Dependable Computing

A Multilevel Approach

Behrooz Parhami

University of California, Santa Barbara

parhami@ece.ucsb.edu
http://www.ece.ucsb.edu/~parhami

This is a draft of the forthcoming book
Dependable Computing: A Multilevel Approach,
by Behrooz Parhami, publisher TBD
ISBN TBD; Call number TBD

Dedication

To my academic mentors of long ago:

Professor Robert Allen Short (1927-2003),
who taught me digital systems theory
and encouraged me to publish my first research paper on
stochastic automata and reliable sequential machines,

and

Professor Algirdas Antanas Avižienis (1932-)
who provided me with a comprehensive overview
of the dependable computing discipline
and oversaw my maturation as a researcher.

About the Cover

The cover design shown is a placeholder. It will be replaced by the actual cover image once the design becomes available. The two elements in this image convey the ideas that computer system dependability is a weakest-link phenomenon and that modularization & redundancy can be employed, in a manner not unlike the practice in structural engineering, to prevent failures or to limit their impact.

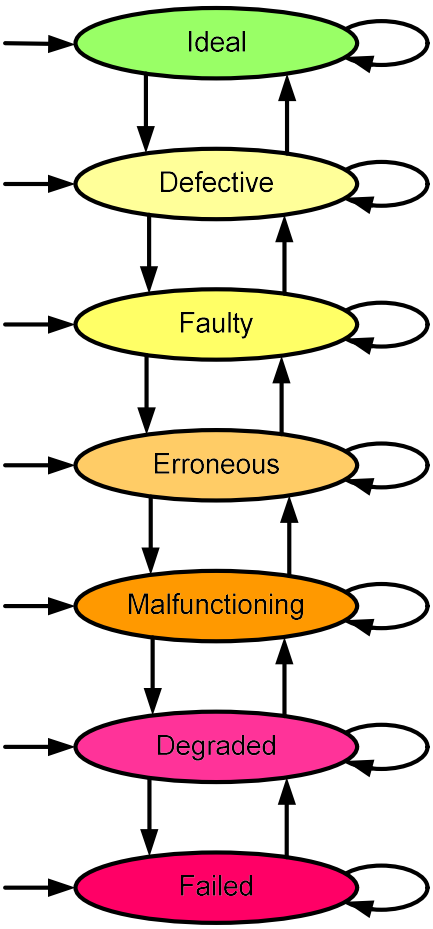
Structure at a Glance

The multilevel model on the right of the following table is shown to emphasize its influence on the structure of this book; the model is explained in Chapter 1 (Section 1.4).

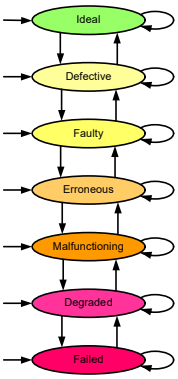
STRUCTURE AT A GLANCE

Part I — Introduction: Dependable Systems (The Ideal-System View)	Goals / Models	1. Background and Motivation 2. Dependability Attributes 3. Combinational Modeling 4. State-Space Modeling
Part II — Defects: Physical Imperfections (The Device-Level View)	Methods / Examples	5. Defect Avoidance 6. Defect Circumvention 7. Shielding and Hardening 8. Yield Enhancement
Part III — Faults: Logical Deviations (The Circuit-Level View)	Methods / Examples	9. Fault Testing 10. Fault Masking 11. Design for Testability 12. Replication and Voting
Part IV — Errors: Informational Distortions (The State-Level View)	Methods / Examples	13. Error Detection 14. Error Correction 15. Self-Checking Modules 16. Redundant Disk Arrays
Part V — Malfunctions: Architectural Anomalies (The Structure-Level View)	Methods / Examples	17. Malfunction Diagnosis 18. Malfunction Tolerance 19. Standby Redundancy 20. Robust Parallel Processing
Part VI — Degradations: Behavioral Lapses (The Service-Level View)	Methods / Examples	21. Degradation Allowance 22. Degradation Management 23. Resilient Algorithms 24. Software Redundancy
Part VII — Failures: Computational Breaches (The Result-Level View)	Methods / Examples	25. Failure Confinement 26. Failure Recovery 27. Agreement and Adjudication 28. Fail-Safe System Design

Appendix: Past, Present, and Future



III Faults: Logical Deviations



"To find fault is easy; to do better may be difficult."

Plutarch

"A fault that humbles a man is of greater value than a virtue that puffs him up."

Anonymous

Chapters in This Part

Faults, defined as circuit-level deviations from a system's specified behavior, can arise in two ways: from defective devices, when a segment of the circuit that contains them is exercised, or directly from incorrect logic-signal values due to external influences, intracircuit interactions, or design/implementation flaws. In this part, we present key concepts pertaining to the complementary methods of fault testing (detecting faults by forcing them to produce observable output errors) and fault masking (using redundancy to ensure that faults do not produce errors). Both fault testing and fault masking are facilitated by an abstract formulation of the causes and manifestations of faults, known as a fault model. Prompted by the complexities encountered in designing, validating, and applying fault detection tests, we examine a number of strategies to improve the testability of such circuits. We conclude this part by a study of a class of logic-level redundancy schemes based on circuit replication and voting, as a simple and widely used example of fault masking techniques.

9 Fault Testing

"As long as there are tests, there will be prayer in schools."

Anonymous

"To test, or not to test; that is the question: Whether 'tis nobler for the tester's soul to suffer the barbs and snickers of outraged designers, or to take arms against a sea of failures, and by testing, end them? To try: to test; to test . . ."

B. Beizer, Software Testing Techniques

Topics in This Chapter

Fault detection by means of testing is used for the validation of engineering prototypes, screening of manufactured devices, and corrective or preventive maintenance in operational systems. The fault testing effort is a combination of test generation, test validation, and test application. In this chapter, after explaining the fundamental notions of test generation for combinational digital circuits, we show that test generation is inherently difficult from a computational standpoint. We then demonstrate how testing becomes even more complicated when the circuit-under-test contains memory. Finally, we discuss how built-in testing and self-test methods partially alleviate the aforementioned difficulties.

9.1 Overview and Fault Models

Fault testing is performed in three contexts. *Engineering tests* aim to ascertain whether a new system (a prototype, say) is correctly designed. *Manufacturing tests* are performed to establish the correctness of an implementation. *Maintenance tests*, performed in the field, check for correct operation, either because some problem was encountered (*corrective maintenance*) or else in anticipation of potential problems (*preventive maintenance*). As shown in Fig. 9.1, fault testing entails the three steps of *test generation*, *test validation*, and *test application*. We present a brief overview of these three steps in the rest of this section, later on focusing exclusively on test generation.

Test generation entails the selection of input test patterns whose application to the circuit would expose the fault set of interest via observing the circuit's outputs. The set of test patterns may be *preset*, meaning that the entire set is applied before reaching a decision, or *adaptive*, where the selection of a test to apply depends on previous test outcomes, with test application stopping as soon as enough information is available.

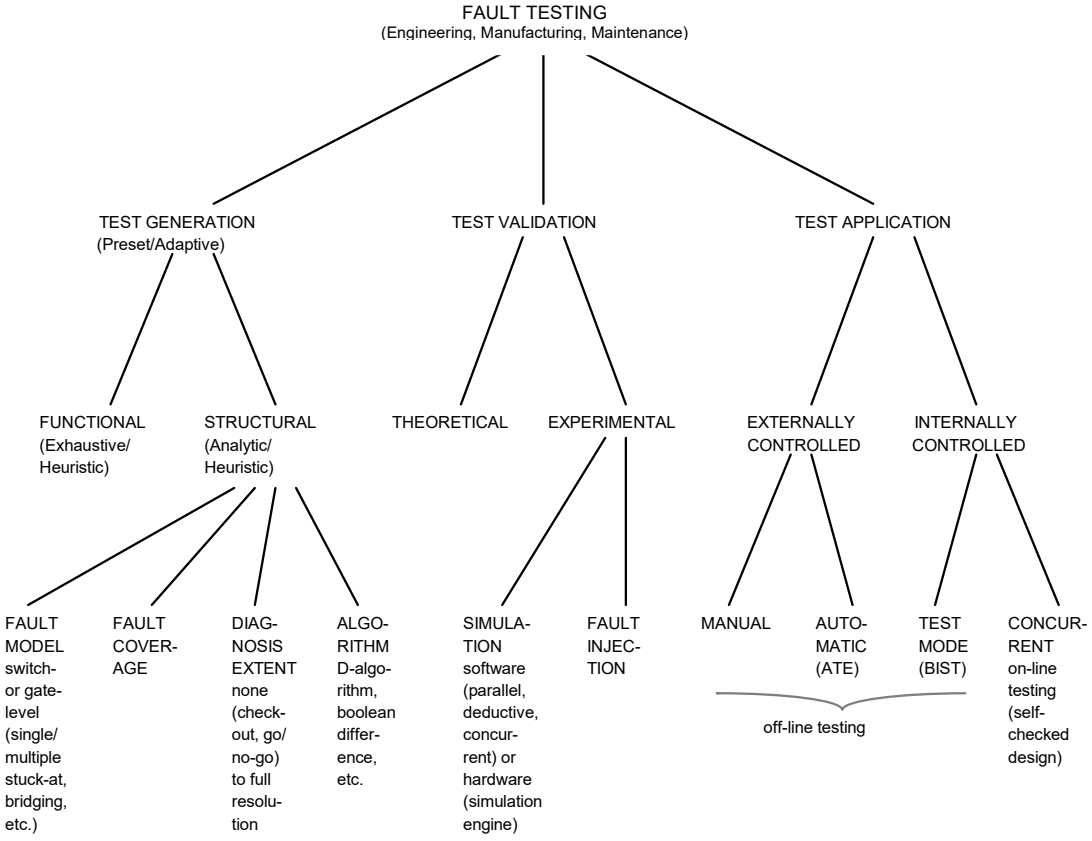


Fig. 9.1 A taxonomy of fault testing.

If we view the circuit or system under test as a black box, perhaps because we know nothing about its internal design, we opt for *functional testing*. For example, functional testing of an adder entails the application of various integers as inputs and checking that the correct sum is produced in each case. A functional test is *exhaustive* if all possible combinations of inputs are applied. Exhaustive testing is practical only for circuits with a relatively small number of inputs (4-bit or 8-bit adder, but not 32-bit adder). Random testing entails the selection of a random sample of input test patterns, which of course provides no guarantee of catching all faults. In *heuristic* functional test generation, we pick the tests to correspond to typical inputs as well as what we believe to be problematic or "corner" cases. In the case of an adder, our selections may include both positive and negative integers, small and large numbers, values that lead to overflow, inputs that generate sums of extreme values, and, perhaps, inputs that generate carry chains of varying lengths.

In *structural testing*, knowledge of the circuit's or system's internal composition is used to make the testing more efficient; that is, selecting inputs that are guaranteed to detect any fault from a fault-set of interest (*analytic*) or do so with high probability (heuristic). Generating structural tests requires that we select a *fault model* (specifying the kinds of faults that are expected and their effects on circuit behavior; more on this in Section 9.2), assess *fault coverage*, contemplate the *diagnosis extent* (whether we would like to pinpoint the location of a detected fault or to simply make a "go/no-go" decision), and devise or select an algorithm for the task.

Once a set of tests has been generated, we need *test validation* to ensure that the chosen tests will accomplish our goal of complete fault coverage or high-probability detection. Some approaches to test validation are *theoretical*, meaning that they can provide guaranteed results within the limitations of the chosen fault model. *Experimental* test validation, which may provide partial assurance, is done via *simulation* (modeling the circuit, with and without faults) or *fault injection* (purposely introducing faults of various kinds in the circuit or system to see if they are detected).

Following validation, we enter the domain of test application, where we expose the circuit or system under test to the chosen inputs and observe its outputs. When test application is *externally controlled*, the circuit or system is placed in test mode and its behavior is observed for an extended period of time. This kind of *manual* or *automatic* test application is known as *off-line testing*. Increasingly, for modern digital systems, test

application is *internally controlled* via built-in provisions for testing and testability enhancement. Internally controlled test application can be off-line, meaning that a special *test mode* is entered during which the circuit or system ceases normal operation, or on-line, where testing and normal operation are *concurrent*.

A circuit or system whose testing requires relatively low effort in test generation and application scores high on *testability*. We will see in Chapter 11 that testability can be quantified, but, for now, we are using testability in its qualitative sense. The set-up for testing is depicted in Fig. 9.2. Testability requires that each point within the circuit under test be *controllable* (through paths leading from primary inputs to that point) and observable (by propagating the state of the desired point to one or more primary outputs). Redundancy in circuits often curtails controllability and observability, thus having a negative impact on testability.

Referring to Fig. 9.2, test patterns may be randomly generated, come from a preset list, or be selected according to previous test outcomes. Test results emerging at the circuit's outputs may be used in raw form (implying high data volumes) or compressed into a *signature* before the comparison. Reference value can come from a "gold" or trusted version of the circuit that runs concurrently with it or from a precomputed table of expected outcomes. Finally, test application may be off-line or on-line (concurrent).

Complete, high-coverage testing is critical, because any delay in fault detection has important financial consequences. The rule of thumb that you lose a few dollars when you catch a component fault, tens of dollars if it goes to the circuit-board level, hundreds of dollars if the faulty component makes it to the system level, and thousands of dollard if in-field corrective action is required, is important to remember.

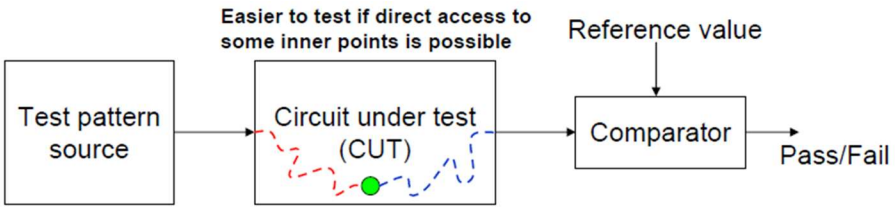
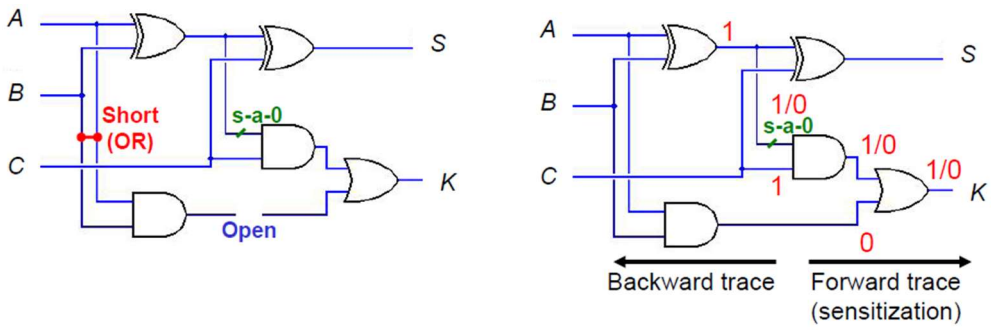


Fig. 9.2. Test set-up, composed of test-pattern source, circuit under test, and comparator. The controllability and observability of a point within the circuit are highlighted by dashed lines.

Except perhaps in the case of exhaustive functional testing, test coverage may be well below 100%, with reasons including the large input space, model inaccuracies, and impossibility of dealing with all combinations of the modeled faults. So, testing, which may be quite effective in the early stages of system design, when there may be many residual bugs and faults, tends to be less convincing when bugs/faults are rare. Paraphrasing Edsger W. Dijkstra, who made an oft-quoted statement in connection with program bugs, we can say that testing can be used to show the presence of faults, but never to show their absence. Also relevant is this observation by Steve C. McConnell: "Trying to improve software quality by increasing the amount of testing is like trying to lose weight by weighing yourself more often."

We conclude this section by presenting an overview of fault models at different levels of abstraction. Fault model, a catalog of the types of deviations in logic values that one expects in the circuit under test, can be specified at various levels: transistor, gate, function, system. Gate-level fault models will be the focus of our discussion, so let us briefly review the other three levels before proceeding. Transistor-level faults, which are caused by defects, shorts/opens, electromigration, transients, and the like, may lead to high current, incorrect output, or intermediate voltage, among others. They can be modeled as stuck-on/off, bridging, delay, coupling, and crosstalk faults. Transistor-level fault models quickly become intractable because of the large model space. Function-level faults are selected in an ad hoc manner based on the defined function of a block (decoder, ALU, memory). We will discuss system-level faults (malfunctions, in our terminology) in Part V of this book.

At the gate or logic level, the most widely considered faults are the so-called "line stuck" faults, where a circuit line/node assumes a constant logic value, independent of the circuit's inputs. We will focus on these stuck-at-0 (s-a-0) and stuck-at-1 (s-a-1) faults in the rest of this chapter. For example, Fig. 9.3a shows the upper input of the rightmost AND gate suffering from an s-a-0 fault. Line bridging faults result from unintended connection between two lines/nodes, often leading to wired OR/AND of the respective signals (Fig. 9.3a). Line open faults (bottom line in Fig. 9.3a) can sometimes be modeled as an s-a-0 or s-a-1 fault. Delay faults (excessively long delays for some signals) are less tractable than the previous fault types. Coupling and crosstalk faults are other examples of fault types included in some models.



(a) Logic circuit and fault examples (b) Testing for a particular s-a-0 fault

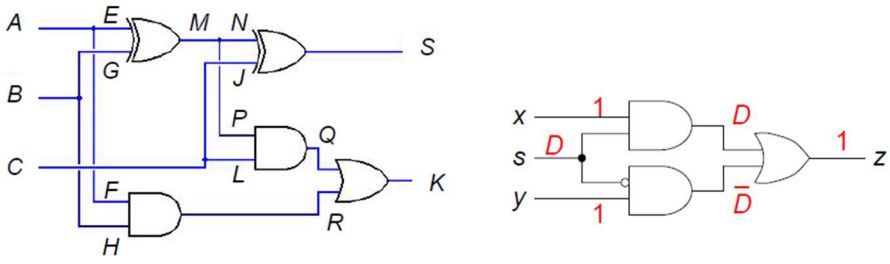
Fig. 9.3 Gate- or logic-level faults and testing for them.

9.2 Path Sensitization and D-Algorithm

The main ideas behind test design are controlling the faulty point from primary inputs and propagating its behavior to some primary output. Considering the s-a-0 fault shown in Fig. 9.3b, a test must consist of input values that force that particular line to 1 and then propagate the resulting value (1 if the line is healthy, 0 if it is s-a-0) to one of the two primary outputs. The process of propagating the 1/0 value to a primary output is known as *path sensitization*. In the example of Fig. 9.3b, the path from the s-a-0 line to output *K* is sensitized by choosing the lower AND gate input to be 1 (a 0 input inhibits propagation, because it makes the AND gate output 0, regardless of the value on the upper input) and the lower OR gate input to be 0.

At this point, we have the following requirements to be satisfied through a suitable assignment of values to the primary inputs *A*, *B*, and *C*: the XOR gate output should be 1, the output of the lower AND gate should be 0, and *C* should be 1. It is easy to see that two test patterns satisfy these requirements: (*A*, *B*, *C*) = (0 1 1) or (1 0 1).

The path sensitization method just discussed is formalized in the *D*-algorithm [Roth66], which is based on the *D*-calculus. A 1/0 on the logic circuit diagram (Fig. 9.3b) is represented as *D* and 0/1 is represented as $\overline{D}$. Then, *D* and $\overline{D}$ values are propagated to outputs via forward tracing (path sensitization) and towards the inputs via backward tracing, eventually producing the required tests. In applying the *D*-algorithm, circuit lines must be considered and labeled separately, as depicted in Fig. 9.DAlg-a. This is required because in some cases, electrically connected lines (such as *M*, *N*, and *P* in Fig. 9.DAlg-a) may not be affected in the same way by a fault on one of them.



(a) Circuit with all lines labeled (b) Reconvergent fan-out example

Fig. 9.DAlg Example circuit for, and a problem with, the D algorithm.

The *D*-algorithm encounters difficulties with reconvergent fan-outs, but an extension of the algorithm, PODEM (path-oriented decision making) [Goel81] fixes the shortcomings. The reconvergent fan-out problem is illustrated in Fig. 9.DAlg-b. The letter *D* on line *s* indicates the intention to test it for s-a-0. Path sensitization dictates that we should force the other input of each of the two AND gates to 1, leading to *D* and $\overline{D}$ on the OR gate inputs and 1 on the *z* output. Thus the s-a-0 fault on line *s* cannot be detected. PODEM gets around this problem by setting the *y* input to 0 instead of 1, causing the output *z* to become *D*.

The worst-case time complexity of the *D*-algorithm is exponential in the circuit size, given that it must consider all path combinations. The presence of XOR gates in the circuit cuases the behavior to approach the worst case. However, the average case is much better and tends to be quadratic in the circuit size. PODEM also has exponential time complexity, but in the number of circuit inputs, not its size.

Once the set of possible tests for each fault of interest has been obtained, the rest of the test generation process is as follows. We construct a table whose rows represent test patterns (circuit input values) and whose columns correspond to faults. An "x" is placed at a row-column intersection if the test associated with that row detects the specific fault associated with the column; a hyphen is placed otherwise. A partial table of this kind is shown in Table 9.1. Our task is completed upon choosing a minimal set of rows that cover all faults. This covering problem can be solved quite efficiently in a manner similar to choosing prime implicants for a minimal sum-of-products representation of a logic function. For example, in the case of Table 9.1, if only the 4 faults shown are of interest, then we have two minimal test sets: {(0, 0, 1), (0, 1, 1)} and {(0, 0, 1), (1, 0, 1)}.

Table 9.1 The covering problem in test generation.

			P		*Q*	
A	*B*	*C*	s-a-0	s-a-1	s-a-0	s-a-1
0	0	0	-	-	-	x
0	0	1	-	x	-	x
0	1	1	x	-	x	-
1	0	1	x	-	x	-

It is easy to see that any test that detects *P* s-a-0 in Fig. 9.DAlg-a also detects *L* s-a-0 and *Q* s-a-0. Such faults are said to be equivalent. In a similar manner, the faults *Q* s-a-1, *R* s-a-1, and *K* s-a-1 are equivalent. Identifying equivalent faults before test generation leads to savings in time and effort, because only one fault from each equivalence class needs to be considered.

9.3 Boolean Difference Methods

Consider a Boolean function of several variables, such the K as a function A, B, and C in Fig. 9.DAlg-a.

$$K = f(A, B, C) = AB \vee BC \vee CA \qquad (9.3.\text{Cout1})$$

The Boolean difference of K with respect to input variable B is defined as:

$$dK/dB = f(A, 0, C) \oplus f(A, 1, C) = CA \oplus (A \vee C) = A \oplus C \qquad (9.3.\text{BD1})$$

Intuitively, $dK/dB = 1$ (satisfied when $A \neq C$) tells us that the value of K changes when B changes from 0 to 1 or that K is sensitive to a change in the value of B. Conversely, $dK/dB = 0$ (satisfied for $A = C$) indicates that the value of K is insensitive to a change in the value of B.

Consider in Fig. 9.DAlg.a the line P being s-a-0. A stuck line behaves as an independent variable rather than as a dependent one. So, considering P as an independent variable, the Boolean equation for the output Q can be obtained as:

$$K = PC \vee AB \qquad (9.3.\text{Cout2})$$

The Boolean difference of K with respect to P is:

$$dK/dP = AB \oplus (C \vee AB) = C(\overline{AB}) \qquad (9.3.\text{BD2})$$

Tests that detect P s-a-0 are solutions to the equation $P\ dK/dP = 1$.

$$P\ dK/dP = (A \oplus B)C(\overline{AB}) = 1 \quad \Rightarrow \quad C = 1, A \neq B \qquad (9.3.\text{Psa0t})$$

Similarly, tests that detect P s-a-1 are solutions to the equation $\bar{P}\ dK/dP = 1$.

$$\bar{P}\ dK/dP = (\overline{A \oplus B})C(\overline{AB}) = 1 \quad \Rightarrow \quad C = 1, A = B = 0 \qquad (9.3.\text{Psa1t})$$

9.4 The Complexity of Fault Testing

From our discussion of the Boolean difference method in Section 9.3, and in particular from equations 9.3.Psa0t and 9.3.Psa1t, we see that the problem of generating tests for a particular fault of interest can be converted to solving an instance of the satisfiability (SAT) problem. The SAT problem is defined, in its decision form, as answering the question: Is a particular Boolean expression satisfiable, that is, can it assume the value 1 for some assignment of values to its variables? According to a well-known theorem of complexity theory [Cook71], SAT is NP-complete. That is, no efficient, subexponential-time algorithm is known that would solve an arbitrary instance of SAT. In fact, even highly restricted versions of SAT remain NP-complete.

Conversion of the fault detection problem to SAT leads to the suspicion that perhaps fault detection is also NP-complete. To show this, we must prove that an arbitrary instance of SAT or some other NP-complete problem can be easily transformed to a fault-detection problem. Such a proof was first constructed by Ibarra and Sahni [Ibar75] and subsequently simplified by Fujiwara [Fuji82]. The demonstration that fault detection is an NP-complete problem makes it unlikely that the task can be performed efficiently by means of a general algorithm any time in the near future. We are thus motivated to seek solutions to the problem in special cases and to devise heuristic algorithms that produce acceptable, near-complete solutions for circuit classes of practical interest.

In the rest of this section, we provide a proof that fault detection is an NP-complete problem. In fact, we will prove that a highly restricted form of the problem, that is, finding tests for stuck-at fault in certain 3-level circuits composed entirely of AND and OR gates (with all primary inputs being uncomplemented) is NP complete, by transforming a known NP-complete problem to it. Circuits composed entirely of AND and OR gates and lacking any complemented inputs are known as *monotone circuits*, so named because if you increase the value of some of the inputs from 0 to 1 the ouput either does not change or it changes from 0 to 1 (it can never go from 1 to 0). Thus, our proof will show that fault detection in certain restricted classes of monotone logic circuits is NP-complete, making the general problem NP-complete as well.

We will use 3SAT as a problem whose NP-completeness is well-established [Cook71] as our starting point. In 3SAT, the logic expression to be satisfied in the product of clauses,

each of which is the logical OR of at most 3 true or complemented variables. For example, an instance of 3SAT might involve the Boolean expression:

$$E = (x'_1 \vee x_4 \vee x'_5)\ (x'_2 \vee x'_3 \vee x_7)\ (x_2 \vee x_3 \vee x_6)\ (x_1 \vee x_3 \vee x'_6) \qquad \text{(9.4.3SAT)}$$

From the NP-completeness of 3SAT, we show that the lesser-known clause-monotone SAT (CM-SAT) is also NP-complete. The CM-SAT is as follows. Given n Boolean variables, we want to determine whether a Boolean expression of the following form is satisfiable, that is, wheter it can assume the value 1 for some combination of input values.

$$E = \Pi(a_i \vee b_i \vee c_i \vee \dots) \qquad \text{(9.4.CM1)}$$

Each of the ANDed (multiplied) clauses on the right-hand side of equation 9.4.CM1 is the logical OR of n or fewer terms, all being either true variables or complemented variables, but not both in the same clause. As an example, an instance of CM-SAT expression with 7 variables might be the following, which contains two clauses with only complemented variables and two clauses with only uncomplemented variables.

$$E = (x'_1 \vee x'_4 \vee x'_5 \vee x'_6 \vee x'_7)\ (x'_2 \vee x'_3 \vee x'_7)\ (x_2 \vee x_3 \vee x_4 \vee x_6)\ (x_1 \vee x_3 \vee x_6) \qquad \text{(9.4.CM2)}$$

It is easy to convert an arbitrary instance of 3SAT to an instance of CM-SAT: all we need is to observe that if a clause in 3SAT has both complemented and uncomplemented variables, we can replace it with the product of two clauses to obtain a CM-SAT problem that is satisfiable iff the original 3SAT problem instance is satisfiable. For example, consider the clause involving x'_k and two uncomplemented variables x_i and x_j. Then:

$$(x_i \vee x_j \vee x'_k) \text{ is replaced with } (x_i \vee x_j \vee v_k)\ (v'_k \vee x'_k) \qquad \text{(9.4.CM3)}$$

The variable v_k is a new variable; thus, the CM-SAT problem obtained may have up to twice as many variables as the original problem. Similarly, when a clause contains a single uncomplemented variable and two complemented ones:

$$(x'_i \vee x'_j \vee x_k) \text{ is replaced with } (x'_i \vee x'_j \vee v'_k)\ (v_k \vee x_k) \qquad \text{(9.4.CM4)}$$

It is easy to see that the original 3SAT problem is satisfiable iff the derived CM-SAT problem is satisfiable.

Corresponding to a CM-SAT instance such as the one in equation 9.4.CM2, we construct a 3-level AND-OR-AND logic circuit as follows.

At the first level, we place a number of AND gates, one for each clause with complemented inputs. Each AND receives the inputs which appear in the clause, but in true form, and produces the complement of the clause as its output. For example, the AND gate associated with the clause $x'_1 \vee x'_4 \vee x'_5 \vee x'_6 \vee x'_7$ in equation 9.4.SAT2 will have the output $x_1x_4x_5x_6x_7 = (x'_1 \vee x'_4 \vee x'_5 \vee x'_6 \vee x'_7)'$. At the second level, there are OR gates, one receiving the outputs of the first-level AND gates as its inputs and the others forming the remaining clauses with only uncomplemented variables. Finally, the outputs of all the OR gates in level 2 are fed to a single AND gate in level 3. As an example, the 3-level circuit constructed as above for the CM-SAT instance of equation 9.4.SAT2 is depicted in Fig. 9.SAT.

In the circuit of Fig. 9.SAT, an s-a-1 fault on line y_1 is detectable iff there exists an input pattern that sets all outputs of the level-1 AND gates to 0 and all outputs of the level-2 OR gates, other than the top one, to 1. It is easy to see that the test will then satisfy the instance of the CM-SAT problem from which the circuit was derived. Given that the conversion time from CM-SAT to fault detection is a polynomial in the number n of variables, the fault detection problem must be NP-complete. We have thus proved:

Theorem 9.fdnpc: The problem of detecting stuck-at faults in three-level monotone AND-OR-AND circuits is NP-complete.

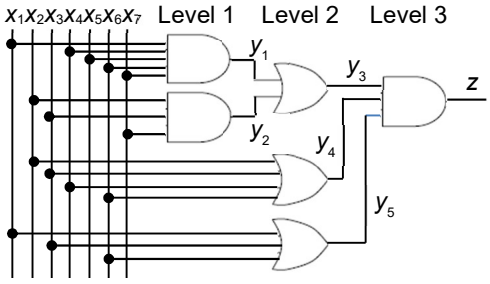


Fig. 9.SAT Example circuit formed by converting the CM-SAT instance of equation 9.4.SAT2 to a fault detection problem.

9.5 Testing of Units with Memory

Even leaving the high complexity of test generation aside, it is still the case that exponentially many (up to 2^n) test patterns may be required for an n-input combinational circuit. This may lead to a significant amount of time spent in applying and analyzing tests. The presence of memory in the circuit expands the number of required test cases, given that the circuit behavior is influenced by its state. To test a sequential machine, we may need to apply different input sequences for each possible initial state. This double-exponential complexity may render testing with 100% coverge impractical.

Memory devices are special sequential circuits for which a wide variety of testing strategies have been devised. A simple-minded approach would be to write the all-0s and all-1s bit-patterns into every memory word and read out to verify proper storage and retrieval. This seems to ensure that every memory cell is capable of storing and correctly reading out the bit values 0 and 1. The problems with this approach include the fact that it does not test the memory access/decoding mechanism (there is no way to know that the intended word was written into and retrieved) and does not allow for pattern-sensitive faults, where cell operation is affected by the values stored in nearby cells. Furthermore, modern high-density memories experience dynamic faults that are exposed only for specific access sequences.

The optimal way of testing memory units changes with each technology shift and evolutionary step in their development. Given the trends in increasing size and sensitivity, both arising from higher densities, built-in self-test appears to be the only viable approach in the long run. A particular challenge in applying built-in self-test methods is that any such method consumes some memory bandwidth, thus requiring some sacrifice in performance. Memory testing continues to be an active research area.

9.6 Off-Line vs. Concurrent Testing

This section not yet written.

Problems

9.1 Testing of binary adders

A half-adder (HA) consists of an XOR gate and an AND gate producing the sum and carry-out bits, respectively, when adding two input bits x and y. A full-adder (FA), with an additional carry-in input, can be built from two HAs and an OR gate.

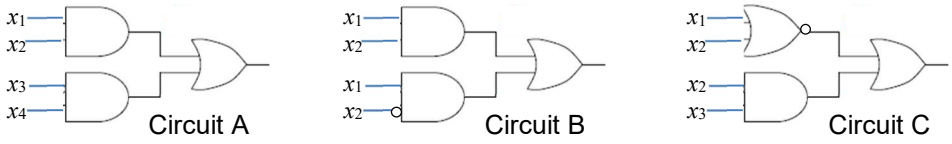
a. Discuss functional and structural testing of a single-bit FA built as above.

b. Repeat part a for a 4-bit ripple-carry adder built of four FAs.

c. How would you enhance the testability of the 4-bit ripple-carry adder if you were allowed to use only one extra pin?

9.2 Path sensitization and D-algorithm

Consider circuits A, B, and C. Answer the following questions using path sensitization and D-algorithm.



a. Identify s-a-0 and s-a-1 tests for the input x_3 in circuit A.

b. Identify s-a-0 and s-a-1 tests for the input x_2 in circuit B.

c. Identify s-a-0 and s-a-1 tests for the input x_2 in circuit C.

9.3 Boolean difference techniques

Redo Problem 9.2 using Boolean difference techniques.

9.4 Testing of XOR circuits

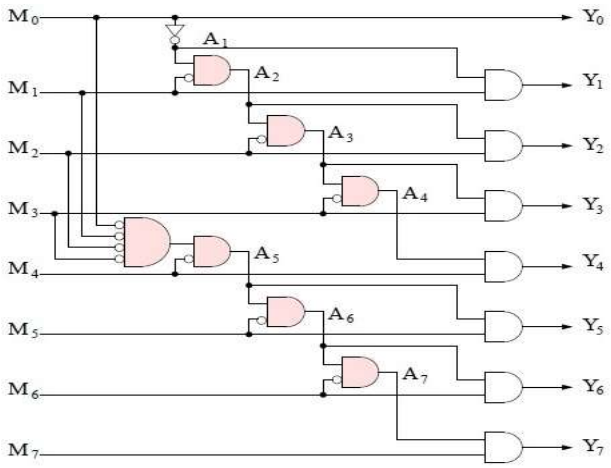
Show that a tree of 2-input XOR gates, implementing the logic function $x_1 \oplus x_2 \oplus \ldots \oplus x_n$, can be tested for all single s-a-0 and s-a-1 faults with only three input test patterns, regardless of the number n of inputs. *Hint:* A single test detects all single s-a-1 faults.

9.5 Testing for stuck-at faults

Consider the following 8-input, 8-output circuit.

a. Derive a set of tests for detecting all stuck-at faults.

b. Describe the function performed by the circuit and relate your response to the tests in part a.



9.6 Testing with false positives

For digital circuits, tests may be incomplete, in the sense of missing some of the possible faults, but we usually don't have to worry about a test incorrectly identifying a healthy circuit as faulty, an outcome known as "false positive." In other areas, such as tests for diseases, false positives are common and must be taken into account in assessing the effectiveness of testing. Consider a diagnostic test for a particular disease that gives the result "positive" with 99% probability if the person tested has the disease and with 2% probability (false positive) if not. Assume that 1% of the residents of a city have that particular disease. A randomly chosen person from the city is administered the test.

a. What is the probability that the test result is positive?

b. If the test result is positive, what is the probability that the person tested has the disease?

c. If the test result is negative, what is the probability that the person tested has the disease?

9.x Title

Intro

a. xxx

b. xxx

c. xxx

d. xxx

References and Further Readings

[Alar06] Al-ars, Z., S. Hamdioui, and A. J. Goor, "Space of DRAM Fault Models and Corresponding Testing," *Proc. Design Automation and Test in Europe*, 2006, pp. 1252-1257.

[Cook71] Cook, S., "The Complexity of Theorem Proving Procedures," *Proc. 3rd Annual Symp. Theory of Computing*, 1971, pp. 151-158,

[Fuji82] Fujiwara, H. and S. Toida, "The Complexity of Fault Detection Problems for Combinational Logic Circuits," *IEEE Trans. Computers*, Vol. 31, pp. 555-559, 1982.

[Fuji85] Fujiwara, H., *Logic Testing and Design for Testability*, MIT Press, 1985.

[Goel81] Goel, P., "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," *IEEE Trans. Computers*, Vol. 30, No. 3, pp. 215-222, March 1981.

[Hamd03] Hamdioui, S. and G. Gaydadjiev, "Future Challenges in Memory Testing," *Proc. PRORISC*, 2003, pp. 78-83.

[Ibar75] Ibarra, O. H. and S. K. Sahni, "Polynomially Complete Fault Detection Problems," *IEEE Trans. Computers*, Vol. 24, No. 3, pp. 242-249, March 1975.

[Jha03] Jha, N. and S. Gupta, *Testing of Digital Circuits*, Cambridge Univ. Press, 2003.

[Krst98] Krstic, A. and K.-T. Cheng, *Delay Fault Testing for VLSI Circuits*, Kluwer, 1998.

[Lala97] Lala, P. K., *Digital Circuit Testing and Testability*, Academic Press, 1997.

[Roth66] Roth, J. P., "Diagnosis of Automata Failures: A Calculus and a Method," *IBM J. Research and Development*, Vol. 10, No. 4, pp. 278-291, July 1966.

[Wang06] Wang, L.-T., C.-W. Wu, and X. Wen (eds.), *VLSI Test Principles and Architectures: Design for Testability*, Elsevier, 2006.

10 Fault Masking

"Don't find fault with what you don't understand."

French proverb

"Life is [like a piano]. The discord is there, and the harmony is there. Study to play it correctly, and it will give forth the beauty; play it falsely, and it will give forth the ugliness. Life is not at fault."

Anonymous

Topics in This Chapter

Fault masking can also be called fault tolerance, but we avoid using the latter term because of its past association with the entire field of dependable computing. There are two ways to mask faults. One way is to build upon the inherent redundancy in logic circuits. This is akin to the redundancy observed in natural language: you may be able to understand a sentence in this book even after removing every vowel, or covering the lower half of every letter. Another way is by using replicated circuits whose outputs feed a fusion or combining circuit, often (inappropriately) called a voter. This approach leads to static, dynamic, and hybrid redundancy methods. Masking of transient faults is also discussed.

10.1 Fault Avoidance vs. Masking

Referring to Fig. 10.1, we note that there is a pleasant symmetry between avoidance and masking strategies for dealing with faults (as explained in the introductory paragraph of this chapter, we prefer to use fault masking in lieu of fault tolerance). Certain faults may be unavoidable, while others may turn out to be unmaskable. Thus, practical strategies for dealing with faults are often hybrid in that they encompass some avoidance and some masking techniques.

In the 9-leaf tree of Fig. 10.1, this chapter's focus will be on the rightmost two leaves labeled "restored" and "unaffected." Masking through restoration requires that a fault be exposed, promptly detected by a monitor, and fully circumvented via reconfiguration. This approach requires the application of *dynamic redundancy*, where redundant features are built into the system but are not brought into service until they are needed to replace faulty elements. Masking through concealment is based on *static redundancy* methods, whereby redundant elements are fully integrated into the circuit in a way that they cover for imperfections in other elements.

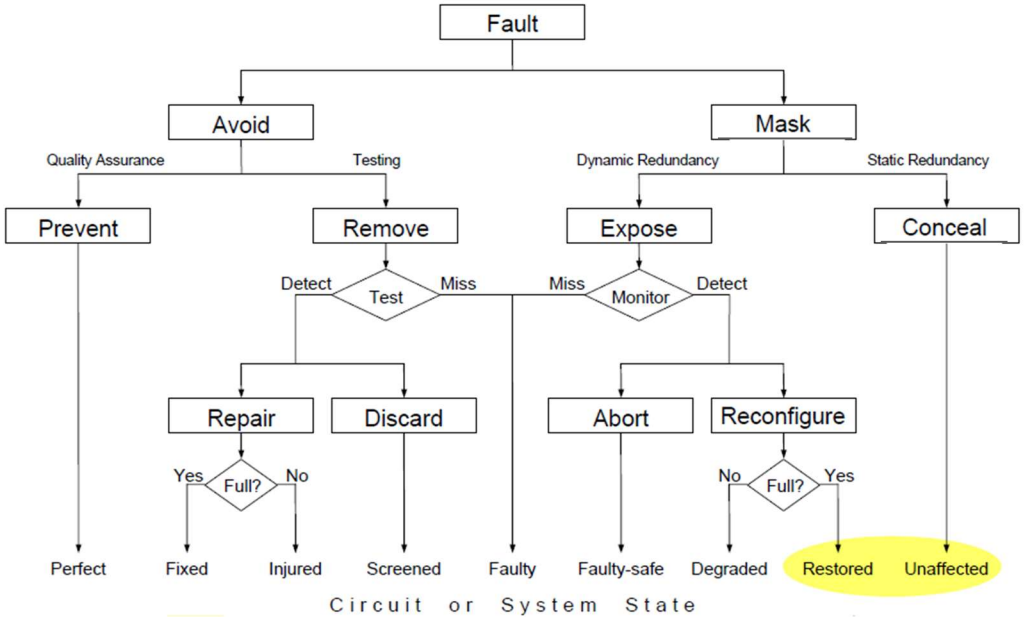
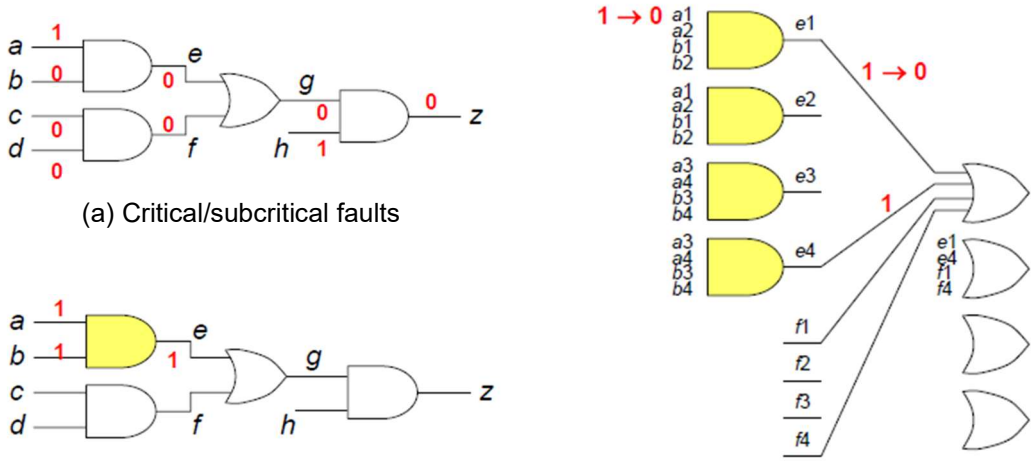


Fig. 10.1 Fault avoidance versus masking: a symmetrical view.

10.2 Interwoven Redundant Logic

Interwoven redundant logic is based on the notions of critical and subcritical faults. Referring to Fig. 10.quad-a, we see that with the given input values, if line *a* is s-a-0, the circuit output will not change. We thus consider *a* s-a-0 a *subcritical fault*. The same holds true for *h* s-a-0, *c* s-a-1, and *d* s-a-1. On the other hand, line *b* s-a-1 will change the circuit output from 0 to 1, making it a *critical fault* for the input pattern shown. Because not all faults are of the stuck-at type, henceforth we characterize each fault as a $0 \rightarrow 1$ flip or a $1 \rightarrow 0$ flip.

We see from the preceding discussion that even a nonredundant logic circuit is capable of masking some faults. This is both good and bad. It is good in the sense that not every fault will affect the correct functioning of the circuit. It is bad in the sense of impacting the timeliness of fault detection. Generally speaking, an AND gate with many inputs is more sensitive to $1 \rightarrow 0$ flips, because they have the potential of causing a $1 \rightarrow 0$ flip at the gate's output. A multi-input OR gate, on the other hand, is more sensitive to $0 \rightarrow 1$ flips. We are thus motivated to seek methods involving alternating use of AND and OR gates for turning this inherent masking capability of logic circuits into a general scheme that also masks critical faults.



(a) Critical/subcritical faults

(b) Original logic circuit

(c) Part of the quadded version

Fig. 10.quad The notion of critical/subcritical faults and connectivity pattern in interwoven 4-way-redundant logic.

Consider, for example, the logic circuit of Fig. 10.quad-b with potentially critical $1 \rightarrow 0$ flips on either input of the shaded AND gate at the top left. Suppose we quadruplicate this AND gate and the OR gate which it feeds, as well as all inputs and other signals, as depicted in Fig. 10.quad-c. The four copies of any signal x are named $x1$, $x2$, $x3$, and $x4$, with the value of x taken to be the 3-out-of-4 majority value among the four signal copies. The connectivity pattern of inputs and other replicated signals is such that any critical flip at the AND layer turns into a subcritical flip at the following OR layer. For example, the critical $1 \rightarrow 0$ flip for $a1$ causes the subcritical $1 \rightarrow 0$ flip at the top input $e1$ to the OR gate it feeds at the next circuit layer.

One can show that to mask h critical faults with this alternating, interwoven AND-OR arrangement, the number of gates must be multiplied by $(h + 1)^2$ and the number of inputs for each gate must increase by the factor $h + 1$. For $h = 1$, this interwoven redundant logic scheme is known as *quadded logic*. Note that the alternating AND and OR layers can be replaced by uniform NAND layers in the usual way.

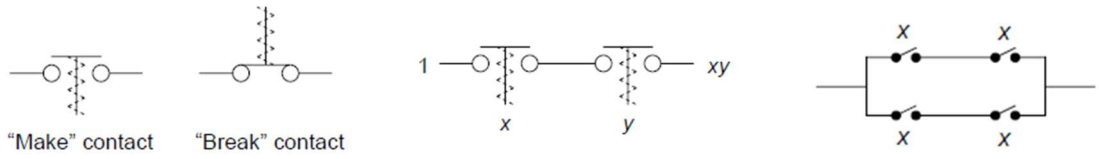
A similar masking effect is observed in the crummy-relays scheme of Moore and Shannon [Moor56]. We consider relays of two kinds (Fig. 10.relay-a): a make contact is normally open and closes when energized, while a break contact is normally closed and opens when energized. Given two make contacts, their series connection computes the AND function, as shown in Fig. 10.relay-b. We define the two probabilities a and c for each relay as follows:

a	prob [contact made | relay is energized]
$1 - a$	prob [contact open | relay is energized]
c	prob [contact made | relay is not energized]
$1 - c$	prob [contact open | relay is not energized]

For make contacts, we have $a > c$, while for break contacts $a < c$ holds. No matter how crummy the relays, that is, how close the values of a and c to each other, one can interconnect many of them in a redundant structure, using series and parallel elements, to achieve an arbitrarily high reliability.

Example 10.M&S: Reliable circuits from crummy relays Consider the parallel-series quad of make relays depicted in Fig. 10.relay-c. Derive the reliability paramters for the quad and determine under what conditions it behaves better than a single make relay.

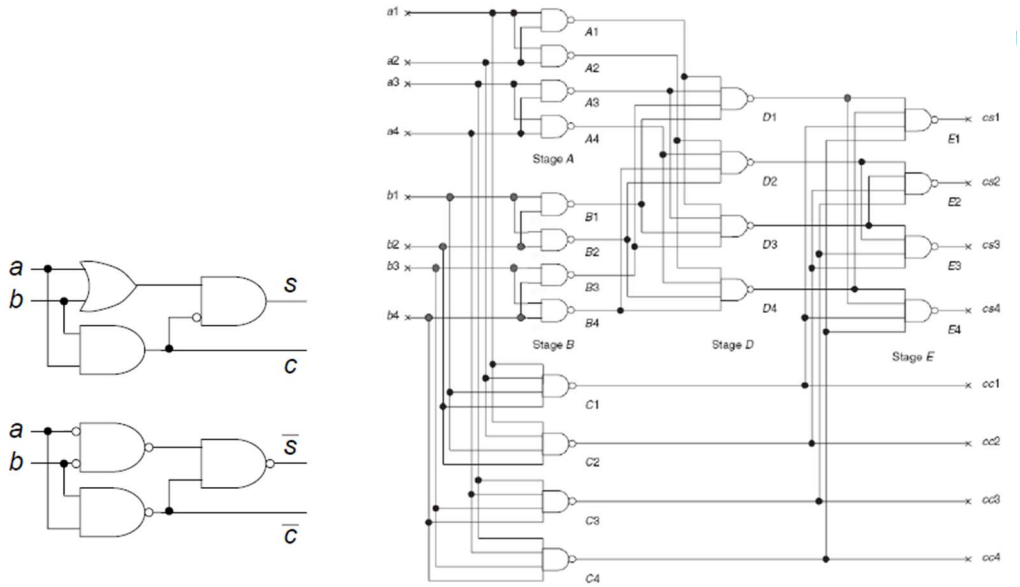
Solution: The a parameter of the quad circuit can be derived to be a_{quad} = prob [connection made | relays energized] = $2a^2 - a^4$. Thus, $a_{quad} > a$ if $a > 0.62$. Similarly, the quad's c parameter can be shown to be c_{quad} = prob [connection made | relays not energized] = $2c^2 - c^4$. It is readily seen that $c_{quad} < c$ for all values of c. Thus, unless the a parameter has an unreasonably low value, the quad circuit offers better reliability than a single make relay.



(a) Relay contact types (b) AND circuit (c) Parallel-series quad

Fig. 10.relay Relays and their use in building an AND circuit and a more reliable make circuit.

Interwoven logic has been proposed as a way of overcoming highly unreliable nanoelectronic circuits and their attendant massive numbers of defects that are inevitable in the mass production of extremely small components. Figure 10.nano-a depicts a half-adder circuit, with AND-OR and NAND-only implementations. The interwoven logic version of the half-adder is depicted in Fig. 10.nano-b, where each of the three 2-input NAND gates is replaced by four 4-input NAN gates and additional NAND gates act as inverters for the signal bundles representing the a and b inputs.



(a) Nonredundant half-adders (b) Interwoven logic version

Fig. 10.nano Interwoven logic for nanoelectronics [Han05].

10.3 Static Redundancy with Replication

A conceptually simple static redundancy scheme is triple-modular redundancy (TMR), which as shown in Fig. 10.tmr-a consists of triplicating the computational module and voting on the results they produce. The voter masks any fault that is limited to only one of the modules. Two faults can potentially lead to an incorrect output from the voter, so denoting the module reliability by R_m, a worst-case reliability analysis leads to:

$$R_{TMR} = 3R_m^2 - 2R_m^3 = R_m[1 + (1 - R_m)(2R_m - 1)] \qquad (10.3.RTMR)$$

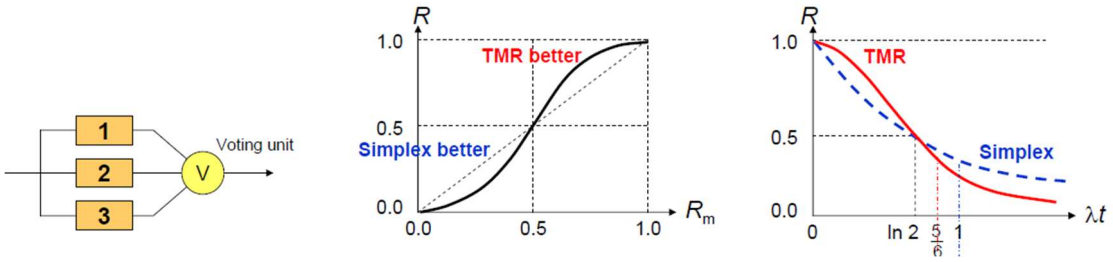
The variation of R_{TMR} as a function of R_m is shown in Fig. 10.tmr-b. The second expression for R_{TMR} in equation (10.3.RTMR) suggests that for the TMR scheme to lead to reliability improvement over a nonredundant module, that is, for $R_{TMR} > R_m$, we must have $(1 - R_m)(2R_m - 1) > 0$ or $R_m > 1/2$. The reliability improvement factor achieved by the TMR scheme is:

$$\text{RIF}_{\text{TMR/Simplex}} = \frac{1-R_m}{1-R_{TMR}} = \frac{1}{1-R_m(2R_m-1)} \qquad (10.3.RIF)$$

Assuming $R_m = e^{-\lambda t}$, the reliability curves for a simplex module and the TMR system are shown as a function of λt in Fig. 10.TMR-c. We know that the MTTF for the simplex system is $1/\lambda$. The MTTF for the TMR system can be shown to be:

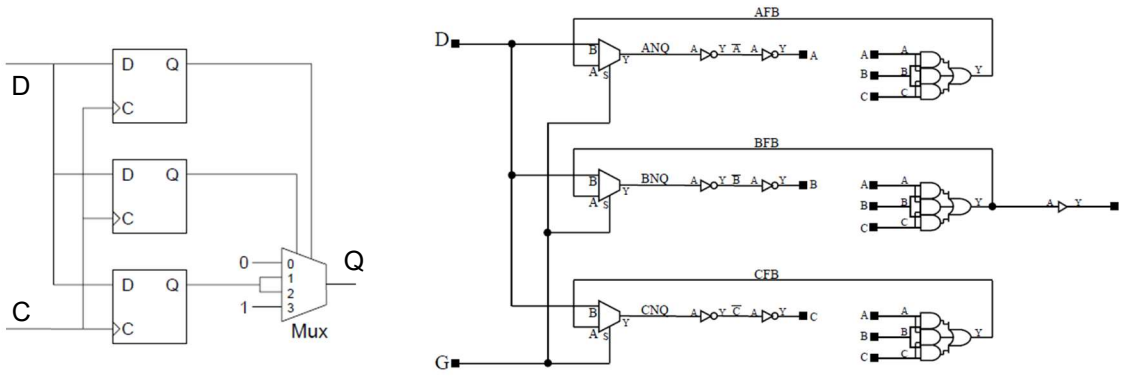
$$\text{MTTF}_{TMR} = (5/6)\lambda \qquad (10.3.MTTF)$$

The shorter MTTF of a TMR system is due to the fact that R_{TMR} falls below R_m and stays there for $\lambda t > \ln 2$, thus leading to a smaller area under the reliability curve.



(a) Simple TMR scheme (b) Reliability vs. R_m (c) Reliability over time

Fig. 10.tmr Relays and their use in building an AND circuit and a more reliable make circuit.



(a) (b)

Fig. 10.seuFF Two types of triplicated flip-flops designed to withstand single event upsets (SEUs).

One recent application of TMR is in the design of flip-flops that are designed to withstand single event upsets (SEUs). As shown in Fig. 10.seuFF, triplication of the FF itself, or both the FF and its correction circuitry, results in single faults to be tolerated.

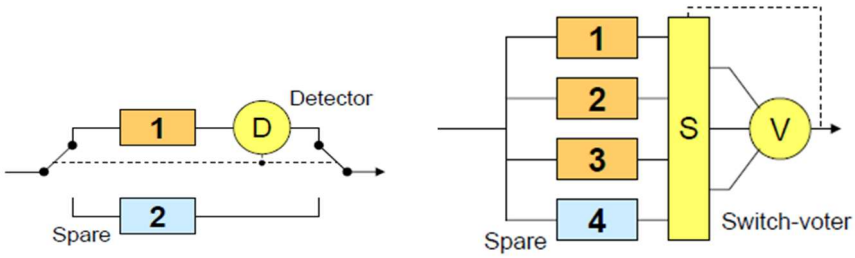
Generalizing triple-modular redundancy to N units, yields N-modular reudnancy (NMR). The parameter N is often taken to be an odd number, allowing the use of an $(N + 1)/2$-out-of-N voter in 5MR, 7MR, and so on. One must note, however that voter complexity rises rapidly with increasing N. It is also possible to use an even value for N. For example, we may use 3-out-of-4 voting in 4MR and also design the voter such that it detects the presence of double faults, leading to greater reliability and safety compared with TMR.

10.4 Dynamic and Hybrid Redundancy

Dynamic redundancy, as the name implies, requires some sort of action to bypass a faulty element, once a fault has been detected. The simplest form of dynamic redundancy entails a fault detector tacked on to an operating unit and a mechanism that allows switching to an alternate (standby, spare) unit (Fig. 10.dynr-a). In contrast, static or masking redundancy (Fig. 10.tmr-a) simply hides the effects of a fault and continues operation uninterrupted, provided that the scheme's masking capacity is not exceeded.

The fault detector in Fig. 10.dynr-a may be of various kinds: a code checker, a watchdog timer, or a comparator, in the event that the operational unit is itself duplicated. The latter scheme is sometimes referred to as "pair and spare."
[Elaborate further on fault detection.]

The standby or spare unit may be "cold," meaning that it is powered off until needed. When the spare is to be switched in, it must be powered on, initialized, and set to an appropriate state to be able to continue the work of the operational unit from where it was interrupted (preferable) or from a restart position. Alternatively, we may use a "hot" spare which runs in parallel with the operational unit and is thus in sync with it. A hot spare can be switched in with minimal delay, thus helping to improve availability. We may also opt for the intermediate case of a "warm" spare that is powered on but not completely in sync with the operational unit.



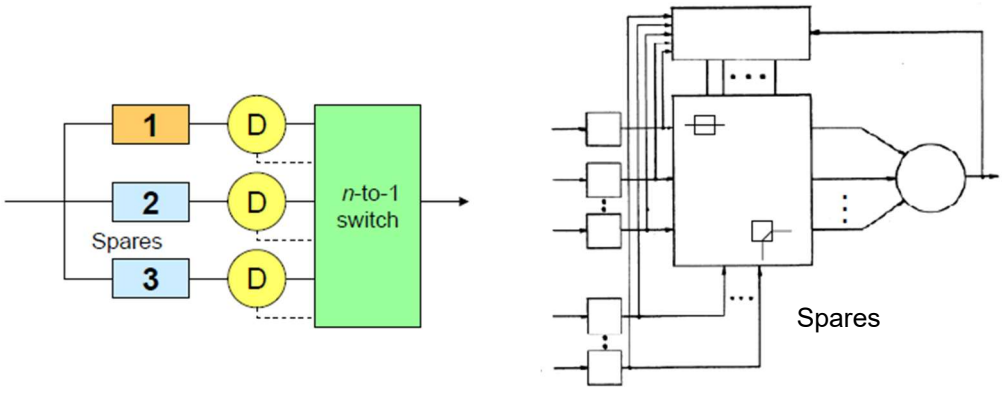
(a) Detect and replace (b) Mask, diagnose, and switch

Fig. 10.dynr Relays and their use in building an AND circuit and a more reliable make circuit.

We can combine the advantages of static (uninterrupted operation) with those of dynamic redundancy (lower hardware and energy costs) into a hybrid redundancy scheme, as depicted in Fig. 10-dynr-b. Initually, the reconfiguration switch S is set so that the outputs of units 1, 2, and 3 are selected and sent to the voting circuit. When a fault occurs in one of the 3 operational units, the voting circuit masks the fault, but the voter output allows the switch S to determine which of the operational units disagreed with the final outcome. The disagreeing unit is then replaced with the spare, allowing the hybrid-redundant system to continue its operation and to tolerate a second fault later on.

It is possible to igmore the first few instance of disagreement from one unit in the expectation that they were due to transient faults. In this scheme, the switch maintains a disagreement tally for each operational unit and replaces it only if the tally exceeds a preset threshold. Another optimization is to switch to duplex or simplex operation when the supply of spares has been exhausted and one of the operational units experiences a fault. Continuing with duplex operation provides greater safety, whereas switching to simplex mode extends the lifetime of the system.

Figure 10.sw depicts high-level designs of switches for standby and hybrid redundancy. [Elaborate on switch design, including the self-purging variant and the associated threshold voting scheme.]



(a) Standby (b) Hybrid

Fig. 10.sw Switches for standby and hybrid redundancy.

We now summarize our discussion of static versus dynamic (and hybrid) redundancy. Static redundancy provides immediate masking and thus uninterrupted operation. It is also high on safety. Its disadvantages include power and area penalties and the fact that the voting circuit is critical to correct operation of the system. Dynamic redundancy consumes less power (especially with cold standbys) and provides longer life by simply adding more spares. Tolerance is not immediate, causing availability and safety concerns. Also, the assumption of longer life with more spares is critically dependant on the coverage factor. In the absence of near-perfect coverage, the addition of more spares may not help or even be detrimental to system reliability. Hybrid redundancy have some of the advantages of both schemes, as well as some of their disadvantages. The switch-voter part of a hybrid-redundant system is both complex and critical.

We will see in Chapter 15 that, via a scheme known as self-checking design, fault detection coverage of standby redundancy can be improved, making the technique more attractive from a practical standpoint.

10.5 Time Redundancy

Instead of replicating a circuit or module, one may reuse the same unit multiple times to derive several results. This strategy, sometimes referred to as *retry*, is particularly effective for dealing with transient faults. Clearly, if the unit has a permanent fault, then the same erroneous result will be obtained during each retry.

One way around the problem presented by permanent faults is to change things around during recomputations so as to make it less likely for the same fault to lead to identical errors, thus reducing the likelihood of the fault going undetected. For example, if an adder is being used to compute the sum $a + b$, we may switch the operands and compute $b + a$ during the second run, or we may complement the inputs and output, thus aiming to obtain the same result via the computation $-(-a - b)$. Similarly, in multiplying a by the even number $2b$, we may try computing $(2a) \times b$ the second time. For arbitrary integer oprands a and b, we can find the product $(2a) \times \lfloor b/2 \rfloor$, while initializing the cumulative partial product to a if $b_0 = 1$ and to 0 otherwise. Yet another alternative, assuming that a and b are not very large, is to compute $(2a) \times (2b)$ and divide the result by 4 through right-shifting by 2 bits.

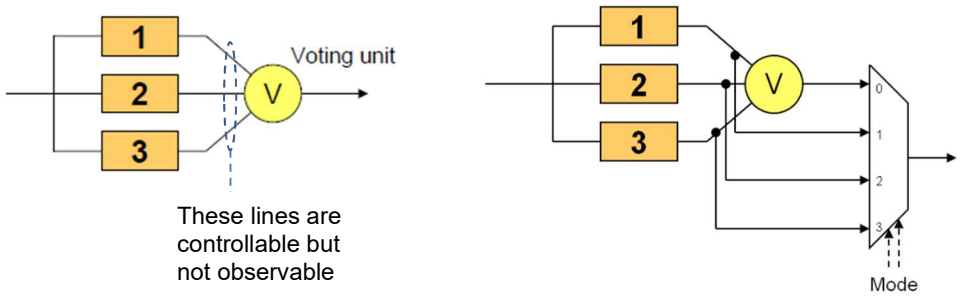
The final example in the preceding paragraph is an instance of the method of recomputing with shifted operatnds.

10.6 Variations and Complications

Both NMR and hybrid hardware redundancy have been used in practice. The Japanese Shinkansen "Bullet" train employed a triple-duplex control system, implying 6-fold redundancy. [Elaborate on the redundancy scheme.] Before they were permanently retired in 2012, NASA's Space Shuttles used computers with 5-way redundancy in hardware and 2-way reudnancy in software. The 5 hardware units originally consisted of 3 concurrently operating units and 2 spare units. Later, the configuration was changed to 4 + 1. Two independently developed software systems were used to protect against software design faults.

One consequence of using static redundancy is reduced testability. The very act of masking faults makes it impossible to detect them via the circuit's inputs and outputs. Consider, for example, the quadded logic scheme of Section 10.2. This scheme is designed to mask a single stuck-at fault. So what happenes if the redundant circuit already contains a fault at the time of its manufacture? The fault cannot be detected by simply testing the redundant circuit, and if a second fault develops during use (which is really the first fault from the user's viewpoint), it may not be masked. Similar difficulties arise with regard to replication-based redundant systems. Thus, incorporating testability features, of the types discussed in Chapter 11, is even more significant for systems employing masking redundancy.

As an example, consider the TMR system of Fig. 10.tmrt-a. If any of the three units becomes faulty after system assembly but prior to its use, then the first fault occurring during the system's operation may lead to an incorrect output. This is because the presence of the first fault in not detectable using only the circuit's primary inputs and outputs. This problem can be fixed by using the arrangement depicted in Fig. 10.tmrt-b. A multiplexer is used to select one of four values as the output from the redundant circuit: option 0 selects the value produced by the voting unit, while options 1-3 select the output supplied by one of the units 1-3. The only detrimental effect of this arrangement is the added multiplexer delay during normal operation of the redundant circuit. This is more than offset by the facility for testing each of the replicas initially as well as periodically during system operation.



(a) Testability problem (b) Testability improvement

Fig. 10.tmrt TMR configuration with provisions for testing.

Problems

10.1 Title

Intro

a. xxx
b. xxx
c. xxx
d. xxx

10.2 MTTF of a TMR system

Prove equation (10.3.MTTF), showing that the MTTF of a TMR system with perfect voting is 5/6 of the MTTF of a simplex module.

10.3 Comparing fault-masking schemes

We are considering two possible implementations of a 4-way redundant system. The redundancy is 300% in both cases, so ignoring the complexity of comparison, voting, and switching mechanisms, cost is not a factor in our decision. Option 1 is the use of pair-and-spare scheme with comparison used for fault detection and the active pair replaced with the spare pair upon a detected fault. Option 2 is triplication with voting and one spare. For both cases, repair is performed to bring the faulty unit or pair back into service. In answering the following questions, no calculation is necessary. However, answers based on quantitative evidence will receive extra credit.

a. Which scheme is preferable with respect to reliability in a safety-critical setting?
b. Which scheme is preferable with respect to system availability?

10.4 Title

Intro

a. xxx
b. xxx
c. xxx
d. xxx

10.5 Modeling for fault masking

Consider the following fault-masking redundancy scheme with voting. There are five modules that feed a 3-out-ot-5 voting unit. When a disagreement is detected among the modules, the disagreeing module is purged (switched out), along with one of the good ones, and the voting unit is reconfigured to act as a 2-out-of-3 unit. The next detected disagreement causes the disagreeing module and one of the good ones to be purged, leaving a simplex system. As usual, we ignore the possibility of simultaneous multiple faults.

a. Construct and solve a state-space reliability model for this system, assuming perfect coverage, switching, and voting. State all your assumptions clearly.
b. Outline a method for improving the reliability of this hardware redundancy scheme that does not involve adding extra modules (only the switching and voting parts can change).

10.6 Comparison-based sorting networks

A comparator is a 2-input, 2-ouput logic circuit that receives the integers a and b as inputs and delivers $c = \min(a, b)$ and $d = \max(a, b)$ as outputs. Thus, a comparator either sends the two inputs straight through or exchanges them to achieve correctly ordered outputs.

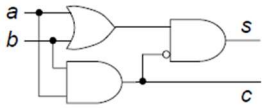
a. Show the design of a 4-input sorting network using 5 comparators.

b. Assume that comparators can get "stuck on straight through" or "stuck on exchange," thus not providing the correct ordering in some cases. How can the network of part a be tested for such stuck-type faults efficiently, assuming at most a single faulty comparator?

c. Is it possible to build a 4-input sorting network that masks the fault of any single comparator? How, or why not?

10.7 Quadded logic with NAND gates

Consider the half-adder circuit shown below.



a. Convert the circuit so that only NAND gates are used. Feel free to change the outputs to $\bar{s}$ and $\bar{c}$ if it makes your job easier.

b. Draw the quadded version of the circuit in part a, using only NAND gates.

c. Is it always possible to convert a quadded AND-OR circuit into a quadded NAND circuit by replacing each AND and OR gate with a NAND gate?

10.8 Voting in NMR systems

a. Does 2-out-of-4 voting make sense in a 4MR system? Discuss.

b. Generalize your answer and discussion for part a to K-out-of N voting in an NMR system, where K isn't $\lceil (N + 1)/2 \rceil$. Consider both the cases of $K \leq N/2$ and $K > N/2 + 1$.

10.9 Design of voters for NMR systems

a. Is it possible to design a 3-out-of-5 voter, using only 2-out-of-3 voters and no other component? Fully justify your answer.

b. Generalize your answer to part a for the case of designing an m-out-of-n voter from 2-out-of-3 voters.

c. Design an efficient 3-out-of-5 voter using 2-out-of-3 voters and inverters, if helpful.

d. Repeat part c for a 5-out-of-9 voter.

References and Further Readings

[Han05] Han, J., J. Gao, P. Jonker, Q. Yan, and J. A. B. Fortes, "Toward Hardware-Redundant Fault-Tolerant Logic for Nanoelectronics," *IEEE Design & Test of Computers*, Vol. 22, No. 4, pp. 328-339, July-August 2005.

[Han15] Han, J., E. Leung, L. Liu, and F. Lombardi, "A Fault-Tolerant Technique Using Quadded Logic and Quadded Transistors," *IEEE Trans. VLSI Systems*, Vol. 23, No. 8, pp. 1562-1566, August 2015.

[Hsu95] Hsu, Y.-M., E. E. Swartzlander Jr., and V. Piuri, "Recomputing by Operand Exchanging: A Time Redundancy Approach for Fault-Tolerant Neural Networks," *Proc. IEEE Int'l Conf. Application-Specific Array Processors*, 1995, pp. 54-

[Jens63] Jensen, P. A., "Quadded NOR Logic," *IEEE Trans. Reliability*, Vol. 12, pp. 22-31, September 1963.

[Lyon62] Lyons, R. E. and W. Vanderkulk, "The Use of Triple Modular Redundancy to Improve Computer Reliability," *IBM J. Research and Development*, Vol. 6, No. 2, pp. 200-209, April 1962.

[Mukh15] Mukherjee, A. and A. S. Dhar, "Fault Tolerant Architecture Design Using Quad-Gate-Transistor Redundancy," *IET Circuits, Devices & Systems*, Vol. 9, No. 3, pp. 152-160, May 2015.

[Pier65] Pierce, W. H., *Failure-Tolerant Computer Design*, Academic Press, 1965.

[Tryo62] Tryon, J. G., "Quadded Logic," in *Redundancy Techniques for Computing Systems*, R. H. Wilcox and W.C. Mann (eds.), Spartan, 1962, pp. 205-228.

11 Design for Testability

"The real fault is to have faults and not to amend them."

Confucius

"When something goes wrong with a computer program or an engineering structure, the scrutiny under which the ill-fated object comes often uncovers a host of other innocuous bugs and faults that might have gone forever unnoticed had the accident not happened."

Henry Petroski, To Engineer Is Human, 1985

Topics in This Chapter
11.1. The Importance of Testablity
11.2. Testability Modeling
11.3. Testpoint Insertion
11.4. Sequential Scan Techniques
11.5. Boundary Scan Design
11.6. Built-in Self-Test

Highly complex digital systems cannot be tested exhaustively to validate their functionalities. Adequate structural testing is also often impractical, given the limited number of primary inputs and outputs that are externally controllable and observable. It has thus become necessary to incorporate testability features in modern digital circuits so as to simplify the testing process and to reduce its cost. Collectively, these features provide better access to the normally hidden internal nodes, thus making it possible to control and observe them without disassembling the circuit under test. Recently, boundary scan design has become the de facto standard for making an assembly of communicating modules testable.

11.1 The Importance of Testability

Whereas a simple circuit with small set of inputs and outputs can be tested with a reasonable amount of time and effort, a complex unit, such as a microprocessor, would be impossible to test solely based on its input/output behavior. Hence, nearly all modern digital circuits are designed and built with special provisions for improved testability.

Because the cost of a fault is greatly reduced when it is caught early, as discussed in Section 9.1, testability features appear at all levels of the digital system packaging hierarchy, from the component or chip level, through board and subassembly levels, all the way to the system level. Our discussions in this chapter pertain mostly to circuit- and board-level techniques, but similar considerations apply elsewhere. In particular, diagnosability features form important considerations at the malfunction level (see Chapter 17).

In order for a unit to be easily testable, we must be able to control and observe its internal points. Thus, *controllability* and *observability* are the cornerstones of testability, as we will see in detail in Section 11.2.

11.2 Testability Modeling

To allow detection of a fault on line L_i of a logic circuit (see Fig. 11.1), we must be able both to control that line from the circuit's primary inputs and to observe its behavior via the primary outputs. Thus, good testability requires good controllability and good obsevability. It is thus natural to define the testability $T(P_i)$ of a line L_i as the product of its controllability $C(L_i)$ and its observability $O(L_i)$. We will discuss suitable quantification of controllability and observability shortly. However, assuming that we have already determined C and O values in the interval [0, 1] for each line L_i, $1 \le i \le n$, within the circuit, the overall testability of the circuit can be taken to be the average of the testabilities for all points. Thus:

$$T_{\text{circuit}} = \frac{1}{n} \sum_{i=1}^{n} C(L_i) \times O(L_i) \qquad \text{(11.2.test)}$$

Note that testability metric defined by equation (11.2.test) is an empirical measure that does not have a physical significance. In other words, if the testability of a circuit turns out to be 0.23, it does not tell us much about how difficult it would be to test the circuit. It is only useful for comparing different circuits, different designs for the same circuit, or different points within a circuit with regard to ease of testing. So, a design with a testability of 0.23 may be preferred to one that has a testability of 0.16, say.

To determine the line controllabilities within a given circuit, we begin from primary inputs, each of which is assigned the perfect controllability of 1, and proceed toward the outputs. The controllability of the single output line of a k-input gate is the product of the gate's controllability transfer factor (CTF) and the average of its input controllabilities:

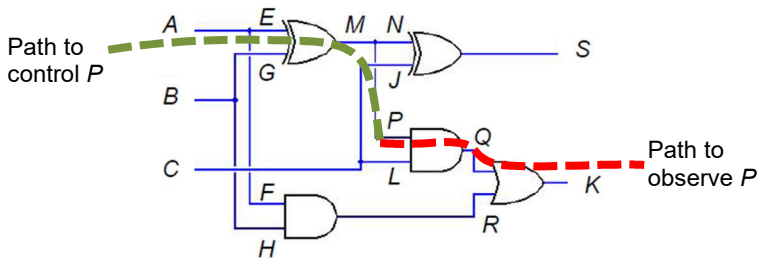
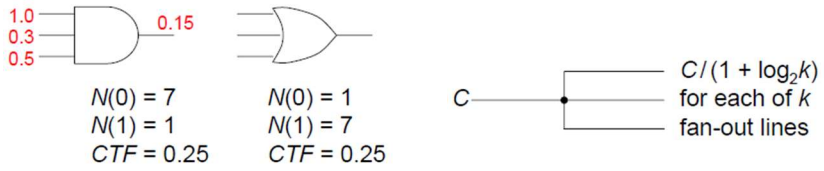


Fig. 11.1 **Good testability with respect to an internal circuit fault on line *P* requires good controllability over *P* from the inputs and good observability of its behavior from the outputs.**



(a) Gate output controllability (b) Controllability after fan-out

Fig. 11.2 Quantifying the controllability of a circuit line.

$$C_{\text{output}} = CTF \times \sum_i C_{\text{input } i}/k \qquad \text{(11.2.contr)}$$

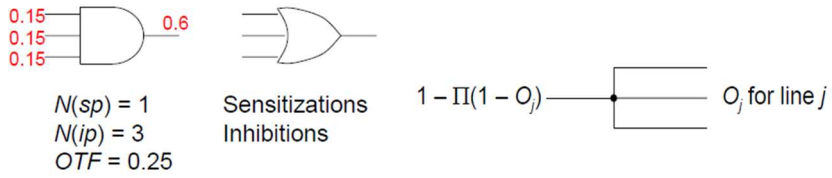
The *CTF* of a gate depends on how easy it is to set its output to 0 or 1 at will. Taking a 3-input AND gate of Fig. 11.2a as an example, 7 of its 8 input patterns set the output to 0 and only one pattern sets the output to 1. The relatively large difference between $N(0) = 7$ and $N(1) = 1$ indicates poor controllability at the gate's output. In general, we use equation 11.2.CTF to derive a gate's *CTF*:

$$CTF = 1 - \left|\frac{N(0)-N(1)}{N(0)+N(1)}\right| \qquad \text{(11.2.CTF)}$$

When an equal number of input patterns set the gate's output to 0 or 1, we have $N(0) = N(1)$, leading to the perfect *CTF* of 1. An XOR gate has the $N(0) = N(1)$ property and is thus a desirable circuit component in terms of testability. In the case of a line that fans out into f lines, as in Fig. 11.2-b, the controllability of each of the output lines is given by:

$$C_{f\text{-way fan-out}} = C_{\text{input}} / (1 + \log_2 f) \qquad \text{(11.2.fo-c)}$$

A line of very low controllability constitutes a good place for the insertion of a testpoint.



(a) Gate input observability (b) Observability before fan-out

Fig. 11.3 Quantifying the observability of a circuit line.

Example 11.contr: Quantifying controllability Derive the controllabilities of lines *P* and *K* in the logic circuit of Fig. 11.1.

Solution: The XOR gate has a CTF of 1, making the controllability of line *M*, and thus line *P*, equal to 1. Two-input AND and OR gates have a *CTF* of 1/2. Thus both *Q* and *R* have a controllability of 1/2, giving *K* a controllability of 1/4.

To determine the line observabilities within a given circuit, we begin from primary outputs, each of which is assigned the perfect observability of 1, and proceed toward the inputs. The observability of each input line of a *k*-input gate is the product of the gate's observability transfer factor (*OTF*) and its output observability:

$$O_{\text{input } i} = OTF \times O_{\text{output}} \qquad \text{(11.2.obser)}$$

The *OTF* of a gate depends on how easy it is to sensitize a path from a gate input to the output. Taking a 3-input AND gate of Fig. 11.3a as an example and considering a fault on one of the inputs, only $N(sp) = 1$ of the 4 patterns on the other two input sensitizes a path to the output, whereas $N(ip) = 3$ patterns inhibit the propagation. The relatively small number of sensitizing options indicates poor observability of the gate's inputs. In general, we use equation 11.2.OTF to derive a gate's *OTF*:

$$OTF = \frac{N(sp)}{N(sp)+N(ip)} \qquad \text{(11.2.OTF)}$$

When there are no inhibiting input patterns, we have $N(ip) = 0$, leading to the perfect *OTF* of 1. An XOR gate has the $N(ip) = 0$ property and is thus a desirable circuit component in terms of observability. In the case of a line that fans out into *f* lines, as in Fig. 11.3b, the observability of each of the input lines is given by:

$$O_{f\text{-way fan-out}} = 1 - \prod_j (1 - O_j) \qquad \text{(11.2.fo-o)}$$

A line of very low observability constitutes a good place for the insertion of a testpoint.

Example 11.obser: Quantifying observability Derive the observabilities of lines *B* and *P* in the logic circuit of Fig. 11.1.

Solution: Two-input AND and OR gates have an *OTF* of 1/2, leading to an observability of 1/2 for *Q* and 1/4 for *P*, tracing back from the primary output *K*. Line *B* has an observability of 1, given the path consisting of two XORs (*OTF* = 1) from *B* to the primary output *S*.

11.3 Testpoint Insertion

If during testability analysis one or more line in the circuit are shown to have low testabilities, we might want to make those lines externally controllable and observable via testpoint placement. Figure 11.tps shows how the testability of a system composed of two cascaded modules can be improved by inserting degating logic at their interface. In this way, each module can be tested separately and then in tandem to ensure both proper module operation and correct interfacing.

Example 11.test: Placement of a testpoint Derive the testabilities of all lines in Fig. 10.quad-a and from them, deduce the location of a single testpoint that would help most with testability.

Solution: To be provided.

If *m* testpoints are to be placed, it may not be a good idea to pick as their locations the lines with the *m* lowest testability values. This is because locating a testpoint on the lowest-testability line will in general affect the testabilities of many other lines. A good strategy would be to recalculate the testabilities of all circuit lines after the optimal placement of a single testpoint and considering it as a primary input/output.

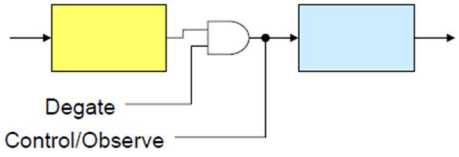
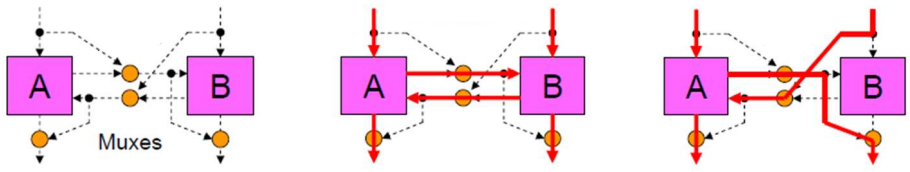


Fig. 11.tps Testpoints improve controllability and observability.



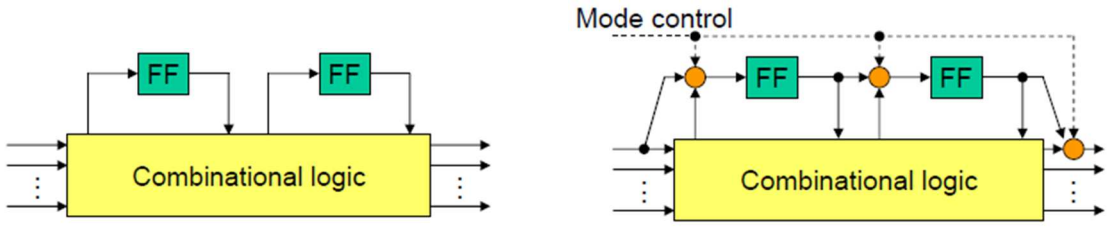
(a) Partitioned design (b) Normal mode (c) Test mode for A

Fig. 11.partn Design partitioning to improve testability.

11.4 Sequential Scan Techniques

Sequential circuits are more difficult to test than combinational circuits, because their behavior is state-dependent. We need exponentially many tests to test the sequential circuit's behavior for each initial condition. One way to reduce the number of test patterns needed is to test the flip-flops' proper operation and the correctness of the combinational part separately. Inputs to the combinational part are the circuit's primary inputs and those coming from the FFs (Fig. 11.scand-a). The ability to load arbitrary bit patterns into the circuit's FFs will allow us to apply desired test patterns to the combinational logic and then observe its response by looking at the primaty outputs and the new contents of the FFs.

Figure 11.scand-b shows one way of accomplishing this aim. All the FFs in the circuit are strung into a long chain, with serial input and serial output. This chain is known as a scan path. There is multiplexer before each FF in the scan path that allows the FF to receive its input from the scan path (test mode) or from the regular source within the circuit (normal operation). During testing, we alternate between test mode and normal mode in many phases. In test mode, we shift a desired pattern into the FFs, while shifting out the stored pattern placed there by the previous phase of testing.

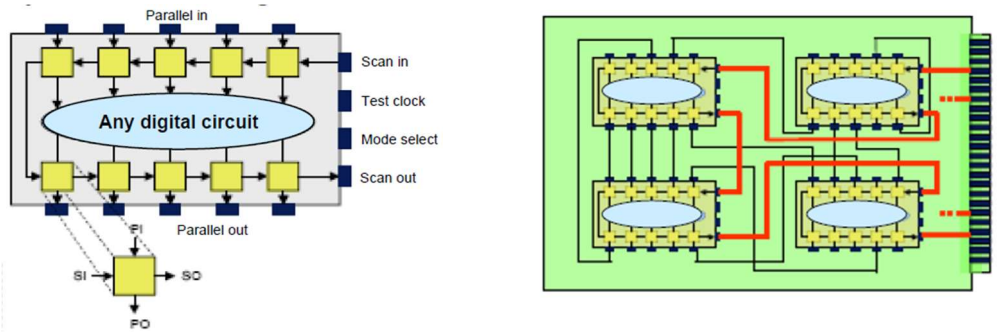


(a) Generic sequential circuit (b) Flip-flops chained in a scan path

Fig. 11.scand Scan design for improving the testability of sequential circuits.

Boundary scan design is a methodology used to facilitate board-level testing of digital systems. For each component to be placed on the board, the input and output lines pass through specially designed *scan cells* (the small square boxes in Fig. 11.scanp-a) that operate in one of two modes. During normal operation, the scan cells are transparent and simply pass the signals from their PI inputs to PO outputs. Thus, during normal operation, the scan cells are invisible and have no effect on the operation of the circuit, except for introducing a slight additional delay resulting from their relaying the signals.

When we want to test the board, . . .



(a) Scan path for a circuit (b) Cascading of scan paths

Fig. 11.scanp Scan path at the circuit and system levels.

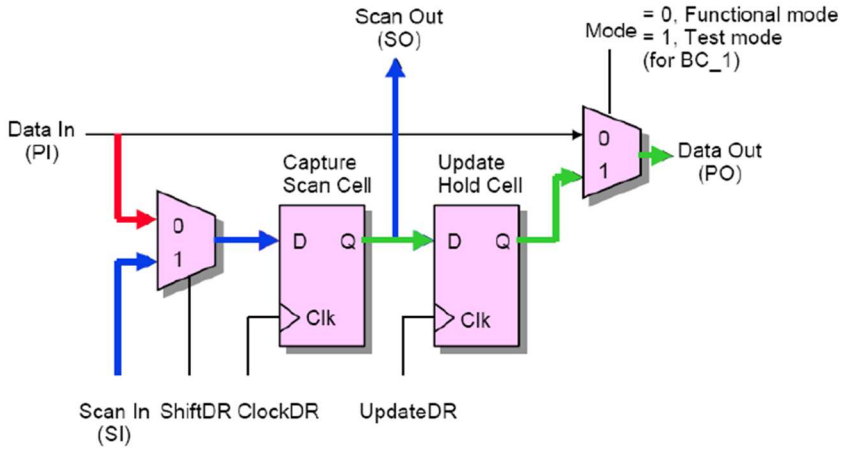


Fig. 11.scanc Basic boundary scan cell.

11.5 Built-in Self-Test

Given the difficulty of testing, built-in self-test (BIST) capability is incorporated into many complex systems. BIST requires the incorporation of test-pattern generation and pass/fail decision into the same package as the circuit under test. As in ordinary testing, the test patterns may be generated randomly at the time of application or may be precomputed and stored in memory. Among the most common methods of on-the-fly test generation is the use of linear feedback shift registers (LFSRs)

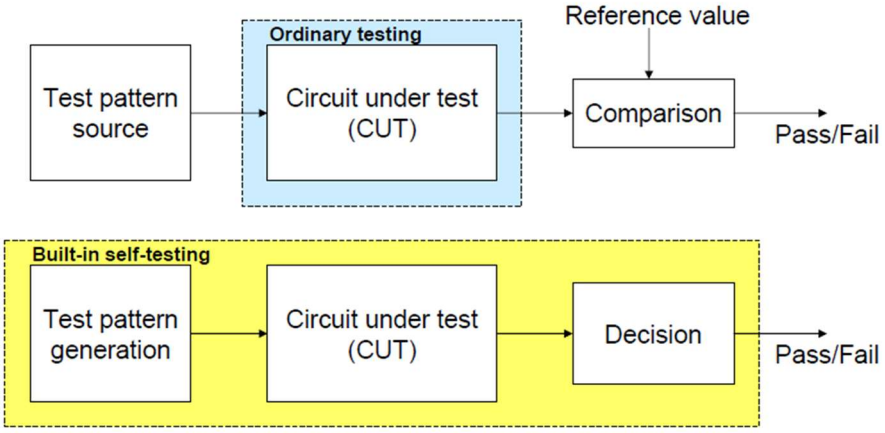


Fig. 11.bist Ordinary testing versus built-in self-test.

11.6 Testing of Analog and Hybrid Circuits

Discuss based on [Liu12].

Problems

11.1 Testability modeling

Quantify the testability of each line of a single-bit full-adder circuit built of two half-adders and an OR gate. Recall that a half-adder consists of an XOR gate and an AND gate producing the sum and carry-out bits, respectively, when adding two input bits x and y. Based on the results obtained, where would you place a single testpoint?

11.2 Testability modeling

a. Suppose we replace a 4-input AND gate (inputs a, b, c, d; output z) with three 2-input AND gates, connected so as to produce the same output $z = abcd$. The 3-gate circuit has internal lines x and y, besides the primary inputs and output. Show how this replacement changes the testabilities of the primary input and output lines.

b. Discuss the reasonableness of the results in part a.

c. Repeat parts a and b with OR gates.

d. Discuss what happens in the case of XOR gates.

11.3 Testability under reconvergent fan-out

a. Consider the circuit of Fig. 9.DAlg-b that contains a reconvergent fan-out. Show that the testability of line s is the same when computed from each of the two paths to the output.

b. Provide an example of a circuit with a reconvergent fan-out where different testability values are obtained for the fan-out point depending on the path chosen.

c. In the situation of part b, how are testability values to be assigned?

11.4 Controllability and Observability transfer factors

a. Derive the *CTF* of 4-bit less-than comparator (two unsigned 4-bit binary inputs, 1 output line).

b. Repeat part a for *OTF*.

c. Repeat part a for a 4-bit less-than comparator for 2's-complement integers.

d. Repeat part c for *OTF*.

11.5 Controllability and Observability transfer factors

Discuss a method for obtaining the *CTF* and *OTF* parameters of a k-input, l-output component. Apply your method to a 4-bit unsigned binary comparator that provides less-than, greater-than, and equality indications (3 output lines).

11.6 Controllability and Observability transfer factors

a. Derive the *CTF* parameter for a 4-input, 1-output logic block that implements the logic function $z = x_0x_2 \vee x_1x_3$.

b. Repeat part a for *OTF*.

c. How would the *CTF* of part a change if we use the block as a multiplexer, by connecting its x_3 input to $\overline{x_2}$?

d. Repeat part c for *OTF*.

References and Further Readings

[Agra93] Agrawal, V. D., C. R. Kime, and K. K. Saluja, "A Tutorial on Built-in Self-Test, Part 1: Principles" (Part 2: Applications), *IEEE Design and Test of Computers*, Vol. 10, No. 1 (2), pp. 73-82 (69-77), January (April) 1993.

[Benn84] Bennetts, R. G., *Design of Testable Logic Circuits*, Addison-Wesley, 1984.

[Fuji85] Fujiwara, H., Logic Testing and Design for Testability, MIT Press, 1985.

[Lala85] Lala, P. K., *Fault Tolerant and Fault Testable Hardware Design*, Prentice-Hall, 1985.

[Lala97] Lala, P. K., *Digital Circuit Testing and Testability*, Academic Press, 1997.

[Liu12] Liu, R. W., *Testing and Diagnosis of Analog Circuits and Systems*, Springer, 2012.

[Step76] Stephenson, J. E. and J. Grason, "A Testability Measure for Register Transfer Level Digital Circuits," *Proc. Int'l Symp. Fault-Tolerant Computing*, 1976, pp. 101-107.

[Wang06] Wang, L.-T., C.-W. Wu, and X. Wen (eds.), *VLSI Test Principles and Architectures: Design for Testability*, Elsevier, 2006.

[Wang10] Wang, L.-T., C. E. Stroud, and N. A. Touba. System-on-Chip Test Architectures: Nanometer Design for Testability, Morgan Kaufmann, 2010.

12 Replication and Voting

"Honest disagreement is often a good sign of progress."

Mahatma Gandhi

"Half of the American people never read a newspaper. Half never voted for President. One hopes it is the same half."

Gore Vidal

Topics in This Chapter

As noted in Chapter 10, a widely used fault masking technique is based on replicated circuits or modules whose outputs feed a fusion or combining circuit, known as vote taker. In this chapter, we augment the abstract discussions of Chapter 10 with practical considerations in designing such redundant circuits. Among implementation questions discussed here are how to synchronize the multiple circuits so that data fusion occurs correctly, how to reduce the volume of data to be compared or fused, how to maximize reliability for a given level of replication, and how to ensure that the switching and fusion elements do not become weak links in the system,

12.1 Hardware Redundancy Overview

Typical hardware units consist of a data path, where computations and other data manipulations take place, and a control unit that is in charge of scheduling and sequencing the data path operations (Fig. 12.dpcg). A small amount of glue logic binds the main two parts together, allowing various optimizations as well as customization of certain generic capabilities for applications of interest. Redundancy methods for dealing with faults in data path, control unit, and glue logic are quite different. Generally speaking, many more options are available for protecting data paths through redundancy, as opposed to control circuitry (far fewer options) and the glue logic (extremely limited).

Options for protecting the data path span a wide range:

- Replication in space, including duplication with comparison, triplication with voting (TMR), pair-and-spare, NMR, and hybrid schemes (rather costly)
- Replication in time, including recomputation with comparison, recomputation with voting, alternating logic, recomputation after shifting, recomputation after swapping, and replication of operand segments (rather slow)
- Mixed space-time replication based on a combination of the methods listed under the preceding two bullet points
- Monitoring via mechanisms, such as watchdog timers and activity monitors, that are economical but have imperfect coverage
- Low-redundancy coding-based schemes, including parity prediction, residue checking, and self-checking design

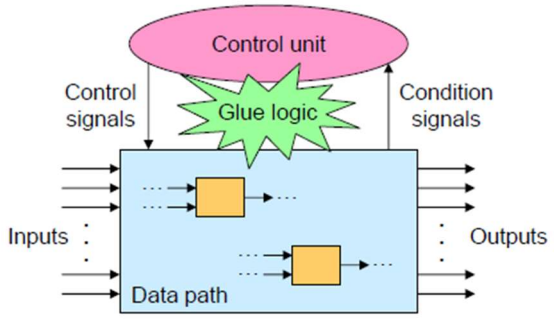


Fig. 12.dpcg Hardware unit with data path, control unit, and glue logic.

Options for the control unit may involve coding of control signals, control-flow watchdogs, and self-checking design. Protection methods for the glue logic are limited to simple replication and self-checking design.

12.2 Replication in Space

The schemes depicted in Fig. 12.space have already been discussed in connection with fault masking. In this section, we take a more detailed look at some of them, with the goal of understanding the trade-offs involved in their use and the extent of protection they offer in data path operations.

Let us begin by a more detailed examination of the TMR scheme of Fig. 12.space-c. Previously, we viewed TMR as a 2-out-of-3 system and derived its reliability with an imperfect voting circuit in equation 10.3.RTMR. In the following example, we consider the effects of an imperfect voter on system reliability.

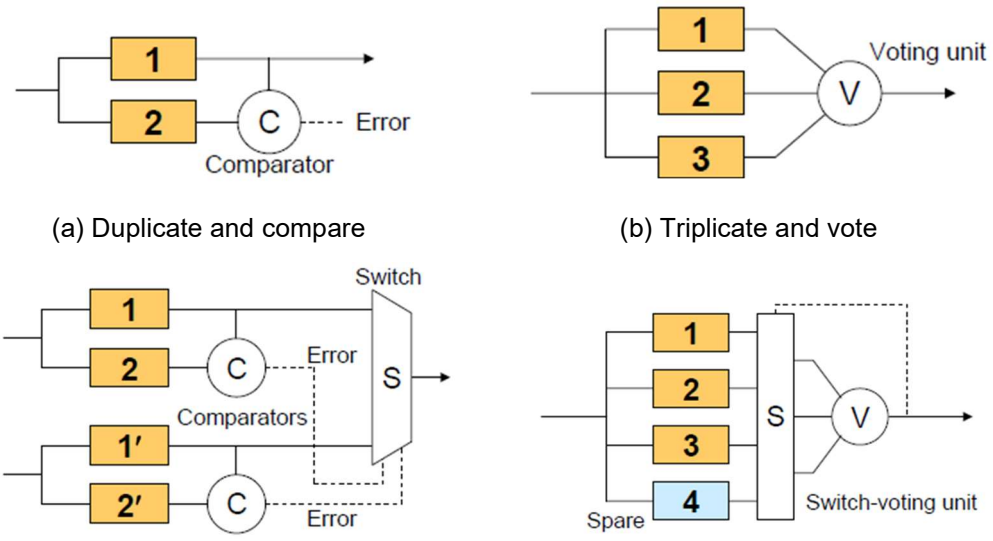
Example 12.TMR1: Modeling of TMR with imperfect voting circuit Consider a TMR system with identical module reliabilities R_m and voter reliability R_v. Under what conditions will the TMR system be more reliable than a simplex module?

Solution: The reliability of the TMR system can be written as $R = R_v(3R_m^2 - 2R_m^3)$, For $R > R_m$, we must have $R_v > 1/(3R_m - 2R_m^2)$. On the other hand, for a given R_v, the system will be more reliable than a module if $(3 - \sqrt{9 - 8/R_v})/4 < R_m < (3 + \sqrt{9 - 8/R_v})/4$ (see Fig. 12.TMR) For example, if $R_v = 0.95$, reliability improvement requires that $0.56 < R_m < 0.94$. In practice, R_v is very close to 1, that is, $R_v = 1 - \varepsilon$. We then have $1/R_v \approx 1 + \varepsilon$ and $(1 - 8\varepsilon)^{0.5} \approx 1 - 4\varepsilon$. Thus, the condition for reliability improvement becomes $0.5 + \varepsilon < R_m < 1 - \varepsilon$. Because modules tend to be much more reliable than voters, improved reliability is virtually guaranteed.

Next, we try to model the effect of compensating errors in TMR.

Example 12.TMR3: TMR with compensating errors Not all double-module faults lead to an erroneous output in TMR. Consider a TMR system in which each of the 3 modules sends a single bit to the voting circuit. Let the module reliability be $R_m = 1 - p_0 - p_1$, where p_0 and p_1 are probabilities of 0-fault and 1-fault, respectively. Derive the system reliability R.

Solution: The system operates correctly if it has no more than one fault or if there are two faults with compensating errors. Thus, $R = (3R_m^2 - 2R_m^3) + 6p_0p_1R_m$, with the last term being the contribution of compendating errors to system reliability. Take for example the numerical values $R_m = 0.998$ and $p_0 = p_1 = 0.001$. These values yield $R = 0.999\ 990 = 0.999\ 984 + 000\ 006$, where 0.000 006 is the improvement to the ordinary TMR reliability 0.999 984 due to the modeling of compensating errors. We can derive the respective reliability improvement factors thus: $RIF_{TMR/Simplex} = 0.002 / 0.000\ 016 = 125$ and $RIF_{Compens/TMR} = 0.000\ 016 / 0.000\ 010 = 1.6$.



(a) Duplicate and compare (b) Triplicate and vote

(a) Pair-and-spare (b) NMR/hybrid

Fig. 12.space Several replication methods in space.

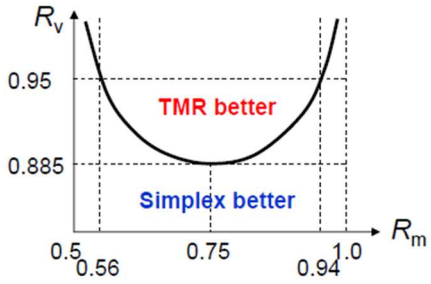
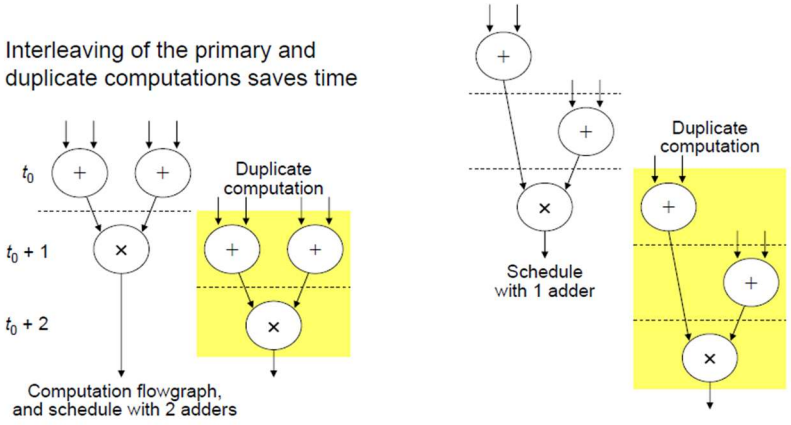


Fig. 12.TMR Analysis of TMR with imperfect voting unit.

12.3 Replication in Time

Section to be based on the following slides:



(a) Using 2 adders and a multiplier (b) Using an adder and a multiplier

Fig. 12.time Duplication in time, with scheduling considerations.

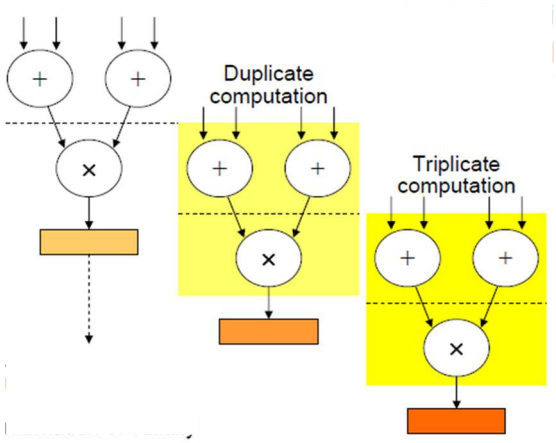


Fig. 12.time2 Triplication in time, with scheduling considerations

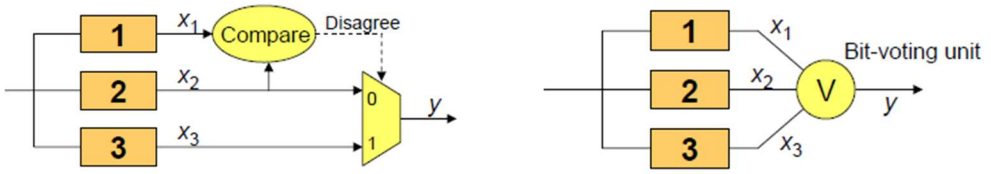
12.4 Mixed Space/Time Replication

Section to be written.

12.5 Switching and Voting Units

Consider a TMR system with a 3-way voting circuit. Assuming that we do not worry about how the system will behave in case of two faulty modules, a simple voter can be designed from a word comparator and a 2-to-1 multiplexer, as depicted in Fig. 12.vote-a. If x_1 and x_2 disagree, then x_3 is chosen as the voting result; otherwise $x_1 = x_2$ is passed on to the output. This comparison-based voting scheme can be readily extended to a larger number of computation channels, but the number of comparators required rises sharply with the number n of channels.

In applications where single-bit results (decisions) are to be voted on, the voting circuit is referred to as a bit-voter. We can synthesize bit-voters from logic gates (Fig. 12.3of5), but the circuit complexity quickly explodes with increasing n. It is thus imperative to find systematic design schemes based on higher-level components that keep the design complexity in check.



(a) Comparison-based voting circuit (b) Basics and notation for bit-voting

Fig. 12.vote Designs for simple voting circuits.

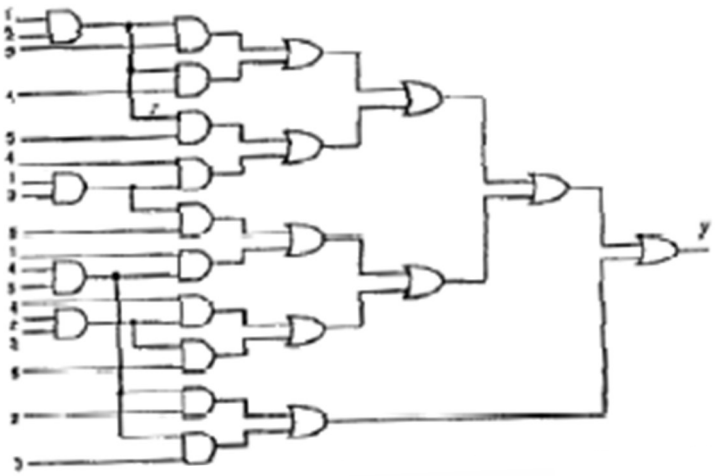
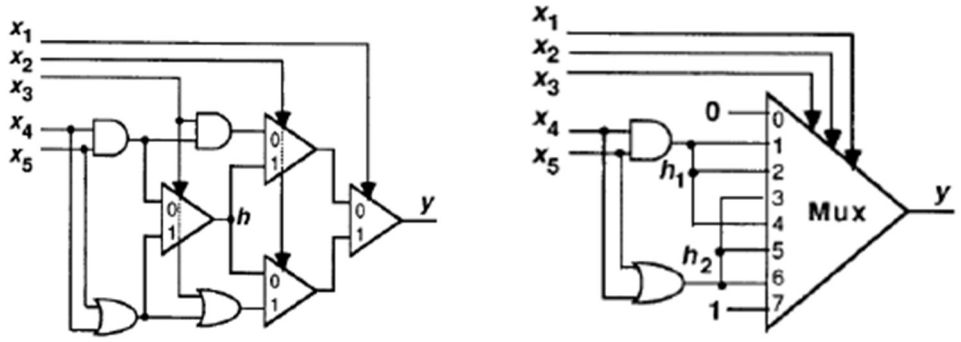


Fig. 12.3of5 Design of a 3-out-of-5 bit-voter from 2-input gates.



(a) Using 2-to-1 muxes (b) Using a single 8-to-1 mux

Fig. 12.muxv Designs for a 3-out-of-5 bitit-voter based on multiplexers.

One possibility is the use of multiplexers, as depicted in Fig. 12.muxv. For example, in Fig. 12.muxv-b, the inputs are partitioned into the subsets $\{x_1, x_2, x_3\}$ and $\{x_4, x_5\}$. If inputs in the first set are identical, then the majority output is already known. If two inputs in the first set are 1s, then the voting result is 1 iff at least one member of the second set is 1. Finally, if one of the 3 inputs in the first set is 1, then $x_4 = x_5 = 1$ is required for producing a 1 at the output.

Other design strategies for synthezising bit-voters include the use of arithmetic circuits (add the bits and compare the sum to a threshold) and the use of selection networks (the majority of bit values is also their median). Synthesis of bit-voters based on these strategies has been performed and the results compared (Fig.12.cmplx). It is readily seen that multiplexer-based designs have the edge for small values of n, but for larger values of n, designs based on selection networks tend to win.

Word-voting circuits cannot be designed based on bit-voting on the various bit positions. To see this, note that word inputs 00, 10, and 11 have majority results of 1 and 0 in their two positions, but the result of bit-voting, that is, 10, is not a majority value. The situation can even be worse. With word inputs 000, 101, and 110, the bitwise majority voting result is 100, which isn't even equal to one of the inputs.

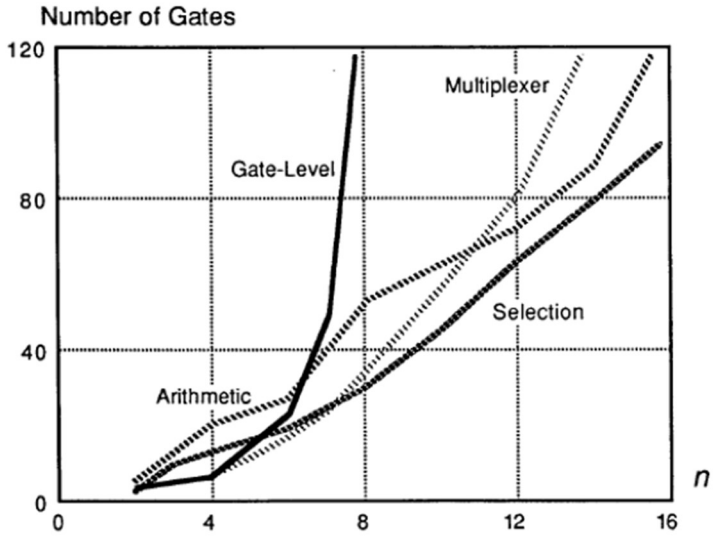


Fig. 12.cmplx Comlexity of bit-voting circuits synthesized based on various strategies.

Recently, a recursive method for the construction of voting networks has been proposed that offers regularlity, scalability and power-efficiency benefits over previous design methods [Parh21], [Parh21a]. The essense of the method is illustrated in Fig. 12.recTCN, which shows an at-least-*l*-out-of-*n* threshold circuit built from a multiplexer (mux), an at-least-*l*-out-of-(*n*–1) threshold circuit, and an at-least-(*l*–1)-out-of-(*n*–1) threshold circuit. Figure 12.5of9 illustrates the unrolling of the recursive construction method in the specific case of a 5-out-of-9 majority voter.

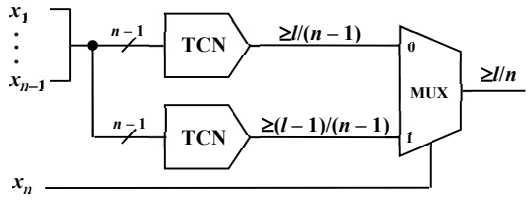


Fig. 12.recTCN Recursive method for building an at-least-*l*-out-of-*n* threshold counting network from a multiplexer and two smaller threshold counting networks. Unrolling the recurrence leads to a multiplxer-based design.

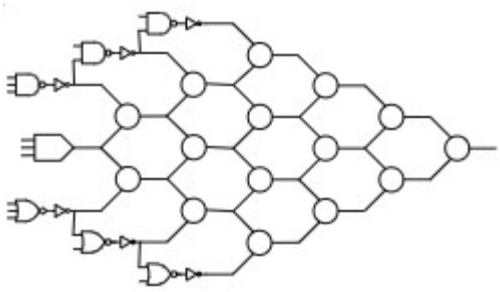


Fig. 12.5of9 **Recursive contruction of a 5-out-of-9 majority voter. Circles represent multiplexers controlled by the inputs x_4, x_5, x_6, x_7, x_8, and x_9, going in columns from left to right. The 3-input gates on the left edge, including the majority gate in the middle-left of the diagram, receive the inputs x_1, x_2, and x_3.**

12.6 Variations and Design Issues

One variation on the theme of replication and voting is that of self-purging redundancy. Instead of having n operational units and s spares, all $n + s$ units contribute to the computation by sending their results to a threshold voting circuit (Fig. 12.purge). When a module disagrees with the voting outcome, it takes itself out of the computatioon by resetting a flip-flop whose output enables the module output to go to the voting circuit. The threshold may be fixed or it may vary as units are taken out of service.

An interesting design strategy is based on alternating logic. The basic strategy is depicted in Fig. 12-alt. Alternating logic takes advantage of the fact that the same fault is unlikely to affect a signal and its complement in the same way. This property is routinely used in data transmission over buses by sending a data packet, then sending the bitwise complement of the data, and comparing the two versions at the destination, allowing the detection of bus lines s-a-0, s-a-1, and many transient faults. Let the dual of a Boolean function $f(x_1, x_2, \ldots, x_n)$ be defined as another function $f_d(x_1, x_2, \ldots, x_n)$ such that:

$$f_d(x'_1, x'_2, \ldots, x'_n) = f'(x_1, x_2, \ldots, x_n) \qquad \text{(12.6.dual)}$$

The dual of a Boolean function (logic circuit) can be obtained by exchanging AND and OR operators (gates). For example, the dual of $f(a, b, c) = ab \vee c$ is $f_d(a, b, c) = (a \vee b)c$. Using the dual of a function instead of its identical duplicate provides greater protection against common-cause and certain unidirectional multiple faults. If a function is self-dual, a property that is heldby many commonly used circuits such as binary adders (both unsigned anc complement), a form of time redundancy can be applied by using the same circuit to compute the function and its dual.

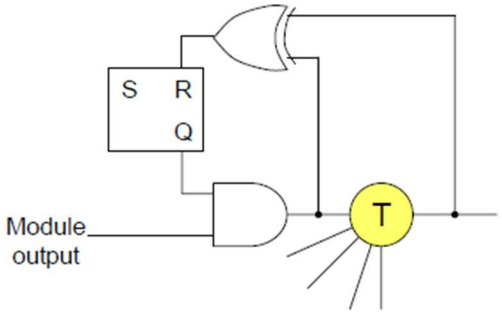


Fig. 12.purge Self-purging redundancy with a threshold voting circuit.

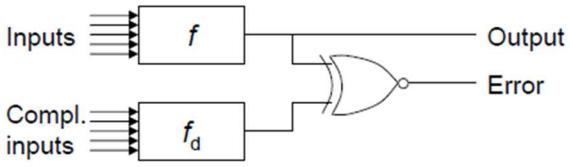


Fig. 12.alt Duplication with alternating logic.

Recomputing with transformed operands (Fig. 12.xform) constitutes another class of strategies for fault detection and masking. The main idea is that the redundant computation is performed on transformed operands in an effort to make it unlikely that similar faults would lead to identical errors in the two results. When *f* is binary addition, we can use shifts for encoding and decoding, given that the sum of shifted operands is a shifted version of the original sum. For operations that are symmetric, recomputation after swapping the two operands will lead to any existing fault to be exercised differently, and thus to produce a different error, which makes it detectable.

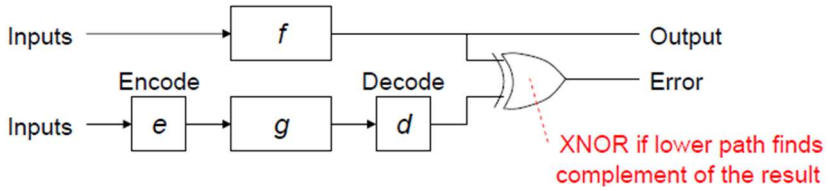
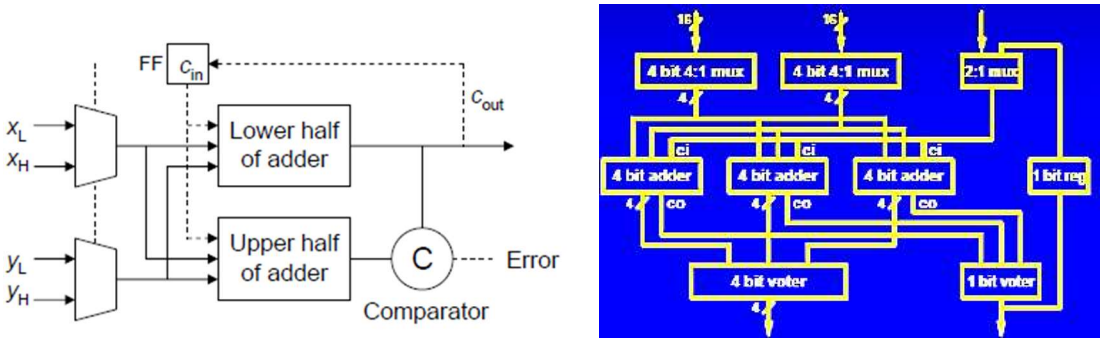


Fig. 12.xform Computing with transformed operands.



(a) Two-way segmented adder (b) A 4-way segmented adder

Fig. 12.segm Two designs for time-redundant, segmented addition.

Problems

12.1 Voting units for TMR and 5MR

a. Show at least one implementation (using logic gates and/or standard building-block combinational circuits, such as comparators, muxes, and the like) for 2-out-of-3 word-voting. The three inputs and the voting unit output are *k*-bit words. The voting unit need not detect the lack of majority, but must produce the majority value, if one exists, at its output.

b. Can your design be extended to a 3-out-of-5 voter?

12.2 How not to do TMR

The International Space Station (ISS) experienced a computer-related crisis in June 2007. According to NASA documents, "On 13 June, a complete shutdown of secondary power to all [three] central computer and terminal computer channels occurred, resulting in the loss of capability to control ISS Russian segment systems." Study this ISS incident using Internet sources and discuss in *one typed page* the nature of the crisis, its underlying causes, how the problems were dealt with, and what we can learn from this experience in terms of how to design dependable systems with diverse subsystems and suppliers.

12.3 Design of switch-voting units

For the redundancy scheme discussed in Problem 10.5, design the required 5-input, 1-output switch-voting unit, with 5 internal FFs indicating which units are contributing to the formation of the output. Assume that the five modules produce 1-bit results. Feel free to use muxes, comparators, and other standard circuits in your design (the design need not be at the gate level).

12.4 TMR with imperfect voting unit

Consider a TMR system with identical logic modules of reliability R and a voting unit of reliability r.

a. Derive the range of R values for which TMR offers improved reliability over a simple module, given the use of a voting unit with reliability r.

b. Plot the upper and lower bounds of part a as functions of r and discuss.

12.5 TMR vs. standby sparing

Which is more reliable over a given mission time T: A 2-out-of-3 system with voter reliability 0.9 over time T or a 2-way parallel system (one working module and one spare) with coverage factor 0.8?

a. If module reliability over time T is 0.9.

b. If module reliability over time T is 0.7.

12.6 Recursively-built voters

Use the recursive design method illustrated in Figs. 12.recTCN and 12.5of9 to build voters and of the following kinds.

a. 4-out-of-7 majority voter

b. 5-out-of-7 supermajority voter

c. 6-out-of-7 near-unanimity voter

d. 6-out-of-9 supermajority voter

e. 8-out-of-9 near-unanimity voter

12.6 Recursively-built threshold circuits

By offering appropriate circuit designs for specific examples, show that the recursive design method illustrated in Figs. 12.recTCN and 12.5of9 can also be applied to the design of various other types of threshold circuits.

a. At-least-3-out-of-7 threshold circuit
b. Less-than-3-out-of-7 inverse-threshold circuit
c. No-more-than-3-out-of-7 inverse-threshold circuit
d. Exactly-4-out-of-7 weight-checking circuit
e. At-least-3-and-no-more-than-4-out-of-7 between-limits threshold circuit

12.x Title

Intro

a. xx
b. xx
c. xx

References and Further Readings

[Lyon62] Lyons, R. E. and W. Vanderkulk, "The Use of Triple Modular Redundancy to Improve Computer Reliability," *IBM J. Research and Development*, Vol. 6, No. 2, pp. 200-209, April 1962.

[Parh91] Parhami, B., "Voting Networks," *IEEE Trans. Reliability*, Vol. 40, No. 3, pp. 380-394, August 1991.

[Parh91a] Parhami, B., "Design of *m*-out-of-*n* Bit-Voters," *Proc. 25th Asilomar Conf. Signals, Systems, and Computers*, 1991, pp. 1260-1264.

[Parh96] Parhami, B., "A Taxonomy of Voting Schemes for Dependable Multi-Channel Computations," *Reliability Engineering and System Safety*, Vol. 52, No. 2, pp. 139-151, May 1996.

[Parh21] Parhami, B., S. Bakhtavari Mamaghani, and G. Jaberipur, "Recursive Construction of Counting Networks," Submitted for publication.

[Parh21a] Parhami, B., S. Bakhtavari Mamaghani, and G. Jaberipur, "Recursive Construction of Large Voters," In preparation.

[Town03] Townsend, W. J., J. A. Abraham, and E. E. Swartzlander Jr., "Quadruple Time Redundancy Adders [Error Correcting Addder]," *Proc. 18th IEEE Int'l Symp. Defect and Fault Tolerance in VLSI Systems*, pp. 250-2562003.

[Yama05] Yamasaki, H. and T. Shibata, "A High-Speed Median Filter VLSI Using Floating-Gate-CMOS-Based Low-Power Majority Voting Circuits," *Proc. 31st European Solid-State Circuits Conf.*, 2005, pp. 125-128.