Stochastic fault simulation of triple-modular redundant asynchronous pipeline circuits

John Daniel Lynch

Presented to the Division of Biomedical Computer Science within

The Department of Science & Engineering

and the Oregon Health & Science University

School of Medicine

in partial fulfillment of

the requirements for the degree of

Doctor of Philosophy

in

Electrical Engineering

5 October 2009

Department of Science & Engineering
School of Medicine
Oregon Health & Science University

CERTIFICATE OF APPROVAL

This is certify that the Ph.D. dissertation of
John Daniel Lynch
has been approved

Dr. Daniel Hammerstrom, Dissertation Advisor,
Associate Dean, Portland State University

Dr. Matthew Heath, Intel Corporation

Dr. Todd Leen, Professor

Mr. John Hunt, Instructor

Acknowledgements

I would like to gratefully acknowledge the contributions of the following individuals for their support of my efforts to complete this dissertation:

Dan Hammerstrom, my advisor and mentor, for inspiration, encouragement, and guidance.

Matt Heath, for thorough review and insightful critique of my early drafts, and for valuable discussions about fault testing and asynchronous circuit design.

Jim Hook and Jan van Santen, consecutive heads of the OHSU Computer Science & Electrical Engineering (now Biomedical Computer Science) department, for enabling me to work on this dissertation while fulfilling my duties as an instructor and program director.

Table of Contents

List of Tables

List of Figures

Abstract

Stochastic Fault Simulation of Triple-Modular Redundant Asynchronous Pipeline Circuits

John Daniel Lynch

Doctor of Philosophy in Electrical Engineering
Division of Biomedical Computer Science within
The Department of Science & Engineering
and the Oregon Health & Science University
School of Medicine
June 2009
Dissertation Advisor: Daniel Hammerstrom

The expected unreliability of nano-scale electronic components has renewed interest in the decades-old field of fault-tolerant logic design. Fault-tolerant design makes it possible to build reliable systems from unreliable components. This has spurred recent research into the application of classical FT techniques to nanoelectronics. Meanwhile, the growing gap between logic gate and wire delays, and the growing power consumption of clock generation and distribution circuits, in nanometer-scale silicon integrated circuits has renewed research in asynchronous, or clockless, logic design. This dissertation examines the application of triple modular redundancy (TMR), one of several FT circuit design techniques, to improve the reliability of a variety of clockless circuits and systems. A new fault model, appropriate for clockless circuits is derived and applied to measure the reliability of nonredundant and triplex micropipelines. A new circuit element that combines the functionality of a Muller C-element and a majority gate is introduced to solve special problems at the simplex-triplex interface. The effectiveness of asynchronous FT circuit design strategies based on the results of Monte Carlo simulation experiments with representative circuits modeled in Verilog hardware description language (HDL) is presented.

1
Introduction

Many fault detection and masking techniques have been developed to improve the reliability of digital electronic systems. Techniques for using information coding and redundant circuitry were developed during the early years of computer design, 1945-1975, when systems were constructed of highly unreliable components (the vacuum tubes used in computers of the 1950s had failure rates of 10^{-4} to 10^{-5} failures/hour, and the integrated circuits used in computers of the 1960s and 1970s had failure rates of 10^{-7} to 10^{-8} failures/hour [1]). A variety of error-detecting and correcting codes addressed faults on signals within combinational logic networks [2], while simple redundancy was applied to timing control and synchronization signals, such as global clocks [3].

Better integrated circuit (IC) design, manufacturing and testing improved component reliability dramatically during the Moore's Law era, 1975-present. Modern IC manufacturing practices assure that nearly all devices shipped to users are free from detectable functional defects [4]. The improved reliability of logic components diminished the necessity of, and hence interest in, the detection and masking of faults within operating logic systems. In industry today fault detection and masking techniques are rarely applied to logic circuits, but are mainly used in communication, memory circuits such as static and dynamic random-access memory (SRAM and DRAM), and magnetic storage media. Examples include Redundant Array of Inexpensive Disk (RAID) storage systems [5] and error-correcting codes used for communication in computer networks.

Manufacturing yield of future nanometer-scale devices is expected to be significantly lower than today's ICs. Also, statistical fluctuations in process parameters make such circuits far less predictable. Furthermore, nanometer-scale devices are expected to be less reliable in operation. As transistors become smaller, and wiring denser, it seems inevitable that circuits will become more susceptible to permanent faults resulting from manufacturing process variations, and to transient faults resulting from noise, cross-talk, and radiation. These trends have recently motivated renewed interest in research into fault-tolerant design techniques.

Although defect rates in of future nanometer-scale technologies are unknown, it is a widely accepted assumption in the research community that they will be significantly higher than with traditional CMOS, even when maximally scaled. For example, the International Technology Roadmap for Semiconductors 2007 Edition [6], referring to the reliability implications of future semiconductor process innovations, states "Understanding and modeling the reliability issues for all these innovations so that their reliability can be ensured in a timely manner is expected to be particularly difficult." Other researchers, lacking quantitative data on defect rates in future technologies, address their expected reliability with qualitative statements such as, "Quantum effects, increased sensitivity to noise, and decreased fabrication tolerances inherent in nanoelectronics will all contribute to reliability losses" [7], and "…the distributed crossbar memory may control the configuration of the system, in particular re-routing it around defective devices which are unavoidable in nanotechnology" [8]. The lack of firm predictions of future defect rates makes it difficult to say when the fault-tolerant design methods that will be discussed in this dissertation will help make systems more reliable.

Shrinking device geometries has other consequences besides reduced manufacturability and reliability. One of the most significant is the growing gap between logic gate delay and wire delay. While gate delays are shrinking with feature size, on-chip wire delay is increasing because wire resistance increases as the wire cross-sectional area shrinks [7]. This growing gap challenges the dominant synchronous design paradigm in which a global periodic clock signal broadcast throughout a digital system provides a common time reference and synchronizes all data transfer within the system. In current very-large scale integrated (VLSI) chips with gigahertz clock frequencies, it can take tens of clock cycles for a signal to traverse the width of a typical chip [7]. And the area and power consumption of the circuitry that generates and distributes the clock signal is increasing faster than the number of transistors on a chip. Thus this circuitry is consuming an ever-larger amounts of power, up to 30% of a modern chip’s power [9].

These difficulties in the scaling of synchronous clocking as Moore’s Laws progresses have motivated renewed interest in asynchronous, or self-timed, digital logic design. In asynchronous systems there is no global clock and data transfer is coordinated locally. Asynchronous computing was originally proposed in the 1950s, but it never became widely used because the common timing model offered by global clocking significantly simplified the design process. During the past 15 years, automated design tools based on synchronous clocking, such as static timing analysis (STA), the logic synthesis of hardware description language (HDL) designs, and structured test methods, contributed to the dominance of the synchronous design paradigm. However, asynchronous logic design is now enjoying a revival. Recent asynchronous designs have shown reduced power consumption [10], higher typical operating speeds [11], greater tolerance of variations in supply Voltage, temperature, and manufacturing variations [12], and even improved data security [13].

This dissertation addresses the convergence of these two trends: the decreasing reliability of nanometer-scale digital electronic components, and the increasing benefits of asynchronous, or clockless, circuits and systems. The question addressed here then is, can classical fault-tolerant digital circuit design methods be successfully applied to clockless systems?

Having stated its motivation – the technology trends that make this problem relevant and interesting - this dissertation is organized accordingly: The remainder of this chapter reviews prior work in fault modeling and fault-tolerance through hardware redundancy, as well as prior work in clockless circuit design, distinguishing the clocked and clockless digital system timing paradigms. Chapter 2 discusses issues that make majority voting in clockless systems more difficult than in clocked systems. Chapter 3 generalizes prior work in fault modeling for clockless systems into a new fault model appropriate for clockless systems that use redundant hardware and majority voting. Chapter 4 describes a methodology for designing fault-tolerant, clockless, pipeline circuits using redundancy and majority voting. Several pipeline circuit examples are presented, as well as a new *CMAJ* circuit element useful in optimizing such circuits. Chapter 5 describes a stochastic fault simulation method that measures circuit fault tolerance using random fault injection in Verilog models of redundant and non-redundant clockless circuits. MATLAB is used to analyze and visualize fault simulation results. Finally, Chapter 6 summarizes the results and draws conclusions.

1.1 Fault Modeling

Microelectronic systems suffer a variety of defects that occur either during manufacture or as devices age. In VLSI chips typical defects include surface impurities, oxide breakdown, electromigration, seal leaks, etc. [4]. A fault is an abstract representation of the manifestation of a physical defect at the functional level. An error is an incorrect signal value resulting from a defect. One such fault model is the *stuck-at model*, which is successfully used as a logic-level abstraction for many kinds of physical defects. In this model a fault is assumed to permanently constrain a logic signal, or circuit node, to a value of 0 (stuck-at-0) or 1 (stuck-at-1) [3]. Other fault models include *bridging faults*, in which two circuit nodes are shorted together; *open faults*, in which a circuit connection is broken; von Neumann faults, in which the defect causes the logic value to be inverted (1→0, 0→1) and *delay faults*, which cause signal transitions on wires to be delayed, that is, arrive later, than those in a fault free circuit.

In addition to the static fault models described above, dynamic fault models provide an abstract functional representation of circuit misbehavior caused by transient phenomena such as crosstalk, noise, and ionizing radiation. Dynamic fault models include Single-Event Upset (SEU), in which a transient event causes the logic value stored in a latch or memory cell to permanently change its state from 1 to 0 or from 0 to 1, and Single-Event Transient (SET), in which the logic value of a circuit node changes for a short time from 0 to 1 to 0, or from 1 to 0 to 1.

1.2 Error-Detecting and Correcting Codes

Data in digital systems frequently travel in groups of related bits, known as bytes or words. A fault can cause one or more of the bits within a group to have the wrong value. Methods for coding data so that bit errors can be detected or corrected have been developed. These methods rely on adding redundant bits to the word and using them to distinguish correct and erroneous codes. Error-correcting codes (ECC) correct data words by performing Boolean operations that map incorrect code words into correct data words. A simplistic error correcting code replicates each bit three times so that when any one copy of the bit is faulty the correct value can be reproduced by majority vote. Begun by Hamming in 1950 [2], decades of research have improved the efficiency of error-correcting codes, making them correct more possible errors, and certain kinds of sequences of errors such as bursts, using fewer redundant bits [14].

Another branch of ECC research concerns the optimal correction of application-specific errors. For example, in 2005 Kuekes, Williams, et al., proposed applying error-correcting codes to mask faults in hybrid CMOS-nanoelectronic devices on signals that cross the boundary from the CMOS to nano domains [15]. This method addresses only faults in signaling but not faults in computing circuits.

Error-correcting codes implicitly assume that all of the bits in the code word are available at some point in time. Correctness can then only be determined if it is known when to evaluate the data, that is when all the bits in a code word – be they right or wrong – are present or valid. The necessity of knowing when to evaluate the code word requires another signal to transmit the temporal information. In synchronous systems this is the global clock; in asynchronous systems it is a locally generated timing signal. In either, a fault on this timing signal can cause the system to fail. ECC can successfully correct errors in data signals, but ignores errors in timing signals.

1.3 Redundant Hardware

ECC guards against faults on the lines that transmit data from place to place, but not against faults in the logic circuits that encode, decode and process data. Redundant logic circuits have been developed to solve this problem. These circuits use coded redundant inputs and outputs, and can deliver correct output values despite faulty circuitry. Examples of redundant logic circuits include N-tuple modular redundant (NMR) logic, which is often applied at the module or functional unit level [16], and interwoven redundant logic, or NAND multiplexing, which is applied at the gate level [17]. The different scale of the redundant units employed in different fault-masking techniques suggests a range of granularity of redundancy. The redundant disk drives units used in RAID are an example of "course-grain" redundancy and the redundant logic gates used in interwoven redundant logic, of "fine-grain" redundancy.

1.3.1 Quadded Logic

Quadded logic is a redundant logic structure that performs computation and can correct errors resulting from both faulty wires and faulty gates. This is accomplished by mixing faulty signals with good ones, while exploiting the logical asymmetries in AND and OR gates.

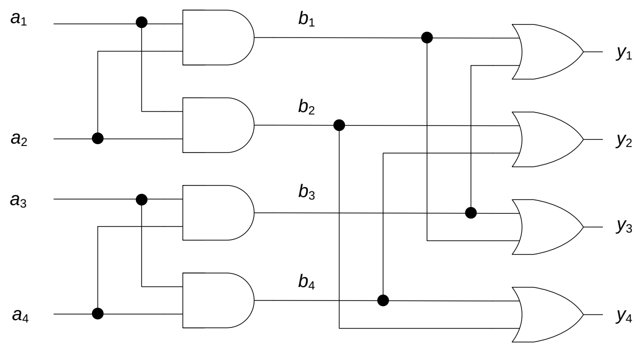


Figure 1. Quadded Logic

Figure 1 shows the basic quadded logic configuration. The circuit receives four redundant input signals a_i, and produces four redundant output signals y_i. If any one bit of the inputs a_i is stuck at 1, then for a 0 input, the AND gates in the first layer correct the error and the value at any b_i is correct. If any one bit of the inputs a_i is stuck at 0, then for a 1 input, two of the b_i bits will be wrong: 0 when they should be 1. However, since the interconnect permutation of b differs from that of a, the two-bit error in b is corrected by the OR gates, producing the correct output at y. This circuit performs no logical function ($y = a$), but it can be shown that quadded logic can be used to compose fault-tolerant combinational logic networks that perform any Boolean function [18].

1.3.2 Interwoven Redundant Logic

Quadded logic is a particular instance of a more general redundant logic configuration known as interwoven redundant logic, or NAND multiplexing. Originally proposed by von Neumann more than 50 years ago [17], this technique corrects errors using the logical asymmetry of NAND gates. Unlike quadded logic in which the permutations of the redundant interconnect between stages are explicitly defined, interwoven logic can use randomly permuted interconnect. Interwoven logic is most effective for high degrees of redundancy, that is, where the number of redundant logic circuits is much greater than the four used in quadded logic.

Figure 2 shows the basic NAND multiplexing configuration. The U boxes represent random permutations of the input signals. Computation is performed in the "executive unit" and the "restorative unit' corrects errors resulting from faults in the executive unit or the interconnect.

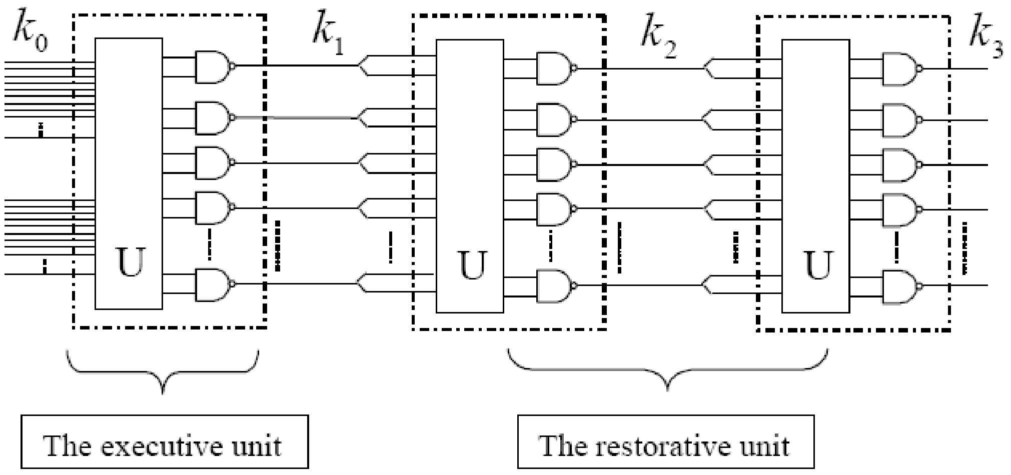


Figure 2. NAND Multiplexing

Recent research on NAND multiplexing focuses on its application to nanoelectronic circuits. Sadek, Nikolić, and Forshaw apply a "parallel restitution" method derived from NAND multiplexing to increasing the noise tolerance of combinational nanoelectronic circuits [19]. Noise in this work can be defined as errors resulting from dynamic faults such as SETs and SEUs. Han and Jonker apply NAND multiplexing to single-electron tunneling circuits and show that, for high degrees of redundancy, it can increase the reliability of such circuits [20]. Roy and Bieu study a variation of NAND multiplexing, called MAJ-3 multiplexing, in which 3-input majority gates replace the 2-input NAND gates in the restoration unit, where they show the superiority of MAJ-3 multiplexing to NMR and NAND multiplexing for small redundancy factors ($R < 10$) [21]; this paper also includes a survey of prior work in fault-tolerant combinational logic circuit design. Bhaduri and Shukla have developed probabilistic model checking tools based on Markovian techniques to evaluate redundancy/reliability trade-offs in NAND multiplexing and NMR combinational circuits [22], [23] and have used them to explore circuit reliability through dimensions of redundancy, granularity, and fault probability for both static and transient faults [24], [25].

1.3.3 Reconfiguration

The Teramac computer built by Heath, Kuekes, Snider and Williams at Hewlett-Packard [26] uses testing and reconfiguration to work around system faults. This approach works well for static faults, but not for dynamic faults. It does not provide continuous error correction so a single transient fault can cause a system failure. Furthermore, faults in certain portions of the system cannot be corrected through configuration, notably, “the wires that are used for clocks”. Snider, Kuekes and Williams later speculate on applying one-time configuration to tolerate defects in future nanoscale CMOS crossbar circuits [27]. Their method is based on two assumptions, first, that defects can be identified and located prior to configuration, and second, that the nanoscale crossbar junctions can be configured. No means of configuration is proposed.

Nicolić, Sadek and Forshaw review N-modular redundancy, cascaded TMR, and NAND multiplexing and compare these classical techniques to the Teramac’s reconfiguration method. They show that reconfiguration is significantly more efficient than the classical alternatives for static faults [28].

Another form of reconfiguration is fault isolation, which is used in multicore processors. Here, failing redundant processor cores can be isolated and the workload redistributed among the nonfaulty cores [29, 30].

1.3.4 N-Tuple Modular Redundancy

In NMR, N identical functional units operate in parallel, and their outputs are compared. The correct output is determined by majority vote (N must be an odd number). In the simplest case of NMR, where *N = 3*, triple modular redundancy (TMR), a two-out-of-three vote produces the correct output if one unit is faulty. TMR can only correct for single errors, but the technique can be generalized to *N*-modular redundancy (NMR) which can correct *N* errors using *2m + 1* copies of the circuit [17]. In the general case, if *N = 2m + 1* then up to *m* faults can be corrected. Of course, there can be faults in the voting logic, so *N* redundant voters are required. Figure 3 shows a basic two-stage TMR circuit configuration. The input and output are redundant signal triplets. Redundant functional modules perform computation while the voters correct errors caused by faults in the modules.

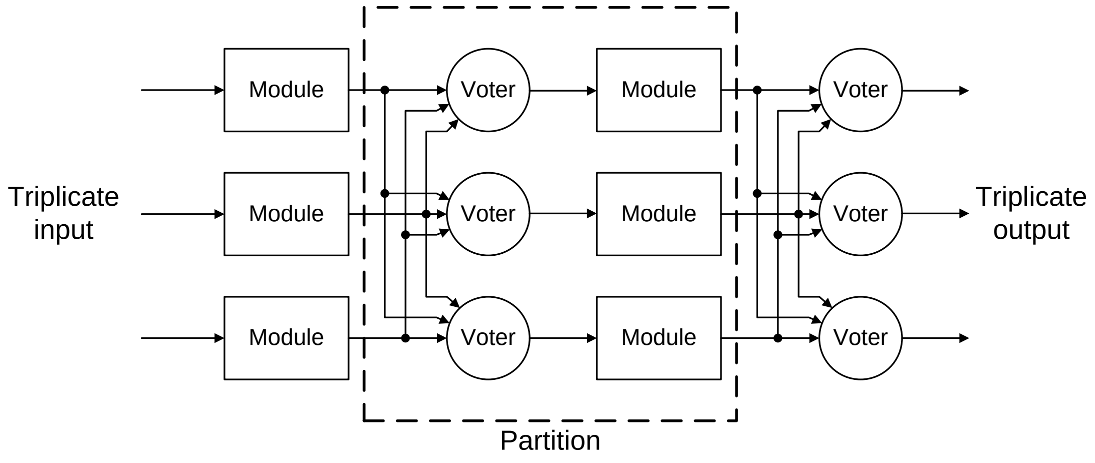


Figure 3. Basic triple modular redundancy configuration

Wakerly [16] calculates the theoretical reliability of a TMR system using the partition shown in Figure 3. The partition’s output is correct – that is, at least two of its three redundant output signals are correct - if the all three voters-module pairs are working correctly, or if any two of the voters-module pairs are working correctly. The expression for the reliability of the partition is

$$r_{part} = (r_m r_v)^3 + 3(r_m r_v)^2 (1 - r_m r_v) \qquad (1)$$

where r_m = reliability of each redundant module, and r_v = reliability of each voter.

The voter can be implemented using a combinational majority circuit, as shown in Figure 4, for each bit. This circuit's output is '1' iff two or more of its inputs are '1'.

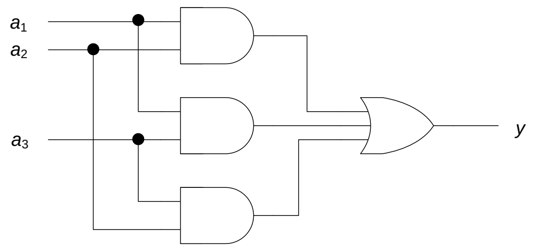


Figure 4. Combinational majority voter circuit

Han has introduced a variation of TMR that he calls "Triplicated Interwoven Redundancy" (TIR) [31], [32], [33]. TIR is important because it shows that the three redundant wires that transmit each signal in the circuit can often be randomly permuted, or interwoven, without reducing circuit reliability. This observation is interesting for future nanocircuits built using stochastic assembly methods. Han observes that in cyclic circuits, that is, sequential circuits with feedback, some interweave patterns are less effective than others [31]. This interweave pattern dependent behavior does not appear in acyclic (combinational) circuits. Han dismisses this interesting result, saying "…this is due to a fanout effect of erroneous signals and a malicious interconnect combination…" However, such "malicious interconnect combinations" exist in all cyclic sequential circuits, clocked or otherwise.

In many of the clockless pipeline TMR circuits that are the subject of this dissertation, the modules and the voters are approximately the same size, that is, each is implemented with approximately the same number of transistors. Therefore, it is reasonable to assume that module and voter have comparable reliabilities.

Let $r_m = r_v$, then Equation 1 becomes

$$\begin{aligned} r_{part} &= \left(r_m r_m\right)^3 + 3\left(r_m r_m\right)^2\left(1 - r_m r_m\right) \\ &= r_m^{\ 6} + 3r_m^{\ 4}\left(1 - r_m^{\ 2}\right) \\ &= r_m^{\ 6} + 3r_m^{\ 4} - 3r_m^{\ 6} \\ &= 3r_m^{\ 4} - 2r_m^{\ 6} \end{aligned} \qquad (2)$$

Equation 2 can be used to compare the reliability of TMR and nonredundant systems for different values of r_m. This provides a baseline indication of whether TMR systems are indeed more reliable than their nonredundant counterparts (for systems in which module and voter are comparably reliable).

The nonredundant counterpart to the partition from which Equation 2 was derived consists of a single module (with no voter), so the reliability of the partition is simply the reliability of the module:

$$r_{part}(nonredundant) = r_m$$

The calculated value of r_{sys} for both TMR and nonredundant partitions as a function of r_m can then be plotted. The result is shown in Figure 5.

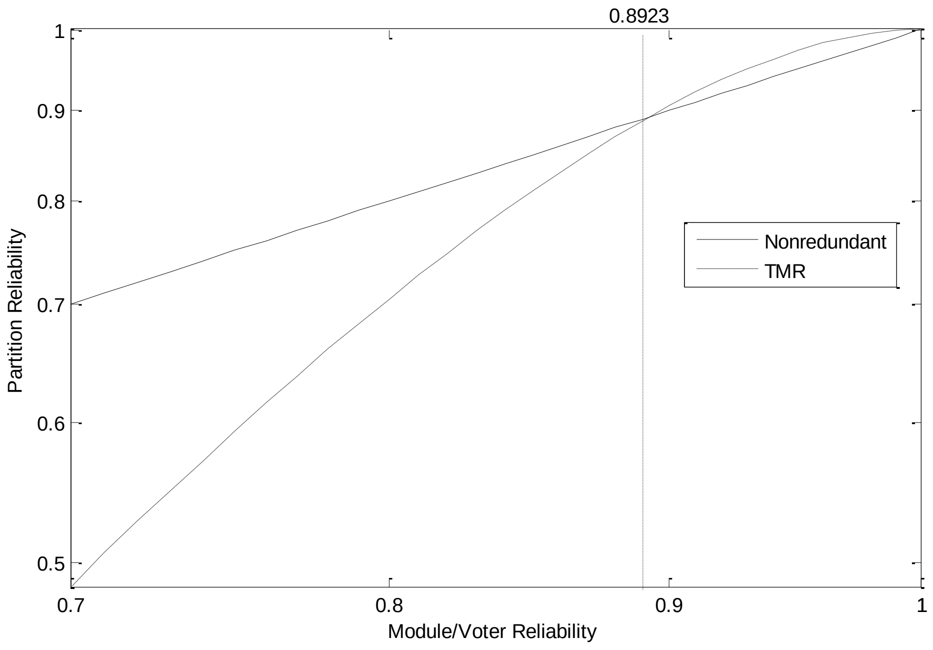


Figure 5. TMR and nonredundant partition reliability for $r_m = r_v$

The intersection of the two curves can be found algebraically by setting

$$r_{part}(nonredundant) = r_{part}(TMR)$$

Then by substitution

$$r_m = 3r_m^{\ 4} - 2r_m^{\ 6} \tag{3}$$

After some algebra, the equation is

$$2r_m^{\ 5} - 3r_m^{\ 3} + 1 = 0 \tag{4}$$

The root of which is 0.8923. Thus the TMR reliability is greater than for the nonredundant partition for r_m greater than 0.8923.

1.3.5 Synchronization of redundant hardware

Just as ECC needs to know when all bits are available in order to perform error correction, NMR voters need to know when all N functional unit results are available in order to vote. This synchronization problem is commonly solved by using a common clock. However, in this case the single clock then becomes an unprotected single point of failure. Davies and Wakerly in 1978 addressed the synchronization of voting in NMR systems and proposed a NMR crystal-controlled clock [34].

Davies and Wakerly also proposed a simple scheme in which, if the number of redundant functional units is $2f + 1$

> "A majority circuit determines when $f + 1$ of the … inputs have been received, then delays for a length of time *d*, and sends [the voted output]." [34]

The value of *d* must be greater than the maximum variation in input arrival times plus the maximum variation in the value of *d* in all participating voters. The problem is how to determine the value of *d*.

Verdel and Makris in 2002 proposed an asynchronous fault-detection (not correction) circuit based on duplicated logic that also relies on similar delayed comparison, [35] but again the problem of determining the delay value is unsolved.

Fischer, et al., proved in 1985 that it is impossible to vote, or “reach consensus”, among completely asynchronous processes [36] - some maximum delay bound must be assumed.

1.4 Asynchronous Circuits

Asynchronous circuits are digital circuits that do not use a periodic clock signal to synchronize data transfer. The field of asynchronous circuit design began in the mid-20th century. Muller and Bartky in the 1950s were among the first to distinguish asynchronous from synchronous circuits, and laid down a theoretical foundation for analysis of asynchronous circuits [37]. Muller also introduced the *Muller C-Element,* an important asynchronous circuit primitive. The Illiac computers of the 1950s and ‘60s at the University of Illinois included asynchronous circuits based on Muller and Bartky’s work [38]. Interest in asynchronous circuits continued as the VLSI era arrived in the 1970s. Seitz applied asynchronous circuit principles to “self-timed” VLSI systems in [39]. Chaney’s pioneering work on synchronization failure in clocked digital circuits produced a quantitative theory of metastability by 1983 [40, 41]. Sutherland’s work on asynchronous “micropipelines” and composable asynchronous circuit primitives revolutionized asynchronous circuit design methodology and helped him receive the Association for Computing Machinery’s Turing Award in 1988 [42]. Chapiro in 1984 showed that clocked logic could be integrated into asynchronous systems through the use of pausable clocks – a method he calls “globally asynchronous locally synchronous” or GALS [43]. Recent textbooks on asynchronous circuit and system design include Sparsø & Furber [44] and Myers [45]. Martin and Nyström in 2006 published a good survey of the current state-of-the art in asynchronous circuit design, including a “brief history” of five decades of asynchronous design [46].

1.4.1 Fault Tolerance in Asynchronous Circuits

Most methods for designing redundant fault-tolerant combinational circuits are either completely static with no notion of time, or assume that all of the redundant signals are present at a known time, such as a clock edge. Only a few researchers have tackled the problem of fault tolerance in a system without a reliable global clock, where there is no signal to indicate when to compare redundant signals. In NMR systems where a majority of redundant signals determines the corrected signal value, the problem can be best expressed as “when to vote”.

1.4.2 Fault Modeling in Asynchronous Circuits

Lu and Tong in 1995 suggest that the traditional “stuck-at” fault model is inadequate for asynchronous circuits and propose adding “transition unable” and “extra transition” to the fault model menagerie [47]. Shirvani, et al., go further, introducing a fault set that they call “DUDES” consisting of “down-enabled”, “up-enabled”, “down-disabled”, and “up-disabled” faults [48]. Down- and up-disabled faults model defects that prevent a circuit from driving its output to logical ‘0’ or ‘1’, respectively. Down-enabled faults model defects that cause a circuit to change its output from ‘1’ to ‘0’ prematurely. Up-enabled faults model defects that enable a circuit to change its output from ‘0’ to ‘1’ prematurely. The DUDES model covers all stuck-at faults on the internal nodes of basic asynchronous circuit elements (C-element, merge, toggle, inverse toggle). This paper also includes a useful summary of work on testing asynchronous circuits published prior to 2000.

The down-enabled, up-enabled, down-disabled, and up-disabled faults in the DUDES model all cause faulty circuit elements to emit early transitions. These early-transition faults are not covered by the stuck-at model. They represent a different class of faults that can be added to the stuck-at model to create a more complete and accurate fault model for asynchronous circuits. The concept of early-transition faults will be generalized in Chapter 3 to create a new fault model used for the stochastic fault simulation described in Chapter 6.

1.4.3 Self-Checking Dual-Rail Signaling

“Self-checking” is different than “fault-tolerant”. A fault-tolerant system delivers correct output for some but not all fault conditions. For example, a fault-tolerant system delivers correct output if it has any single fault, but delivers incorrect output if it has two or more faults. A Self-checking system never delivers an incorrect output, rather it fails either silent (no output), causing system deadlock, or gives a definitive error indication.

Deadlock can be detected using a timer. With a timer there is the problem of determining the time-out value – how long to wait. A self-checking module sending its output using dual-rail signaling either fails silently by maintaining a 00 output, or it indicates an error using the unused code 11. It must never give an incorrect 01 or 10 output.

In an abstract sense, a "what + when" signal indicates a binary digit and null state in which neither binary value is present. The binary digit values are represented with the Boolean values 0 and 1. The null state is variously represented in the literature as "NULL" [49], "NIL" [50], "empty" [44], "clear" [13], "U" [51], and ⊥ [52], the "lowest element" symbol used in set theory. In this dissertation, the symbol ⊥ will be used to represent the null signal state. Thus, a "what + when" signal w is ternary with $w \in (0,1,\perp)$. Possible faults on w then include stuck-at-0, stuck-at-1, and stuck-at-⊥; and all classes of transition faults must include transitions to and from ⊥. Such a signal can be implemented as "dual-rail", that is with 2 binary wires, w_1 and w_0. Several researchers have proposed using the otherwise unused 11 state to indicate that an error has occurred. They have named the 11 state "alarm" [13], "error" [53], or left it unnamed [35, 51, 54]. Continuing to use set theory notation for a partially-ordered set, the dual-rail signal encoding table can be completed as shown in Table 1.

Table 1. Dual-Rail Binary Encoding with Error Symbol

w_1w_0	value
00	⊥
01	0
10	1
11	⊥

1.4.4 SEU and SET Tolerance in Asynchronous Circuits

As logic devices shrink and ever smaller amounts of electric charge are used to store state information, they become more vulnerable to transient, or "soft", errors such as SEU and SET, caused by naturally-occurring background radiation. Designers of memory devices such as SRAM and DRAM have been aware of these soft errors for many years, but at the nanoscale, logic gates and flip-flops have recently become just as vulnerable to soft errors [55].

A method of tolerating single-event upsets using both temporal and spatial redundancy combines the outputs of duplicated gates or storage elements in such a way that if the value of only one changes, then the previous combined output value is maintained until both inputs gain change together [56]. Gong calls this “Dual Modular Redundancy” or DMR. Gong also proposes another method called ‘Temporal Spatial Triple Modular Redundancy” or TSTMR. In this method skewed triplicate clock signals clock triplicated D flip-flops (FFs) at slightly different times. The idea is that if an SET or SEU strikes at the vulnerable sampling interval of one FF it will not affect the other two.

Jang and Martin propose a similar duplication-based, SEU-tolerant method for clockless circuits [57]. Other works also propose employing C-elements to compare the outputs of duplicated latches for SEU-tolerant *clocked* circuits [58-60]

Monnet, Renaudin and Leveugle have analyzed the SET sensitivity of the Muller C-Element, a key clockless system component [61, 62], and propose several duplication-based techniques for hardening against SET-induced errors in both clockless [63] and clocked circuits [64].

Almukhaizim and Makris address the *when to check* problem, that is, checking synchronization, for error detection for asynchronous systems, applying a transition-detecting synchronization method to both duplication and EDC-based schemes [65, 66]. They point out that their duplication-based method is as effective in masking transient faults as TMR, and has a lower cost. This duplication based-method, however, does not mask permanent stuck-at faults.

1.5 Review of Digital System Timing

Digital systems can be broadly divided by their timing properties into two families: *Synchronous systems* in which timing is controlled a global clock signal, and *asynchronous systems* in which timing is controlled locally throughout the system. The distinction can be thought of as the degree of timing granularity. In synchronous systems the number of clock signals is small, typically 1-10, and each clock signal controls the timing of large blocks (100-1,000,000 gates) of logic, thus timing control is “course grained”. In asynchronous systems there are many (100-1,000,000) timing control signals and each timing signal controls a relatively small number of gates (1-100 gates), thus timing control is “fine grained.”

1.5.1 Synchronous signaling

The vast majority of digital systems designed today use synchronous signaling because using a common clock simplifies the design process and is well-supported by industrial design automation software. In synchronous systems, there are two types of signals: data and clock, as shown in Figure 6. The dashed arrows in the figure indicate how the data must be set up some time before the rising edge of the clock signal, and held for some time after rising edge. The clock is a regular periodic signal that is broadcast throughout the system to synchronize data transfer between storage elements (state changes). It is generally free-running with a constant period, usually generated externally by a crystal oscillator or phase-locked loop. It carries no data per se, but it does carry information of when the data signals can be evaluated (“when” data). The combinational methods based on Boolean algebra assume signals have values of only 0 or 1. In clocked sequential systems the arrival of a clock signal transition establishes when data signals are ready, that is, they have settled to [0, 1].

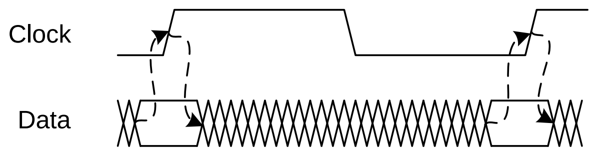


Figure 6. Synchronous signal timing

Designers of synchronous logic use automated methods such at static timing analysis to constrain circuit delays so that the data signals are always stable at the time of the clock transition.

1.5.2 Asynchronous Signaling

Asynchronous systems can be divided into two general classes: Those that use *bundled* signaling and those that use *dual-rail* or *1-of-N* signaling (1-of-n can be generalized into m-of-n [67]).

In bundled systems, data signals use Boolean ‘0’ and ‘1’ levels to encode information, and the time at which the data signals are valid is indicated by locally-generated Request-Acknowledge handshake signals [44]. The timing relationship of data signals to Request-Acknowledge is similar to that of data to clock in synchronous systems, as shown in Figure 7. The dashed arrows in the figure indicate the sequence of event on the Request, Acknowledge, and data signals. The sender must assure that all data signals must be valid before the Request signal rises. The rising edge of Request signals the receiver that data are valid. Then the rising edge of Acknowledge indicates that the receiver has read the data, and the sender need not keep it available. Request then returns to zero, followed by Acknowledge to complete the transaction.

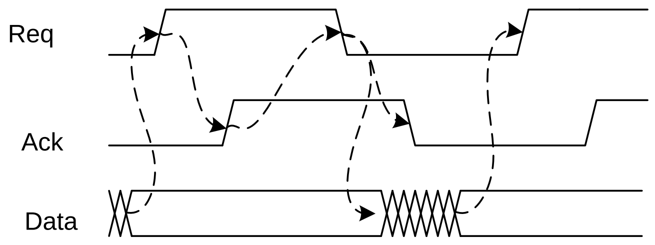


Figure 7. 4-Phase Bundled Data Timing

In *dual-rail* or *1-of-N* signaling clockless signaling, data and timing information are encoded in two, or N signals, using a ternary [0, 1, null] alphabet [49]. This can be encoded onto a single wire a pair of wires using codes 10, 00, 01 respectively. In dual-rail or 1-of-N signaling, there is no distinction between timing and data signals; all signals are coded such that each indicates both what the value of the bit being sent is, and when it is valid. In dual-rail signaling two wires (rails) code each bit: one of the two is asserted to represent logical 1, the other is asserted to represent logical 0, and if no data are present than neither is asserted, as shown in Figure 8. The figure shows the transfer of a logical 1 followed by a logical 0. The dashed arrows in the figure indicate the sequence of events.

Only one wire may be asserted at a time. Both being asserted simultaneously is an error condition.

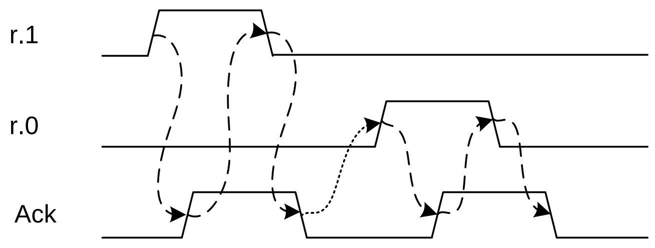


Figure 8. Dual-Rail Signal Timing

1.6 Two-Phase Bundled Data (Micropipeline)

Sutherland's micropipeline is a classic example of a clockless circuit [8]. It is an event-driven, asynchronous, elastic pipeline, which can also incorporate internal processing. The events that drive the micropipeline are rising or falling digital signal transitions. Rising and falling transitions are equivalent and are used to synchronize the transfer of "bundled" data as shown in Figure 9.

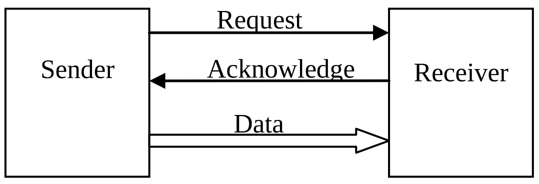


Figure 9. Clockless Bundled Data Interface

The diagram in Figure 10, adapted from [8], shows the timing of data transfer. The arrows in the figure indicate the sequence of events; the solid arrows show the sender's actions; the dashed arrows show the receiver's actions. The sender assures that all data signals are valid before changing the value of request. The transition on the request signal (either rising or falling) indicates that data are valid. The subsequent transition on the acknowledge signal (either rising or falling) indicates that the receiver has taken the data.

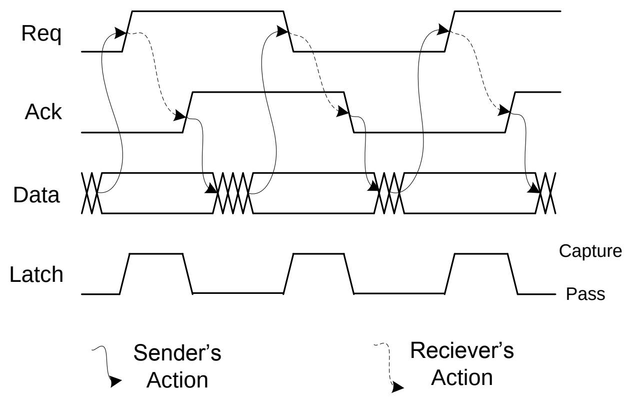


Figure 10. Two-phase bundled signaling used in micropipeline

The control circuit that synchronizes data flow through a micropipeline is built from Muller C-elements connected as shown in Figure 11. The pipeline control circuit is composed of a linear array of interlocked Muller C-elements. The figure shows four-stages. This circuit is also called a *dataless* pipeline, since only control signals are moved down the pipe.

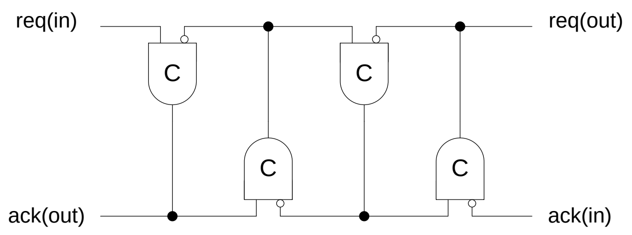


Figure 11. Asynchronous pipeline control circuit or "dataless" pipeline

A key component of the pipeline circuit, which determines when the pipeline latches capture and release data, is the Muller C-element [10]. It is a sequential device whose outputs follow its inputs when they are equal and holds its state when they are not. Its function can be described as a logical "join" or "last-of".

Table 2 specifies the C-element function, and Figure 12 shows the C-element schematic symbol representation (a) and timing diagram (b). The timing diagram shows output *c* changing only after both inputs *a* and *b* have changed. The dashed arrows indicate which input transition event causes output *c* to change.

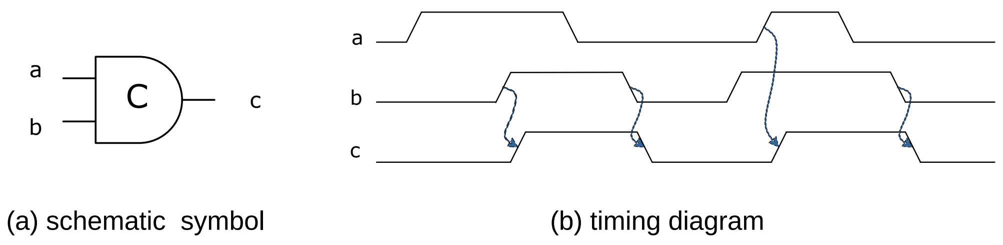


Figure 12. Muller C-element

Table 2. Muller C-Element Truth Table

a	b	c
0	0	0
0	1	no change
1	0	no change
1	1	1

The data in the micropipelines are stored in event-controlled elements called “capture-pass” latches as shown in Figure 13. The capture-pass latch has two control inputs called “capture” and “pass” (“C” and “P” in the figure). When the two control inputs are equal, the latch is transparent, allowing data to pass from input to output. When the control inputs differ, the latch holds its stored value. Thus the operation of the latch is completely symmetrical with regard to the ‘0’ and ‘1’ logic levels of its control inputs.

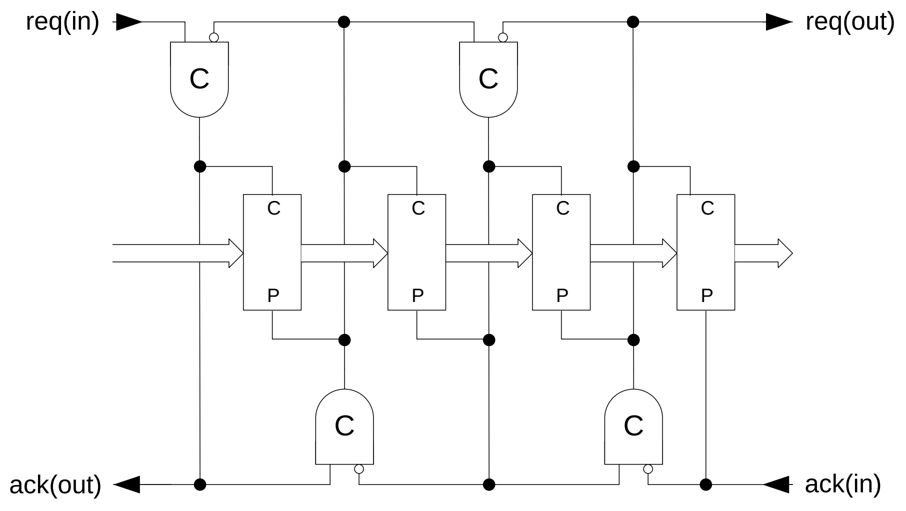


Figure 13. Micropipeline circuit with capture-pass data latches

The micropipeline circuit, shown in Figure 13, implements an elastic first-in-first-out buffer without processing. It can be made into a processor by adding combinational logic in the data path between the capture-pass latches, and adding delays matched to that of the combinational logic to the forward control signal paths as shown in Figure 14.

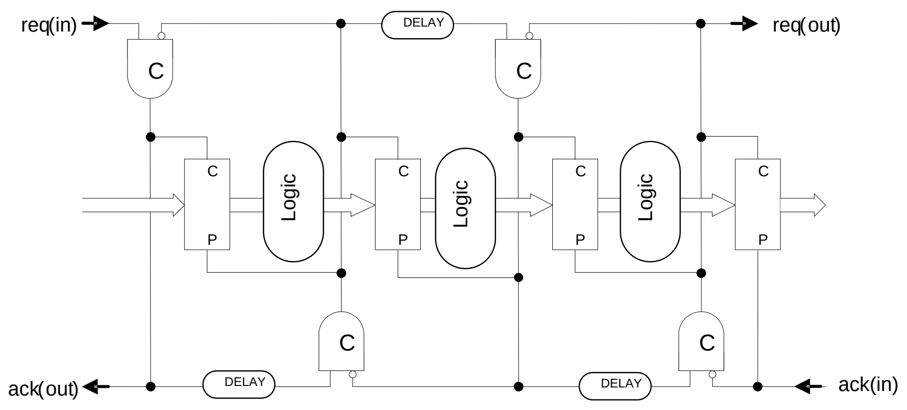


Figure 14. Micropipeline with processing

A useful metric of micropipeline performance is the micropipeline's period, *P* [9]. The period is defined as the delay between the input of one data bundle and the input of the next data bundle. This comprises the forward propagation of the request signal (through the C-element and the interstage delay), $L_{forward}$, and the reverse propagation of the acknowledge signal (through the C-element and the inverter at its right input), $L_{reverse}$. Therefore the minimum period of a micropipeline with processing is:

$$\begin{aligned} P &= L_{forward} + L_{reverse} \\ &= t_C + t_{DELAY} + t_{INV} + t_C \\ &= 2t_C + t_{DELAY} + t_{INV} \end{aligned} \quad (5)$$

Where t_C is the delay through the C-element, $t_{DELAY,}$ is the delay through the delay element, and t_{INV} is the delay through the inverter (indicated by the bubble) at the right input of the C-element. In a micropipeline without processing, t_{DELAY} may be zero, or the minimum value necessary to assure that the data latch's set-up/hold timing constraints are satisfied. In any micropipeline implementation, the values of t_C and t_{INV} will vary from instance to instance. Because all the delay loops within a micropipeline are interlocked, the overall pipeline period will be the maximum value of all of the $2t_C + t_{INV}$ loops in the pipeline.

1.7 Four-Phase Dual-Rail Pipeline

The 4-phase dual-rail pipeline shown in Figure 15 is another example of an event-driven, asynchronous, elastic pipeline. This clockless pipeline circuit encodes data using the dual-rail signaling shown in Figure 8 rather than bundling data. The pipeline circuit in the figure carries just one bit of data; additional bits can be added by adding more dual-rail pipelines in parallel. Unlike the micropipeline, rising and falling transitions are not equivalent. A rising transition indicates the presence of a valid data bit; the bit remains present as long the signal remains high; and a falling transition indicates that the bit is no longer valid.

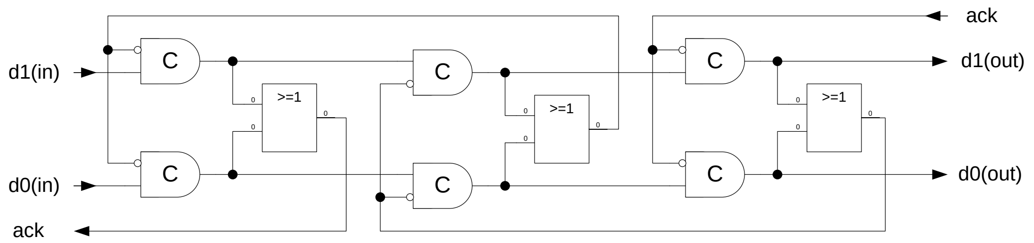


Figure 15. 1-bit, 2-rail, 3-stage, 4-phase pipeline

The period of a 4 phase, 2 rail (4P2R) pipeline can be calculated in a manner similar to that of the micropipeline. In addition to the $2t_C + t_{INV}$ loop delay of the micropipeline, the 4P2R loop includes the delay of the OR gate. Thus,

$$\begin{aligned} P &= L_{forward} + L_{reverse} \\ &= 2t_C + t_{INV} + t_{OR} \end{aligned} \tag{6}$$

1.8 Chapter Summary

This chapter has described the motivation for, and introduced the subject of this dissertation. It has also reviewed prior work in two different fields of digital circuit design: fault-tolerant circuit design using hardware redundancy, and asynchronous or clockless circuit design. Having done this, I now move on to the particular problems encountered at the intersection of these two fields: Majority voting in redundant clockless circuits.

2
Majority Voting In Asynchronous Systems

This chapter discusses the particular problems of majority voting in redundant systems without a global timing reference. It begins by contrasting voting in clocked and clockless systems. It discusses several issues in designing effective voting circuits for clockless systems. It also discusses the effects of single-event transients (SETs) and single-event upsets (SEUs) in clockless systems and reviews prior work on a SET-resistant voting circuit for clockless systems.

2.1 When to Vote

In order to perform error correction by majority voting – i.e., reaching "consensus" or "agreement" of redundant hardware - it is necessary to know when to vote, that is, when the outputs of the redundant modules are ready. In other words, the module outputs must be synchronized.

In systems using a common clock signal, the clock signal edge (rising or falling) indicates completion of each cycle of computation, that is, when the results produced by combinational logic can be evaluated. In clocked redundant systems, voting is accomplished by comparing the redundant values present at the time of the clock event. All values are compared at the same time, that is, all values participate in the vote; so voter turnout is always 100%. A fault can only change the value of a "ballot" - a data input to the voter - from 0 to 1 or vice versa, but not prevent a ballot from being counted.

Figure 16 shows a TMR circuit synchronized by a broadcast clock signal. A problem with this method of synchronization is that the common clock signal becomes a single point of failure: A single fault on any wire transmitting the clock signal can cause an uncorrectable failure. One solution to this problem is to use phase-locked redundant clocks [34].

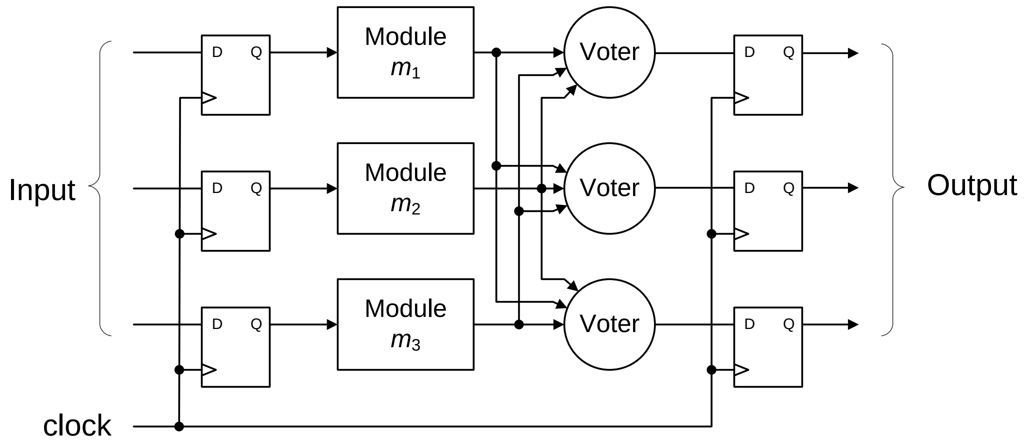


Figure 16. Clocked triple modular redundant logic

In asynchronous, self-timed, or clockless systems, no clock is used. Rather, modules synchronize communication locally using handshake signals. Handshake signaling makes no assumptions about the absolute time when events occur, but its rules provide for the strict ordering of events. Here, ballots can arrive at different times. A fault can change the value of a ballot, change the time when a ballot arrives, or prevent a ballot from arriving at all. Because ballots arrive at different times, voting can be sequential. It is not necessary to wait until all ballots have been received to vote. In fact, because a fault can prevent a ballot from arriving at all, if voting waits for all ballots then a fault can prevent voting from ever occurring and cause system deadlock. Voting begins when the first ballot arrives and ends when a majority is established, even if not all the ballots have arrived. Ballots received after a majority has been decided cannot change the result of voting, and therefore are ignored. In clockless systems using bundled data, a request-acknowledge signal pair indicates when the data signals are valid and can be voted upon.

Figure 17 shows an implementation of voting for a redundant dataless or “bare” handshake channel [46]. The modules on the left send redundant “request” signals through a bank of voters to the modules on the right. Symmetrically, the modules on the right send redundant “acknowledge” signals to the modules on the left. Voters in both the request and acknowledge paths correct errors in both directions. The voting scheme shown for the bare channel can be generalized for channels using both dual-rail and bundled signaling. For dual-rail signalling, additional voters are added for each rail. In bundled signaling, the data wires carry no timing information and can be handled using classic combinational error-correcting techniques.

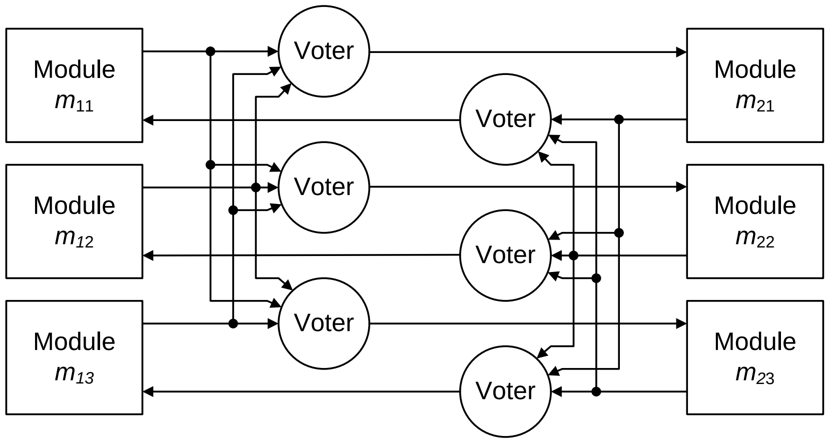


Figure 17. TMR implementation of a dataless handshake channel

The arrival times of the signals at the voter inputs are unknown and unordered. In such a system it is impossible to tell whether a module output is missing or is just arriving very late. So how can it be decided when to vote? In other words, when should the voter send its result?

2.2 Strong vs. Weak Ordering in Clockless Voters

Clockless functional modules can be characterized as either *strongly indicating* or *weakly indicating* [44]. Weakly indicating modules obey Seitz's "weak conditions": they can deliver some outputs after *some*, but not *all* inputs are received [39] . Strongly indicating modules, in contrast, wait for *all* inputs to arrive before delivering *any* outputs. Weak ordering is illustrated in Figure 18 and strong ordering is illustrated in Figure 19.

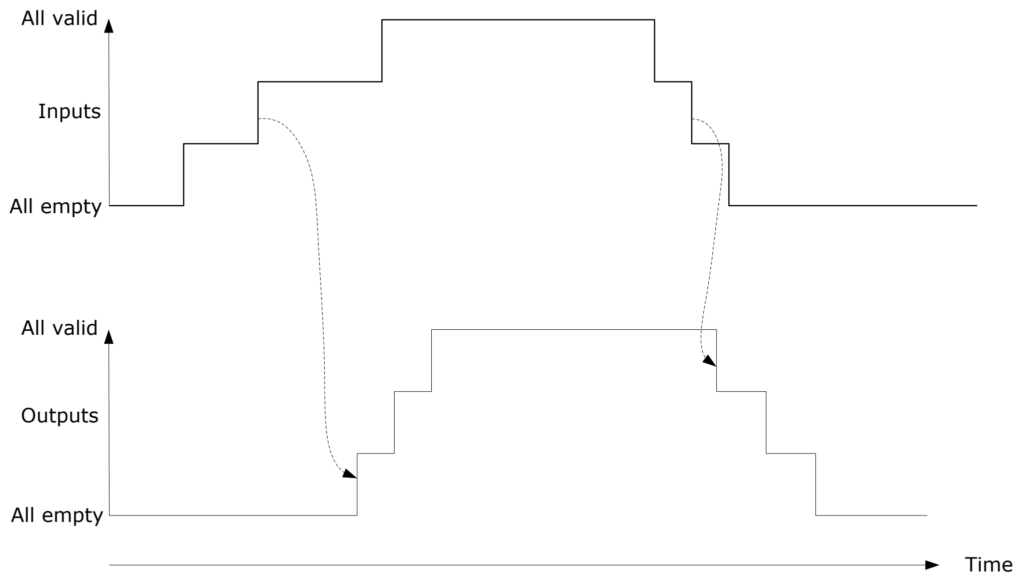


Figure 18. Weak input/output event ordering

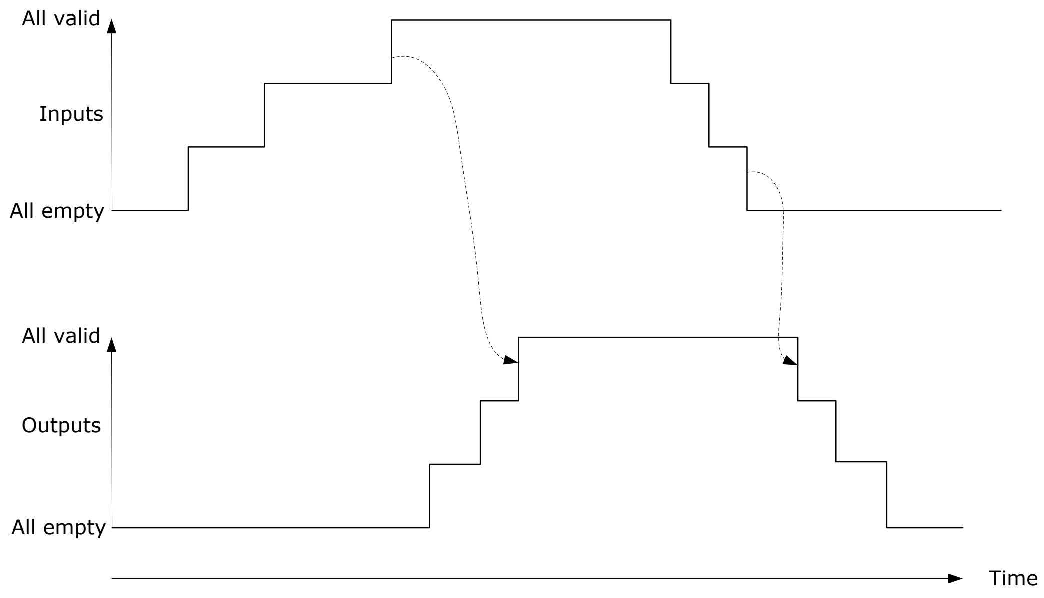


Figure 19. Strong input/output event ordering

Seitz’s weak conditions also require a module to produce all valid outputs until all valid inputs have been received. Seitz uses a full adder as an example. If it receives the carry-generate (11) or carry-kill (00) input conditions then it can produce the carry output without waiting for the carry input. But it cannot produce the last output (sum) until the last input (carry) has been received.

In fault-tolerant, redundant clockless systems, a faulty module may cause a voter input to never arrive. If the voter uses strong ordering then it may wait forever for all inputs to arrive. A missing input that never arrives means that the voter will never issue an output, potentially halting the entire system. Therefore, clockless voters cannot use strong ordering. A clockless voter can issue its output when enough inputs have arrived to determine the majority vote result. So a 3-input voter circuit can issue its output after at least two inputs in agreement have arrived, without waiting for the third input.

Figure 20 presents a decision process for a clockless majority voting circuit. In the figure, i1, i2 and i3 represent the first, second and third input values received by the voter. If i1 = i2 then the majority value is known after receipt of i2, and the value of i3 cannot change it. Thus the voter can issue its output immediately after receiving i2, and ignore i3 when it eventually arrives. If i1 ≠ i2, then the majority value is unknown after receipt of i2, and the voter must wait for i3 to arrive to break the tie. A problem with issuing the majority output value immediately after receiving i2 is in determining whether i3 is missing or just late. In the first case, the vote is complete and the next input to arrive, whether it is i1, i2 or i3, initiates a new round of voting. In the second case the arrival of i3 completes the current round of voting. Without some time boundary, it is impossible to determine whether the arrival of i3 completes the current round of voting or begins a new round [36]. Therefore some kind of timeout mechanism is required in any fault-tolerant asynchronous system.

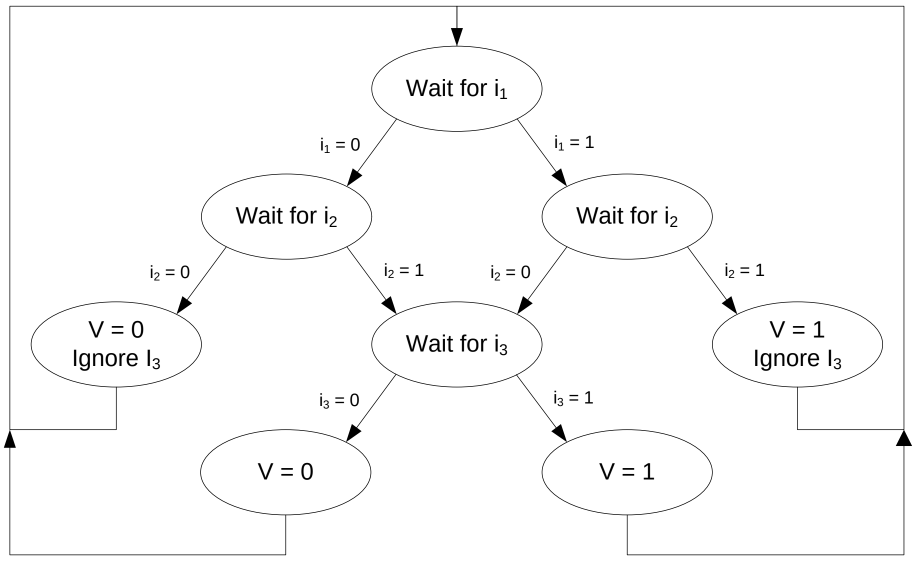


Figure 20. Asynchronous consensus decision flowchart

This section began by arguing that strongly-ordered voting can deadlock waiting for an input that will never arrive. It then showed that weakly-ordered voting will not deadlock, but requires a timeout mechanism to maintain event order. But a timeout mechanism can also be used to detect deadlock in strongly-ordered voting. Thus, with timeout protection, both strongly- and weakly-ordered voting can be used in clockless systems. Which is better?

2.3 Designing a Clockless Voter Circuit

The following discussion considers the circuit design of the “voter” modules shown in Figure 17. Arbitrary Boolean function blocks for dual-rail signaling can be constructed using the *delay-insensitive minterm synthesis* (DIMS) method [44, 68]. The DIMS method resembles the traditional sum-of-products approach to constructing ordinary combinational logic circuits, except that C-Elements, rather than AND gates, generate the minterms. The minterms are combined in OR gates, just as they are in ordinary combinational circuits. The C-Elements implicitly implement the necessary handshaking, making the resulting circuit delay-insensitive.

Building a majority function, or voter, using the DIMS method illustrates some interesting challenges. For the purpose of clarity, the terms *stuck-low* and *stuck-high* will be used in the following discussion of stuck-at faults in circuits that use dual-rail signaling. Consider the 2-of-3 majority functional block shown in Figure 21.

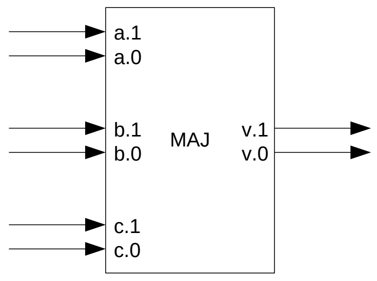


Figure 21. 4-phase dual-rail majority function block

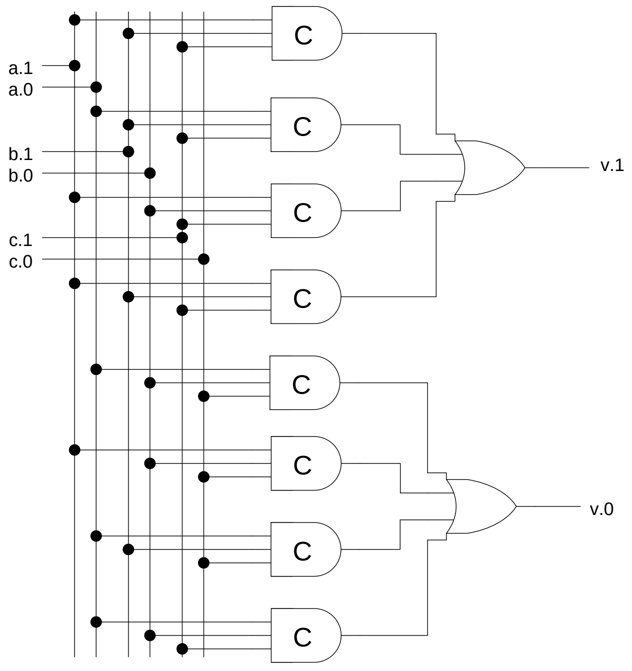


Figure 22. Strongly indicating 3-input majority circuit using 4-phase dual-rail signaling

A DIMS implementation of this function produces the PLA-like circuit shown in Figure 22. The circuit is strongly indicating. That is, it waits until all 3 inputs are valid (∈ 0,1) before producing a valid output. If either rail of any input is stuck low then both output rails will also be stuck low, causing deadlock. Thus the circuit fails to restore any stuck-low faults. A stuck-high fault on either rail of any input propagates to the inputs of the relevant four C-elements. This will prevent the outputs of those C-elements from ever returning to zero, which, in turn, will cause one or both rails of output *v* to become stuck high. Thus the circuit also fails to restore any stuck-high faults.

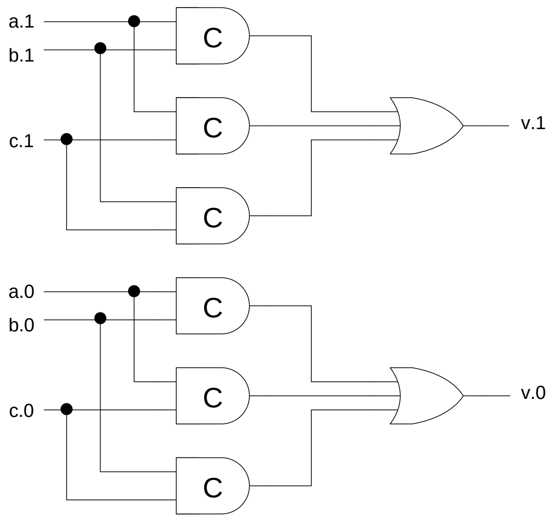


Figure 23. Weakly-indicating 3-input majority circuit using 4-phase dual-rail signaling

A weakly-indicating optimized DIMS implementation of the dual-rail majority function block is shown in Figure 23. If either rail of any input is stuck low then two of the C-element outputs for that rail will be stuck low, causing two subcritical stuck-low faults at the OR gate input. Thus, the circuit can restore stuck-low input faults. However, if either rail of any input is stuck high then two of the C-element outputs for that rail will be stuck high, causing critical stuck-high faults at the OR gate input. Thus the circuit cannot restore stuck-high faults.

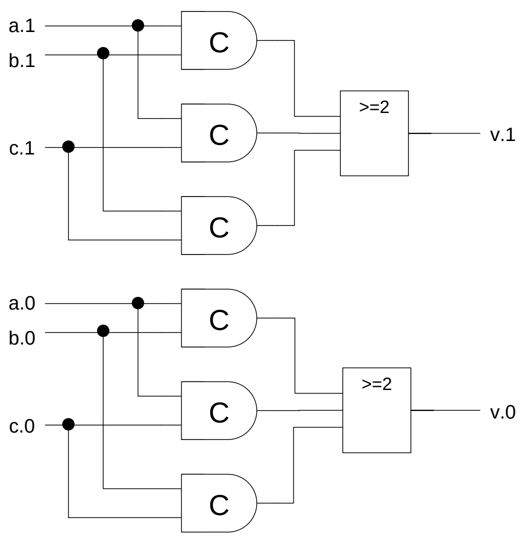


Figure 24. Weakly-indicating majority function, 4-phase dual-rail, majority gate implementation

An alternate 4P2R majority voting circuit is shown in Figure 24. The OR gate in the circuit in Figure 23 is replaced with a 3-input combinational majority (MAJ) gate. A single stuck-low or stuck-high fault on any of the circuit's primary inputs causes a stuck-low or a stuck-high fault, respectively, on two of the C-element outputs. Unlike the OR gate, the MAJ gate has no critical input faults, but it still cannot correct faults on two of its inputs. Thus a stuck-at fault at any one of the circuit's inputs causes the same stuck-at fault at the circuit's output.

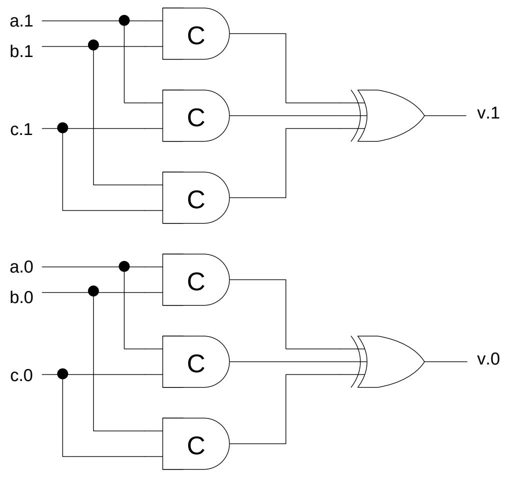


Figure 25. 4-phase dual-rail majority circuit, weakly indicating, XOR implementation

A final variation of a DIMS majority implementation is shown in Figure 25. Here the C-element outputs are merged using an XOR gate. This variant can handle the cases where a single stuck-low or stuck-high fault on any of the circuit's primary inputs causes a stuck-low or stuck-high fault on two of the C-element outputs, which enable the XOR gate to pass the third C-element output reliably. However, it has a problem with non-faulty inputs: Any variation in delay, or skew, between the XOR gate inputs can cause a glitch, or *extra-transition* fault, on the output.

All of the DIMS voter circuits suffer from the fact that a stuck-at fault at any primary input cause faults at a majority of the C-element outputs: 4 of 4 in the strongly indicating circuit and 2 of 3 in the weakly indicating circuits, making it difficult to find a logic function that can successfully merge the C-element outputs. This leads us back to using the combinational majority gate as a voter.

2.4 Asynchronous voting with majority gates

A CMOS 3-input majority logic gate can be implemented using 12 transistors [69]. The majority gate is also known as a 'threshold' gate. Beiu has written an excellent summary of the decades-long history of threshold gate implementations in [70]. The use of the majority gate is also interesting for some kinds of nanoelectronic circuits, since the basic logic gate of quantum dot cellular automata [71] is the majority gate. Furthermore, a 3-input majority gate with its output fed back to one of its inputs forms a Muller C-element. Thus, a fault-tolerant asynchronous pipeline can be built by just using majority gates.

In the weakly indicating circuits, there is no interaction between the ‘0’ rail inputs and the ‘1’ rail inputs. The ‘0’ output rail depends only on the ‘0’ input rails and the ‘1’ output rail depends only on the ‘1’ input rails. The ‘0’ rail subcircuit is identical to, and separable from, the ‘1’ rail subcircuit. Thus the common subcircuit is the same as that used for a bare handshake. Consequently, the following discussion can focus on the bare handshake case without loss of generality.

In using a majority gate to correct faults in asynchronous sequential networks of C-elements, it can be assumed that the non-faulty inputs to the majority gate are not random, but constrained by the circuit’s sequencing constraints such as those shown in Figure 26.

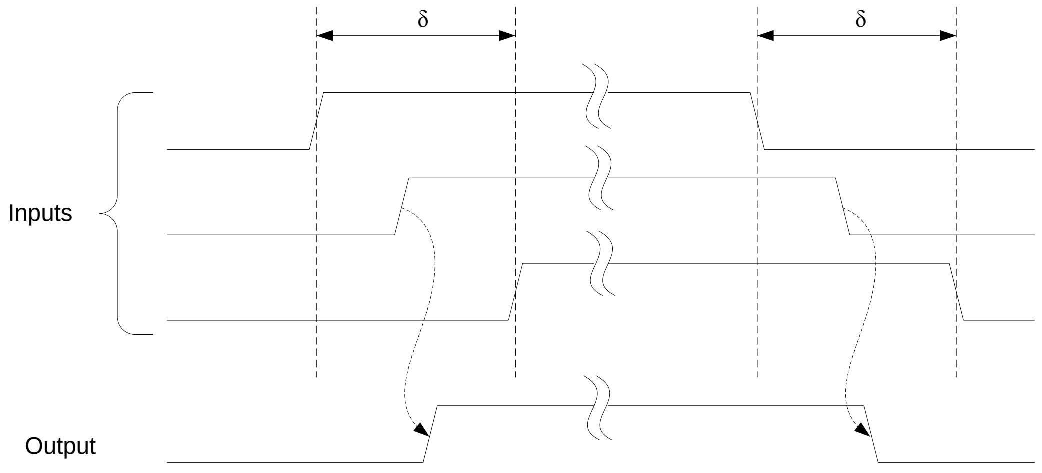


Figure 26. Majority gate voter timing showing redundant input signal skew

The majority gate performs timing restoration, correcting early transition faults, by emitting an output transition immediately after receiving congruent transitions on any two of its inputs as shown in Figure 26 (Congruent here means transitions of the same direction - either ↑ or ↓ - that arrive at different times). The output transition time is determined by the median input arrival time. The last input transition to arrive does not affect the output transition. The time skew between the arrivals of the first and last of the three congruent transitions, δ, results from varying delays in the circuits and wires that produce and deliver the redundant signals. This skew value, δ, will be used in circuit analysis later in this dissertation.

Consider the 3-input, 3-output majority-based restoration circuit shown in Figure 27. It can correct any one input stuck-high or stuck-low: Input codes 000, 001, 010 and 100 all produce output 000, and input codes 111, 110, 101 and 011 all produce output 111. Weakly indicating, it corrects an early, late, or “never” transition on any one input. As the input transitions arrive it emits transitions on all outputs after the arrival of the median input transition. Thus it can perform temporal restoration of delay faults by reducing the skew of redundant signals.

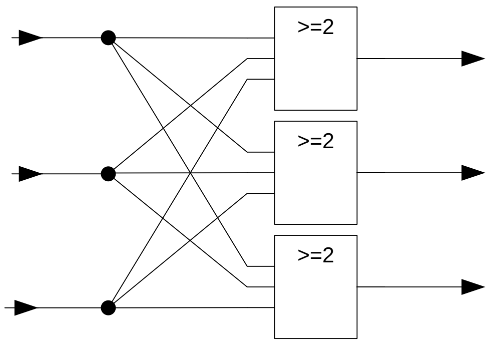


Figure 27. Triplex majority gate restoration circuit

2.5 TMR asynchronous systems using majority gate voting

N-modular redundancy can be applied to clockless systems by replicating all signal lines and circuit elements N times, then inserting majority restoration circuits in selected lines. Figure 28 shows a triple modular redundant half-buffer circuit. Half-buffers are found in both 2- and 4-phase asynchronous pipelines. They are called ‘half-buffers’ because two half-buffer stages are required to hold one data token [72].

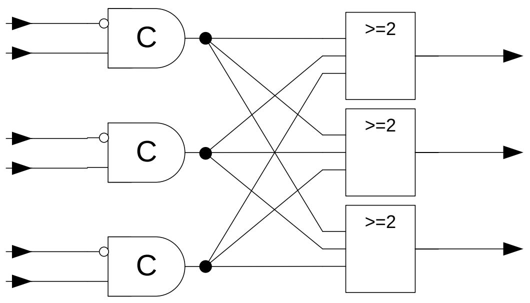


Figure 28. Triplex asynchronous pipeline half buffer

2.6 Single-event-transients in clockless circuits

Single-event transients (SETs) are dynamic faults in which a circuit node switches momentarily to the opposite logic value then returns to its initial value. SETs have two effects in micropipeline circuits. An SET on a signal that controls a capture-pass latch can temporarily change the latch mode from capture to pass, causing data stored in the latch to be lost, or can temporarily change the latch mode from pass to capture, storing duplicate data. In the latter case, the latch soon returns to pass mode with no permanent data corruption.

2.6.1 Single-event-transients In C-elements

An SET on a C-element input can, depending on when it arrives, cause the C-element to change state prematurely, thus causing an SEU. As described in [61], an input SET causes an SEU only when the C-elements differ and the polarity of the SET pulse is the complement of the current stored value of C. Figure 29 illustrates when the C-element is susceptible to SETs causing SEUs. Table 3 summarizes these conditions.

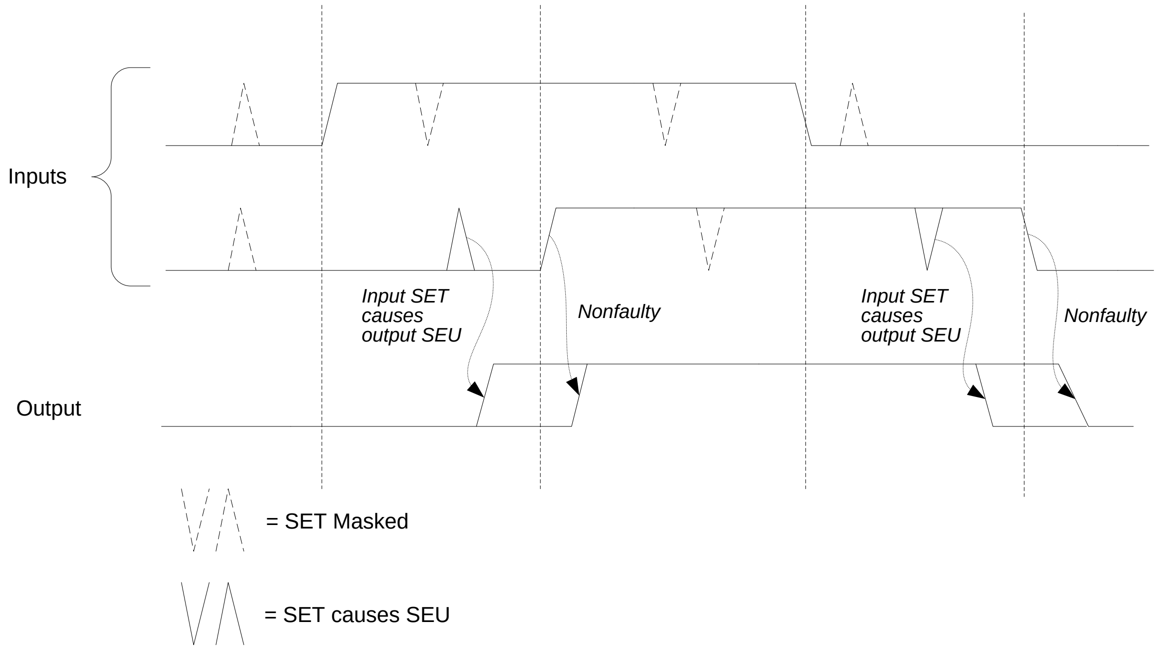


Figure 29. C-element SET while inputs differ causes output SEU

Table 3. Effects of SET on C-Element

a/b	**b/a**	**c, initial**	**c, final**	**error**
1	$0 \to 1 \to 0$	0	1	SEU
1	$0 \to 1 \to 0$	1	1	none
1	$1 \to 0 \to 1$	1	1	none
0	$0 \to 1 \to 0$	0	0	none
0	$1 \to 0 \to 1$	0	0	none
0	$1 \to 0 \to 1$	1	0	SEU

C-elements are only susceptible to SETs and SEUs when their inputs differ and they are storing state. When their inputs are equal, they mask SETs and are invulnerable to SEUs because they have no state to upset. In a micropipeline control circuit the number of C-elements storing state varies dynamically with data flow. All C-elements store state when the pipeline is either empty or full. As each C-element passes data to its neighbor, it releases its stored state and then stores a new state. The probability that a C-element in a micropipeline emits a premature transition caused by a SET is the product of the SET rate, λC, and the fraction of time that the C-element stores state, ξ:

$$\lambda_C = \lambda_{SET}\xi \qquad (7)$$

Because ξ varies dynamically during circuit operation, it cannot be determined statically from a circuit netlist. However, ξ could be measured using logic simulation driven by stimuli that dynamically models the circuit's operating environment.

2.6.2 Single-event-transients in majority gates

The majority gate can mask some, but not all, SET events. The circuit is invulnerable to input SETs when the non-faulty input is 000 or 111. The circuit is vulnerable to SETs when the inputs are not in agreement. The interval during which the circuit is vulnerable to pass SET errors begins with the first transition of a voting cycle arrives and ends when the second transition arrives. Thus the circuit's vulnerability to SETs is proportional to the variance of the arrival times of the redundant input signals. Figure 30 shows SETs superimposed on the majority gate timing diagram. When all three of the majority gate's inputs are equal, a SET on any one is masked, and it is not passed on to the output. When one of the inputs differs from the other two, an SET on the minority input is masked, but SETs on either of the matching inputs are not masked, and the SET propagates to the output.

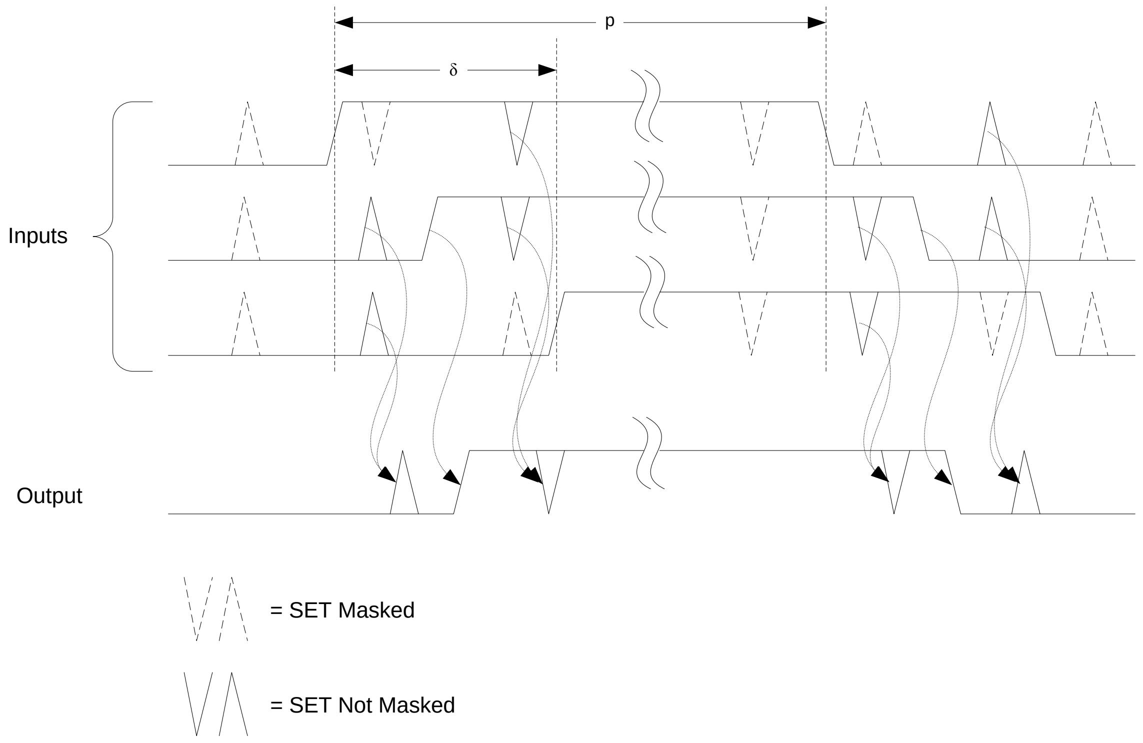


Figure 30. Majority Gate SET timing diagram

The duration of the interval when one of the inputs differs from the other two is determined by the variation in the delays, or skew, δ, of the three paths leading into the majority gate. The majority gate's SET masking ability depends on the skew between the circuit's redundant signals.

Every transition of the redundant signal triplet causes a skew interval. Thus, the fraction of time when the inputs are not unanimous is the skew divided by the switching interval, *p*: $\frac{\delta}{p}$. The switching frequency, *f*, is given by $f = \frac{1}{p}$, so this fraction can be rewritten as $\bar{\delta} f$. (In asynchronous circuits, signal transitions are not necessarily periodic, so *p* and *f* are average values.)

Let λ_{SET} be the SET rate for each circuit node. The majority gate has three inputs so that its total input SET rate can be approximated as $3\lambda_{SET}$, if it is assumed that the SET rate is low enough, and the SET pulse width is small enough, that the effect of overlapping SETs occurring on different majority gate inputs can be ignored. When the inputs are not unanimous the gate does not mask SETs on 2 of the 3 inputs so the SET-caused failure rate of the gate, λ_{MAJ}, is

$$\begin{aligned} \lambda_{MAJ} &= \left(\frac{2}{3}\right) 3\lambda_{SET} f \bar{\delta} \\ &= 2\lambda_{SET} f \bar{\delta} \end{aligned} \tag{8}$$

2.7 Hazard-Free Majority Voter

Almukhaizim and Sinanoglu recognize that majority gate voters in asynchronous circuits are vulnerable to SET-induce errors [73], and propose a "hazard-free majority voter" that freezes the voter's output from the time that two inputs agree until all three inputs agree. This method effectively masks input SETs like the ones shown in Figure 30. Thus, for the HFMV, Equation 8 can be expressed as $\lambda_{HFMV} = 0$.

Almukhaizim and Sinanoglu's proposed HFMV implementation is shown in Figure 31. They explain its operation as follows:

"The output of a typical majority voter drives a multiplexer; whose other data input is the output of the proposed hazard-free voter. The multiplexer selects the hazard-free output of the voter at i1 when the error sensitization condition is met (i.e. the "output freezing state" should be entered), and the output of the typical majority voter at *i*0, otherwise. The select line of the multiplexer is driven by a flip-flop output. The clock input of this flip-flop is connected to an XOR gate along with a delay unit, i.e., a transition detector circuit. A transition on the output of the hazard-free voter indicates that the error sensitization condition is met, which triggers a rising transition on the clock input of the flip-flop and stores the logic value 1. This event indicates the entry to the aforementioned "output freezing state". At this point, the multiplexer sensitizes the feedback loop, forcing the output of the voter to retain its value. Once the error sensitization condition is no longer met, i.e., when all of the inputs to the voter agree, the "output freezing state" is exited by changing the state of the flip-flop to logic 0. This is performed using the equality indicator, which resets the state of the flip-flop. Subsequently, the multiplexer selects the output of the typical majority voter."

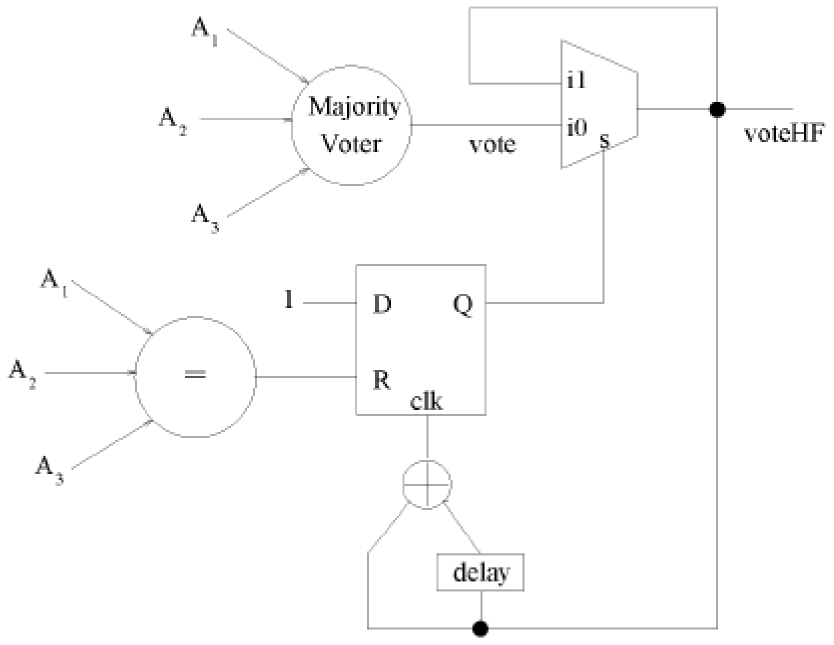


Figure 31. Hazard-free majority voter implementation from [73]

Almukhaizim and Sinanoglu state that their implementation can be realized in CMOS using 64 transistors, including “those necessary for inversions and for the delay unit”. I propose a simpler HFMV implementation shown in Figure 32. Its function is identical to that of Almukhaizim and Sinanoglu’s implementation, but it does not require a delay unit and can be implemented in CMOS using only 50 transistors.

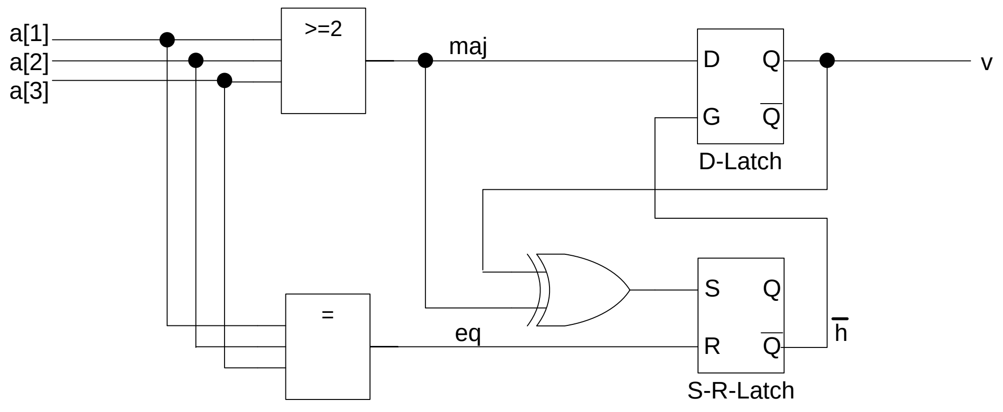


Figure 32. Alternate hazard-free majority voter implementation

A Verilog behavioral model of the 3-input HFMV implementation in Figure 32 is shown below.

```
module hfmv (output m, input [1:3] a);
      reg v = 0, // voter output
          h = 0; // hold bit - voter output cannot change while h = 1

      // 3-input combinational majority function
      wire maj = a[1]&a[2] | a[2]&a[3] | a[3]&a[1];

      // all 3 inputs are the same
      wire eq  = (a == 3'b000 || a == 3'b111);

      always @*              // S-R-latch
            if (eq) h <= 0;   // reset hold bit if all inputs are equal
            else
         if (maj ^ v) h <= 1; // set hold bit when a majority of inputs
                              // cause voter output to switch

      always @*
            if (!h) v <= maj; // D-latch holds voter output while
                              // hold bit is true

      endmodule
```

The HFMV masks stuck-at and early/late-transition faults like a combinational majority gate voter. Because the HFMV has a higher transistor count than the simple majority-gate voter, it presents a “bigger target”, and is therefore more susceptible to faults than the majority gate.

The HFMV corrects stuck-at faults at any one of its inputs like a combinational majority gate, but its temporal behavior while doing so differs slightly from the majority gate. This timing difference is shown below in Figure 33, which shows the outputs of both the majority gate and the HFMV correcting a single stuck-at-0 input. Both voter outputs rise at the same time, but the latching action of the HFMV causes its output to fall after *all* of the inputs return to zero, instead of when *a majority* of the inputs return to zero as with the combinational voter.

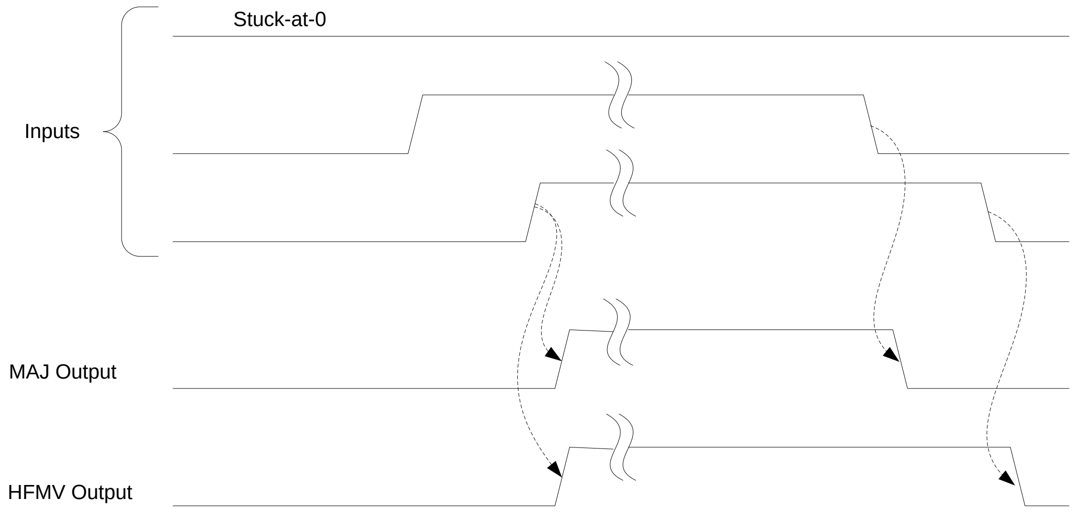


Figure 33. HFMV timing correcting with a stuck-at-0 input

Unlike the combinational voter, the HFMV does not produce correct output when one input is stuck-at-0 and another input is stuck-at-1. In the combinational voter, the opposing stuck-at inputs cancel each other and the one unstuck input determines the output. The HFMV output, however, becomes latched at 1 or 0; the latch never again opens because the opposing stuck-at inputs prevent the 000 or 111 input patterns, which are needed to reopen the latch, as shown in Figure 34.

Thus it can be hypothesized that a TMR system using HFMVs will have a lower level of stuck-at fault tolerance than one using combinational voters because HFMV's larger size makes it more vulnerable to faults, and because the HFMV cannot correct opposing stuck-at input faults where the combinational voter can.

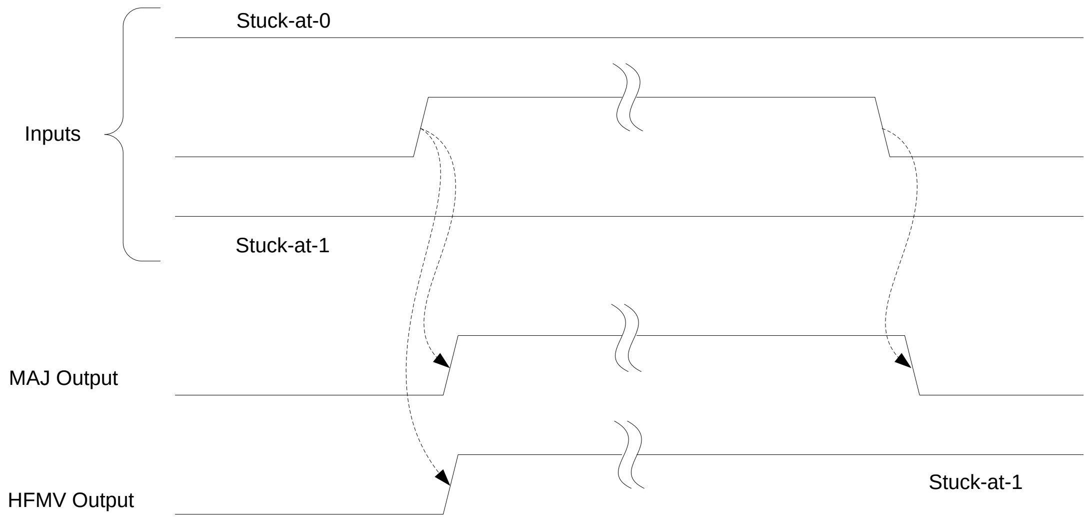


Figure 34. HFMV and MAJ output behaviors with two opposing stuck-at inputs

HFMV delay – The delay from the HFMV's primary inputs to its primary output is greater than that of the simple combinational majority gate. The critical delay path within the HFMV traverses the combinational majority gate and the D-to-Q path through the D-latch. The delays of the exclusive-or gate and the S-R latch do not impact the overall delay, if it is assumed that the exclusive-or and S-R latch delays are greater than that of the wire that connects the majority gate output to the D-latch input. If the delay of the simple combinational majority voter is t_{MAJ} then the delay of the HFMV is $t_{MAJ} + t_{D-LATCH}$.

HFMV fault model – For simplicity, the HFMV is modeled as being susceptible to the same faults as the combinational majority voter. That is, its output can be stuck-at-0, stuck-at-1; or suffer early-rise, early-fall, late-rise, or late-fall faults. The HFMV model used for stochastic fault simulation (Chapter 0) assumes that the HFMV is more likely to be faulty than the combinational majority gate in proportion with its larger transistor count.

$$\frac{Prob. HFMV\ is\ faulty}{Prob. majority\ gate\ is\ faulty} = \frac{No.\ of\ transistors\ in\ HFMV}{No.\ of\ transistors\ in\ maj. gate} = \frac{50}{12} \cong 4 \qquad (9)$$

Therefore the HFMV's greater SET tolerance comes at the cost of possibly introducing more static faults into the system. This trade-off can be quantified using equations 7 and 8. For both voter types, the total output fault probability is the sum of the probability of a static fault in the voter itself, and the probability output fault caused by input SETs. Using λ_V to represent the SET-caused failure rate of either voter type, and T to represent the duration of voter operation, this relationship can be expressed as:

$$P_{TOTAL} = P_{STATIC} + \lambda_V T \qquad (10)$$

Substituting equations 7 and 8 for each voter type yields an equation for the total output fault probability for each:

$$\begin{aligned} P_{MAJ} &= P_{STATIC} + 2\lambda_{SET} f \bar{\delta} T \\ P_{HFMV} &= 4P_{STATIC} + 0 \end{aligned} \qquad (11)$$

If is set P_{MAJ} equal to P_{HFMV} then some algebra yields:

$$3/2 = \lambda_{SET} f \bar{\delta} T \qquad (12)$$

Thus, the HFMV is more reliable if $\lambda_{SET} f \bar{\delta} T > 3/2$. Of course, in TMR systems not all uncorrected SETs on voter outputs and stuck voter outputs propagate to cause system errors. But in general, systems with higher skew, higher switching rates, and higher SET rates, favor the use of HFMVs over combinational majority voters.

2.8 Chapter Summary

This chapter has explored issues concerning majority voting in clockless systems, particularly the difficulty in deciding when to vote in absence of a clock. Several different clockless voting circuits have been considered – some of which were shown not to work. Two effective voting circuits have survived this analysis: the combinational majority gate, and the hazard-free majority voter which has better resistance to single-event transient faults but less resistance to static faults. A prior hazard-free voter circuit implementation was reviewed, and a better (lower-cost) implementation derived. This lower-cost implementation is used in the stochastic fault simulation experiments described in Chapter 5. I now move on to fault modeling in clockless circuits.

3
The Dyschronous Fault Model

This chapter introduces a novel fault model appropriate for clockless circuits. This fault model refines the classic stuck-at fault model by including a set of temporal faults that can cause errors in clockless systems. It generalizes prior work on fault modeling in asynchronous circuits, and it will be used for stochastic fault simulation discussed in Chapter 5.

This model generalizes Sirvani, et al.'s *DUDES* fault model [48], by including late-transition, as well as early-transition faults. I coin a new term to describe these early- and late-transition faults: *Dyschronous*, from Greek dys- "bad, abnormal, difficult" + khronos "time." Thus, dyschronous means happening at the wrong, or "abnormal," time. Dyschronous faults cause signal transitions to occur at the wrong time. The term dyschronous fault model is used to describe the whole fault model that includes stuck-at, single-event, and dyschronous faults.

The information conveyed with digital circuits is not only spatial (what), but also temporal (when). For example, a signal that transitions from 0 to 1 at the wrong time can cause a system to fail. This fault model considers not only errors in logic circuits as a loss of spatial information in the sense of logic values represented on circuit nodes having erroneous 1 or 0 values at an arbitrary time, but also errors that are a loss of temporal information, which can also lead to erroneous data. One possible consequence of a transition at the wrong time is that the circuit stores incorrect data; the dyschronous "when" fault is transformed into a spatial "what" error.

A fault is an abstract representation of the impact of a physical defect on the function of a logic circuit. An error is an incorrect signal value resulting from a defect.

One such fault model is the stuck-at model, which has been successfully used as a logic-level abstraction for a number of physical defects. In the stuck-at model a fault is assumed to permanently constrain a logic signal, or circuit node, to a value of 0 (stuck-at-0) or 1 (stuck-at-1) [11]. Stuck-at-0 is abbreviated here as s-a-0 and stuck-at-1, as s-a-1.

The stuck-at fault model is static because it represents fault behavior that is time-invariant. Static faults typically result from manufacturing defects, or appear as devices age. In addition to static fault models, dynamic fault models, in which fault behavior may vary in time, provide an abstract functional representation of circuit misbehavior caused by transient phenomena such as crosstalk, noise, and ionizing radiation. Dynamic fault models include single-event upset (SEU), where a transient event causes the logic value stored in a latch or memory cell to permanently change its state, from 1 to 0 or from 0 to 1 and single-event transient (SET), in which the logic value of a circuit node changes for a short time, from 0 to 1 to 0, or from 1 to 0 to 1.

3.1 Signal Taxonomy

To help understand the problems of masking faults on timing signals, I propose, in Table 4, a taxonomy of the signals in a digital system, which distinguishes those that carry value information, or data (what), and those that carry timing information (when), indicating when data are valid.

Table 4. Signal taxonomy based on temporal information content

		Signal carries timing information	
		Yes	**No**
Signal carries value information	**Yes**	Dual-rail (*what* + *when*)	Data (*what*)
	No	Clock (periodic) Req-Ack (aperiodic) (*when*)	Uninteresting

In synchronous systems, the clock signal carries periodic timing information, synchronizing data transfer between registers, but no value information. The other signals in the system carry value information, but no timing information. The clock and data signals are completely orthogonal in this respect. In asynchronous systems using bundled data, the request and acknowledge signals carry timing information, indicating when the data signals in their bundles are valid. Bundled data signals, like synchronous data signals, carry value information but no timing information. In dual-rail signaling, all signals carry both timing and value information.

Both synchronous and asynchronous systems are vulnerable to faults in their synchronizing ("when") signals. Synchronous systems are vulnerable to faults on their clock signal. Asynchronous systems are vulnerable to faults on their request and acknowledge signals.

3.2 Fault effects in clocked and clockless systems

Table 5 lists, for each signal type identified in Table 4, the consequences of a stuck-at fault on the signal, and common design strategies used to mask the fault.

Table 5. Stuck-at fault effects and masking strategy for different signal types

Signal type	**Information carried**	**Stuck-at fault consequence**	**Fault masking strategies**
Data	What	Incorrect data *Time-safe, value-fail*	ECC, NMR, interwoven redundancy
Clock	When (periodic)	Freeze *Value-safe, time-fail*	Redundant phase-locked clock generation and distribution
Req-Ack	When (aperiodic)	Deadlock *Value-safe, time-fail*	NMR + deadlock detection
Dual-Rail	What + When	Deadlock *Time-fail, value-fail*	NMR + deadlock detection

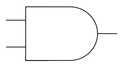
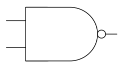
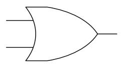
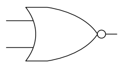
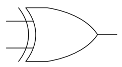
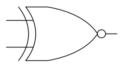
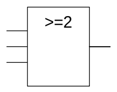
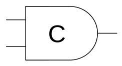
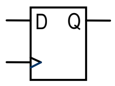
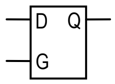
A stuck-at fault on a data signal can cause the associated bit in a field of data to be wrong. Such an error can be readily corrected using one of the many error-correcting codes, or the several varieties of redundant logic. A stuck-at fault on a clock signal does not cause any data error; rather it causes the whole system to freeze. The only method for masking a clock fault is for $N \geq 3$, that is N-modular redundancy of the clock oscillator and distribution network. A stuck-at request or acknowledge signal does not directly create erroneous data; rather it can cause data corruption by preventing a local data transfer from being completed. This can cause a pipeline blockage upstream from the location of the fault and data starvation downstream from the location of the fault. Since dual-rail signals carry both value and timing information, a fault on a dual-rail signal will have the same effect as a req/ack signal fault, but it will be data-dependent.

3.3 Error classification in logic primitives

Circuits that can function correctly in the presence of faults require redundant gates, redundant inputs, or both. The following logic gates can correct, or mask, errors at their inputs: AND/NAND, OR/NOR, MAJ. XOR/XNOR gates, incidentally, cannot correct input errors. An error at one of the gate’s inputs is said to be critical if it causes an error at the gate’s output. Whereas an error at one of the gate’s inputs is said to be subcritical if it does not cause the gate’s output to be incorrect. Thus the gate corrects subcritical errors, but not critical errors.

Kohavi classifies logic gates by their error correcting properties in [18] (p. 224). Kohavi’s error classification table can be extended to include both timing and data signals by the use of stateful elements such as the D-flip-flop, the D-latch, and the Muller C-element. It can be further modified to use input fault (rather than error) behavior. Using faults rather than errors allows the use of a more general fault set including DUDES faults [48]. Notice that the Muller C element has no subcritical input faults and therefore cannot be used as a “restoring organ”.

Table 6. Fault classification of clocked and clockless logic primitives.

Function	Symbol	Subcritical input faults	Input pin	critical input faults	Output fault as result of critical input fault
AND		s-a-1	all	s-a-0	s-a-0
NAND		s-a-1	all	s-a-0	s-a-1
OR		s-a-0	all	s-a-1	s-a-1
NOR		s-a-0	all	s-a-1	s-a-0
XOR		none	all	s-a-0 s-a-1	y $\rightarrow \neg$y
XNOR		none	all	s-a-0 s-a-1	y $\rightarrow \neg$y
MAJ		s-a-0, s-a-1	all	none	none
Muller C		none	all	s-a-0 s-a-1	up-disable (UD) down disable (DD)
D-FF		none	D	s-a-0 s-a-1	up-disable (UD) down disable (DD)
			Clock	s-a-0 s-a-1	transition disable (TD) transition disable (TD)
D-Latch		none	D	s-a-0 s-a-1	up-disable (UD) down disable (DD)
			G	s-a-0 s-a-1	transition disable (TD) transition enable (TD)

3.4 Fault Analysis of the Muller C-Element

Fault models for asynchronous circuits have already been developed by analyzing the behavior of faulty C-elements. Lu and Tong [12] derive transition-unable and extra-transition faults. Shirvani, et al.'s [13] "DUDES" fault model defines down-enabled, up-enabled, down-disabled, and up-disabled faults. Extra-transition and up/down-enabled faults cause the C-element to emit output transitions (either ↑ or ↓) prematurely, possibly causing the sequence of transition arrivals at the inputs of another circuit element to differ from that of a non-faulty system.

Figure 35, adapted from [12], shows the CMOS circuit of a dynamic C-element. Table 8 shows C-element's behavior that result from various internal stuck-at and bridging faults.

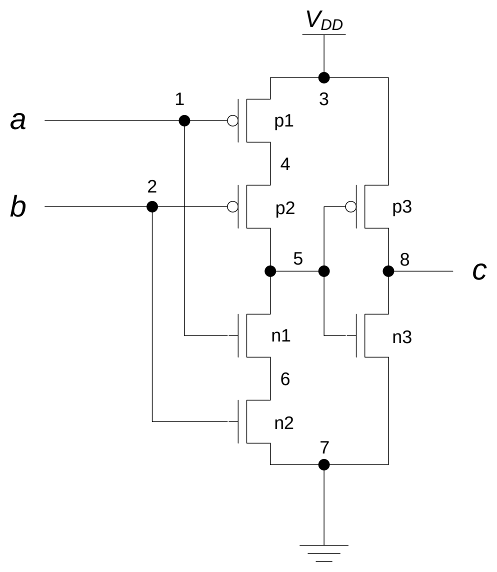


Figure 35. CMOS Dynamic C-Element, adapted from [12]

Table 7. Faulty behavior of C-element, adapted from [12]

Faults	**Logic Behavior**	**Model**
Bridging: (2,4) (3,8) (5,7)	C stuck-at-1	Stuck-at
Bridging: (1,6) (7,8) (3,5)	C stuck-at-0	
p1,p2,n3 stuck-open	no high-to-low transition	transition unable
Bridging of (1,3) (2,3)		
p3,n1,n2 stuck-open	no low-to-high transition	transition unable
Bridging of (1,7) (2,7)		
p1 stuck-on, bridging (3,4)	$C = AB + BC'$	extra transition
p2 stuck-on, bridging (4,5)	$C = AB + AC'$	
n1 stuck-on, bridging (5,6)	$C = B + AC'$	
n2 stuck-on, bridging (6,7)	$C = A + BC'$	
Bridging of (1,4)	$C = \bar{A}\bar{B} + AB + \bar{A}C'$	
Bridging of (2,6)	$C = A\bar{B} + \bar{A}BC'$	
Bridging of (1,2)	$C = AB$	
Bridging of (1,5)	$C = \bar{A}$	
Bridging of (2,5)	$C = \bar{B}$	
Bridging of (1,8)	$C = A$	
Bridging of (2,8)	$C = B$	
p3, n3 stuck-on	indeterminate	parametric
Bridging of (5,8)		

In the table, C′ represents the prior state of C.

The “DUDES” up/down enabled faults, identified by Shirvani, et al., also cause C-elements to emit premature transitions. Table 8, adapted from Shirvani, shows the equivalence between single stuck-at and stuck-open faults on the internal nodes of the CMOS C-element shown in Figure 36 and up/down enabled faults at the pins of the C-element. The rightmost column of the table shows the dyschronous fault that corresponds to the listed stuck-at or stuck-open circuit fault(s).

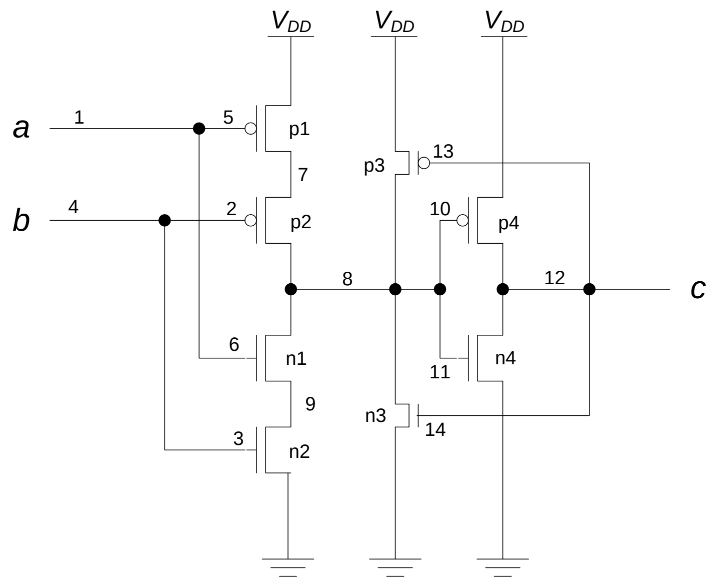


Figure 36. CMOS static C-element, adapted from [13]

Table 8. CMOS static C-Element DUDES faults, adapted from [13]; dyschronous faults added

Line No.	Logic Behavior	Node/Stuck-at (stuck open)	DUDES pin Fault	Dyschronous Fault
1	ab + ac + bc	None	None	non-faulty
2	0	8/1, 12/0	c/0	S-A-0
3	a + c	4/1	b/1	→ S-A-1
4	b + c	1/1	a/1	→ S-A-1
5	a + b	14/1	a-ue and b-ue	early ↑
6	ac + bc	3/0, 6/0, 10/1 (n1, n2, p4)	c-ud	→ S-A-0
7	ab + bc	2/0	a-de	early ↓
8	ab + ac	5/0, 7/1	b-de	early ↓
9	a + bc	6/1, 9/0	b-ue	early ↑
10	b + ac	3/1	a-ue	early ↑
11	c + ab	2/1, 5/1, 11/0 (p1, p2, n4)	c-dd	→ S-A-1
12	ac	4/0	b/0	→ S-A-0
13	bc	1/0	a/0	→ S-A-0
14	ab	13/0	a-de and b-de	early ↓
15	$a\bar{b}c$	9/1	c-ud	→ S-A-0
16	$a + \bar{b} + c$	7/0	c-dd	→ S-A-1
17	1	8/0, 12/1	c/1	S-A-1

Line No.	**Logic Behavior**	**Node/Stuck-at (stuck open)**	**DUDES pin Fault**	**Dyschronous Fault**
18	-	10/0, 11/1 (p3, n3)	(Iddq)	-
19	-	13/1, 14/0	(dynamic CMOS)	-

The timing diagram, Figure 37, of the effects of up/down enable/disable faults in the C-element shows clearly how these faults manifest as premature transitions of the C-element. The solid lines indicate non-faulty output transitions. The dashed lines indicate the premature transitions caused by up-enable and down-enable faults for both *a* and *b* inputs. The dotted lines indicate missing transitions caused by up/down disabled faults.

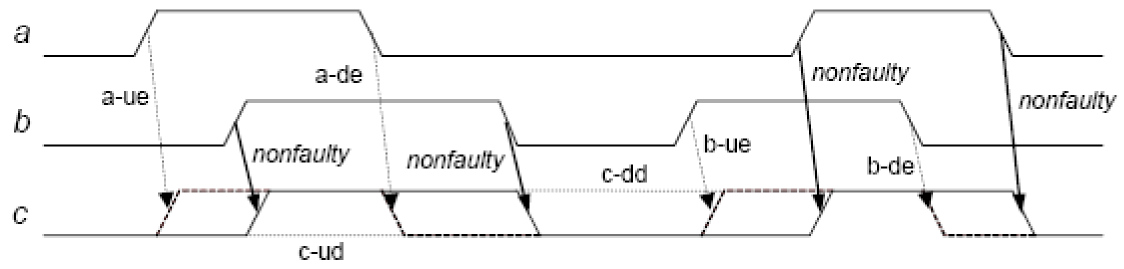


Figure 37. Dyschronous timing of DUDES faults

The Boolean expressions for all the early ↑ cases in Table 8 (lines 5, 9, and 10) can be combined to yield an expression for the equivalent faulty C-element.

$$\begin{aligned} c &= (a+b)+(a+bc)+(a+bc) \\ &= a+b+c(a+b) \\ &= a+b \end{aligned} \tag{13}$$

Similarly, the expression for a C-element with an early ↓ temporal fault (lines 7, 8, and 14) is

$$\begin{aligned} c &= (ab+bc)(ab+ac)(ab) \\ &= ab+(bc)(ac) \\ &= ab+abc \\ &= ab \end{aligned} \tag{14}$$

Adding the non-faulty case (line 1), and the case where the C-element is afflicted with both early ↑ and early ↓ faults (line 14) gives a complete temporal model for a faulty C-element

Table 9. Boolean equations for C-element DUDES faults

Fault(s)	Equation
none	$c = ab + c(a+b)$
early ↑	$c = a + b$
early ↓	$c = ab$
SA0	$c = 0$
SA1	$c = 1$

An informal, but intuitive, explanation of the premature transition fault in a C-element is based on the fact that a correctly-functioning C-element fires when the last of its two input events arrives. A faulty C-element fails to perform this function and fires when the first of two input events arrives.

A premature transition emitted from a C-element output in a micropipeline control circuit can cause corruption of the data captured by the register controlled by the C-element. Therefore, any strategy for masking faults in a micropipeline control circuit must prevent premature transitions from reaching the registers.

3.4.1 Stuck-at faults

Stuck-at faults are static faults in which a circuit node (that is, a wire) is permanently stuck to logic ‘0’ or logic ‘1’. The prior work cited addresses stuck-at faults. The transition-unable faults possible in C-elements, as identified by Lu and Tong [12], and the down-disabled and up-disabled faults identified by Shirvani, et al. [13], are latent or potential stuck-at faults, and will eventually manifest as stuck-at faults after the conditions that activate the disabled transition occur. Before the enabling condition occurs, the node is non-faulty; after the enabling condition occurs, the node is stuck-at 0 or stuck-at 1.

3.4.2 Single-Event-Upsets

Another source of premature transitions is single-event upset faults [14]. Single-event-upsets are dynamic faults in which a stored bit spontaneously changes from 1 to 0 or vice versa. A single-event upset in a C-element causes its output to change state at an unexpected random time. This occurrence can be considered a premature transition, as shown in Figure 38. Note that SEUs cannot occur within the C-element itself when its inputs, *a* and *b*, are equal, because the C-element stores no state during those times. However, if there is feedback from C-element’s output back to its input, as is often the case in clockless circuits, a single-event transient affecting the C-element’s output may trigger an incorrect acknowledgement (if the single-event transient lasts long enough to be recognized by the acknowledge circuit), which in turn changes the value of the inputs to the C-element, resulting in a permanent change, or SEU.

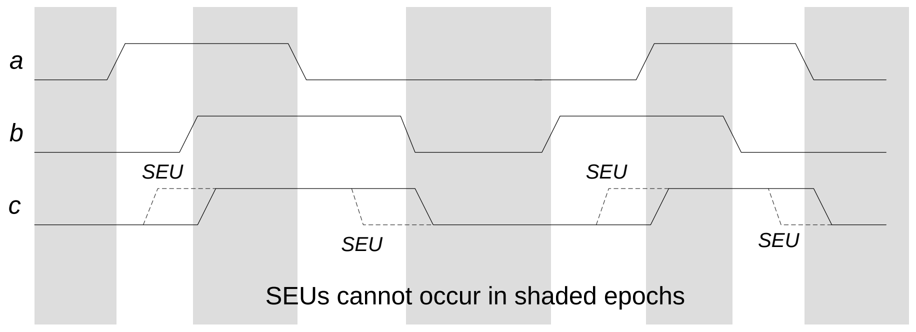


Figure 38. SEU Faults in C-elements cause premature transitions

The transitions emitted by C-elements caused by SEUs can occur spontaneously, at random times, while those caused by dyschronous faults occur following a transition on one of the C-element’s inputs, but both cause premature transitions which results in an error.

3.5 Effects of C-Element faults in Pipeline Circuits

Let us consider the effect of each of the previously identified C-element faults on the operation of the pipeline. The faults are assumed to appear on the C-element output.

Because the operation of the pipeline is symmetrical with respect to the ‘0’ and ‘1’ signal levels, stuck-at-0 and stuck-at-1 faults have equal effect and can be combined into a single stuck-at fault case. Similarly, early-rise and early-fall faults can be combined into a single early-transition fault case.

Stuck-at faults in the pipeline prevent forward propagation of request signals or backward propagation of acknowledge signals. One or more stuck-at faults in the pipeline will eventually freeze the pipeline and deadlock the system in which the pipeline resides. This kind of fault can be detected using a timeout counter that indicates system failure if an expected transition event does not occur within a predefined time limit.

An early transition fault or an SEU fault permit a C-element to fire prematurely without waiting for synchronizing events from their neighbors, corrupting the order of control events within the pipeline. This permits the latches controlled by the faulty C-element to open or close prematurely causing loss or duplication of the data stored in the latches.

3.6 Discussion of Delay “Faults”

Delay faults are a class of faults in which a defect causes a signal transition to arrive at a circuit element either earlier or later than expected. In asynchronous systems, most signal transitions follow rules of order, but there is no inherent expectation for when a signal transition will arrive. Asynchronous systems are generally tolerant of delay variations. The handshake protocols used assume that a circuit will wait as long as necessary for a delayed signal to arrive before proceeding. At the extreme, a circuit will wait forever for an infinitely delayed signal. A circuit in this situation is, for all practical purposes, deadlocked, just as it would be if it had experienced a stuck-at fault. Any distinction between an asynchronous circuit that is deadlocked and one that is operating very slowly is an arbitrary one. The only practical way to make that distinction is to use a timer circuit with an arbitrary time limit. If the expected transition arrives before the timer expires, then the timer is reset in anticipation of the next transition. If the timer expires before the expected transition, then the circuit is considered deadlocked. Thus delays less than the timer limit are not faults at all, but merely normal delay variation and delays greater than the time limit can be considered stuck-at faults. Calculation of an appropriate time limit must consider circuit delay variations, as will be discussed in Chapter 4.

Alternately, a delay fault can cause a signal transition to arrive too soon. In micropipeline control circuits, this is only a problem for delay-matched signals. For example, if a defect in one of the delay elements in Figure 14 causes it to emit a transition from its output before the corresponding logic in the datapath has completed its computation, then the following register could latch erroneous data. In its practical effect, such a delay fault is equivalent to a premature transition fault. Thus, for practical purposes, delay faults having the effect of delivering transitions early can be considered as premature-transition faults.

A generalized fault set derived by merging fault sets described in [47] and [48] with the stuck-at and single-event fault sets includes the following faults:

- s-a-0, s-a-1 faults
- Transition-disable faults including up-disable, down-disable. These are potential stuck-at faults because circuit activity can cause them to manifest at stuck-at faults in the future.
- Premature-transition faults including up-enable, down-enable. Single-event upsets (SEU) fall into this class of faults
- Late-transition faults, a.k.a delay faults.
- Extra-transition faults. Single-event transients (SET) fall into this class of faults

Table 10 summarizes the dyschronous fault set for the Muller C-element.

Table 10. C-Element fault model summary

Type	**Fault**	**Timing**	**Description**
Static	Stuck-at-0/1	Permanent	Circuit node permanently stuck at logic 0 or logic 1
Dyschronous	Early transition	Deterministic	Circuit element emits output transition in response to wrong input transition
Dynamic	SEU	Random	State element spontaneously changes output state from 0 to 1, or 1 to 0.

Type	Fault	Timing	Description
	SET		Circuit node spontaneously changes from 0 to 1 and back to 0, or 1 to 0 and back to 1.

3.7 Fault Analysis of Majority Gate

The previous section only discussed fault modeling in C-elements. Since the triplex micropipeline architecture uses majority gates to mask faults, the proposed fault model needs to be shown to apply to majority gates as well.

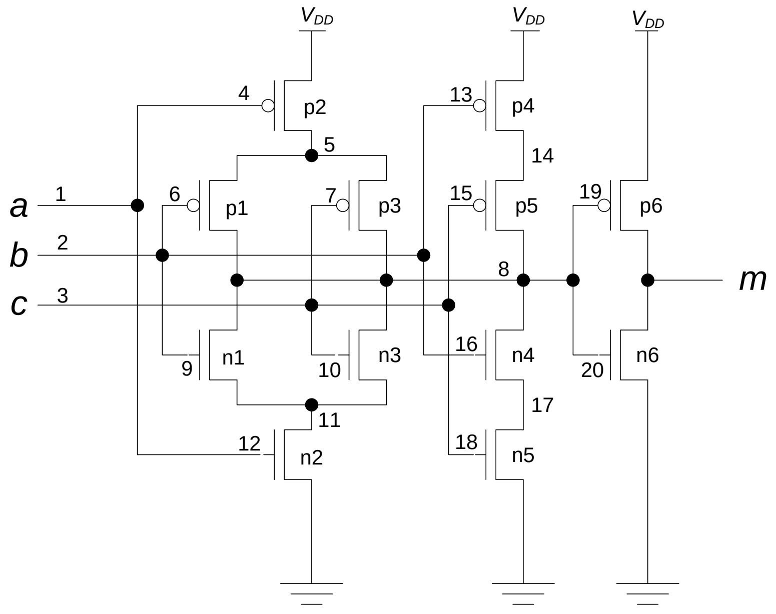


Figure 39. CMOS 3-input majority gate

Figure 39 shows the schematic diagram of a CMOS majority gate. Using an analysis similar to Shirvani's approach to the C-Element, it can be shown that single stuck-at faults on the majority gate's internal nodes are equivalent to transition faults at the gate's output.

Table 11 lists the output transition faults that are equivalent to all single stuck-at and stuck-open faults on internal nodes of the CMOS majority gate in Figure 39. In cases where a transistor is stuck open, some input combinations cause node 8 to be undriven. In these cases, the capacitance of node 8 is assumed to hold its value for the undriven period. This assumption, of course, only holds for short durations. The faults map to either stuck-at faults on the output, or early or late output transitions. In cases where a transistor is stuck on, some input combinations cause simultaneous conducting paths from VDD to ground. In these cases, the logical output of the gate is unknown, and is represented as *x* in the table, and in simulation.

Table 11. Fault equivalence between stuck-at and stuck-open faults and dyschronous output faults for the CMOS majority gate shown in Figure 39

Node/Stuck-at (stuck-open)	**Logic Behavior**	**Equivalent DUDES fault**	**Dyschronous fault(s)**
none	m = ab + ac + ab	-	-
1/0	m = bc	-	late ↑, early ↓
1/1	m = b+c	-	late ↓, early ↑
2/0	m = ac	-	late ↑, early ↓
2/1	m = a + c	-	late ↓, early ↑
3/0	m = ab	-	late ↑, early ↓
3/1	m = a+b	-	late ↓, early ↑
4/0, 5/1	$a(b \oplus c) \rightarrow m = x$ else m = ab + ac + ab	-	late↑ or early ↓
4/1, 5/0 (p2)	$m = bc + \bar{a}m$	$\bar{a}(b \oplus c) \rightarrow m/dd$	late ↓
6/0, 7/0	$\bar{a}bc \rightarrow m = x$ else m = ab + ac + ab	-	early↑ or late ↓
6/1 (p1)	$m = ab + ac + bc + \bar{a}\bar{b}cm$	$\bar{a}\bar{b}c \rightarrow m/dd$	late ↓
7/1 (p3)	$m = ab + ac + bc + \bar{a}b\bar{c}m$	$\bar{a}b\bar{c} \rightarrow m/dd$	late ↓
8/0, m/1	m = 1	m/1	S-A-1
8/1, m/0	m = 0	m/0	S-A-0
9/0 (n1)	$m = abm + ac + bc$	$ab\bar{c} \rightarrow m/ud$	late ↑
9/1, 10/1	$a\bar{b}\bar{c} \rightarrow m = x$ else m = ab + ac + ab	-	-
10/0 (n3)	$m = ab + acm + bc$	$a\bar{b}c \rightarrow m/ud$	late ↑
11/0, 12/1	$\bar{a}(b \oplus c) \rightarrow m = x$ else m = ab + ac + ab	-	-
11/1, 12/0 (n2)	$m = bc + am$	$a(b \oplus c) \rightarrow m/ud$	late ↑
13/0, 14/1	$ab\bar{c} \rightarrow m = x$ else m = ab + ac + ab	-	-
13/1, 14/0, 15/1 (p4, p5)	$m = ab + ac + \bar{a}bcm$	$a\bar{b}\bar{c} \rightarrow m/dd$	late ↓
15/0	$a\bar{b}c \rightarrow m = x$ else m = ab + ac + ab	-	-
16/0, 17/1, 18/0 (n4, n5)	$m = a(b + c) + bcm$	$\bar{a}bc \rightarrow m/ud$	late ↑

Node/Stuck-at (stuck-open)	**Logic Behavior**	**Equivalent DUDES fault**	**Dyschronous fault(s)**
16/1	$\bar{a}\bar{b}c \rightarrow m = x$ else m = ab + ac + ab	-	-
18/1, 17/0	$\bar{a}b\bar{c} \rightarrow m = x$ else m = ab + ac + ab	-	-
19/0	$ab + ac + bc \rightarrow m = x$ else m = ab + ac + ab	-	-
19/1 (p6)	-	m/ud	$\rightarrow$ S-A-0
20/0 (n6)	-	m/dd	$\rightarrow$ S-A-1
20/1	$\bar{a}\bar{b} + \overline{ac} + \bar{b}\bar{c} \rightarrow m = x$ else m = ab + ac + ab	-	-

The timings of early and late output transitions are shown in Figure 40. The solid lines indicate non-faulty transitions. The dashed lines indicate early and late output transitions caused by dyschronous faults in the gate. Table 12 gives the Boolean equations for the majority gate in the non-faulty, dyschronous fault, and stuck-at fault cases.

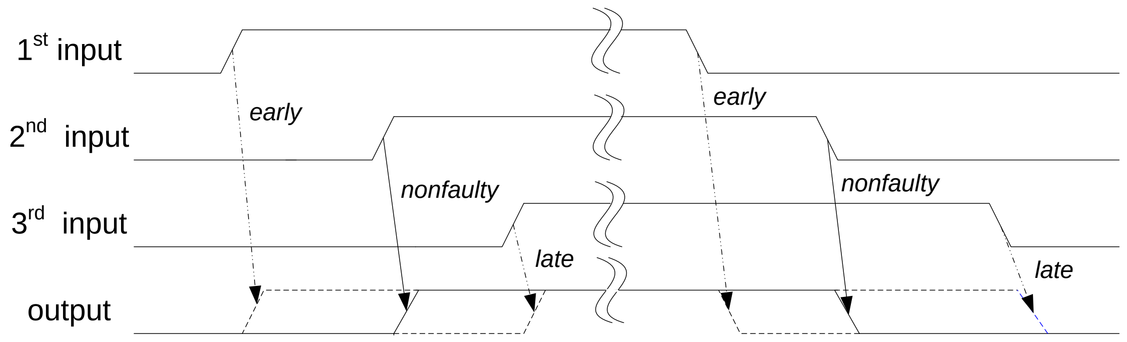


Figure 40. Timing of dyschronous faults in majority gate output

Table 12. Boolean equation for majority gate dyschronous faults

Fault(s)	**Equation**
none	$m = ab + ac + bc$
early ↑ and late ↓	$m = a + b + c$
late ↑ and early ↓	$m = abc$
early ↑ only	$m = \bar{m}(a + b + c) + m(ab + ac + bc)$
late ↑ only	$m = \bar{m}(abc) + m(ab + ac + bc)$
early ↓ only	$m = \bar{m}(ab + ac + bc) + m(abc)$
late ↓ only	$m = \bar{m}(ab + ac + bc) + m(a + b + c)$

Fault(s)	**Equation**
early ↑ and early ↓	$c = ab$
SA0	$m = 0$
SA1	$m = 1$

Premature transition faults result from faulty transistors in the CMOS circuit’s pull-up or pull-down networks that are stuck-on. They can exist only in circuits in which some input combinations do not determine the output value. That is, the circuit is either dynamic or stores state. A stuck-on transistor disables a guard condition or enables a premature state change.

In combinational circuits such as the majority gate, a stuck-on transistor fault will cause both the gate’s pull-up and pull-down networks to conduct simultaneously for one or more input combinations. When this occurs, the gate’s output will be indeterminate and thus modeled as an ‘x’ value in the logic simulation. This defect may cause the gate to draw excessive current from its power supply, and result in damage to the gate. But analyzing these consequences is beyond the scope of this dissertation.

In triplex circuits, when the majority gate’s inputs are non-faulty, that is, either 000 or 111, no single stuck-on fault has effect, that is, none of the possible stuck-on faults is activated unless the inputs differ.

As applied in micropipeline control circuits, C-elements perform the function of joining two input events by emitting an output event when the second of two input events has arrived. Majority gates perform a somewhat similar function: they join three input events by emitting an output event when the second of three input events has arrived. A faulty C-element can emit an output event when the first of two input events has arrived, but a faulty majority gate can fail in one of two ways: it can, like a faulty C-element, emit an early output event when the first input event has arrived, or it can emit a late output event when the third of three input events has arrived. Thus, a faulty majority gate can emit either premature or late output transitions

3.8 Majority gate masking of dyschronous faults

This section explains how the majority gate can mask the static and dyschronous faults listed in Table 12.

- Stuck-at faults – The majority gate corrects the following stuck-at fault cases, emitting a corrected signal from its output:
    - One input S-A-1 or S-A-0 and two non-faulty inputs. In this case, the two non-faulty inputs correct the majority gate's output by majority vote.
    - One input S-A-1 and one input S-A-0 and one non-faulty input. Here, the opposite stuck inputs cancel each other, enabling the single non-faulty input to be the "swing vote" and determine the majority gate's output

  The majority gate cannot correct:

- Two inputs S-A-0 or S-A-1. The two concurring stuck inputs form a permanent majority and stick the majority gate's output. The third input can be non-faulty or stuck either way; it has no effect on the majority gate's output.
- Dyschronous faults – The majority gate emits an output transition immediately after receiving any two of three congruent input transitions, as shown in Figure 40. This corrects an early transition on any one input and two non-faulty inputs. In this case, the two non-faulty inputs correct the gate's output by a temporal majority vote.
- SEU faults – The majority gate itself is not vulnerable to SEU faults because it has no stored state to "upset". An SEU in a state element, such as a C-element, in the transitive fan-in of the majority gate will result in an early transition on the majority gate's input, and can be corrected as described above. However, if the majority gate is in a feedback path that includes C-elements, as is likely when the majority is used as a voter in a clockless system, a single-event transient affecting the majority gate's output may trigger an incorrect acknowledgement, which in turn changes the values of one or more inputs to C-elements in the path, resulting in a permanent change of the C-element state.

3.9 When dyschronous faults cause errors

C-elements in clockless circuits are susceptible to early-transition faults. An early-transition fault will cause an error if the difference between the arrival times of the two input transitions is less than the propagation delay of the faulty gate.

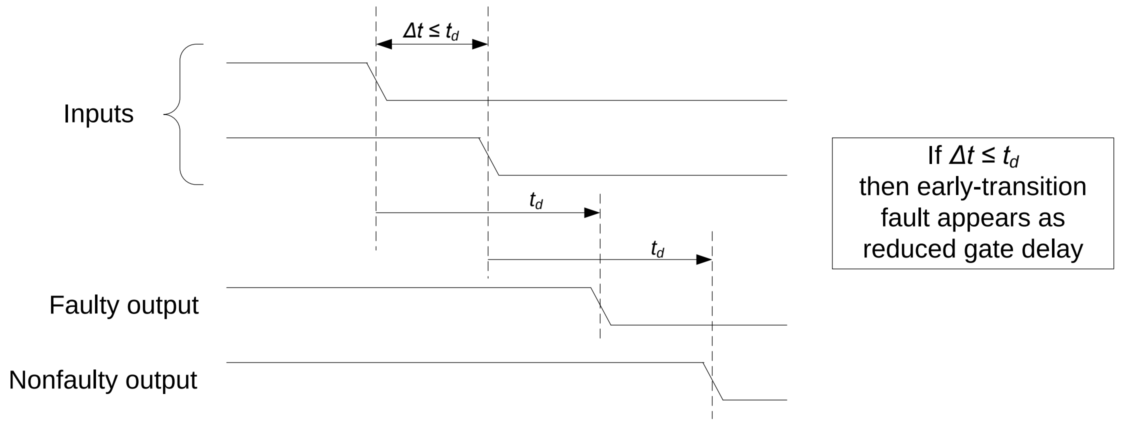


Figure 41. Early-transition fault timing diagram

Figure 41 shows the timing of faulty and nonfaulty outputs from a C-element that may have an early-fall fault. The difference between the arrival times of the two input transitions is less than the propagation delay of the faulty gate. The falling edge case is shown; the rising edge case is similar. The time difference between the arrivals of the input transitions on the C-elements two inputs is Δt. The propagation delay of the C-element is t_d. If Δt is less than or equal to t_d then the early-fall fault does not change the sequence of events. The early-fall fault causes the output to fall earlier than it would without the fault, but because Δt is less than or equal to t_d, the output still falls after the second input falls, maintaining the same sequence as the nonfaulty gate. This is indistinguishable from a gate with reduced propagation delay. Thus whether an early-transition fault in a C-element causes an error depends on the relative timing of its inputs. The timing of the inputs varies with the dynamic operation of the circuit. Some of the faulty C-element's output events will cause errors and others will not.

Majority gates in clockless circuits are susceptible to both early- and late-transition faults. A late-transition fault causes the majority gate to emit its output transition after receiving the third, rather than the second, input transition. Thus the late-transition fault causes the majority gate to emit its output transition later than it otherwise would. This is indistinguishable from the gate having increased propagation delay. A late-transition fault in a majority gate will cause an error only if the gate's third input never arrives because of a stuck-at fault at its input.

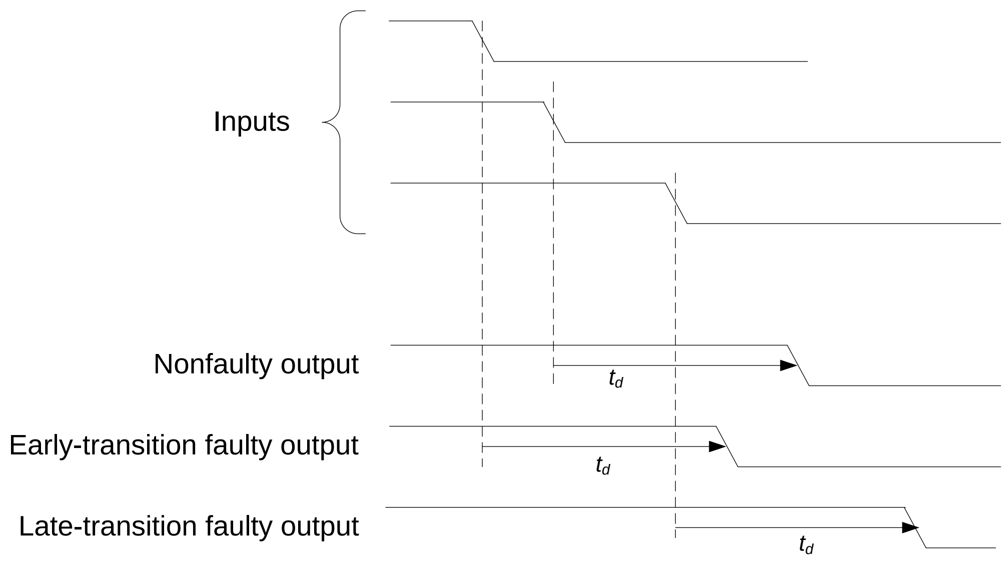


Figure 42. Majority gate early- and late-transition fault timing diagram

Figure 42 shows the timing of early- and late-fall faults in a majority. An early-transition fault causes the majority gate to emit its output transition after receiving the first, rather than second, input transition. Thus the early-transition fault causes the majority gate to emit its output transition earlier than it otherwise would. This is indistinguishable from the gate having reduced propagation delay. An early-transition fault in a majority gate will cause an error only if the gate's first input arrives early because of an early-transition fault in another circuit element connected to the input.

The examples in the preceding discussion show that some faults in majority gates in clockless circuits do not cause errors, though they do allow propagation of errors introduced by faulty upstream circuit elements. In the triplex circuits that are the subject of this dissertation, majority gates are connected to C-elements as shown in Figure 28.

3.10 Dyschronous Fault Analysis of OR gate

In addition to the Muller C-element, and majority gate, a fault analysis of the OR gate is needed because OR gates are essential components in 4-phase dual-rail clockless circuits. As applied to 4-phase clockless logic, the OR gate's (nonfaulty) inputs are mutually exclusive – only one of the inputs can be '1' at any time. As shown in Figure 43, the OR gate emits an output transition is response to a single input transition. This is in contrast to the C-element and the majority gate, both of which implement a more complex temporal function. The C-element emits an output transition in response to the second of two input transitions. The majority gate emits an output transition in response to the second of three input transitions. A dyschronous fault in a majority gate or C-element causes it to emit an output transition in response to the wrong input transition. A faulty OR gate cannot emit a output transition in response to the wrong input transition because its temporal function involves only one input transition .

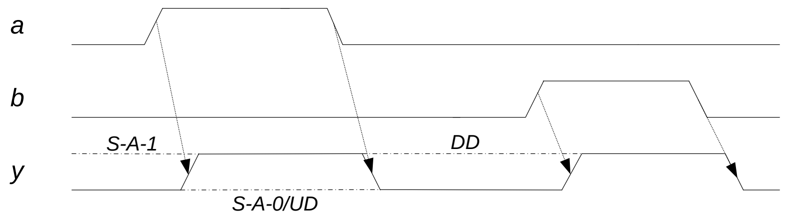


Figure 43. Timing of OR gate applied in asynchronous circuit

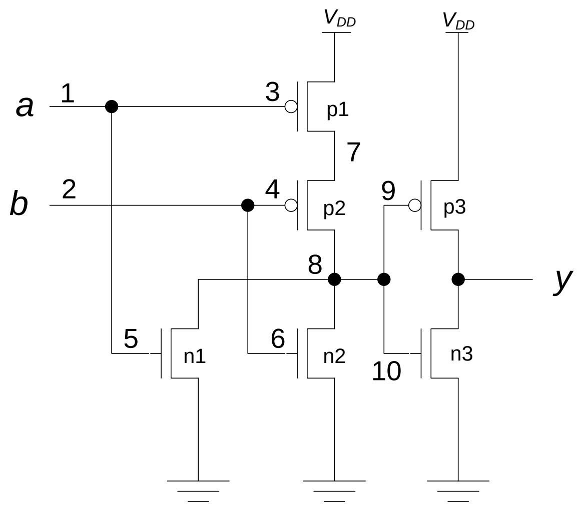


Figure 44. CMOS OR Gate

Table 13. Stuck-at and dyschronous faults in CMOS OR gate

Node/Stuck-at (stuck-open)	**Logic Behavior**	**Output Fault**	**Notes**
none	$y = a + b$	-	nonfaulty
1/0	$y = b$		
1/1, 5/1, 8/0, 2/1, 6/1, 9/0	$y = 1$	S-A-1	
2/0	$y = a$		
8/1, 10/1	$y = 0$	S-A-0	
3/1, 4/1, 5/0, 6/0, 7/0, 10/0 (p1, p2, n1, n2, n3))		DD	nmos or pmos stuck off
3/0, 7/1	$\bar{a}b \rightarrow y = x$ else $y = a + b$		pmos stuck on
4/0	$a\bar{b} \rightarrow y = x$ else $y = a + b$		pmos stuck on
9/1(p3)		UD	pmos stuck off

3.11 Dyschronous Fault Model Summary

Table 14 summarizes the majority gate fault model. Because it subsumes the C-element fault model (Table 9), it serves as a complete fault model for any circuit composed of C-elements and majority gates. Like all fault models, the dyschronous fault model is an abstract representation of the manifestation of physical defects at the functional level.

Table 14. C-Element and majority gate fault model summary

Type	**Fault**	**Timing**	**Description**
Static	Stuck-at-0	Permanent	Circuit node permanently stuck at logic 0 or logic 1
	Stuck-at-1		
Dyschronous	Early transition	Deterministic	Circuit element emits output transition response to wrong input transition
	Late transition		
Dynamic	SEU	Random	State element spontaneously changes output state from 0 to 1, or 1 to 0.
	SET		Circuit node spontaneously changes from 0 to 1 and back to 0, or 1 to 0 and back to 1.

This chapter has developed a new *dyschronous* fault model, useful in analyzing faults in clockless circuits, that extends the well-known stuck-at fault model to include early/late transition temporal faults. This fault model was derived from examining some of the underlying fault mechanisms in CMOS logic gates, but the dyschronous model is general enough to be useful for other logic technologies. In general, dyschronous faults cover any defect that causes a logic gate to emit an output transition at the wrong time.

4
Triplex Clockless Pipeline Architectures

This chapter describes a triplex clockless architecture that applies triple-modular redundancy and majority voting to increase the reliability of clockless circuits and systems. Two different clockless pipeline circuits, a two-phase bundled-data pipeline and a four-phase dual-rail pipeline, are presented as examples of the triplex architecture. The area and performance of equivalent triplex and simplex circuits are also compared.

In the diagrams in this chapter, I use, somewhat arbitrarily, cardinal numbering for redundant signals because, unlike data signals, they are not ordered and have no place value.

4.1 Dataless Pipeline

Triple-modular redundancy can be applied to clockless pipeline circuits by triplicating each wire and each primitive gate (C-element, etc.), then inserting majority voters at appropriate points in the circuit. The simplex dataless pipeline in Figure 11 can be made into a triplex fault-tolerant circuit by triplicating every element in the simplex circuit, then inserting majority voters, as shown in Figure 45. Additional single-event transient (SET) tolerance may be gained by using hazard-free majority voters (HFMVs) rather than combinational majority voters. This voting method makes redundant C-elements functionally equivalent, so the three wires in any redundant control triplet can be randomly permuted without any loss of function. This observation is similar to Han's observation in [33] that "the reliability of a ... circuit with random interconnections is, in general, comparable with that of its [nonrandom] equivalent". This permutation-tolerance feature may be interesting for nanocircuits built using stochastic assembly methods.

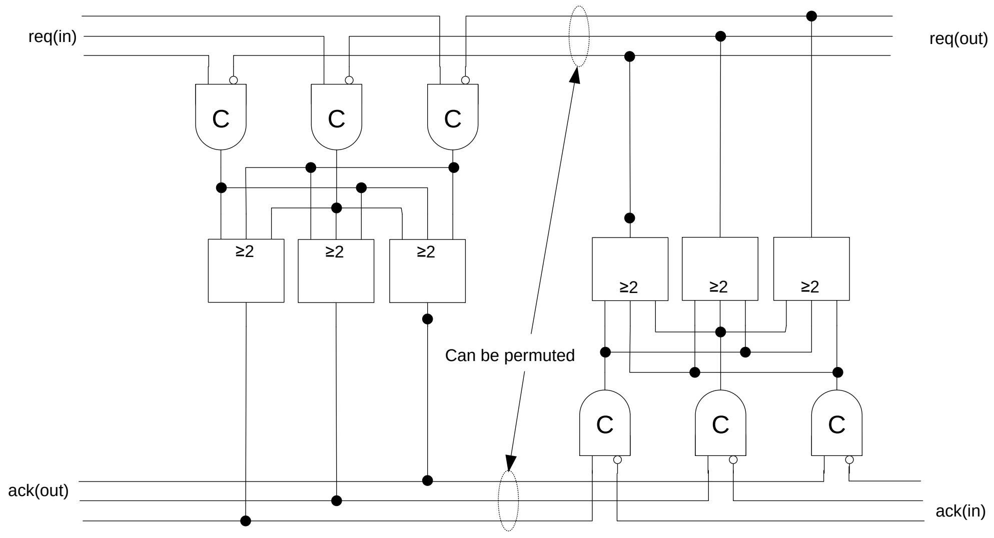


Figure 45. Triplex dataless pipeline

In any system that includes both simplex and triplex circuits, it is necessary to design simplex-triplex interface circuits. The error rate of the simplex portion of the circuit must be assumed to be acceptably low. The dataless pipeline has only request and acknowledge signals to interface. Converting the simplex signal to triplex can be accomplished by simple fanout. Converting the triplex signal to simplex requires a vote in case one of the three redundant copies of the signal is faulty (either stuck or suffering dyschronous transitions). Figure 46 shows a dataless simplex-triplex interface. Additional SET tolerance may be gained by using an HFMV rather than a combinational majority gate.

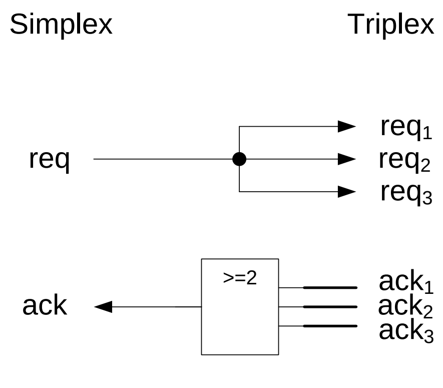


Figure 46. Triplex-simplex handshake interface

The figure shows request (req) flowing from the simplex domain to the triplex domain, and acknowledge flowing from triplex to simplex. By interchanging the signal names *req* and *ack*, the same circuit serves to pass a request signal from triplex to simplex, and an acknowledge signal from simplex to triplex.

4.2 Triplex 2-Phase Bundled-Data Micropipeline

TMR can be applied to the 2-phase bundled-data micropipeline by triplicating each wire, C-Element, and data latch, then inserting majority voters into the circuit. The voters can be either combinational majority gates or, for increased SET tolerance, hazard-free majority voters. The control portion of the triplex micropipeline circuit is identical to the triplex dataless pipeline circuit described in the previous section. The triplex bundled datapath is implemented with triple redundant capture-pass latches. Error-correcting combinational majority gates can then be inserted between latch stages.

Figure 47 shows a triplex micropipeline in which the voter outputs drive the capture-pass latch inputs.

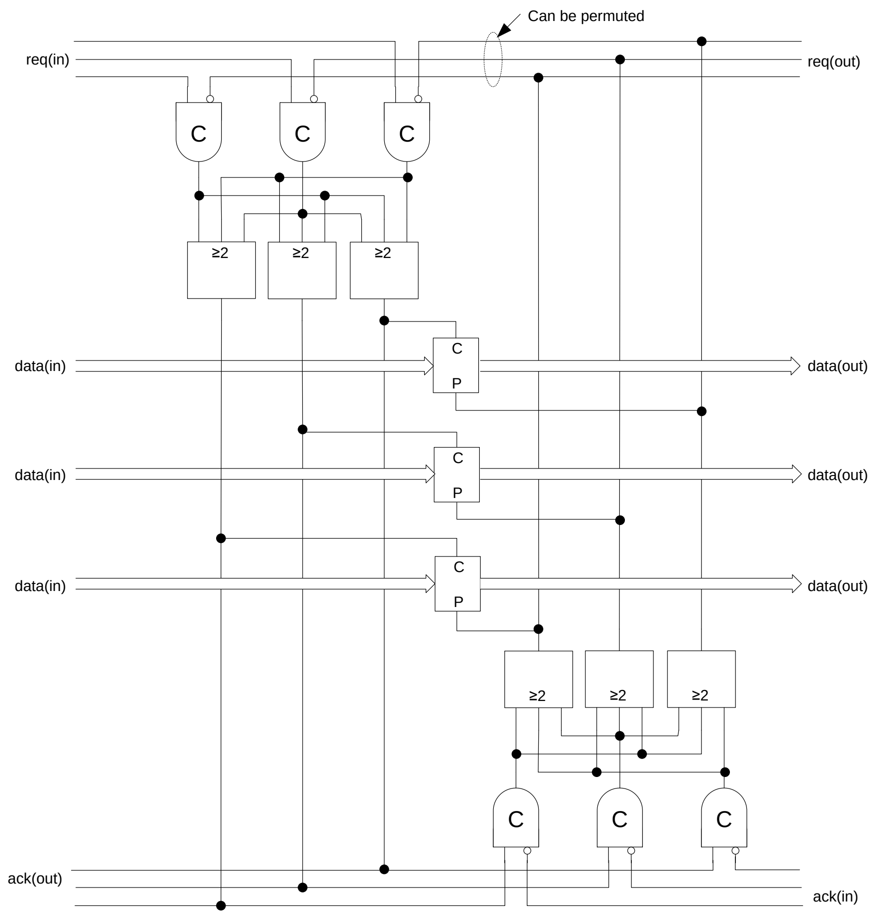


Figure 47. Triplex micropipeline

A scenario that can cause data errors in a fault-free triplex pipeline is illustrated in the timing diagram in Figure 48. The top three traces in the diagram represent a redundant request (C-element output) signal triplet at some stage of the pipeline. Delay variation causes skew between the three signals in the triplet; the skew between rising transitions on the second and third signals is denoted δ. The three redundant request signals are input to voters. The fourth trace in the diagram shows the output of one voter. The voter output signal transitions following the second request signal transition. The delay of the voter is denoted t_V. The voter output drives the capture control input of the latch. The latch's input set-up time requirement[1] is denoted by t_S. If $\delta > t_V + t_S$ then the

[1] The set-up time requirement is the minimum time before the capture input transition that the data must be stable in order to assure that the latch captures the input data accurately

latch setup time requirement is violated, possibly causing the data captured in the latch to be corrupted.

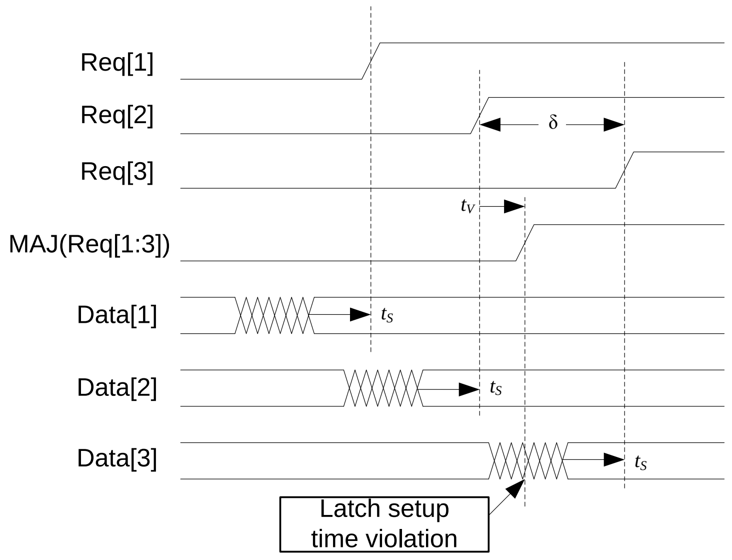


Figure 48. Voter/latch timing hazard in triplex micropipeline

If data are corrupted in only one of the three redundant data latches then the correct data value can be restored by majority vote (using additional voters in the redundant data path). But if data are corrupted in this manner in two or more of the redundant latches at the same time then the correct value cannot be restored.

A solution to this problem is to constrain the minimum voter delay, t_V, to always be greater than the maximum skew, δ. This is nothing more than implementing Davies and Wakerly's suggestion in [34]. This approach may require adding delay circuitry to all of the voters, increasing system cost, and reducing system performance. The added delay circuitry will also be susceptible to faults and therefore reduce system reliability. The maximum skew, δ, may be difficult to determine in practice since it depends on delay variation among redundant gates and wires, and will vary with manufacturing process, voltage, and temperature. Another solution is to design the voter with minimum delay greater than maximum input skew, but cut short the delay if all three inputs agree.

4.3 The CMAJ Element

A new circuit element that combines the functionality of a 3-input C-element and a 3-input majority gate provides an alternate solution to this dilemma. The 3-input “CMAJ” element initially functions as a C-element, waiting for all three input events to arrive, then after some delay, it functions as a 2-of-3 majority gate. The delay must be greater than the maximum skew among the three signals in the triplet. The CMAJ begins each cycle strongly indicating, and then becomes weakly indicating if necessary. Thus the CMAJ element modifies Davies and Wakerly’s solution (“A majority circuit determines when f + 1 of the … inputs have been received, then delays for a length of time *d*, and sends [the voted output].” [34]) by skipping the delay if all of the inputs arrive.

The CMAJ element can be constructed from a 2-of-3 majority gate, a delay element, and a 3-of-5 majority gate as shown in Figure 49. The 3-of-5 majority gate’s output is fed back to its input in a manner similar to that of the 2-of-3 majority gate whose feedback forms a 2-input C-element. If the three inputs quickly come to agreement then the 3-of-5 majority gate acts like a C-element. If they do not reach agreement before the delay has expired, then the delayed 2-of-3 majority result tips the 3-of-5 balance.

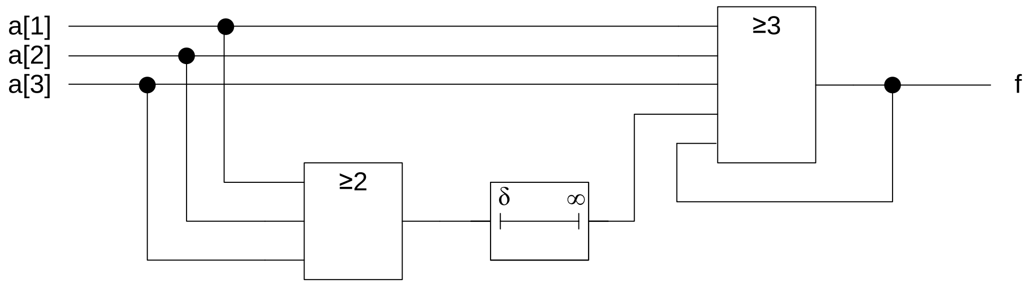


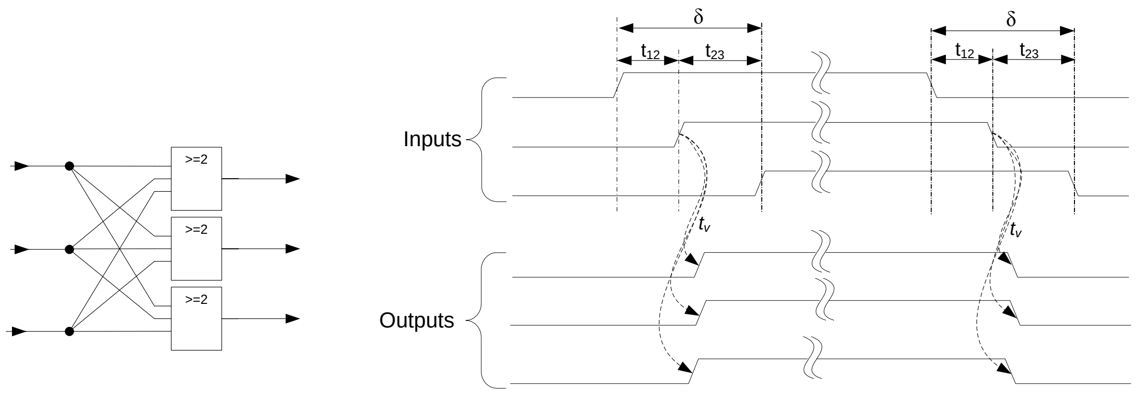
Figure 49. CMAJ element schematic diagram

A Verilog behavioral model of the 3-input CMAJ element in Figure 49 is shown below.

```
module cmaj
   #(parameter TIMELIMIT = 10) (input [1:3] a, output reg c = 0);
   reg delayed_maj = 0;
   integer n = 0;

   always @* // 2 of 3 majority gate with delayed output
      delayed_maj <= #TIMELIMIT (a[1]&a[2] | a[1]&a[3] | a[2]&a[3]);

   always @* // 3 of 5 threshold gate
      // count how many inputs are '1'
      n = delayed_maj + a[1] + a[2] + a[3] + c;

   always @* // set c = 1 if at least 3 of 5 inputs are '1'
      c = #1 (n > 2);

endmodule
```

4.4 Triplex vs. Simplex Micropipeline Performance

The performance of the triplex micropipeline can be measured by its period, and compared to the simplex micropipeline period described in Chapter 1. The period of a triplex micropipeline is defined not by a single delay loop, but rather by three delay loops that interact in the voters.



a. Circuit a. Timing Diagram

Figure 50. Triplex Majority-Gate Voter

The triplex voter shown in Figure 50 delays each of the three redundant input signals by a different amount. Each majority gate emits an output transition after it has received the second of three input transitions. The second of the of the three input transitions to arrive is delayed by the delay through the majority gate, t_V. The third of the three input signal transitions is delayed by t_V plus the skew between the 2nd and 3rd transitions, Δt_{23}. The first of the three signal input transitions is delayed by t_V minus the skew between the 1st and 2nd transitions, Δt_{12}. This value can be negative. The total skew of all three signals, δ, in the triplet is a random variable that depends on manufacturing variations and local circuit conditions such as voltage, temperature and crosstalk. If the distribution of these signal delays is assumed to be symmetric, then the expected value of Δt_{23} and Δt_{12} then is $\delta/2$. So the expected voter delay is $t_V + \delta/2$.

The voter delay must be added to both the forward and reverse latencies in the triplex micropipeline. The forward propagation delay of the request signal, $L_{forward}$, includes the delay through the voter, $t_V + \delta/2$, in addition to the delay through the C-element, t_C. The reverse propagation delay of the acknowledge signal also includes the delay through the voter, $t_V + \delta/2$, in addition to the delay through the C-element, t_C, and the inverter at the right input of the C-element, t_{INV}., $L_{reverse}$. Thus the triplex period, P, is:

$$\begin{aligned} P &= L_{forward} + L_{reverse} \\ &= \left(t_C + t_V + \frac{\delta}{2} \right) + \left(t_C + t_{INV} + t_V + \frac{\delta}{2} \right) \\ &= 2\left(t_C + t_V + \frac{\delta}{2} \right) + t_{INV} \end{aligned} \tag{15}$$

The triplex micropipeline's period is two voter delays greater than the simplex micropipeline's. This equation gives the period of a triplex pipeline without processing; the period of a triplex pipeline with processing must include the interstage delay. Thus the period becomes

$$P = 2\left(t_C + t_V + \frac{\delta}{2} \right) + t_{INV} + t_{DELAY} \tag{16}$$

Since the C-element and the majority gate have roughly similar structures, the majority gate delay will be roughly similar to the C-element delay. If $t_c \gg t_{DELAY}$, as in a nonprocessing clockless FIFO, the triplex micropipeline's period will be approximately twice the simplex micropipeline. If $t_c \ll t_{DELAY}$, as in a processing pipeline, the triplex micropipeline's period will be only somewhat greater than the simplex.

4.5 Triplex 4-phase dual-rail pipeline

Unlike the two-phase signaling used in Sutherland's Micropipeline, 4-phase signaling is not symmetrical with respect to '0' and '1' logic levels. '0' represents the absence of data, while '1' represents the presence, or validity, of data. The dual-rail encoding shown in Table 1 shows that '00' is a legal code, but '11' is illegal because it would represent the presence of '0' and '1' at the same time. A consequence of this is that an early-rise fault in a C-element in a 4-phase dual-rail (4P2R) system can cause the illegal '11' code. The triplex fault-tolerant architecture can be applied to the four-phase dual-rail pipeline by triplicating each circuit element, and adding restoring majority voters to each C-element output. Figure 51 shows one stage of a triplex 1-bit 4P2R pipeline.

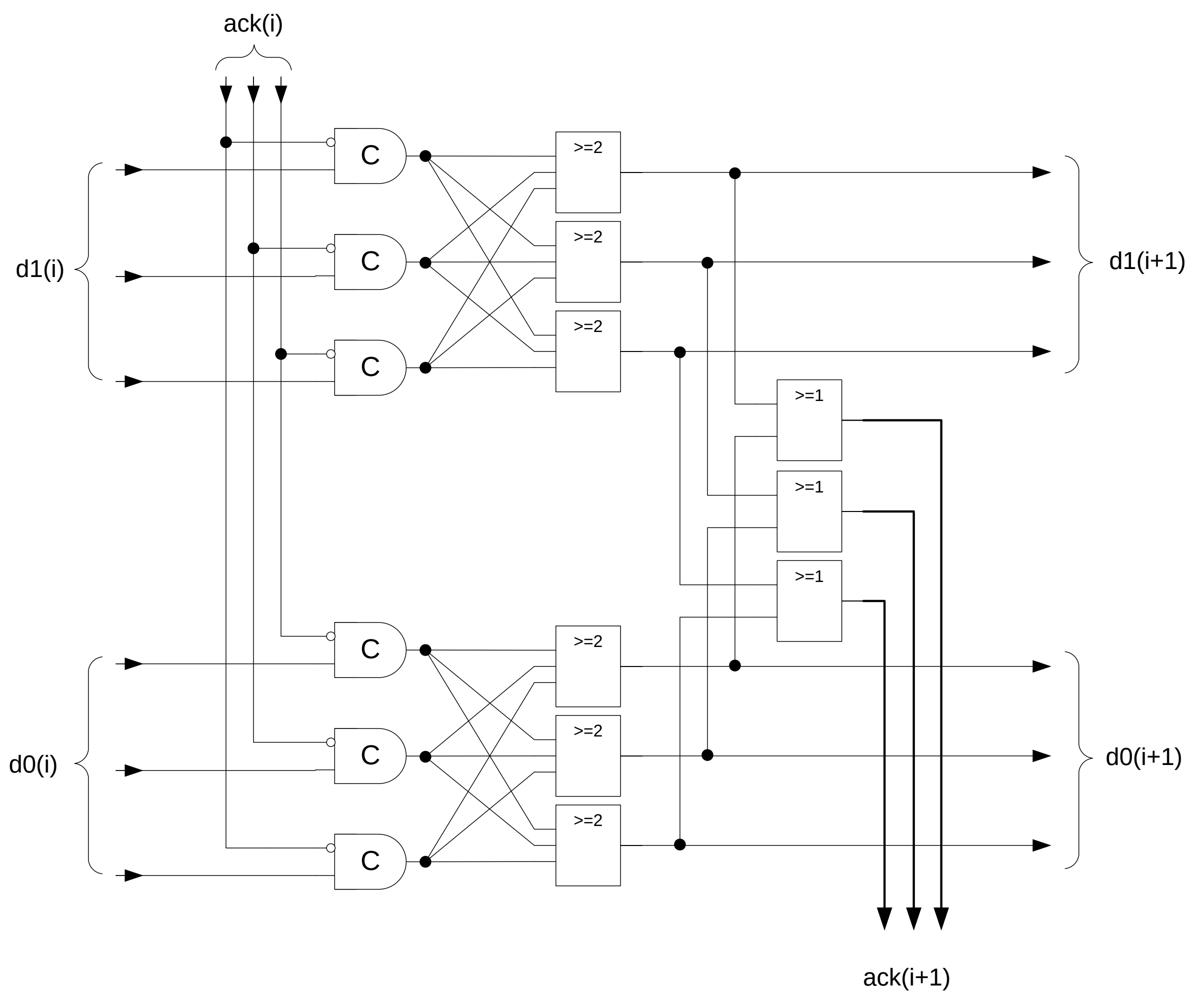


Figure 51. Triplex 1-bit 4-phase dual-rail pipeline stage

4.6 Bundled Simplex-Triplex Interface

At some point the triplex circuit must interface with a nonredundant simplex circuit. Converting simplex timing or data signals to triplex is accomplished simply by connecting the simplex signal directly to the three signals in a triplet, as shown in Figure 46. However, converting a triplex signal triplet that controls bundled data to a simplex signal is not as simple because of the timing problem illustrated in Figure 48. Transitions on the three wires of the triplet may arrive at different times. In a bundled data triplex-to-simplex interface there are three bundles of data; the time when each bundle is valid is indicated by a unique request signal. Correcting data errors by majority requires waiting until all three bundles are valid, i.e., waiting until all three request events have occurred. This can be accomplished by combining the three request signals in a 3-input C-element. The C-element's output then indicates when the data voting can occur. Such a C-element circuit can correct for a dyschronous fault on one of the request lines – one of the data bundles may not be valid at the time of voting but the two correctly-timed bundles will constitute a correctable majority. However, the strongly-indicating C-element circuit cannot correct even one stuck-at fault on a request line: it will deadlock waiting for the last request event which will never arrive.

Combining the three request lines in a weakly-indicating majority gate masks stuck-at faults by emitting an output event after it has received the second input transition, without waiting for the third input transition which may never arrive. In a bundled data interface the majority gate's output event should occur when two of the bundles are valid, and corrected simplex data can be generated by majority vote. However, a dyschronous fault can cause a request transition to arrive early, before its bundle of data is valid. In this case the majority gate's output event occurs when only one bundle of data is valid and the simplex output cannot be generated properly.

Figure 52 shows a bundle-data triplex-to-simplex interface. Here the CMAJ element combines the three redundant request signals that indicate when the data bundles are valid. The simplex data bundle is produced by latching the majority vote result of the redundant data bundles at the time of the CMAJ element's output transition.

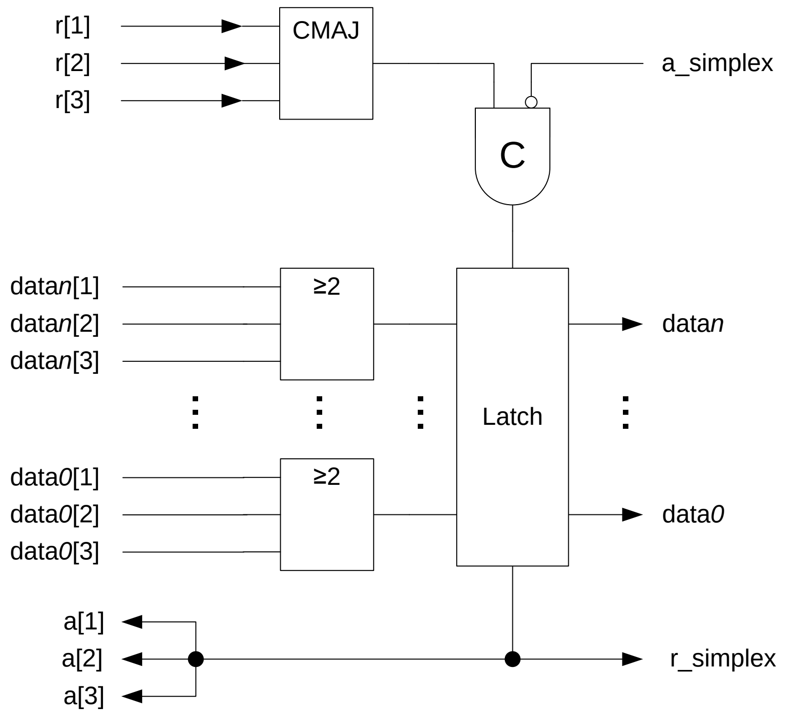


Figure 52. Triplex-to-simplex bundled data interface

Figure 53 shows the CMAJ element used in a simplex-to-triplex interface. The triplex request and data signals are produced by fanning out the corresponding simplex signals. Because the timing of data removal is bundled with the acknowledge signal a CMAJ element combines the triplex acknowledge signals to produce a single simplex acknowledgement.

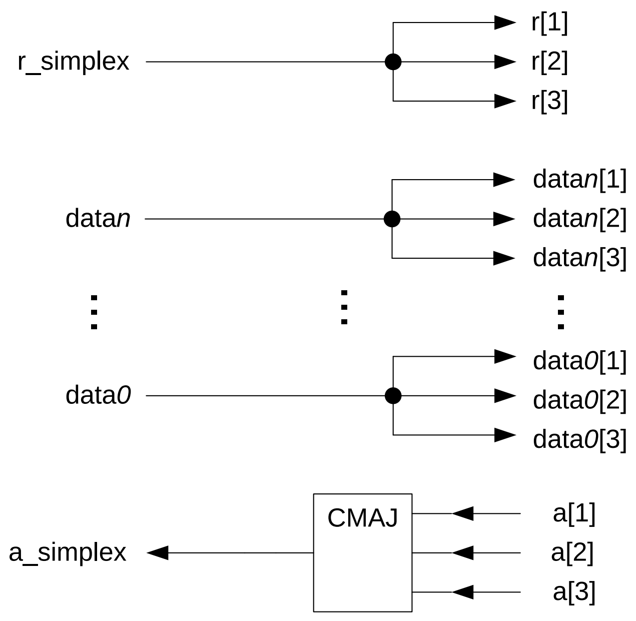


Figure 53. Simplex-to-triplex bundled data interface

4.7 Four-phase dual-rail simplex-triplex interface

The simplex-triplex interface for 4P2R pipelines is easily derived from the simplex-triplex dataless interface. For simplex-to-triplex, the two request signals are each fanned-out to form triplets, as shown in Figure 54. The simplex common acknowledge signal is generated from the acknowledge triplet using a majority voter, either combinational or HFMV. For triplex-to-simplex, the ‘0’ and ‘1’ request triplets feed majority voters, and the simplex common acknowledge signal is fanned out to form the triplex signal, as shown in Figure 55. Both figures show the interface circuits for a single bit. Multi-bit interface circuits can be constructed by replicating the single-bit circuit as needed.

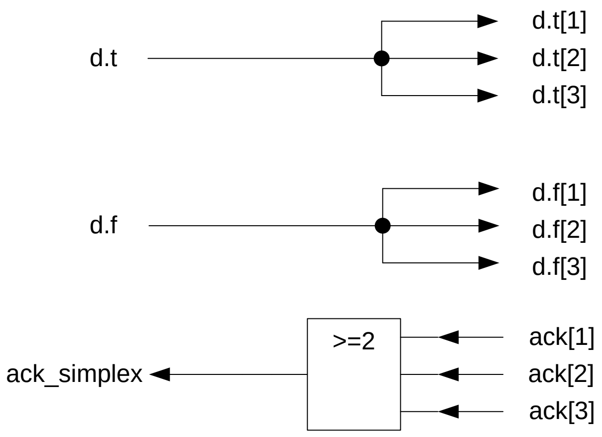


Figure 54. Four-phase dual-rail simplex-to-triplex interface

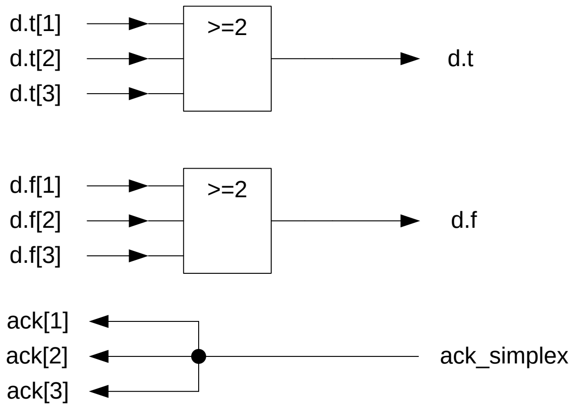


Figure 55. Four-phase dual-rail triplex-to-simplex interface

4.8 Triplex vs. Simplex Circuit Area

In order to compare the costs of simplex and triplex pipeline circuits the circuit area for each must be calculated. First, the dataless pipeline: Using the equivalent gate count from Table 15, each simplex dataless pipeline stage consists of a single C-element with one inverting input.

$$Simplex\ dataless\ stage\ equivalent\ gates = 2.5 \tag{17}$$

As shown in Figure 45, each triplex dataless pipeline stage consists of 3 C-elements with one inverting input, and 3 majority gates.

$$Triplex\ dataless\ stage\ (comb.voter)\ equivalent\ gates \\ = 3(2.5 + 3) = 16.5 \tag{18}$$

Thus the dataless triplex pipeline using combinational majority voter transistor area is 6.6 times that of the simplex pipeline. Alternately, one triplex dataless pipeline stage consists of 3 C-elements with one inverting input, and 3 HFMV elements of 50 transistors, or 12.5 equivalent gates, each.

$$Triplex\ dataless\ stage\ (HFMV)\ equivalent\ gates \\ = 3(2.5 + 12.5) = 45 \tag{19}$$

Thus the transistor area of the triplex dataless pipeline stage is 18 times that of the simplex pipeline. The relative circuit areas of the three dataless pipeline circuit alternatives are summarized in Table 15

Table 15. Dataless pipeline circuit relative area

Circuit Type	**Voter Type**	**Equivalent gates/stage**	**Normalized**
Simplex	N/A	2.5	1.0
Triplex	comb. Maj. gate	$3(2.5 + 3) = 16.5$	6.6
Triplex	HFMV	$3(2.5 + 12.5) = 45$	18

Each wire in the simplex circuit is replaced by three wires in the triplex circuit; and additional wiring is needed to implement the 3x3 cross-connection of the C-element outputs to the voter inputs. Thus the triplex wire area is somewhat more than three times that of the simplex circuit.

Next, the 4-phase dual-rail pipeline is considered. Using the equivalent gate count from Table 16, one simplex 4P2R pipeline stage consists of 2 C-elements with one inverting input and 1 OR gate for each bit transmitted.

$$Simplex\ 4P2R\ pipeline\ stage\ equivalent\ gates/bit \\ = 2 \times 2.5 + 1.5 = 6.5 \tag{20}$$

One triplex 4P2R pipeline stage consists of 6 C-elements with one inverting input, 6 majority gates, and 3 OR gates for each bit.

$$Triplex\ \ 4P2R\ pipeline\ stage\ (comb.voter) equivalent\ \frac{gates}{bit} = 3(2 \times 2.5 + 2 \times 3 + 1.5) = 37.5 \tag{21}$$

Thus the triplex 4P2R pipeline area is approximately 5.8 times the simplex pipeline area per bit. Alternately, one triplex 4P2R pipeline stage using HFMV elements rather than combinational majority gates has the following gate count.

$$Triplex\ \ 4P2R\ pipeline\ stage\ (HFMV) equivalent\ \frac{gates}{bit} = 3(2 \times 2.5 + 2 \times 12.5 + 1.5) = 94.5 \tag{22}$$

The relative circuit areas of the three 4P2R pipeline circuit alternatives are summarized in Table 16.

Table 16. 4P2R pipeline circuit relative area per bit

Circuit Type	Voter Type	Equivalent gates/stage	Normalized
Simplex	N/A	6.5	1.0
Triplex	comb. Maj. gate	37.5	5.8
Triplex	HFMV	94.5	14.5

4.9 Triplex vs. Simplex Circuit Power Consumption

Each of the three C-elements and majority voters within a redundant triplet has the same switching activity. The only difference, in a non-faulty system, will be that the gates in a triplet can have different switching times caused by the delay variation in the circuit's wires and gates. It can be reasonably assumed that that the dynamic switching energy of combinational majority gates and C-elements is approximately the same. Thus a non-faulty triplex pipeline circuit using combinational majority voters consumes approximately six times the power of the equivalent simplex circuit. A triplex pipeline circuit using HFMVs or CMAJ voters will consume even more power than a faulty triplex pipeline circuit using combinational majority voters because the both the HFMV and CMAJ are more complex than the combinational majority voters. The power consumption of a circuit with faults is difficult to estimate because the faults themselves may cause abnormal current flow.

4.10 Triplex Clockless System Building Blocks

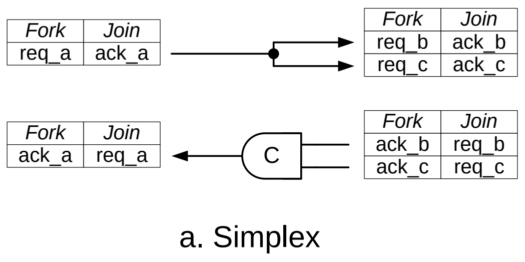
Building complex clockless systems requires a set of flow-control components in addition to the pipeline structures considered so far. The components include unconditional flow-control elements *FORK*, *JOIN* and *MERGE*, conditional flow-control elements *MUX* and *DEMUX*, and mutual-exclusion element *MUTEX* [44]. This section discusses the triplex implementation of these components and shows several example circuits.

The FORK, JOIN, MERGE, MUX and DEMUX elements are composed of C-elements, OR gates, and combinational multiplexers. The triplex form of each can be derived from the simplex by triplicating each wire and gate, then inserting a majority gate restoring stage at the outputs of each C-element triplet, as shown in Figure 28.

4.10.1 Triplex FORK/JOIN

In clockless systems, the FORK and JOIN components are used to handle parallel computations. FORK splits a process into two parallel streams. JOIN synchronizes and unites two parallel streams into a single process. The implementation of the FORK and JOIN elements are quite symmetrical: A FORK circuit connected in reverse implements a JOIN, and vice versa. As shown in Figure 5.1 of [44], the basic FORK-JOIN circuit is easily adapted to serve bundled and dual-rail signaling. For dual-rail signaling the request path of both FORK and JOIN elements is duplicated to form a request-0/request-1 pair. The FORK element for bundled data simply copies each data bit to both outputs. The JOIN element for bundled data brings the data from both inputs together; how the data are combined is not specified by the JOIN – the JOIN simply assures that both groups of data are ready for computation.

Figure 56 a. shows the basic simplex FORK-JOIN circuit. The triplex FORK-JOIN is composed by triplicating all elements of the simplex circuit and adding a majority gate restoring stage, as shown in Figure 56 b. Alternatively, inserting a triplex majority voter stage before splitting the requests/acknowledges in the top circuit in Figure 56 b would mask any single fault on the input at left and block propagating of the fault onto the outputs at right.



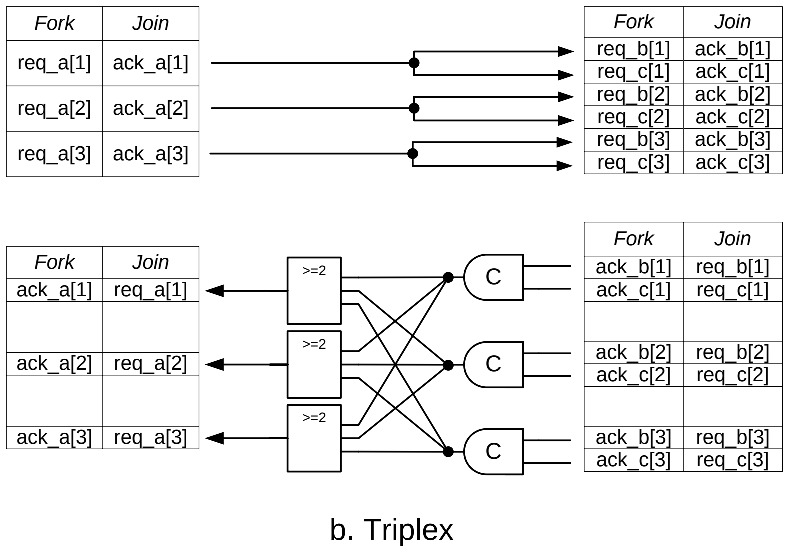


Figure 56. FORK-JOIN circuit

4.10.2 Triplex MERGE

The MERGE element implements the handshakes necessary to merge two mutually-exclusive data streams. Following the method outlined above, a triplex bundled-data MERGE shown in Figure 58 can be derived from the simplex bundled-data MERGE shown in Figure 57. The triplex dual-rail MERGE is not shown, but it can be derived from the simplex dual-rail MERGE shown in [44], Figure 5.1.

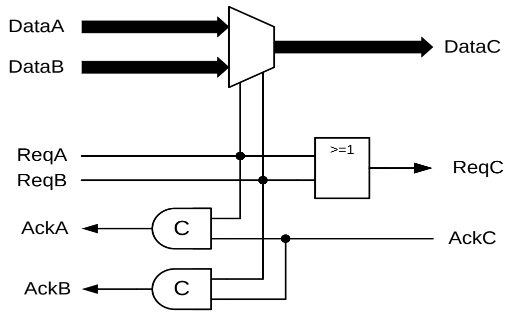


Figure 57. Simplex bundled-data MERGE element

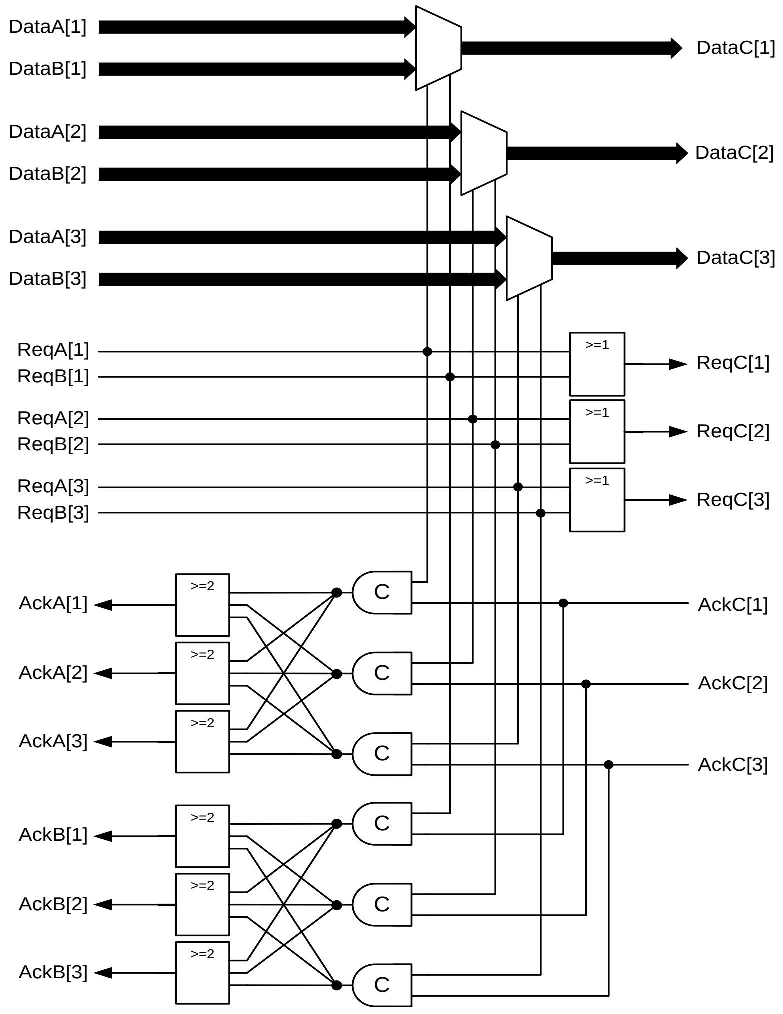


Figure 58. Triplex bundled-data MERGE element

4.10.3 Triplex MUX/DEMUX

The triplex MUX and DEMUX elements can be derived from the simplex elements shown in [44], Figure 5.19 using the method previously described.

4.10.4 Triplex MUTEX

Clockless circuit building blocks such as merge require that their inputs be mutually exclusive. So it is often necessary for a clockless system to arbitrate among concurrent asynchronous inputs. The mutual exclusion element, or MUTEX, serves this need. It receives two concurrent inputs, arbitrates between them, and provides two outputs that are mutually exclusive: no more than one of the two outputs can be true at a time. Figure 59 shows the MUTEX symbol and a possible implementation.

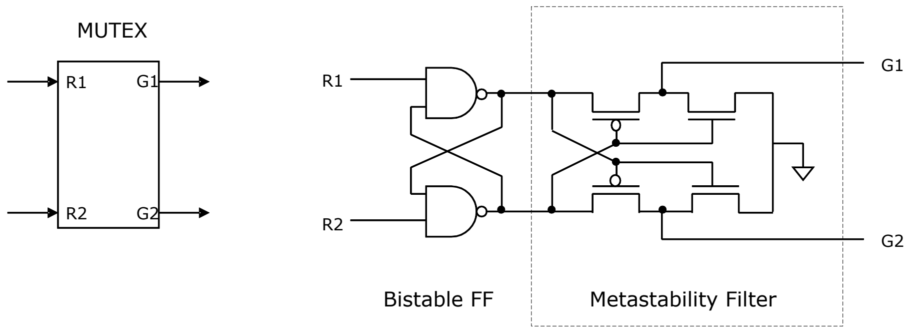


Figure 59. Mutual exclusion element (adapted from [44])

The triplex MUTEX is derived from the simplex in the same manner as fork, join, and merge: the outputs of three parallel MUTX elements feed two majority voter restoring stages, as shown in Figure 60. Skew between the three signals in each of the MUTEX's two input triplets can cause the three MUTEX outputs to disagree, even if all three MUTEX elements are fault-free. When this occurs, the voters pass the results of the majority (2 of 3) MUTEX outputs, and block the minority (1 of 3) result. Thus the voted outputs always concur.

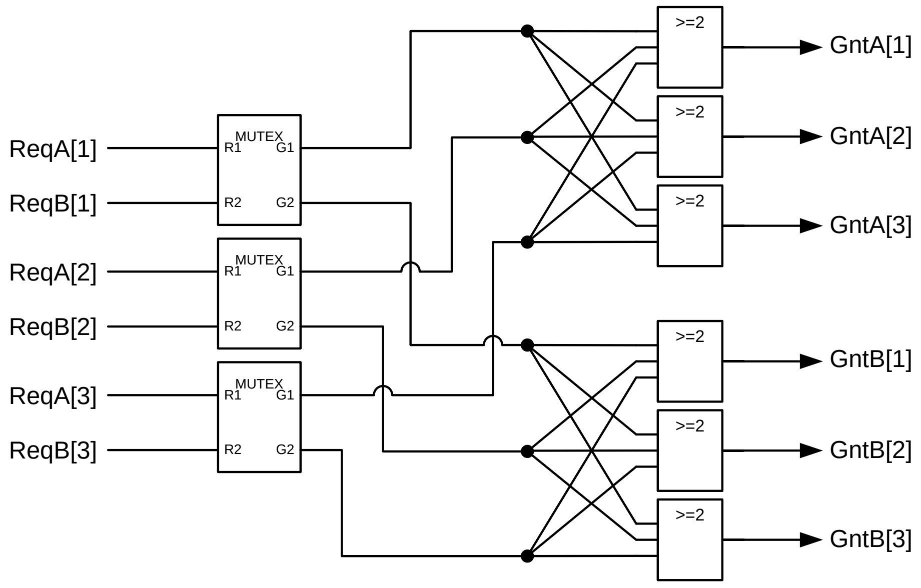


Figure 60. Triplex MUTEX

4.11 Triplex Clockless Architecture summary

This chapter presented a method for designing triple-modular redundant clockless pipeline circuits. Examples of two-phase bundled-data and four-phase dual-rail pipeline circuits were shown. Circuit area, performance, and power consumption for triplex and simplex configuration were compared: triplex pipelines with combinational majority gates were shown to require over 6 times the area of simplex pipelines, and triplex pipelines with hazard-free majority voters were shown to require 14 to 18 times the area of simplex pipelines A new circuit element, useful in triplex clockless circuits, the CMAJ that combines functionality of a C-element and a majority gate was presented. Finally, triplex versions of several clockless circuit building were introduced to show how to extend the triplex architecture beyond linear pipelines.

The next chapter describes a simulation method to approximately measure the actual fault tolerance of the clockless pipeline circuits introduced in this chapter. This simulation method will be used for both two-phase bundled-data and four-phase dual-rail pipeline circuits, and will compare the fault tolerance achieved using either combinational or hazard-free voters.

5
Stochastic Fault Simulation Method and Results

This chapter presents estimates of the reliabilities of functionally equivalent simplex and triplex clockless circuits derived from Monte Carlo simulation with random injection of stuck-at and dyschronous faults. Two representative clockless pipeline circuit models have been simulated: Sutherland's two-phase bundled-data "micropipeline" and a four-phase dual-rail (4P2R) pipeline. These two examples have been chosen because of their familiarity to designers of clockless circuits, and because they cover both two-phase and four-phase signaling, and both bundled and dual-rail data encoding. Simplex and triplex versions of both the micropipeline and 4P2R pipeline circuits have been modeled using the Verilog hardware description language (HDL) and simulated using the *ModelSim* logic simulator. The logic simulation results have been recorded in ASCII text files, analyzed, and plotted using *MATLAB.*

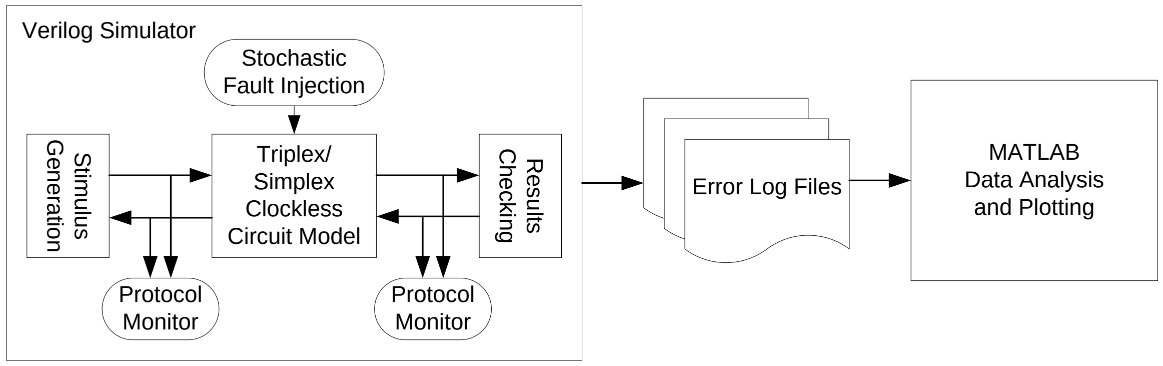

Verilog Simulator
Stochastic Fault Injection
Stimulus Generation
Triplex/ Simplex Clockless Circuit Model
Results Checking
Protocol Monitor
Protocol Monitor
Error Log Files
MATLAB Data Analysis and Plotting

Figure 61. Stochastic fault simulation environment

Figure 61 illustrates the simulation environment. The modules shown within the Verilog portion of the simulation environment are described in Section 5.1 .

5.1 Verilog HDL Circuit Simulation

The Verilog HDL circuit simulation includes the following modules:

1. A model of the pipeline circuit under study. Four different pipeline circuit models were used: simplex micropipeline, triplex micropipeline, simplex four-phase dual-rail (4P2R), and triplex 4P2R. The micropipeline models included an 8-bit-wide bundled data path. The 4P2R models were one data bit in width. The simplex and triplex pipeline circuit Verilog models are *structural* in that they describe the interconnection between instances of the primitive cells that compose the circuit: C-elements, majority gates, OR gates, and multiplexers. These primitive cells were modeled *behaviorally*, by defining their input/output function. Transistor-level Verilog modeling was not used
2. A *stimulus generation* module generated a sequence of inputs to the pipeline model using either 2-phase bundled or four-phase dual-rail handshake signaling as needed. The stimulus generator used handshake signals with random arrival times, and produced a random sequence of data in both cases.
3. A *results checker* that compared the actual data value received from the pipeline model output against the expected data value. The expected data value was the data value sent into the pipeline by the stimulus generator. Like the stimulus generator, the results checker used handshake signals with random arrival times. Thus the checking process was independent of delays within the pipeline. The operation of the stimulus generator and the results checker were coordinated in a manner that assured that each logic simulation run filled the pipeline circuit to its capacity then emptied it, thereby covering both the pipeline-full and pipeline empty conditions.
4. *Protocol monitors* that checked the pipeline model's input and output handshake signals for deadlock and sequence violations such as an acknowledge indication without a prior request.

5.1.1 Dynamic Random Delay Modeling

Each primitive cell model within each stage of the pipeline circuit includes a dynamic random delay to simulate the effects of circuit delay variation. Each transition emitted from a cell output is delayed by a random time from the causal input transition. The same cell instance can have different delays at different times. All random cell delays have a uniform distribution within a range limited by an arbitrary upper bound. This is *not* a realistic distribution model for actual gate and wire delays. The uniform distribution was chosen to stress the Verilog circuit model in order to better verify its functional correctness over a wide range of timing variations.

5.1.2 Stochastic Fault Injection

Each primitive cell model included a stochastic fault injection feature. At the beginning of each logic simulation run, each cell generated a random number that determines whether it will function normally (nonfaulty) or implement one of the following stuck-at or dyschronous faults: stuck-at-0, stuck-at-1, early-rise, early-fall, late-rise or late-fall. The injected faults were independent and identically distributed, and were static in that once they are injected their behavior does not change during the logic simulation run. After each logic simulation run was completed, all faults were removed before injecting new faults for the next run. This random fault determination was guided by two independent parameters: the stuck-at fault probability, P_{SA}, and the dyschronous fault probability, P_{Dys}. In order to collect system reliability data over a range of gate fault probabilities, the Verilog testbench iterated logarithmically through 28 values each of P_{SA} and P_{Dys}, ranging from 10^{-4} to $1.6x10^{-1}$. This range of fault probabilities was sufficient to provide many examples of a single fault in the circuit being analyzed, which TMR is expected to correct. It also generated a number of complex multiple-fault cases.

The simulated range of fault probabilities is well above current semiconductor industry manufacting defect rates. But as CMOS IC technology moves to 22nm and below, transistor variability will increase dramatically and may lead to many of the fault and delay effects simulated here. Furthermore, the simulated range of fault probabilities is possibly more realistic for future nano-device manufacting defect rates.

10,000 simulation runs at each P_{SA} , P_{Dys} value pair yielded reasonable total run times. Thus a total of 28 x 28 x 10,000, or 7,840,000 logic simulations were run for each simplex and triplex pipeline model. Each of the 3-D plots in Sections 5.2 and 5.3 visualizes the data generated by 15,680,000 simulation runs.

Each logic simulation run exercises the fault-injected pipeline model by asynchronously pushing data into and popping data out of the pipeline. Each run stresses the pipeline by covering both pipeline full and empty conditions. Each run was scored as a “fail” if the micropipeline deadlocked, violated request-acknowledge ordering at its input or output, or showed any mismatch between its input and output data. Otherwise the run scored as a “pass,” resulting in successful pipeline operation. The pipeline error rate shown in the plots in Section 5.2 and 5.3 is computed by dividing the number of failing runs by the total number of runs.

The nature and distribution of defects in future digital circuit implementation technologies is not known, therefore the random fault injection uses the following simple assumptions about fault distribution:

- Faults are uniformly distributed throughout the circuit.
- Stuck-at-1 and stuck-at-0 are equally likely in all gates.
- Early-rise and early-fall are equally likely in C-elements.
- Early-rise, early-fall, late-rise and late fall are equally likely in majority gates.
- The relative probabilities of faults in C-elements, majority gates and OR gates are proportional to the area of each gate as measured by the number of transistors in each gate. One equivalent gate is four transistors. The equivalent gate count for each primitive is obtained by dividing its transistor count by 4.

These assumptions yield the relative fault probabilities listed in Table 17. The entries in the six bottom rows indicate the relative probabilities used for fault injection for each cell instance as a function of global fault probability iterators, P_{SA} and P_{Dys}.

Table 17. Random fault injection relative probabilities

Cell	C-Element with one inverting input	Majority Gate	OR Gate	Hazard-free majority voter (HFMV)
number of transistors	10	12	6	50
equivalent gates	2.5	3	1.5	12.5
$P_{Stuck\text{-}at\text{-}1}$	1.25 P_{SA}	1.5 P_{SA}	0.75 P_{SA}	6.25 P_{SA}
$P_{Stuck\text{-}at\text{-}0}$	1.25 P_{SA}	1.5 P_{SA}	0.75 P_{SA}	6.25 P_{SA}
$P_{Early\ Rise}$	1.25 P_{Dys}	0.75 P_{Dys}	0	3.125 P_{Dys}
$P_{Late\text{-}Rise}$	1.25 P_{Dys}	0.75 P_{Dys}	0	3.125 P_{Dys}
$P_{Late\text{-}Fall}$	0	0.75 P_{Dys}	0	3.125 P_{Dys}
$P_{Early\text{-}Fall}$	0	0.75 P_{Dys}	0	3.125 P_{Dys}

Figure 62 shows a flowchart of the Verilog stochastic fault simulation method. Simulation results were recorded in a text file and analyzed post-simulation using MATLAB. The simulation results are plotted to show what fraction of the simulation runs passed for a given gate fault probability, and to show what fraction of the simulation runs passed for a given number of gate faults per 10-stage pipeline, for both simplex and triplex circuits. Stochastic fault Simulations were run on 2-phase bundled data pipelines (micropipelines) and 4-phase dual-rail pipelines. Each of these two pipeline styles was simulated with combinational majority gate voters and hazard-free majority voters.

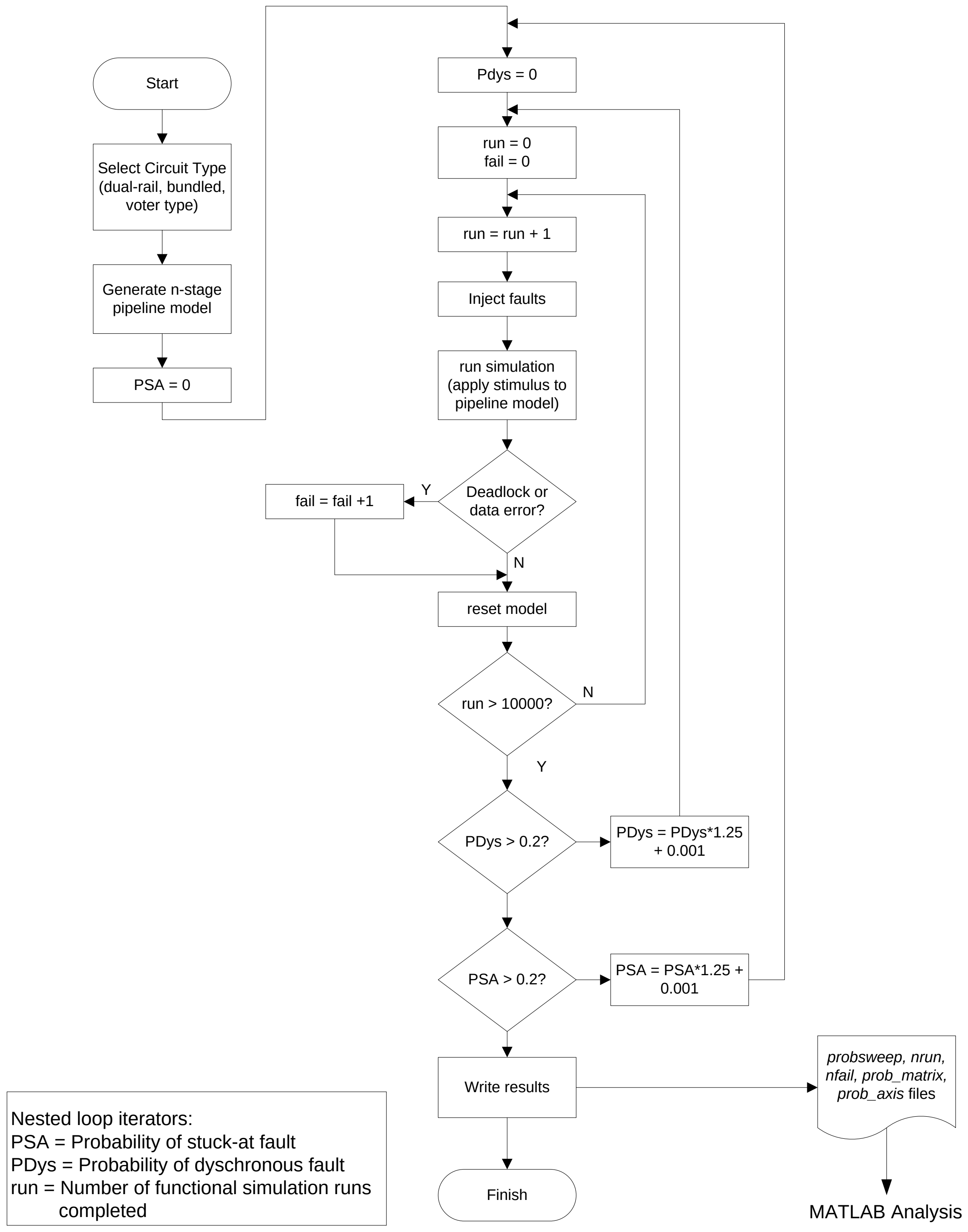

Start
Select Circuit Type (dual-rail, bundled, voter type)
Generate n-stage pipeline model
PSA = 0
Pdys = 0
run = 0
fail = 0
run = run + 1
Inject faults
run simulation (apply stimulus to pipeline model)
Deadlock or data error?
Y
fail = fail +1
N
reset model
run > 10000?
N
Y
PDys > 0.2?
PDys = PDys*1.25 + 0.001
PSA > 0.2?
PSA = PSA*1.25 + 0.001
Write results
probsweep, nrun, nfail, prob_matrix, prob_axis files
MATLAB Analysis
Finish
Nested loop iterators:
PSA = Probability of stuck-at fault
PDys = Probability of dyschronous fault
run = Number of functional simulation runs completed

Figure 62. Stochastic fault simulation flowchart

5.2 Micropipeline Simulation Results

Figure 63 plots the measured pipeline error rates of 10-stage simplex and triplex micropipelines, with combinational and hazard-free voters, for dyschronous faults only (stuck-at fault probability = 0), stuck-at faults only (dyschronous fault probability = 0).

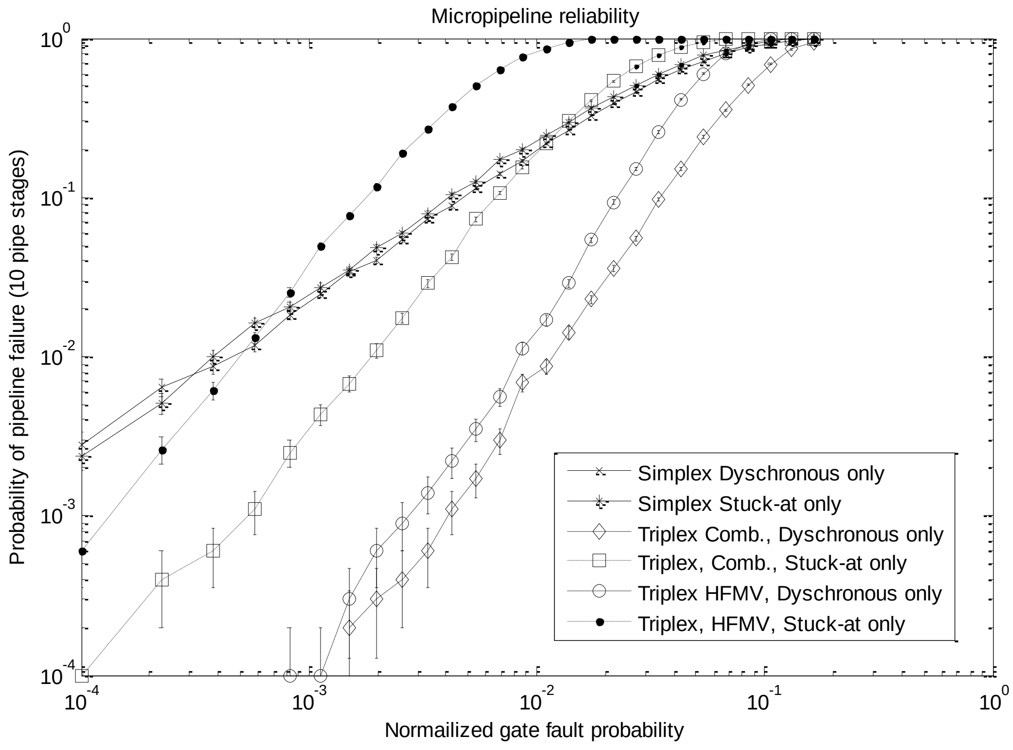


Figure 63. Micropipeline error rate vs. gate fault probability

Each data point represents the fraction of the 10,000 pipeline logic simulation runs that failed against the normalized gate fault probability used for random fault injection for those runs. The error bars indicate plus and minus one standard deviation

$$\sigma_{\bar{X}} \cong \frac{\sigma}{\sqrt{n}}$$

Where $\sigma = \sqrt{\bar{X}(1-\bar{X})}$ is the standard deviation of the Bernoulli distribution, $\bar{X}$ is the sample mean; and n = 10,000 is the number of simulations in the statistical ensemble.

The plotted results for both simplex pipelines confirm the expectation that the simplex pipeline failure probability depends on the gate fault probability according to the following equation:

$$P\left(PipeFail\right)=1-\left[1-P\left(GateFail\right)\right]^{10} \qquad (23)$$

This is expected because in the 10-stage simplex micropipeline, each stage consists of one C-element, and the stage fails if the C-element is faulty. The pipeline fails if one or more stages are faulty. MATLAB curve fitting confirms that the plotted data correlates well with Equation 23. MATLAB fit the data to the general model

$$f(x) = 1 - (1 - ax)^b \qquad (24)$$

where *x* is the normalized gate fault probability, and *f*(*x*) is the pipeline failure probability. Coefficients *a* and *b* for both stuck-at only and dyschronous only data, as well as the sum of squares due to error (SSE) and the coefficient of multiple determination (R-square), are listed in Table 18. The fitted values of coefficient *a* correlates well with the factor 2.5 (listed in Table 17) that multiples the normalized gate fault probability to yield the C-element fault probability. The fitted values of coefficient *b* correlate well with the expected value, 10, the number of stages in the simulated micropipeline model.

Table 18. Simplex micropipeline curve fit results

Data set	Coefficients (with 95% confidence bounds)		SSE	R-Square
	a	*b*		
stuck-at only	2.582 (2.093, 3.07)	9.722 (7.783, 11.66)	0.0003548	0.9999
dyschronous only	2.195 (1.864, 2.526)	9.993 (8.409, 11.58)	0.0002099	0.9999

The triplex pipeline failure rate for both voter types is lower than the simplex failure rate for sufficiently low gate fault probabilities. However, at higher gate fault probabilities the triplex circuit has a higher pipeline failure rate than the simplex because, , it is larger and therefore has more “fault targets,” even though the triplex circuit can mask one fault in each pipeline stage. This effect is more significant with hazard-free majority voters than with combinational voters because the HFMV is larger than the combinational voter and thus presents a bigger fault target. For stuck-at faults, the triplex pipeline using HFMVs has a higher failure rate than one using combinational voters for any gate fault probability. These results are roughly consistent with the calculated triplex/simplex pipeline stage failure rates discussed in Chapter 1, and plotted in Figure 5. The measured system simulation results differ somewhat from the calculated stage results, due to miscellaneous circuit details and the different voter reliabilities, as shown in Table 17.

MATLAB curve fitting of the triplex results yielded good fits to the general model

$$f(x) = 1 - (1 - x^a)^b \tag{25}$$

where *x* is the normalized gate fault probability, and *f*(*x*) is the pipeline failure probability. Coefficients *a* and *b* for both stuck-at-only and dyschronous-only data, and for both voter types, as well as the sum of squares due to error (SSE) and the coefficient of multiple determination (R-square), are listed in Table 19. The fitted values of coefficient *a* vary only slightly with voter type, depending more strongly on fault type. The larger values of *a* for dyschronous faults may reflect the result that dyschronous faults are less potent than stuck-at faults in causing system failure. The fitted values of coefficient *b* depend strongly on fault type. The larger values of *b* for the HFMV than the combinational voter suggest that *b* is a function of the number of gates in the model. But beyond these observations it is not clear why the fitted values of *b* are so large.

Table 19. Triplex micropipeline curve fit results

Voter type	Data set	Coefficients (with 95% confidence bounds)		SSE	R-Square
		a	*b*		
Comb.	S-A only	1.61 (1.585, 1.635)	354.6 (320.2, 388.9)	0.0006393	0.9999
	Dys. only	2.18 (2.169, 2.192)	156.6 (152, 161.1)	5.624e-005	1.0000
HFMV	S-A only	1.578 (1.546, 1.611)	2573 (2143, 3003)	0.001038	0.9998
	Dys. only	2.458 (2.426, 2.49)	1194 (1078, 1309)	0.000303	0.9999

In order to show the pipeline reliability improvement offered by the triplex architecture, Figure 64 plots the ratio of triplex/simplex pipeline error rate vs. gate error probability for combinational majority voters in a two-phase bundled data pipeline (micropipeline). The three curves plotted represent dyschronous faults only (stuck-at fault probability = 0), stuck-at faults only (dyschronous fault probability = 0), and a “50/50” curve where stuck-at gate fault equals dyschronous gate fault probability.

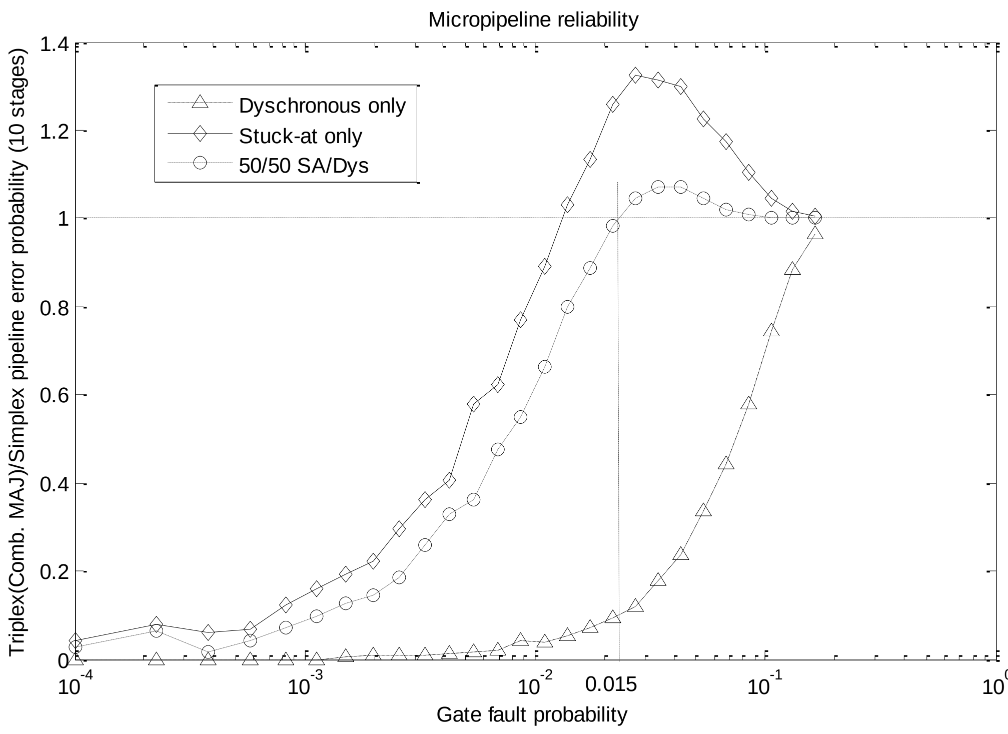


Figure 64. Triplex (comb. voter) / simplex micropipeline error rate vs. gate fault probability

The plots show that stuck-at faults are the dominant cause of pipeline errors unless dyschronous gate fault probability is approximately an order of magnitude greater than stuck-at fault gate fault probability. This result is expected because, as discussed in Chapter 3, whether dyschronous faults cause errors or not depends on the timing of input signals to the faulty C-elements or majority voters. The plot also shows that triplex does not reduce error rate for gate stuck-at fault probability > 0.015. Below 0.015, triplex circuits are significantly more reliable than their simplex equivalents. Above 0.015 additional fault targets in the voters outweigh triplex fault masking ability, making simplex more reliable. This simplex reliability benefit peaks at stuck-at fault probability ≈ 0.03. At higher stuck-at fault probabilities, both triplex and simplex are overwhelmed by high fault counts and the ratio of their reliabilities approaches 1.0.

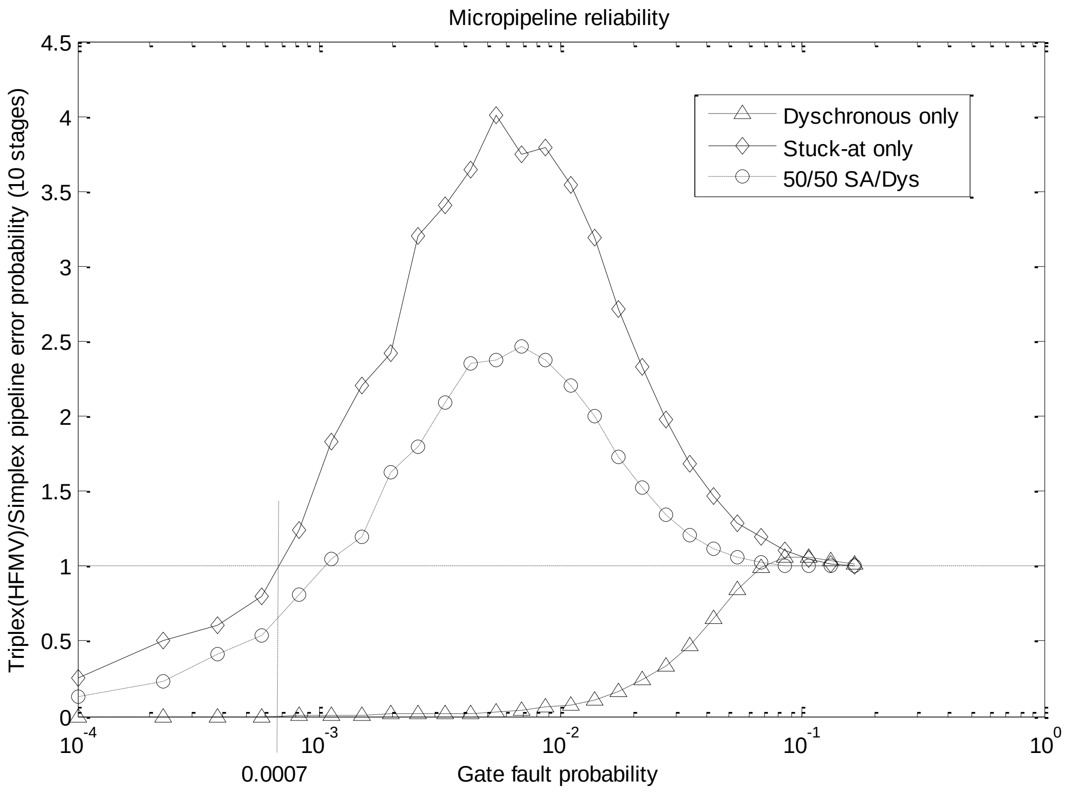


Figure 65. Triplex (HFMV) / simplex micropipeline error rate vs. gate fault probability

Figure 65 shows the same results plotted for a pipeline with hazard-free rather than combinational voters. The effect of the larger, and therefore less reliable voter, is apparent in the plot. Here, the triplex micropipeline has a higher error rate for gate stuck-at fault probability > 0.0007; and for gate stuck-at fault probabilities around 0.005 the triplex circuit is four times less reliable than the simplex. These results support the conclusion of Section 2.7, derived from Equation 12, that using HFMVs rather than combinational voters is not justified unless the SET rate, switching rate and skew are all high.

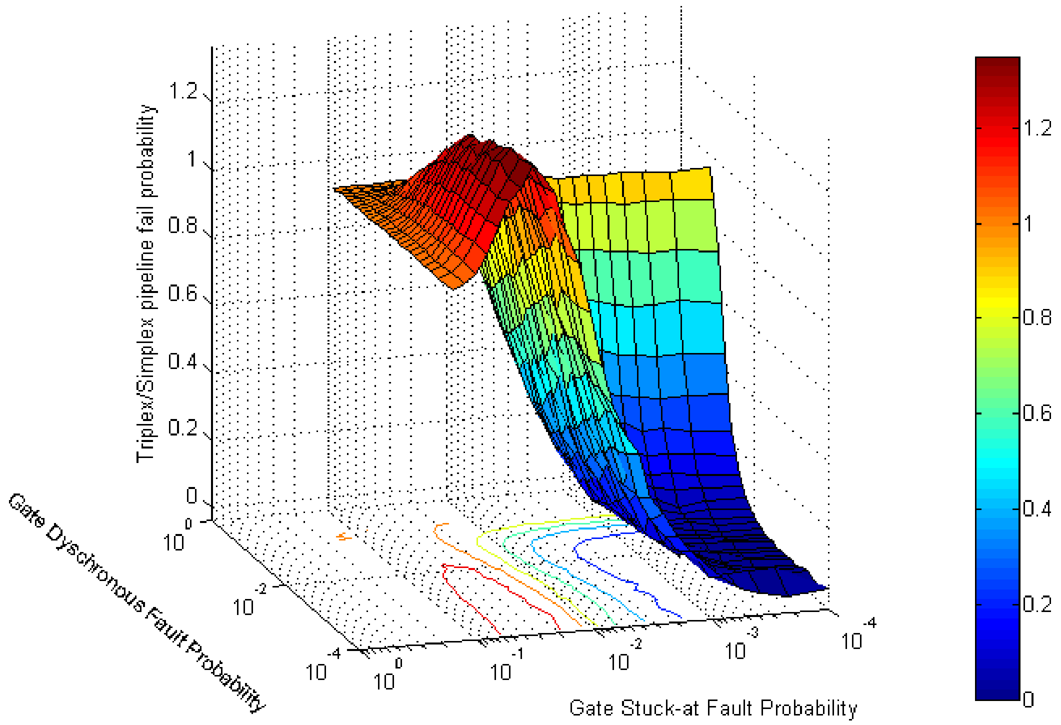


Figure 66. Triplex (comb. voter) / simplex micropipeline error rate vs. stuck-at and dyschronous gate fault probability

Figure 66 shows a 3-D plot of the ratio of triplex (combinational voter) /simplex pipeline error rate as a function of stuck-at gate fault probability and dyschronous gate fault probability. The smaller the ratio (z value), the more triplex architecture improves system reliability over the simplex architecture. The contours projected onto the z = 0 plane show the regions of fault probability where triplex improves reliability and the region (z > 1) where triplex actually reduces reliability. If an implementation technology's gate fault probabilities lie within this region then a triplex system will be both less reliable and more costly than an equivalent simplex system.

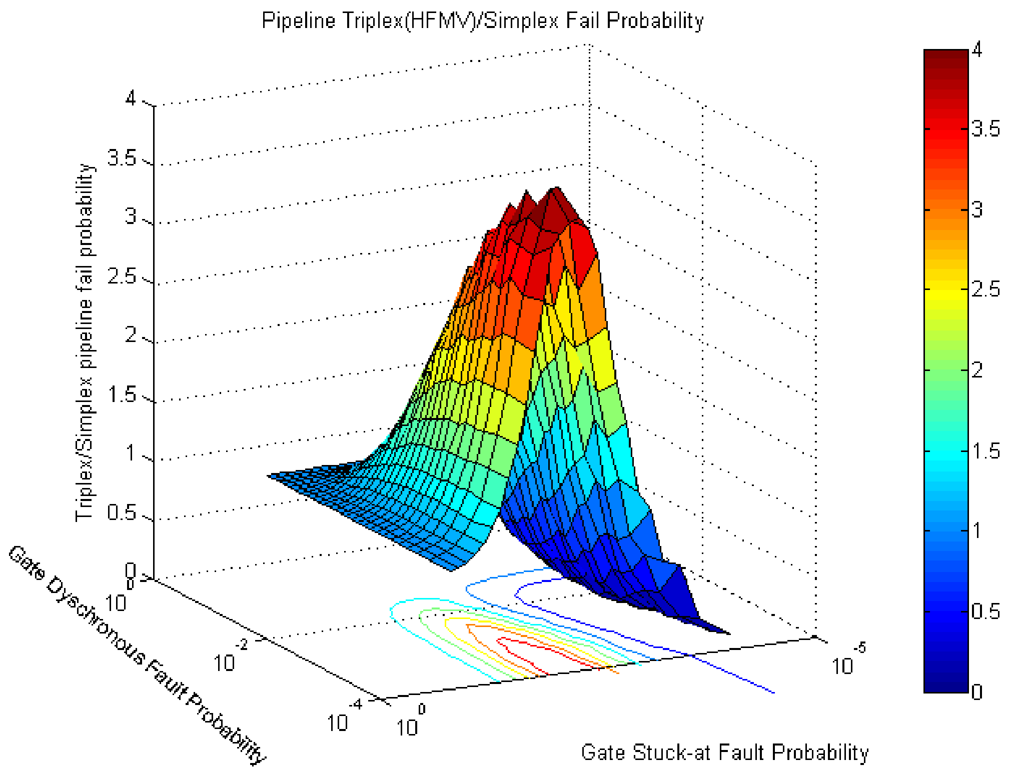


Figure 67. Triplex (HFMV) / simplex micropipeline error rate vs. stuck-at and dyschronous gate fault probability

Figure 67 shows the same results plotted for a pipeline with hazard-free rather than combinational voters. Here, the region where triplex reduces pipeline reliability is in approximately the same location on the gate fault probability (X-Y) plane, but it has a larger footprint and greater *z* magnitude.

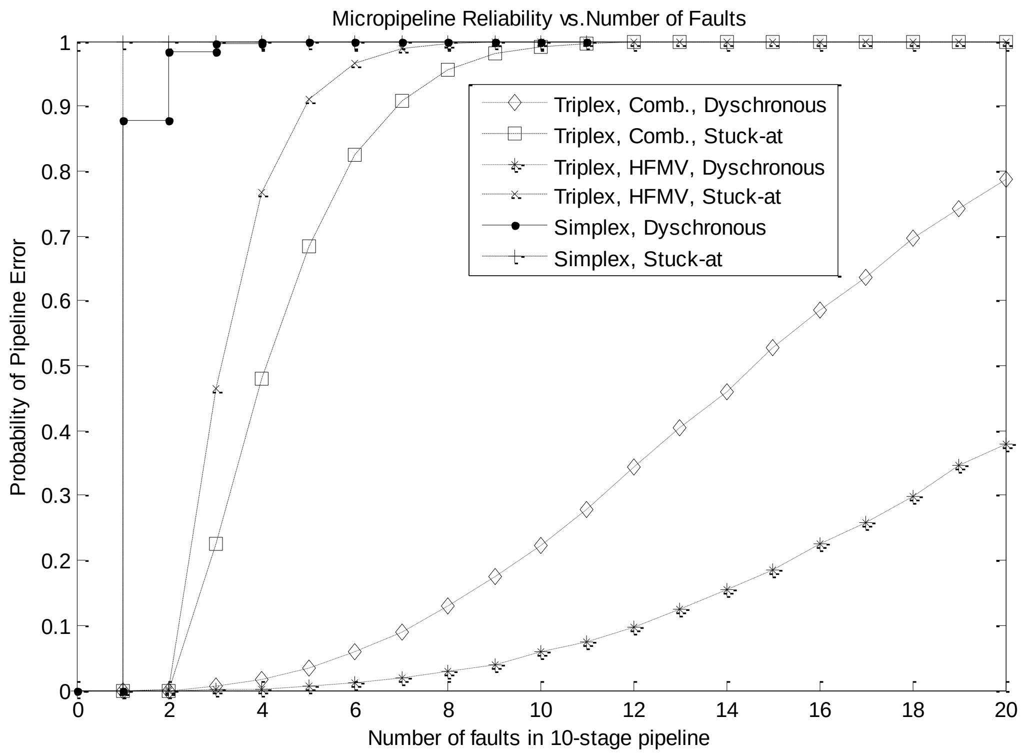


Figure 68. Micropipeline error rate vs. number of faults

Figure 68 plots triplex and simplex micropipeline error rate vs. total number of gate faults in the 10-stage pipeline for both combinational and hazard-free voters. The markers show the discrete data points. Error bars are not shown because the standard error magnitude is smaller than the marker height. Note that the simplex pipeline error rate is 1.0 for any number of stuck-at faults greater than zero. In other words, a simplex micropipeline circuit with any number of stages will fail (deadlock) if there is even one stuck-at fault anywhere in the pipeline circuit control path through the C-elements (A stuck-at fault in the data path will cause data corruption, but not deadlock). Also note that the triplex pipeline error rate is zero for zero or one faults. Triplex majority voting always corrects any single fault in the system. The pipeline error rate for two or more stuck-at faults is higher for hazard-free than for combinational voters. This result is expected because the HFMV present a larger fault target. The rate of pipeline errors caused by dyschronous faults is lower for hazard-free than for combinational voters. This result may seem counterintuitive and therefore requires some explanation.

In a pipeline that uses hazard-free, rather than combinational, voters a greater proportion of the total number of faults in the system will be in voters, not the C-elements. This follows from the fact that the hazard-free voter is larger, and therefore more likely to contain faults than the combinational voter. The fault simulation assumes each C-element has 10 transistors, each combinational majority voter has 12 transistors, and each HFMV has 50 transistors. Gate fault probabilities are assumed to be proportional to the number of transistors in each gate. Table 20 compares the number of transistors per pipeline stage and shows that the gate fault probability of the C-element and combinational voter are about equal, thus in pipelines that use combinational voters about half of the faults are in voters and about half are in C-elements. But in pipelines that use HFMVs, only one sixth of the faults are in C-elements and the remaining five sixths are in voters.

Table 20. Number of Transistors per Pipeline stage for Hazard-Free and Combinational Voters

	Number of transistors in			**Fraction of transistor in voters**
	3 x C-Element	**3 x Voter**	**pipeline stage (half buffer)**	
Comb. MAJ	30	36	30 + 36 = 66	36/66 = 0.55
HFMV	30	150	30 + 150 = 180	150/180 = 0.83

The horizontal axis in Figure 68 plots the total number of faults in the pipeline. For a given number of faults, the majority of faults in and HFMV pipeline will be in voters, rather than C-elements. For example, in a pipeline with six total faults, for the HFMV case, it is likely that, five of the faults will be in voters and one in a C-element; and for the combinational voter case, it is likely that three faults will be in voters and three faults in C-elements.

Stuck-at faults in voters are *more* likely to cause pipeline errors than stuck-at faults in C-elements because any single fault in a triplet of C-elements is masked by the voters, but any single fault in a voter triplet causes an error that propagates unmasked to the next pipe stage. Dyschronous faults in voters are *less* likely to cause pipeline errors than dyschronous faults in C-elements because, as explained in Section 3.9, a late-transition dyschronous fault in a voter will cause an error only if the voter’s third input never arrives, because of an upstream stuck-at fault and an early-transition dyschronous fault in a voter will cause and error only if the voter's first input arrives early because of an upstream early-transition dyschronous fault. Without these prerequisite upstream faults, the early- and late-transition dyschronous voter faults are indistinguishable from the voter having reduced or increased propagation delay, respectively.

5.3 Four-phase dual-rail stochastic fault simulation results

Figure 69 plots the error rates of simplex and triplex four-phase dual-rail (4P2R) pipelines for dyschronous faults only (stuck-at fault probability = 0), stuck-at faults only (dyschronous fault probability = 0) in the manner that Figure 63 plots these results for two-phase bundled-data pipelines.

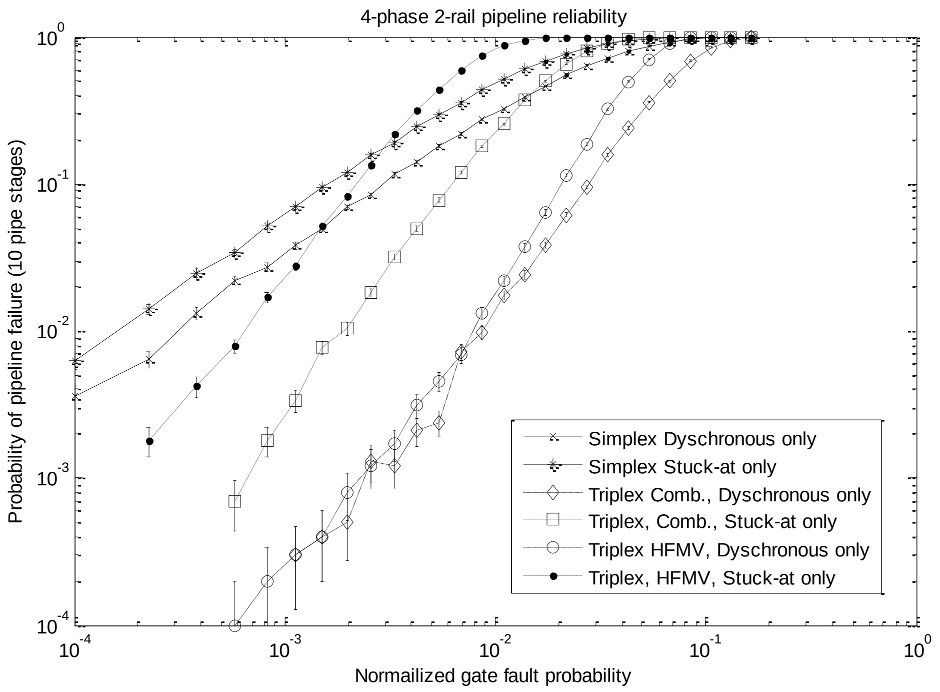


Figure 69. Four-phase dual-rail pipeline error rate vs. gate fault probability

As in Figure 63, each data point plots the fraction of the 10,000 pipeline logic simulation runs that failed against the normalized gate fault probability used for random fault injection for those runs. The error bars indicate plus and minus one standard error deviation

$$\sigma_{\bar{X}} \cong \frac{\sigma}{\sqrt{n}}$$

Where $\sigma = \sqrt{\bar{X}(1-\bar{X})}$ is the standard deviation of the Bernoulli distribution, $\bar{X}$ is the sample mean; and *n* = 10,000 is the number of simulations in the statistical ensemble.

MATLAB fit the simplex 4P2R data to the same general model to which it fit the simplex micropipeline data,

$$f(x) = 1 - (1 - ax)^b$$

where *x* is the normalized gate fault probability, and *f*(*x*) is the pipeline failure probability. Coefficients a and b for both stuck-at only and dyschronous only data, as well as the sum of squares due to error (SSE) and the coefficient of multiple determination (R-square), are listed in Table 21. The fitted values of coefficient a is within 10% of the fitted a values for the simplex micropipeline results, similarly correlating with the factor of 2.5, from Table 17, that multiples the normalized gate fault probability to yield the C-element fault probability. The fitted values of coefficient b correlate roughly with the number of pipeline stages (10) in the simulated model multiplied by a factor related to the number of gates per pipeline stage. Using numbers from Equations 17 and 20 in Section 4.8, a 4P2R simplex pipeline stage has 6.5/2.5 = 2.6 times the number of gates as a simplex micropipeline stage. The fitted values of coefficient b are within 33% of the above factor (2.6) times the number of pipeline stages (10).

Table 21. Simplex 4P2R pipeline curve fit results

Data set	Coefficients (with 95% confidence bounds)		SSE	R-Square
	a	b		
stuck-at only	2.254 (1.348, 3.159)	28.88 (17.06, 40.69)	0.0001656	1.0000
dyschronous only	2.061 (1.404, 2.719)	17.34 (11.64, 23.03)	0.0003014	0.9999

MATLAB curve fitting of the 4P2R triplex results yielded good fits to the same general model as the triplex micropipeline results

$$f(x) = 1 - (1 - x^a)^b$$

where *x* is the normalized gate fault probability, and *f*(*x*) is the pipeline failure probability. Coefficients *a* and *b* for both stuck-at only and dyschronous only data, and for both voter types, as well as the sum of squares due to error (SSE) and the coefficient of multiple determination (R-square), are listed in Table 22. As with the triplex micropipeline curve fit results (Equation 25 and Table 19), the fitted values of coefficient *a* vary only slightly with voter type, and more strongly with fault type, and the fitted values of coefficient *b* depend strongly on fault type. The larger values of *b* for the HFMV than the combinational voter suggest that *b* is a function of the number of gates in the model. But beyond these observations it is not clear why the fitted values of *b* are so much larger than for the simplex case.

Table 22. Triplex 4P2R pipeline curve fit results

Voter type	Data set	Coefficients (with 95% confidence bounds)		SSE	R-Square
		a	*b*		
Comb.	S-A only	1.828 (1.808, 1.848)	1152 (1060, 1245)	0.0002762	0.9999
	Dys. only	2.143 (2.124, 2.162)	230.8 (219.2, 242.5)	0.0001534	0.9999
HFMV	S-A only	1.816 (1.794, 1.838)	7625 (6778, 8473)	0.0003176	0.9999
	Dys. only	2.587 (2.565, 2.609)	2361 (2199, 2522)	0.0001215	1.0000

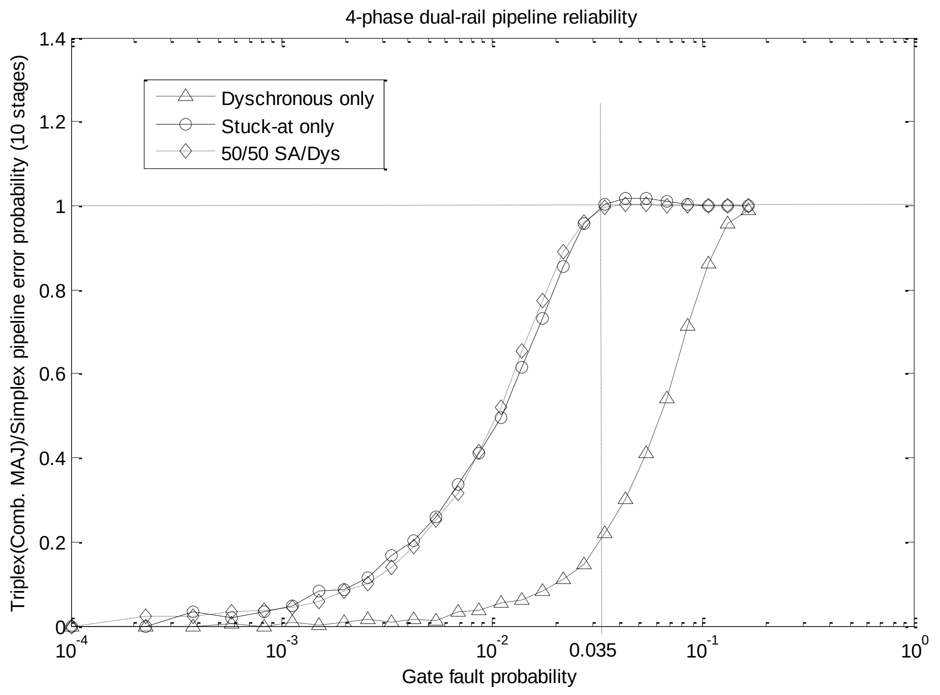


Figure 70. Four-phase dual-rail triplex (comb. MAJ) / simplex pipeline error rate vs. gate fault probability

Figure 70 plots the ratio of triplex (combinational voter) /simplex 4P2R pipeline error rate vs. gate error probability. The three curves plotted represent dyschronous faults only (stuck-at fault probability = 0), stuck-at faults only (dyschronous fault probability = 0), and a “50/50” curve where the stuck-at gate fault probability equals the dyschronous gate fault probability. As in Figure 64, the plots show that stuck-at faults are the dominant cause of pipeline failure. The plot shows that the use of the triplex technique increases error rate for gate stuck-at fault probability > 0.035.

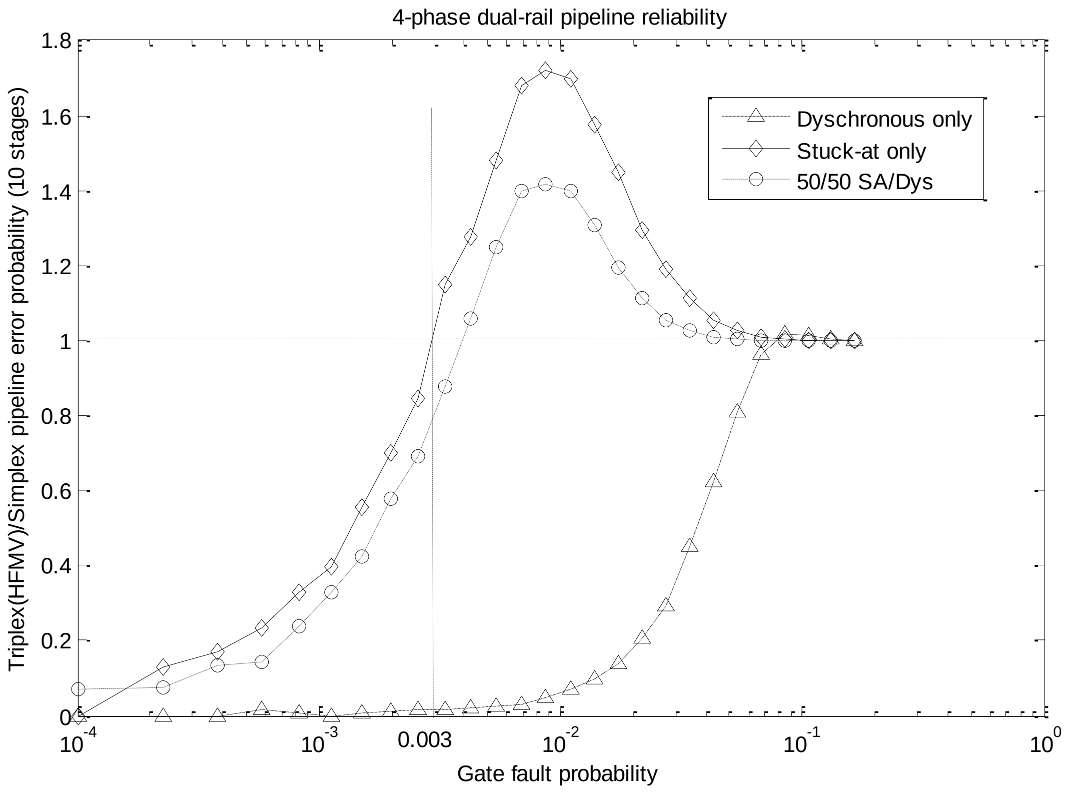


Figure 71. Four-phase dual-rail triplex (HFMV) / simplex pipeline error rate vs. gate fault probability

Figure 71 shows a similar plot of the ratio of triplex/simplex 4P2R pipeline error rate vs. gate error probability with a “50/50” line for a triplex pipeline using hazard-free rather than combinational voters. Here the results show the effect of the larger (which is more vulnerable to faults) voter; the triplex pipeline is significantly less reliable than the simplex for gate stuck-at fault probabilities greater than 0.003.

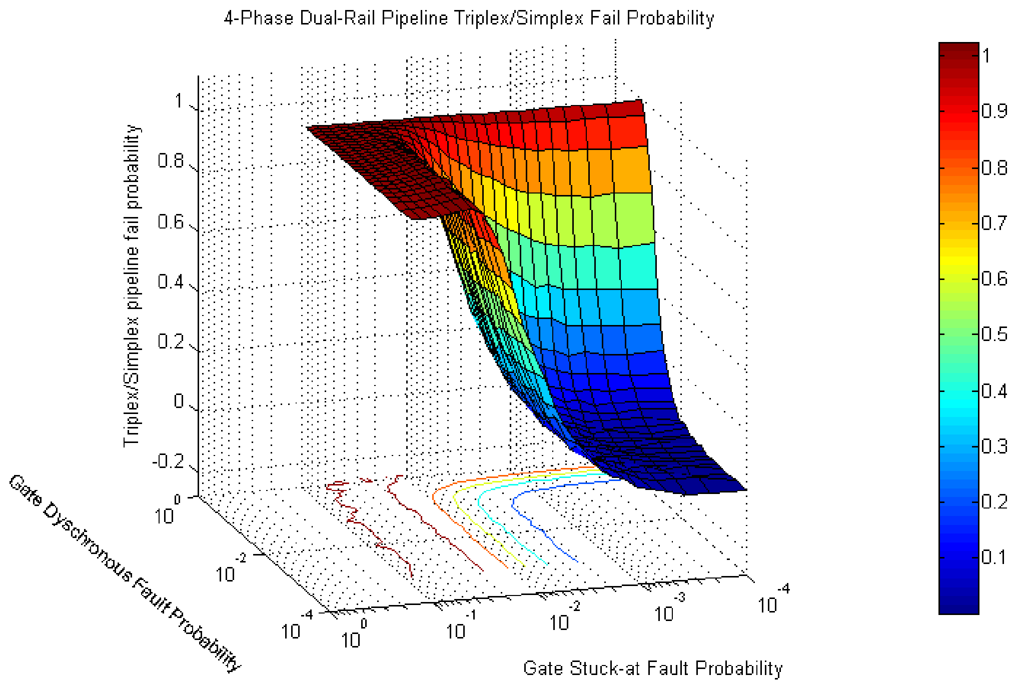


Figure 72. Triplex(Comb. MAJ)/simplex pipeline error rate vs. stuck-at and dyschronous fault probabilities.

Figure 72 shows a 3-D plot of the ratio of triplex (combinational voter) /simplex 4P2R pipeline error rate as a function of stuck-at gate fault probability and dyschronous gate fault probability in the manner that Figure 66 shows these results for 2-phase bundled micropipelines. The smaller the ratio, the more the triplex architecture improves reliability over the simplex. The contours projected onto the z = 0 plane in the figure show the regions of fault probability where triplex improves reliability, and the region where triplex actually reduces reliability. The red contour line bounds an area of stuck-at fault probability between roughly 1×10^{-1} and 3×10^{-2} where the redundant triplex circuit is slightly less reliable than its nonredundant simplex counterpart.

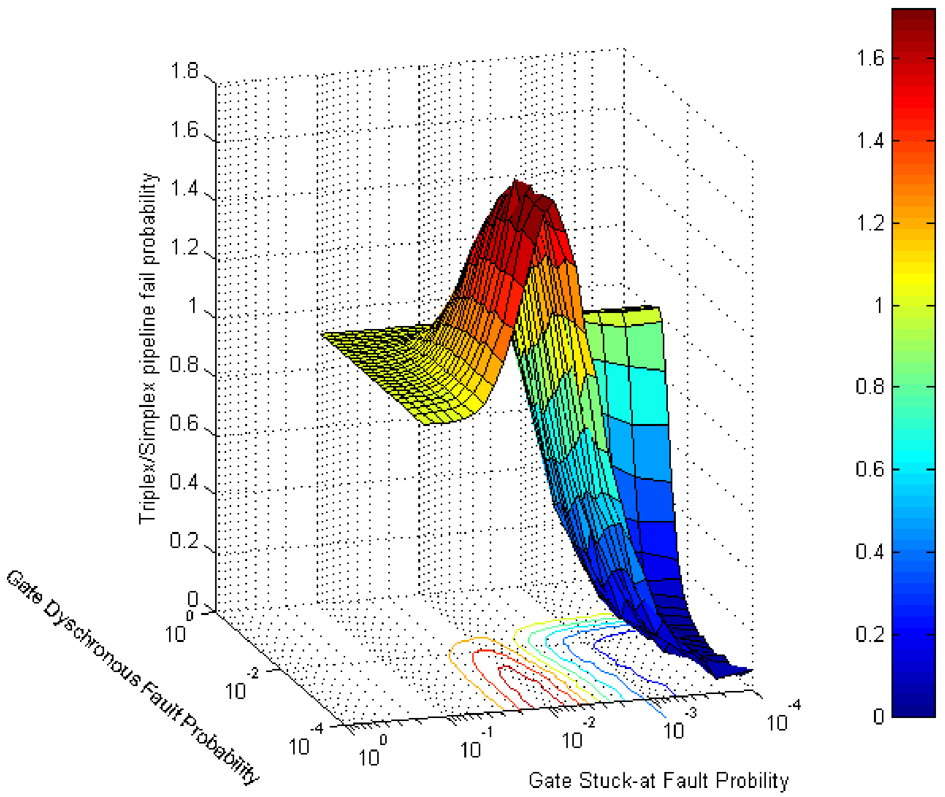


Figure 73. Triplex (HFMV)/Simplex pipeline fail rate vs. gate fault probability

This effect is more pronounced in triplex circuits that use hazard-free majority voters, as shown in Figure 73. Here, the HFMV presents a larger “fault target” than the combinational majority-gate voter. The HFMV reduces system reliability against stuck-at and dyschronous faults by as much as an order of magnitude compared to the combinational majority voter. The data show that nowhere in the stuck-at/dyschronous fault probability plane does the HFMV yield better reliability than the same system using simpler combinational majority gates. However, these results do not consider single-event transient faults, but they do confirm that there is no reliability benefit from the HFMV unless the system switching rate, skew, and SET rate are high, as discussed in section 2.5.

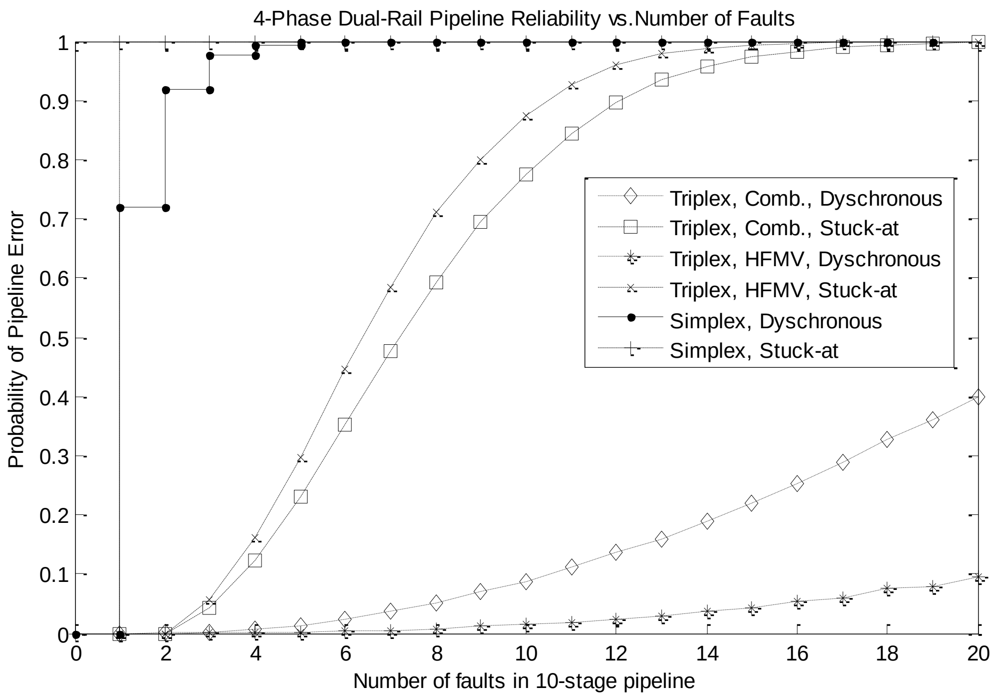


Figure 74. 4-phase dual-rail pipeline error rate vs. number of faults

Figure 74 plots triplex and simplex pipeline error rates vs. the total number of gate faults in the 10-stage pipeline. The markers show the discrete data points. Error bars are not shown because the standard error magnitude is smaller than the marker height. The simplex pipeline error rate is 1.0 for any number of stuck-at faults greater than zero.

5.4 Comparison of 2-Phase Bundled and 4-Phase 2-Rail Results

While it is difficult to draw meaningful conclusions from comparing the results from two such different circuits, some useful observations about these two circuit models are possible.

Comparing the simplex results shows that the nonredundant 4P2R pipeline has higher system error rates than the 2-phase bundled-data micropipeline. For low gate fault probabilities (~10^{-4} to ~10^{-3}) the 4P2R pipeline has system error rates as much as 2.6 times greater than those of the micropipeline for stuck-at faults, and as much as 1.5 times greater than those of the micropipeline. At higher gate fault probabilities the difference becomes less as both system error rates approach 1.

Each 2P bundled pipeline stage consists of one C-element and one capture-pass data latch. Each 4P2R pipeline stage consists of two C-elements and one OR gate. No faults are injected into the data latch in these simulations in order to focus on faults in the timing control logic path. Consequently the lower reliability of the nonredundant 4P2R pipeline model is a result of it having 2.5 times more fault targets per stage than the 2P bundled pipeline model.

Redundancy is more beneficial for the 4P2R pipeline than for the 2P bundled micropipeline. Comparing Figure 64 to Figure 70 shows that redundancy reduces system error rate for a wider range of gate stuck-at fault probabilities in the 4P2R pipeline than for the 2P bundled micropipeline. Redundancy reduces micropipeline system error rate for gate fault probabilities below 0.015, but in the 4P2R pipeline redundancy reduces system error rate for gate fault probabilities below 0.035.

6
Summary and Conclusion

This chapter summarizes the major results and original contributions of this dissertation, and suggests some directions for future research.

6.1 Summary of Results

This dissertation has studied the problems of voting in triple-modular redundant asynchronous circuits and systems, and proposed practical solutions for concurrent fault masking.

Chapter 2 reviewed prior work on fault tolerance, particularly concurrent fault masking using triple modular redundancy (TMR) with majority voting. Also discussed was prior work on a Hazard-Free Majority Voter (HFMV). A lower-cost HFMV implementation that uses 20% fewer transistors was proposed.

Chapter 3 discussed prior work on fault models for asynchronous circuits, particularly the DUDES model. The DUDES fault model was presented, which includes early-transition temporal faults, and which was derived from a fault analysis of the C-element. The original DUDES paper applied the model to fault detection, but not fault masking. Because majority gates are used to mask faults in TMR systems, the DUDES fault analysis method was applied to majority gates. The majority gate fault analysis added late-transition faults to the DUDES fault menagerie, generalized the DUDES fault model into the *dyschronous* fault model.

Chapter 4 presented a general triplex architecture for TMR asynchronous pipelines. Both dual-rail and bundled-data pipeline circuit examples were presented. The cost (area) and performance penalties of TMR were articulated. Possible timing problems with using voters in the data latch control lines in bundled-data pipelines were explored, which resulted in minimum delay voter constraints that varied with the amount of skew between redundant control lines. Chapter 4 also developed a new voter element "CMAJ" that solves the problem with bundled-data and improves system performance.

Chapter 5 presented a general Verilog based stochastic fault modeling environment for asynchronous pipelines that estimates the circuit failure rate for varying probabilities of stuck-at, dyschronous and single-event transient (SET) faults. Stochastic fault simulation results of dual-rail and bundled-data, simplex and triplex pipelines, using combinational majority gate voter and hazard-free majority voters, were plotted. Stuck-at and dyschronous gate fault probabilities are modeled independently. The results show how pipeline error rate varies as a function of these two independent variables. The following conclusions were drawn from the simulation results:

- Stuck-at faults are about ten times more potent in causing system errors than dyschronous faults for a wide range of gate fault probabilities.
- TMR circuits have a lower error rate than their nonredundant counterparts for gate fault rates below a certain threshold. Above the threshold, additional faults in the TMR circuit's voters counteract the voters' error-correcting ability. The value of the threshold depends on pipeline implementation and voter size. Increasing voter size, as exemplified by the larger HFMV, reduces the threshold, thereby reducing the range of gate fault probability for which TMR increases system reliability. Increasing SET tolerance by using HFMVs comes at the trade-off of reducing stuck-at fault tolerance. Because the threshold value depends on circuit implementation, no simple formula for calculating it is derived. Rather, Chapter 5 describes a simulation methodology for determining this value from detailed circuit models.

Also presented in Chapter 5 are equations derived by curve-fitting the stochastic fault simulation results, that relate pipeline error rate to gate fault probability. One equation holds for both the simplex micropipeline and the simplex 4P2R pipeline results.

This equation closely matches what is expected from basic probability theory. Another equation holds for both the triplex micropipeline and triplex 4P2R pipeline results. However, it is not exactly clear why this is the case and how these equations relate to the structure and parameters of the triplex circuits.

This dissertation has shown that triple-modular redundancy can be used successfully to improve the fault tolerance of a variety of asynchronous circuits, including both bundled and dual-rail data encoding. A general method for designing TMR asynchronous circuits can be described:

1. Starting with a nonredundant asynchronous circuit, triplicate every circuit primitive (C-element, MERGE, data latch, etc.) and every wire.
2. In the resulting triplex circuit, insert triplex majority-gate voters at the output of every C-element and every mutual-exclusion (MUTEX) element.
    a. The minimum delay of each voter must exceed the maximum skew of the voter’s three redundant input signals.
    b. If the system’s SET rate, switching rate, and skew are high relative to the static fault rate, then the use of hazard-free instead of combinational majority voters may be justified.
3. In bundled-data circuits, insert triplex majority voters at selected points in the data path. How to locate these points in the data path is circuit-specific and not described in this dissertation. But the stochastic fault simulation method used in this dissertation can be used to compare the reliability trade-offs of voter location alternatives.
4. Add the triplex-simplex interface circuits described in Section 4.6 to complete the TMR implementation. For bundled data triplex-simplex interfaces, use CMAJ elements, as described, to combine triplex handshake signals in order to produce the equivalent simplex handshake signals that are correctly timed relative the the bundled data.

The fault simulation results show that TMR increases system reliability only for gate stuck-at fault rates below approximately 10^{-2} and dyschronous fault rates below approximately 10^{-1}. This result is roughly consistent with the theoretical value derived from Equations 2 - 4, which demonstrated that TMR improves system reliability only if module reliability, r_m, is greater than 0.8923, or the module error rate 1- r_m, is less than 0.1077.

TMR’s increased reliability comes at the cost of increasing circuit area and power by approximately a factor of six (more if HFMVs are used). TMR also reduces the performance of asynchronous circuits because of the added voter delay. TMR’s performance penalty will vary with circuit design, but can reduce throughput by as much as a factor of two. A designer considering using TMR in asynchronous circuits must weigh these cost and performance trade-offs against the increased reliability offered by TMR.

6.2 Original Contributions

The original contributions of this dissertation include:

- Proposed a lower-cost implementation of Almukhaizim and Sinanoglu’s hazard-free majority voter that uses 20% fewer transistors than the original.
- Quantified the cost-performance trade-off of using HFMVs vs. combinational majority voters. HFMVs are more costly than combinational voters, as measured by circuit area and power consumption. Because they present larger fault targets than combinational voters, systems using them are less tolerant of static stuck-at and dyschronous faults. Thus HFMVs are only cost-effective in systems with high SET rates, switching rates, and triplet skew, as given by Equation 11.
- Proposed a new *dyschronous* fault model for fault analysis of clockless circuits. The dyschronous model has been derived from Shirvani, et al.’s *DUDES* fault model by generalizing it to include late-transition as well as early transition faults. The dyschronous fault model covers static temporal faults not covered by the well-known *stuck-at* fault model.
- Proposed a new *CMAJ* circuit element that combines the functions of the Muller C-element and combinational majority gate. The CMAJ is useful in triplex clockless circuits in synchronizing bundled data transfer at the triplex-simplex interface. The CMAJ may have other applications in triplex clockless circuits as well (see Section 6.3 Future Work).

- Proposed a general design methodology for TMR clockless circuits, and discussed the cost and performance trade-offs of TMR in clockless circuits.
- Demonstrated through logic simulation that triple-modular redundancy using majority-gate voters can increase the reliability of asynchronous pipeline circuits for gate fault rates below approximately 10^{-2}.
- Showed that Wakerly’s timing caveat manifests in clockless circuits as a minimum voter delay constraint.
- Developed a HDL simulation stochastic fault-simulation methodology that measures overall circuit reliability for several classes of static spatial and temporal faults over a range of gate fault probabilities.

6.3 Future Work

The stochastic fault simulation done for this dissertation measured the reliability of only two clockless pipeline circuit models: a 2-phase bundled micropipeline and a 4-phase 2-rail pipeline. More information should be possible gained from stochastic fault simulation of other clockless circuits. For example, comparing the results of the stochastic fault simulation of simplex and triplex 4-phase bundled-data pipelines to the simulation results of 2-phase bundled-data pipelines would illuminate the implications of the choice of 2-phase vs. 4-phase signaling in fault-tolerant clockless system design.

In the stochastic fault simulations, the weights used to inject different types of faults were derived from simple assumptions based on the transistor count of different types of circuit cells, as described in Section 5.1.2. It may be possible to derive more realistic fault weights from analysis of circuit defect models such as those presented in Chapter 3, Table 8, Table 11, and Table 13. Stochastic fault simulations that use more realistic fault weights would yield more accurate results.

This dissertation discussed single-event transient (SET) and single-event upset (SEU) faults, and the application of hazard-free majority voters to improve SET-fault-tolerance, but none of the stochastic fault simulations that were run included SET or SEU faults. Stochastic fault simulation of clockless circuits with random injection of SET/SEU could measure the circuit’s tolerance of these faults, and quantify the improved SET tolerance promised by the HFMV.

The CMAJ element, developed in this dissertation to solve a particular synchronization problem at the simplex-triplex interface of bundled data pipelines may have other applications with TMR clockless pipeline circuits. It may offer a way to improve the performance of TMR pipelines by reducing voter delay.

In the pipeline circuits discussed in this dissertation the minimum voter delay is constrained by Wakerly’s caveat [34]:

> “A majority circuit determines when *f + 1* of the … inputs have been received, then delays for a length of time *d*, and sends [the voted output].”

Where the number of redundant inputs is *2 f + 1* and the value of *d* must be greater than the maximum variation in input arrival times plus the maximum variation in the value of *d* in all participating voters.

If the CMAJ element is used as a voter, it will send its output immediately after all of its inputs have been received. Thus, for the CMAJ, Wakerly’s caveat can be restated as: a majority circuit sends its output when it determines that at least one of the following has occurred:

1) *f + 1* of the inputs have been received, and then delay for a length of time *d.*
2) All *2f + 1* of the inputs have been received.

The CMAJ voter does not delay for a length of time *d* if all of its inputs have been received. Only those voters in a pipeline whose inputs are faulty will incur the delay, thus reducing the overall latency of a pipeline with few faults.

Further work in investigating the cost-performance-reliability trade-offs of TMR asynchronous pipelines using CMAJ voters would begin by developing a CMOS transistor implementation and fault model of the CMAJ. The CMAJ is a more complex circuit than the combinational majority gate and consequently a larger fault target. Stochastic fault simulation results could quantify the performance gain and reliability loss that would result from using CMAJ voters instead of combinational majority gate voter in asynchronous pipelines.

MATLAB curve fitting of the stochastic fault simulation results for all of the triplex pipeline circuit models results yielded good fits to the equation

$$f(x) = 1 - (1 - x^a)^b$$

where *x* is the normalized gate fault probability, and *f*(*x*) is the pipeline failure probability. Coefficients *a* and *b* vary with pipeline type (2P bundled or 4P2R), fault type (stuck-at only or dyschronous only) and voter type (combinational majority gate or HFMV). It is not clear why this equation fits the simulation results so well, or how the values of the coefficients are related to the circuit. In order to develop this equation into a useful tool for circuit designers, further study is needed to understand how the coefficient values relate to circuit structure or parameters such as number of pipeline stages, or number or type of gates.

References

[1] E. D. Colbourne, G. P. Coverley, and S. K. Behera, "Reliability of MOS LSl circuits," *Proceedings of the IEEE*, vol. 62, pp. 244-259, 1974.

[2] R. W. Hamming, "Error detecting and error correcting codes," *The Bell System Technical Journal*, vol. 26, pp. 147-160, 1950.

[3] D. Siewiorek and R. Swarz, *The theory and practice of reliable system design*. Bedford, Mass.: Digital Press, 1982.

[4] M. Bushnell and V. Agrawal, *Essentials of electronic testing for digital, memory, and mixed-signal VLSI circuits*: Kluwer, 2000.

[5] D. Patterson and J. Hennessy, *Computer Organization and design*, Third ed. San Francisco: Morgan Kaufmann, 2005.

[6] "International Technology Roadmap for Semiconductors," 2007.

[7] R. Ho, K. Mai, and M. Horowitz, "The future of wires," *Proceedings of the IEEE,* vol. 89, 2001.

[8] K. Likharev, "CMOL Technology: Devices, Circuits, Architectures, and Possible Applications," Stony Brook University,, 2008.

[9] I. Sutherland and J. Ebergen, "Computers without clocks," in *Scientific American*, 2002.

[10] H. van Gageldonk, K. van Berkel, A. Peeters, D. Baumann, D. Gloor, and G. Stegmann, "An asynchronous low-power 80C51 microcontroller," presented at Fourth International Symposium on Advanced Research in Asynchronous Circuits and Systems, 1998.

[11] A. J. Martin, A. Lines, R. Manohar, M. Nystrom, P. Penzes, R. Southworth, U. Cummings, and L. Tak Kwan, "The design of an asynchronous MIPS R3000 microprocessor," presented at Seventeenth Conference on Advanced Research in VLSI, 1997.

[12] L. S. Nielsen, C. Niessen, J. Sparso, and K. van Berkel, "Low-power operation using self-timed circuits and adaptive scaling of the supply voltage," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 2, pp. 391-397, 1994.

[13] S. Moore, R. Anderson, R. Mullins, G. Taylor, and J. J. A. Fournier, "Balanced self-checking asynchronous logic for smart card applications," *Microprocessors and Microsystems,*, vol. 27, pp. 421-430, 2003.

[14] P. K. Lala, *Self-checking and fault-tolerant digital design*. San Francisco: Morgan Kaufmann, 2001.

[15] P. J. Kuekes, W. Robinett, G. Seroussi, and R. S. Williams, "Defect-tolerant interconnect to nanoelectronic circuits: internally redundant demultiplexers based on error-correcting codes," *NANOTECHNOLOGY*, vol. 16, pp. 869–882, 2005.

[16] J. F. Wakerly, "Microcomputer reliability improvement using triple-modular redundancy," *Proceedings of the IEEE*, vol. 64, pp. 889-895, 1976.

[17] J. von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," in *Automata Studies, Annals of Mathematics Studies*, C. E. Shannon and J. McCarthy, Eds. Princeton, New Jersey: Princeton University Press, 1956, pp. 43-98.

[18] Z. Kohavi, *Switching and finite automata theory*. New York: McGraw-Hill, 1970.

[19] A. S. Sadek, K. Nikolic, and M. Forshaw, "Parallel information and computation with restitution for noise-tolerant nanoscale logic networks," *NANOTECHNOLOGY*, pp. 192-210, 2004.

[20] J. Han and P. Jonker, "A system architecture solution for unreliable nanoelectronic devices," *Nanotechnology, IEEE Transactions on*, vol. 1, pp. 201-208, 2002.

[21] S. Roy and V. Beiu, "Majority multiplexing-economical redundant fault-tolerant designs for nanoarchitectures," *Nanotechnology, IEEE Transactions on*, vol. 4, pp. 441-451, 2005.

[22] D. Bhaduri and S. K. Shukla, "NANOPRISM: a tool for evaluating granularity vs. reliability trade-offs in nano architectures " presented at 14th ACM Great Lakes symposium on VLSI Boston, 2004.

[23] D. Bhaduri and S. Shukla, "NANOLAB-a tool for evaluating reliability of defect-tolerant nanoarchitectures," *Nanotechnology, IEEE Transactions on*, vol. 4, pp. 381-394, 2005.

[24] D. Bhaduri and S. K. Shukla, "Reliability evaluation of von Neumann multiplexing based defect-tolerant majority circuits," presented at 4th IEEE Conference on Nanotechnology, 2004.

[25] D. Bhaduri, S. Shukla, P. Graham, and M. Gokhale, "Comparing Reliability-Redundancy Trade-Offs for Two von Neumann Multiplexing Architectures," *IEEE Transactions on Nanotechnology*, vol. 6, pp. 265-279, 2007.

[26] J. R. Heath, P. J. Kuekes, G. S. Snider, and R. S. Williams, "A Defect-Tolerant Computer Architecture: Opportunities for Nanotechnology," *Science*, vol. 280, pp. 1716-1721, 1998.

[27] G. Snider, P. Kuekes, and R. S. Williams, "CMOS-like logic in defective, nanoscale crossbars," *NANOTECHNOLOGY*, vol. 15, pp. 881–891, 2004.

[28] K. Nikolic, A. Sadek, and M. Forshaw, "Architectures for reliable computing with unreliable nanodevices," presented at 1st IEEE Conference on Nanotechnology, 2001.

[29] N. Aggarwal, P. Ranganathan, N. P. Jouppi, and J. E. Smith, "Isolation in Commodity Multicore Processors," *Computer*, vol. 40, pp. 49-59, 2007.

[30] D. Hammerstrom, "A Digital VLSI Architecture for Real World Applications," in *An Introduction to Neural and Electronic Networks*: Academic Press, 1995.

[31] J. Han and P. Jonker, "From massively parallel image processors to fault-tolerant nanocomputers," presented at 17th International Conference on Pattern Recognition, 2004.

[32] J. Han, "Fault-tolerant Architectures for Nanoelectronic and Quantum Devices," Delft University of Technology, 2004.

[33] J. Han, J. Gao, Y. Qi, P. Jonker, and J. A. B. Fortes, "Toward Hardware-Redundant, Fault-Tolerant Logic for Nanoelectronics," *IEEE Design & Test of Computers*, 2005.

[34] D. Davies and J. F. Wakerly, "Synchronization and Matching in Redundant Systems," *IEEE Transactions on Computers*, vol. C-27, pp. 531-539, 1978.

[35] T. Verdel and Y. Makris, "Duplication-based concurrent error detection in asynchronous circuits: shortcomings and remedies," presented at 17th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems 2002.

[36] M. Fischer, N. Lynch, and M. Patterson, "Impossibility of Distributed Consensus with One Faulty Process," *Journal of the ACM*, vol. 32, pp. 374-382, 1985.

[37] D. E. Muller and W. S. Bartky, "A theory of asynchronous circuits," presented at International Symposium on the Theory of Switching, 1959.

[38] H. C. Brearley, "ILLIAC II - A Short Description and Annotated Bibliography," *Electronic Computers, IEEE Transactions on*, vol. EC-14, pp. 399-403, 1965.

[39] C. L. Seitz, "System Timing," in *Introduction to VLSI Systems*, C. Mead and L. Conway, Eds. Reading, MA: Addison Wesley, 1980.

[40] T. J. Chaney and C. E. Molnar, "Anomalous Behavior of Synchronizer and Arbiter Circuits," *IEEE Transactions on Computers*, vol. C-22, pp. 421-422, 1973.

[41] T. J. Chaney, "Measured Flip-Flop Responses to Marginal Triggering," *IEEE Transactions on Computers*, vol. C-32, pp. 1207-1209, 1983.

[42] I. Sutherland, "Micropipelines," *Communications of the ACM*, vol. 32, pp. 720-738, 1989.

[43] D. M. Chapiro, "Globally-Asynchronous Locally-Synchronous Systems," in *Computer Science*, vol. Ph.D. Palo Alto: Stanford University, 1984.

[44] J. Sparsø and S. Furber, *Principles of Asynchronous Circuit Design - A Systems Perspective* Boston: Kluwer Academic Publishers, 2001.

[45] C. J. Myers, *Asynchronous Circuit Design*: Wiley, 2001.

[46] A. J. Martin and M. Nystrom, "Asynchronous techniques for system-on-chip design," *Proceedings of the IEEE*, vol. 94, pp. 1089-1120, 2006.

[47] D. Lu and C. Q. Tong, "High level fault modeling of asynchronous circuits," presented at 13th IEEE VLSI Test Symposium, 1995.

[48] P. P. Shirvani, S. Mitra, J. C. Ebergen, and M. Roncken, "DUDES: a fault abstraction and collapsing framework for asynchronous circuits," presented at Sixth International Symposium on Advanced Research in Asynchronous Circuits and Systems, 2000.

[49] K. M. Fant and S. A. Brandt, "NULL Convention Logic^{TM}: a complete and consistent logic for asynchronous digital circuit synthesis," presented at International Conference on Application Specific Systems, Architectures and Processors, Chicago, 1996.

[50] M. Pease, R. Shostak, and L. Lamport, "Reaching Agreement in the Presence of Faults," *Journal of the ACM* vol. 27, pp. 228-234, 1980.

[51] I. David, R. Ginosar, and M. Yoeli, "Self-timed is self-checking," *Journal of Electronic Testing: Theory and Applications*, vol. 6, pp. 219-228, 1995.

[52] M. Herlihy and N. Shavit, "The topological structure of asynchronous computability," *Journal of the ACM*, vol. 46, pp. 858-923, 1999.

[53] D. A. Rennels and K. Hyeongil, "Concurrent error detection in self-timed VLSI," presented at 24th International Symposium on Fault-Tolerant Computing Austin, 1994.

[54] S. Peng and R. Manohar, "Efficient Failure Detection in Pipelined Asynchronous Circuits," presented at 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'05) 2005.

[55] H. Stork, "Reliability Challenges for Sub-100nm CMOS System on Chip Technologies (Keynote Address)," presented at IEEE International Reliability Physics Symposium, San Jose, 2006.

[56] R. Gong, W. Chen, F. Liu, K. Dai, and Z. Wang, "Modified Triple Modular Redundancy Structure based on Asynchronous Circuit Technique," presented at 21st IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems, 2006.

[57] W. Jang and A. J. Martin, "SEU-tolerant QDI circuits," presented at 11th IEEE International Symposium on Asynchronous Circuits and Systems, 2005.

[58] M. Zhang, S. Mitra, T. M. Mak, N. Seifert, N. J. Wang, Q. Shi, K. S. Kim, N. R. Shanbhag, and S. J. Patel, "Sequential Element Design With Built-In Soft Error Resilience," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 14, pp. 1368-1378, 2006.

[59] M. Omana, D. Rossi, and C. Metra, "Novel transient fault hardened static latch," presented at International Test Conference, 2003.

[60] Y. Zhao and S. Dey, "Separate dual-transistor registers: a circuit solution for on-line testing of transient error in UDMC-IC," presented at 9th IEEE On-Line Testing Symposium, 2003.

[61] Y. Monnet, M. Renaudin, and R. Leveugle, "Asynchronous circuits sensitivity to fault injection," presented at 10th IEEE International On-Line Testing Symposium, 2004.

[62] Y. Monnet, M. Renaudin, and R. Leveugle, "Asynchronous circuits transient faults sensitivity evaluation," presented at 42nd Design Automation Conference, 2005.

[63] Y. Monnet, M. Renaudin, and R. Leveugle, "Hardening techniques against transient faults for asynchronous circuits," presented at 11th IEEE International On-Line Testing Symposium, 2005.

[64] S. Mitra, M. Zhang, S. Waqas, N. Seifert, B. Gill, and K. S. Kim, "Combinational Logic Soft Error Correction," presented at IEEE International Test Conference, 2006.

[65] S. Almukhaizim and Y. Makris, "Concurrent Error Detection Methods for Asynchronous Burst-Mode Machines," *Computers, IEEE Transactions on*, vol. 56, pp. 785-798, 2007.

[66] S. Almukhaizim, F. Shi, and Y. Makris, "Coping with Soft Errors in Asynchronous Burst-Mode Machines," presented at 14th IEEE International Symposium on Asynchronous Circuits and Systems, 2008.

[67] W. J. Bainbridge, W. B. Toms, D. A. Edwards, and S. B. Furber, "Delay-insensitive, point-to-point interconnect using m-of-n codes," presented at Ninth IEEE International Symposium on Asynchronous Circuits and Systems 2003.

[68] J. Sparso and J. Staunstrup, "Delay-insensitive multi-ring structures," *Integration, the VLSI Journal*, vol. 15, pp. 313-340, 1993.

[69] D. Hampel, K. J. Prost, and N. R. Scheinberg, "Threshold Logic Using Complementary MOS Device," U. S. Patent, Ed., 1975.

[70] V. Beiu, J. M. Quintana, and M. J. Avedillo, "VLSI implementations of threshold logic-a comprehensive survey," *Neural Networks, IEEE Transactions on*, vol. 14, pp. 1217-1243, 2003.

[71] I. Amlani, A. O. Orlov, G. Toth, G. H. Bernstein, C. S. Lent, and G. L. Snider, "Digital Logic Gate Using Quantum-Dot Cellular Automata," *Science*, vol. 284, pp. 289.

[72] R. O. Ozdag and P. A. Beerel, "High-speed QDI asynchronous pipelines," presented at Eighth International Symposium on Asynchronous Circuits and Systems, 2002.

[73] S. Almukhaizim and O. Sinanoglu, "A Hazard-Free Majority Voter for TMR-Based Fault Tolerance in Asynchronous Circuits," presented at 2nd International Design and Test Workshop, 2007.

Biographical Sketch

John Daniel Lynch was born in Portland, Oregon on 22 March 1957. He attended Portland Community College, The Evergreen State College in Olympia, Washington, and received a BS in Electrical Engineering, *Cum Laude*, from the University of Utah in Salt Lake City in 1979.

Since 1979 he has worked extensively in the high-tech industry as a computer engineer, including positions at Floating Point Systems, Intel, AMD, Pyramid Technology, and Adaptive Solutions. From 1995 to 1998 he managed ASIC Design Engineering for InFocus Corporation. In 1998 he moved to Pixelworks, Inc. where he was Director of IC Design Engineering. In 2002 he joined the School of Science and Engineering (formerly the Oregon Graduate Institute) of Oregon Health & Science University as an Instructor, where he developed and taught courses in a variety of computer engineering subjects. In 2007 he was appointed Director of OHSU's Computer Engineering and Design Education Program.

Patents

U.S. Patent No. 5,787,095: Multiprocessor computer backplane bus.

U.S. Patent No. 5,581,713: Multiprocessor computer backplane bus in which bus transactions are classified into different classes for arbitration.

U.S. Patent No. 7,406,632: Error reporting network in multiprocessor computer

Publications

J. Lynch, “Teaching Digital System Timing: A Comprehensive Approach”, *IEEE Transactions on Education*, Vol. 51, Issue 2, August 2008

J. Lynch, "A Novel Graduate Course Takes a Systematic Approach to Teaching Digital System Timing", *Proceedings of the 2007 IEEE International Conference on Microelectronic Systems Education*

J. Lynch, D. Hammerstrom and R. Kravitz, "A Coherent FPGA-Based System-on-Chip Design Curriculum", *Proceedings of the 2005 IEEE International Conference on Microelectronic Systems Education*

J. Lynch and H. Schiefer, “Concurrent Engineering Delivers at the Chip and System Level,” *Integrated System Design*, December 1997.

R. Fazzari and **J. Lynch**, “The FPS Mark II T Series: An Enhanced Parallel Supercomputer,” in *Proceedings of the Third Conference on Hypercube Concurrent Computers and Applications*, 1988.