault in column P6, shown in blue, may generate aliased results with a processing element containing a fault in column P5, P6, or P7. No aliasing was observed for faults pairs that are more than one column apart.

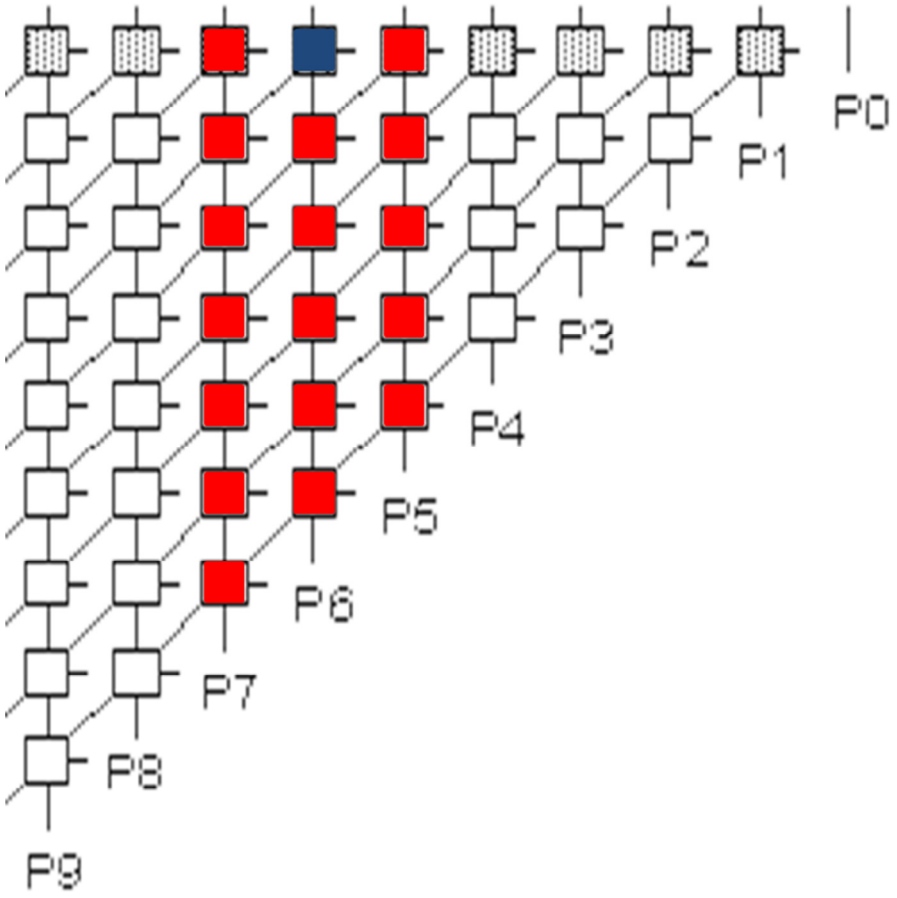


Figure 27: Aliasing Fault Proximity example [3]
A fault in column P6 (blue) will have aliasing fault pairs located in all adder modules from columns P5, P6, and P7.

The number of propagation paths for a fault also contributes to aliasing and multiple distribution modes. Figure 28 shows the gate level schematics for the adders used in the C6288. Faults that can propagate to both Sum and Carry outputs will have a higher frequency of aliasing fault pairs than faults that only propagate to a single output. In Figure 28, the outputs of g1 and g5 have propagation paths that include both the sum and the carry output for the adder. Faults at g1 and g5, or faults equivalent to these, will have more aliasing fault pairs because they have more propagation paths.

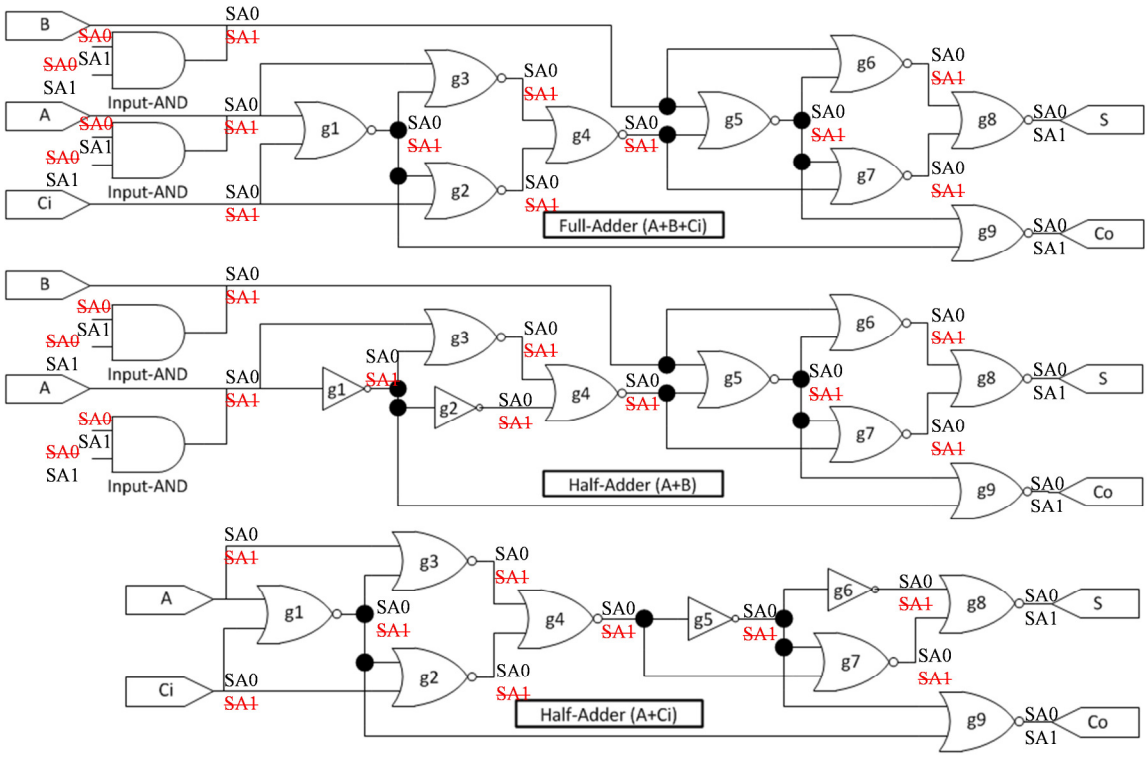


Figure 28: Gate level schematics of the adders used in the C6288
Gates g1 and g5 may be propagated to both the Sum and Carry outputs. Faults that propagate to multiple adder outputs have a higher number of aliasing fault pairs. Input-AND gates do not appear on each adder. They are shown here as options. All internal stuck-at faults for the adders are shown here. Faults shown in red are equivalent, collapsible equivalent faults that may be detected using patterns that would activate their equivalent counterparts.

Figure 29 shows the spatial relationship of aliasing faults and three distribution modes from Figure 25. The X-axis contains the fault node index and are ordered as follows:

1. Range 1-64 contains SA0 and SA1 faults for the 32 input nodes.
2. Range 65-576 contains SA0 and SA1 faults for the “and” gates that provide inputs to the adders.
3. Range 577-4,896 contains faults within the matrix of adders. The adder fault nodes are ordered from right to left and top to bottom. Each “spike” represents one of the 16 matrix rows (partial product) in the C6288.

The scatter plot in Figure 29 shows that the number of aliasing fault combinations is heavily dependent on the placement of the singular fault within a C6288 matrix row. Each matrix row in the C6288 is a partial product used to perform the multiplication function. A heavier concentration of aliasing fault combinations exists in the horizontal center of the C6288. The horizontal center of the C6288 is recognizable in the scatterplot by the 16 local maxima points. The local maxima points correspond to faults located in the centermost adder module of each partial product row in the C6288 adder matrix from Figure 21.

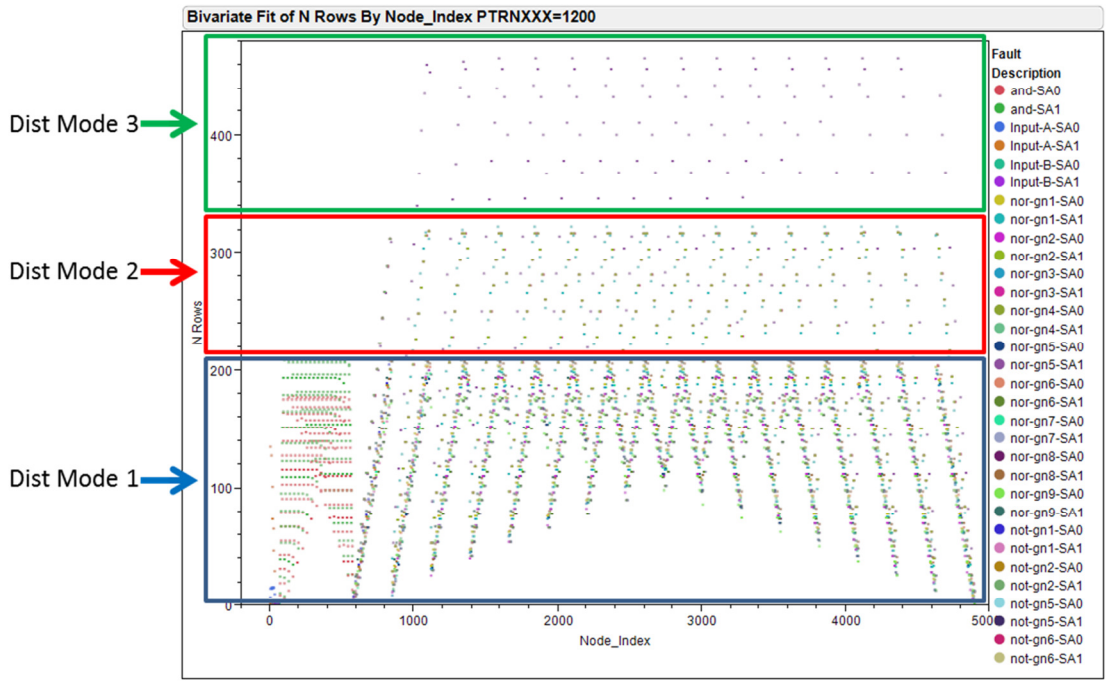


Figure 29: Fault Alias Spatial dependence
The Node index (x-axis) provides information about the location of each fault in the C6288. Range 1-64 contains SA0 and SA1 faults for the 32 input nodes. Range 65-576 contains SA0 and SA1 faults for the "and" gates that provide inputs to the adders. Range 577-4,896 contains faults within the matrix of adders. The adder fault nodes are ordered from right to left and top to bottom. Each "spike" represents one of the 16 matrix rows (partial product) in the C6288.

In Figure 28, the faults shown in red are collapsible equivalent faults. Equivalent faults exhibit the same syndrome. In single stuck-at fault testing, equivalent faults are collapsed such that only one fault is targeted for testing. Any equivalent faults are then observable when testing the single targeted fault. To make testing more efficient, all possible faults are collapsed to a minimal target set. A minimized set of input patterns is computed to activate the target fault set. During test, the circuit operates on the minimal set of test patterns to activate all possible faults. If no incorrect results are observed, the circuit is deemed healthy. The documentation for the C6288 benchmark provides the minimum set of 12 input test vectors. On average, with 4,851 observable faults in the

C6288, each input test vector activates approximately 404 faults. High fault coverage with a small input set is a benefit of using the single stuck-at fault model. Exhaustive research has been done to improve the efficiency of fault collapsing and test pattern generation. The C6288 is frequently used as a benchmark for evaluating the efficiency and effectiveness of algorithms to collapse faults and minimize test patterns [8][9].

Identifying equivalence in fault pairs is necessary to make testing of large logic circuits manageable. Testing a minimal target fault set with a minimal input pattern set requires that the tester already know the correct result for each input pattern beforehand. A comparator is then used to identify any mismatched results and label an element as faulty. Lockstep TMR methodology inherits the assumption that, at most, only a single fault is present. This assumption is maintained even though the processing element hardware must be triplicated and additional hardware added to instantiate the voting logic. The correct result is not known beforehand, and reliability is wholly dependent on the presence of two healthy processing elements.

If two PE's contain equivalent faults, then aliased results should be expected. In this thesis, aliasing has been observed in fault pairs that are not equivalent. Lockstep TMR may be a riskier scenario than conventional testing because the occurrence of aliased results extends beyond equivalent faults. The data collected in this work for the C6288 multiplier suggests that of all observable fault combinations, approximately 3.2% generate aliased results for a subset of input patterns. When a second processing element contains a fault, Lockstep TMR becomes less robust. The likelihood of propagating incorrect aliased results increases. Time distributed voting (TDV) is proposed in this thesis to provide active fault tolerant coverage when multiple PE's are faulty.

4.4 Fault Coverage using Time Distributed Voting

The design proposed in this thesis uses time-distributed voting (TDV) to determine if processing elements (PE) are healthy or faulty. The tabulated result files contain *G_Tally*, *F1_Tally*, and *F2_Tally*. These *tally* values represent the final weights after accumulating majority voting results for all input test patterns. All *tallies* are first initialized to an integer value of zero. With each pseudorandom test pattern, a vote is executed to determine if and how the *tally* values for each PE are to be adjusted. If the three PE's generate identical results, then the *tally* values will remain unchanged. If a majority of PE's generate identical results, while a minority of PE's generate mismatched results, the *tally* values for PE's in the majority are incremented and the *tally* values for PE's in the minority are decremented. If all three PE's generate mismatched results, no majority is observed and no adjustment is made to the *tally* values. After all test patterns have been processed, the PE(s) with the highest *tally* is deemed healthy. The TDV decisions are non-biased and are in no way skewed to favor the golden processing element.

TDV is not immune to aliasing, but it relies on averaging to provide robustness when multiple PE's are faulty. When faulty elements generate correct results, they vote with the Golden element(s) to increase its *tally*. Ideally, after all test patterns have been processed, the Golden element(s) represents the dominant *tally*. Figure 30 contains the Venn diagram of the expected behavior for non-aliasing fault pairs. The green overlapping regions in the diagram indicate majority voting results that favor the Golden PE and to its TDV *tally* value. Ideally, TDV would always favor the Golden element(s).

In simulation using the C6288 PE, TDV was able to correctly identify the Golden PE for 96.8% of fault pairs with no observed aliasing.

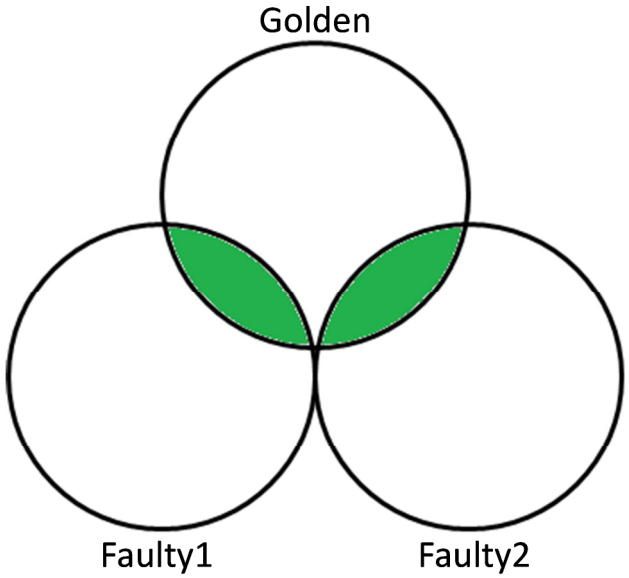


Figure 30: Expected Venn diagram of Time-Distributed Voting.
When Faulty1 or Faulty2 generate correct results (green), they vote with the Golden element. After processing all test patterns the Golden element is correctly identified.

Over 99% of faults in the C6288 benchmark have aliasing fault pairs. The aliasing fault pairs may be represented using the Venn diagram shown in Figure 31. Whenever Faulty1 and Faulty2 results are identical, but different from the Golden result, the *G_Tally* metric is decremented. If *G_Tally* is decremented too frequently, the outcome after processing all test patterns may incorrectly identify the Golden element as faulty as seen in Figure 32.

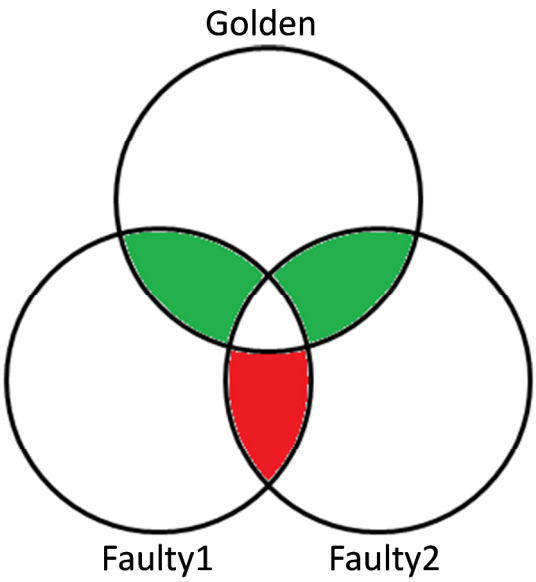


Figure 31: Venn diagram for aliasing fault pairs
Here, the overlapping regions between Faulty1 & Faulty2 (red) adversely affect vote averaging for the Golden element. As long as the overlap of Faulty1 & Golden + Faulty2 & Golden (green) is greater than Faulty1 & Faulty2 (red), TDV is able to correctly resolve the Golden element.

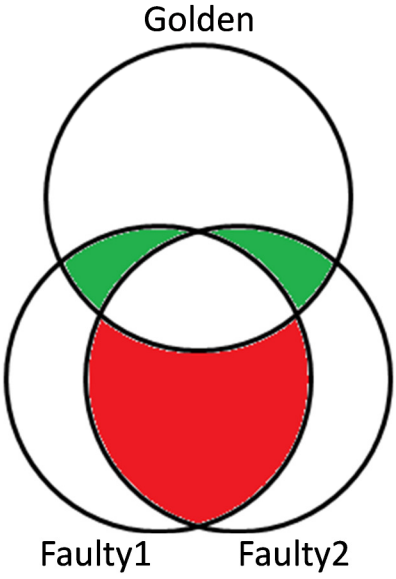


Figure 32: Venn diagram when Golden element is evicted
When Faulty1 & Faulty2 (red) alias too frequently, the Golden element may be erroneously evicted.

Pseudorandom input patterns are used in the simulation to activate faults and evaluate fault coverage. If the faulty PE's contain aliasing fault pairs, the PE favored by the TDV may change based on the input set size. Table 8 shows TDV outcomes for select aliasing fault pairs using different pseudorandom input pattern set sizes. In Table 8, a *0* indicates the TDV outcome was correct for the indicated number of input patterns. A *1* indicates the TDV outcome was incorrect. The selected fault pairs show that TDV outcomes may toggle between correct and incorrect as more patterns are presented to the system. While aliasing coverage percentage remained essentially flat, the aliasing fault pairs that are covered may shift with different input pattern sets. Coverage may not be achievable for all aliasing fault pairs, but designers may be able to engineer input pattern sets to optimize coverage of aliasing fault pairs.

Table 8: Aliasing fault pair TDV outcomes vary with input set size

PE1	PE2	PE3	200 Patterns	400 Patterns	800 Patterns	1200 Patterns
Golden	N997	N2263	0	0	0	1
Golden	N4297	N4796	0	0	1	0
Golden	N752	N2514	0	0	1	1
Golden	N411	N3247	0	1	0	0
Golden	N1000	N2759	0	1	0	1
Golden	N997	N2008	0	1	1	0
Golden	N872	N1121	0	1	1	1
Golden	N868	N1121	1	0	0	0
Golden	N3162	N4676	1	0	0	1
Golden	N1823	N2577	1	0	1	0
Golden	N3508	N4513	1	0	1	1
Golden	N843	N3362	1	1	0	0
Golden	N3745	N4760	1	1	0	1
Golden	N435	N2694	1	1	1	0
Mean(XYZ) voting results			60.83	121.61	243.29	364.88
Mean(XXX)voting results			11.62	23.01	45.75	68.91

The bottom two rows in Table 8 shows the number of patterns for each input set size that generated identical results (XXX) and different results (XYZ). The occurrence of XXX and XYZ results trends linearly with the input set size. In total, there are approximately 11.9 million fault combinations that were tested during the TDV simulation. The simulation used three C6288 PE's. One element was ideal (Golden) while 2 elements contained different faults (Faulty1 and Faulty2). The processing elements are intended to operate on independent data streams with test patterns interleaved into the data streams. Aliasing fault pairs were observed for over 99% of the activated observable singular faults in the circuit. Observing common test patterns in the three data streams created voting opportunities.

Table 9: TDV outcomes for 12 ATPG patterns

Test Patterns	12 ATPG Patterns
% Correct No-Aliasing Pairs	97.52%
% Aliasing pairs	2.48%
% Correct Aliasing Pairs	0.50%
% Total Correct Pairs	98.02%
% Incorrect Aliasing Pairs	1.98%

Table 9 shows TDV fault coverage for multiple faulty PE's using the 12 engineered test patterns for the C6288. Table 10 shows fault coverage as a count of passing and failing fault pairs with different pseudorandom input pattern set sizes. Table 11 shows fault coverage as a percentage of all possible fault pairs. TDV correctly identified both healthy and faulty elements in approximately 96.8% of all fault combinations because no aliasing was observed. An additional 1.8% coverage is obtained by correctly identifying healthy and faulty elements in over half of the aliasing fault pairs. The remaining 1.4% of fault pairs are escapes in which the healthy element is

evicted in favor of a faulty element. Test escapes in which both faulty PE's are deemed healthy and the golden element evicted account for 0.02% of fault pairs. An equivalent Lockstep TMR system provides no coverage for multiple faulty PE's. In this single point sample using the C6288, the proposed TDV design provides the same fault coverage as lockstep TMR for single faulty PE's and extends coverage to 98.6% of fault pairs that may occur when two PE's are faulty.

Table 10: Fault Pair Coverage Statistics by pattern set size (Integer count)

Test Patterns	Total Fault Pairs	Correct No-Aliasing Pairs	Aliasing Pairs	Correct Aliasing Pairs	Total Correct Pairs	Incorrect Aliasing Pairs	Test Escapes
200	11,899,881	11,522,854	377,027	212,446	11,735,300	164,581	2724
400	11,899,881	11,521,064	378,817	214,242	11,735,306	164,575	2276
800	11,899,881	11,520,506	379,375	216,910	11,737,416	162,465	1942
1,200	11,899,881	11,520,444	379,437	217,408	11,737,852	162,029	1902

Table 11: Fault Pair Coverage Statistics by pattern set size (Percentage)

Test Patterns	Total Fault Pairs	% Correct No-Aliasing Pairs	% Aliasing Pairs	% Correct Aliasing Pairs	% Total Correct Pairs	% Incorrect Aliasing Pairs	% Test Escapes
200	11,899,881	96.83%	3.17%	1.79%	98.62%	1.38%	0.0229%
400	11,899,881	96.82%	3.18%	1.80%	98.62%	1.38%	0.0191%
800	11,899,881	96.81%	3.19%	1.82%	98.63%	1.37%	0.0163%
1,200	11,899,881	96.81%	3.19%	1.83%	98.64%	1.36%	0.0160%

Chapter 5 Conclusion and Recommendations

This thesis has explored challenges facing semiconductor manufacturers and designers as defects become increasingly difficult to prevent. Fault tolerant methods, such as TMR, have been discussed as options to protect yield and improve reliability. The shortcomings of TMR are the motivation for research and the new fault tolerant methodology proposed herein. TMR's effectiveness is limited to cases where no more than one processing element (PE) may be faulty. In practice, once a part is put into use, the onset of multiple faults is neither predictable nor preventable. When multiple processing elements are faulty in a TMR system, aliasing may cause the correct result, if there is one, to be evicted and an incorrect result to be propagated.

The methodology proposed in this thesis is an active fault tolerant technique with the ability to distinguish faulty elements from healthy elements in the presence of single or multiple faults. Time distributed voting (TDV) is proposed as a technique that accumulates voting results over time to recognize healthy PE's. TDV has been implemented in a Verilog HDL prototype design. The prototype design evaluates TDV using the ISCAS '85 C6288 benchmark as a PE. Simulations have been completed to verify the design and evaluate fault coverage provided by TDV with faults injected into multiple faulty processing elements.

The TDV technique was able to correctly identify the healthy processing element for 98.6% of all fault pairs. TDV even provided coverage for some aliasing fault pairs (1.84%). Based on results using the C6288 benchmark as the processing element, TDV has successfully extended fault coverage to a system with single and/or multiple faulty processing elements. TDV does not ensure detection of faulty elements in all cases.

When PE's contain aliasing fault pairs TDV may evict the healthy element as was demonstrated with 1.4% of the fault pairs in the C6288.

Designers who consider using TDV should analyze the target PE to determine the frequency of aliasing fault pairs. This work has shown that aliasing extends beyond equivalent faults. Conventional fault collapsing does not capture all aliasing fault pairs. TDV may not be an appropriate design choice for PE's with a large number of aliasing fault pairs. TDV may provide an effective alternative to lockstep TMR and enable fault tolerant design of systems in the presence of multiple faults. The following recommendations may further enhance the capabilities discussed in this work:

- Adding more PE's may further reduce the probability of aliasing faulty PE's to conspire against the healthy PE.
- Probability of aliasing may be reduced by using PE's with the same function, but different implementation.
- It may be possible to engineer a minimal test pattern set to further reduce aliasing and better evaluate the PE.

References

[1] K. Chakrabarty and J.P. Hayes. Zero-aliasing space compaction of test responses using multiple parity signatures. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 6(2):309 –313, June 1998.

[2] B. El-Kareh, A. Ghatalia, and A.V.S. Satya. Yield management in microelectronic manufacturing. In *Electronic Components and Technology Conference, 1995. Proceedings., 45th*, pages 58 –63, May 1995.

[3] J.P. Hayes. Iscas high-level models. 1999. http://www.eecs.umich.edu/ jhayes/iscas/c6288.html.

[4] M. Hunger and S. Hellebrand. The impact of manufacturing defects on the fault tolerance of tmr-systems. In *Defect and Fault Tolerance in VLSI Systems (DFT), 2010 IEEE 25th International Symposium on*, pages 101 –108, Oct. 2010.

[5] C. A. L. Lisboa, E. Schuler, and Luigi Carro. Going beyond tmr for protection against multiple faults. In *Integrated Circuits and Systems Design, 18th Symposium on*, pages 80 –85, Sept. 2005.

[6] K. Matsumoto, M. Uehara, and H. Mori. Evaluating the fault tolerance of stateful tmr. In *Network-Based Information Systems (NBiS), 2010 13th International Conference on*, pages 332 –336, Sept. 2010.

[7] K. Pagiamtzis and A. Sheikholeslami. Content-addressable memory (cam) circuits and architectures: a tutorial and survey. *Solid-State Circuits, IEEE Journal of*, 41(3):712 – 727, March 2006.

[8] W. Qiu and D.M.H. Walker. Testing the path delay faults of iscas85 circuit c6288. In *Microprocessor Test and Verification: Common Challenges and Solutions, 2003. Proceedings. 4th International Workshop on*, pages 19 – 24, May 2003.

[9] R.K.K.R. Sandireddy and V.D. Agrawal. Diagnostic and detection fault collapsing for multiple output circuits. In *Design, Automation and Test in Europe, 2005. Proceedings*, pages 1014 – 1019 Vol. 2, March 2005.

[10] Chua-Chin Wang, Chia-Hao Hsu, Chi-Chun Huang, and Jun-Han Wu. A self-disabled sensing technique for content-addressable memories. *Circuits and Systems II: Express Briefs, IEEE Transactions on*, 57(1):31 –35, Jan. 2010.