VLSI DESIGN
2001, Vol. 12, No. 4, pp. 515–525
Reprints available directly from the publisher
Photocopying permitted by license only

Test Generators Need to be Modified to Handle CMOS Designs*

JACOB SAVIR

ECE Dept., New Jersey Institute of Technology, University Heights, Newark, NJ 07102-1982

(Received 15 August 1999; In final form 11 September 2000)

CMOS designs have some unique properties that prevent existing test generators from computing a test vector for a fault when one might exist. The problem lies in the premises laid out on what it takes to detect a stuck-at fault. The basic premise that states that it is required to set a line to 0(1) in order to detect a stuck-at 1(0) fault, and then propagate the error to an observable point, needs to be re-examined. This is due to the existence of indeterminate states throughout the logic. The paper distinguishes between the traditional test vector (here called a *hard-detect*), and a potential test vector (here called a *soft-detect*). Our proposed test set is the union of hard and soft-detects. We also re-examine the issue of redundancy and show that it needs to be re-defined in order to comply with CMOS technology behavior.

This paper shows several examples to illustrate the problem; describes what it takes in order to remedy it; proposes possible enhancements to existing test generation algorithms, and outlines the risks faced in the event that no correcting steps are taken.

Keywords: Testability; Sensitized path; Tri-state drivers; Pass-gate transistors

1. INTRODUCTION

CMOS designs make use of circuit elements that do not lend themselves to gate level modeling (using And, Or, Nand, Nor and Not gates). Circuits such as pass-gate multiplexors and tri-state drivers are just a few examples. Use of such elements pose unique problems to test pattern generators.

Normally a test generator requires a *test view* description. A test view is a circuit description at the gate level. This test view is only an approximation to the actual circuit schematic that is described at the transistor level. Test generators use test views in order to simplify the test generation process that would otherwise be prohibitive if done at the transistor level. There has been sufficient experimental evidence to indicate that patterns generated at the gate level detect most of the transistor level defects.

Figure 1 shows a basic two-way pass-gate multiplexor. This device cannot be accurately

* This work was supported by NJCST under the Center for Embedded System on a Chip Design.

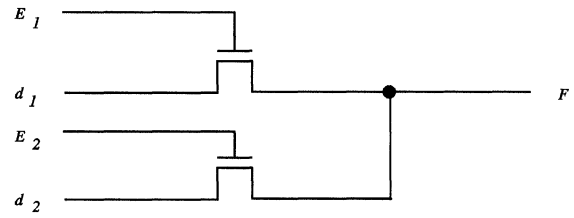


FIGURE 1 A two-way pass-gate multiplexor.

modeled using gates because of two reasons. The first reason is that in the presence of a pattern such as $E_1 = E_2 = d_1 = 1$, $d_2 = 0$ the output F is indeterminate (in some technologies this pattern will actually damage the device). This indeterminate (u) response from the circuit cannot be depicted if the pass-gate multiplexor is modeled using And-Or gates. Secondly, in the presence of a pattern that disables both pass-gate transistors ($E_1 = E_2 = 0$) the multiplexor response is, again, u, and has the same problem discussed earlier (although this pattern is not damaging).[1] In order to be able to generate patterns that are both safe (not damaging) and more representative of the actual circuit behavior the pass-gate multiplexor is modeled at the test view level using tri-state elements, as shown in Figure 2.

TRD1 and TRD2 are the test view representation of the pass-gate transistors. A TRD will pass the value on its data input to the output, provided it is enabled, and produce an indeterminate state at the output in case it is disabled.

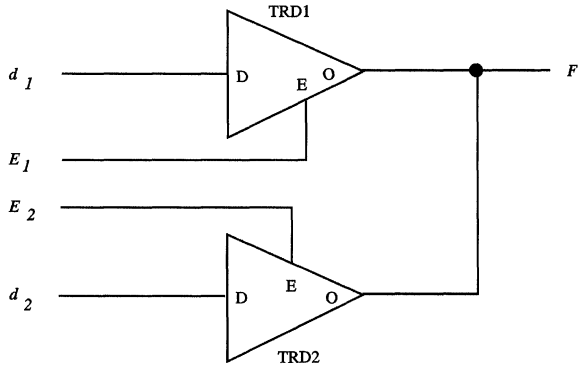


FIGURE 2 Test view of a two-way pass-gate multiplexor.

In order to safe-guard against cases where both enable lines are turned off (it may be prevented from occurring in normal circuit operation, but is likely to happen in test mode and/or in a faulty machine condition) a pull-up device may be added to the output. Figure 3 shows the two-way pass-gate multiplexor with the pull-up device controlled by the test line, *TEST*.

The pull-up transistor is designed and sized in such a way that it will force the output F to become 1 whenever the enable lines are turned off, but will be "defeated" when exactly one of the enable lines is turned on. In the case where both pass-gate transistors are turned on and driving opposite values, the test pull-up is normally not strong enough to drive the output node to a 1. In this case the output is still indeterminate (u).

Some CMOS designs may use test pull-down devices rather than test pull-ups.

CMOS devices may use tri-state buses that are driven by tri-state drivers. The test views of these tri-state drivers are the same as for the pass-gate transistors. We will, therefore, refer to either occurrence by denoting the element by the symbol TRD.

Test generation algorithms work under the assumption that the function F, realized by the combinational logic inside the circuit, performs a mapping F: $\{0, 1\}^n \rightarrow \{0, 1\}^m$, where n and m are the

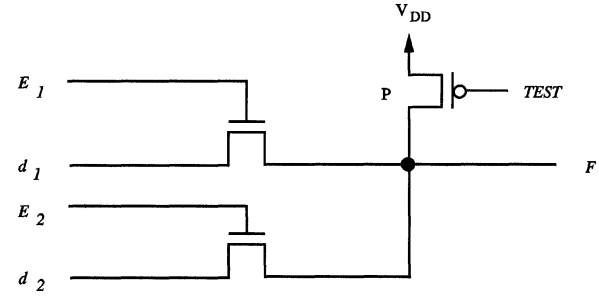


FIGURE 3 A two-way pass-gate multiplexor with a pull-up device.

[1]This is the High-Z state.

number of inputs and number of outputs respectively. A test generator would try to compute a test vector that would distinguish the fault-free F from a faulty function F' that performs a different mapping F': $\{0,1\}^n \rightarrow \{0,1\}^m$. We call such a vector a *hard-detect*. This fundamental assumption does not hold in CMOS designs due to the existence of indeterminate states (u) throughout the logic (we hereby distinguish between an indeterminate state, u, and a unknown state, x. An indeterminate state is an indeterminate logic level. An unknown state is a well-defined logic level that might be either 0 or 1). A more accurate representation of the mapping performed by the output F would be F: $\{0,1\}^n \rightarrow \{0,1,u\}^m$. Even though it is possible to impose some design restrictions that will guarantee the good-machine mapping to be F: $\{0,1\}^n \rightarrow \{0,1\}^m$, it is impossible to guarantee it under an arbitrary faulty-machine condition. Thus, the faulty behavior of the circuit is always of the form F': $\{0,1\}^n \rightarrow \{0,1,u\}^m$. This fundamental difference is the source of the problem when it comes to compute test vectors for stuck-at faults in CMOS designs.

The problem with existing test generation algorithms is that they try to detect a fault on a line using a hard-detect, *i.e.*, by first driving an opposite value to that line, and then propagating the error to an observable point. This is sufficient when the circuit is free of tri-state elements. When the circuit has embedded tri-state elements in it, some of the values existing in the circuits are indeterminates (u). As a result, a faulty machine behavior may transform the value on a line in a variety ways, none of which are within the scope of the test generation algorithm. The uncovered cases are:

1. good value $= u$; bad value $= 0$
2. good value $= u$; bad value $= 1$
3. good value $= 0$; bad value $= u$
4. good value $= 1$; bad value $= u$

Whenever any one of these cases propagates to an observable point, we get a different detection class called *possible detect* (PD) [1]. Note that Cases 3 and 4 occur at an output of a TRD when a fault changes the value of the enable line from 1 to 0.

A fault classified as a PD may still be detected by the computed pattern. Let $p \rightarrow q$ denote a change of a good-machine value p to a bad-machine value q. In order to illustrate this point, assume a case where, as a result of a fault, $a\ u \rightarrow 1$ propagates to a primary output. The computed pattern may or may not detect the fault in question. It all depends on whether or not the indeterminate state (u) happens to be within a voltage range that is recognized as a 0 logic level. If this is the case, then the computed vector detects the fault. If this is not the case, then the computed pattern does not detect the fault. Since there is no way for a test generation algorithm to determine whether or not the indeterminate state u will resolve itself in a favorable way, the fault can only be classified as a PD, and the computed pattern only as a *soft-detect* (potential test vector).

Large CMOS designs may have as much as 20% of the total fault universe in a PD class. If the test generator cannot compute soft-detecting vectors for these faults, then the fault coverage achievable will only be around 80%. If the test generator can cover the PD class and add the patterns that potentially detect these faults to the pattern set, a potential fault coverage of 100% might be achieved. If we assume a random resolution of the u states between 0 logic level and 1 logic level in such a way that half of the time it is "favorable" to the fault, and half of the time it is not, then the fault coverage will be closer to 90%. If, on the other hand, 6% of the 20% do not resolve to either 0 or 1 logic levels, and the other 14% resolve themselves equally in a "favorable" and "unfavorable" way, then the fault coverage would be around 87%.

Since the PD class is a relatively large class in a CMOS design, it is important to cover it by soft test patterns. In order to be able to do this, test generators need to be modified.

For the remaining of this paper we assume that the circuit has been designed to conform with the level-sensitive scan design (LSSD) rules

[4]. We are, therefore, concentrating our discussion on the unique test problems associated with the combinational logic between the scan banks.

Section 2 revisits the redundancy issue and extends its definition to fit CMOS designs. Section 3 shows experimental results supporting the new redundancy definition. Section 4 revisits the test generation issue and describes possible modifications to existing test pattern generators so that they will be able to cover unique CMOS test problems. Section 5 outlines some of the risks that may be faced if existing test pattern generators are used to detect faults in CMOS designs. Section 6 summarizes the results of this paper and provides some further concluding remarks.

2. REDUNDANCY REVISITED

2.1. Some Examples

Before we present a number of examples showing the problems associated with the notion of redundancy in CMOS designs, let us introduce a simple example to show how redundancy is used today to simplify logic circuits.

Figure 4(a) shows a 3-gate circuit. The fault m stuck-at 1 ($m/1$) is undetectable. Since this fault is undetectable the circuit can be further simplified by injecting the value of the undetected fault and propagating it as far forward as possible. This simplification process leads to the circuit of Figure 4(b). This circuit only has 2 gates, and it realizes the same function as Figure 4(a) (since $x_1x_2 + \bar{x}_2 = x_1 + \bar{x}_2$).

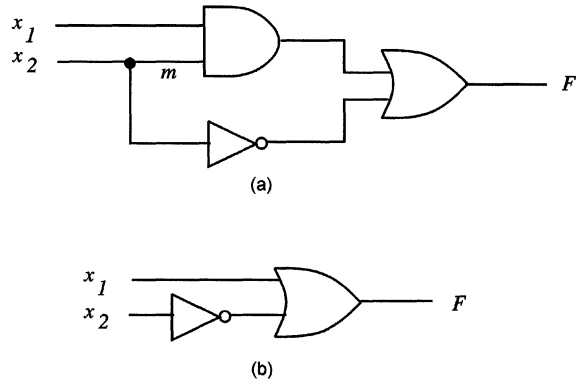


FIGURE 4 (a) A redundant circuit. Fault $m/1$ undetectable. (b) A simplified circuit after injecting $m/1$.

Test generators [3, 5–9] have been trying to identify redundant faults in order to suggest, among other things, that the circuit may be simplified. One of the corner stones of most test generator algorithms is to have a fault declared redundant *if it can be proven that no test pattern (i.e., hard-detect) can detect the fault.* Thus, a test generator will correctly declare fault $m/1$ in Figure 4(a) as redundant.

As soon as we move into the domain of CMOS designs we realize that there are some very serious problems with the current definition of redundancy and the way it is being used in test generation algorithms. Figure 5(a) shows two tri-state drivers feeding a bus line that is terminated by a pull-up device. The enable inputs are fed orthogonally from primary input C. The faults $m/1$ and $n/1$ are undetectable. (The faults $m/0$ and $n/0$ are detectable due to the pull-up device.) As a result, these two faults may be claimed "redundant", and circuit simplification may be initiated. We first remove one of these faults by injecting it with a constant value of "1". After removing this first "redundancy", we check if the other fault is still undetectable. Since it does remain undetectable we remove it also. The "simplified" circuit that results from this process

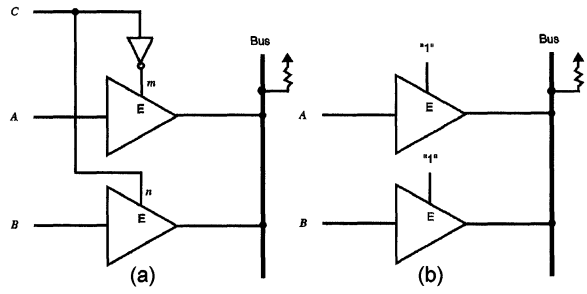


FIGURE 5 (a) Two tri-state drivers with orthogonal Enable signals. Faults m/1 and $n/1$ are undetectable. (b) A simplified circuit after injecting $m/1$ and $n/1$.

is shown in Figure 5(b). Not only has this process failed to simplify the circuit (as a matter of fact the circuit of Figure 5(a) cannot be further simplified), but if actually implemented will cause damage to some logic, because inputs A and B can now drive opposite values into the bus. So, obviously, the traditional notion of *undetectability* does not immediately translate into *redundancy* in this case.

Figure 5(a) is a special case of the circuit shown in Figure 6. The "1-Hot Encoder" is a circuit whose property is that only one of its outputs can have the value "1" at any given time. Addition of such a circuit is intended to guarantee a "safe" operation of the tri-state drivers in normal mode; to prevent damaging the device during scan, and also forbid the test generator from computing a damaging test pattern. The problem, though, is that this "protection circuit" will, in the most part, be wiped out once the "redundant faults" on the enable lines of the tri-state drivers are identified. This will happen for the same reasons it happened to faults $m/1$ and $n/1$ in Figure 5(a).

More sporadic occurrences of *false* redundancy identification will occur whenever Muxes appear inside the logic (and there are normally many such instances).

2.2. Correcting the Problem

So how can this problem be filtered out? There are, certainly, several ways of handling it. An obvious way would be to re-examine all faults driving the Muxes and tri-state drivers that were declared redundant, before actually embarking on circuit simplification. What we are interested here, though, is in an automatic solution to the problem that will entail no human intervention.

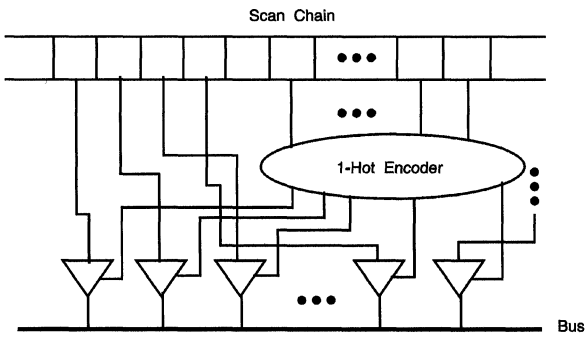


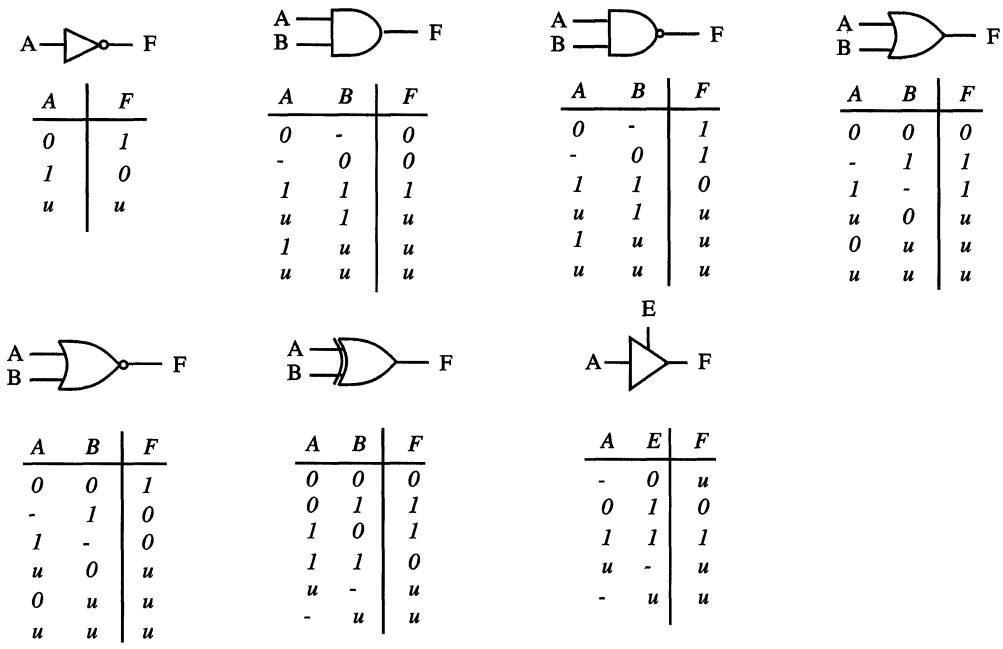
FIGURE 6 "1-Hot" Encoder added to logic to assure enable signal orthogonality.

A possible way of doing it is to define a logical symbol u whose meaning is "indeterminate". The truth table for Inverter, And, Nand, Or, Nor, and Xor gates and a tri-state element, using this indeterminate value, is shown in Figure 7. Also added to the rules of Figure 7 is the rule that any clash between a 0 and 1 on a bus, or bus-like wire (like shared output line of a Mux), is considered a u, as well as any other clash where one of its members is a u.

Before we introduce our proposed new definition of a redundant fault, we would like to draw the distinction between the notion of "hard-detect" and the notion of "soft-detect". A *hard-detect* is the case where a traditional detecting test pattern exists; *i.e.*, there exists a test pattern TP_i for fault f_i, for which the presence of the fault will cause at least one of the primary outputs to either change from an expected value of 0 to a wrong value of 1, or from an expected value of 1 to a wrong value of 0. A *soft-detect*, on the other hand, is a case where a hard-detect does not exist, but where a test pattern TP_i' exists for which the presence of the fault will cause at least one of the primary outputs to either change from a determinate value (0 or 1) to an indeterminate value (u) or vice versa. Given this distinction, we can now redefine the notion of a redundant fault.

DEFINITION A fault is redundant if it does not have either a hard-detecting nor a soft-detecting pattern.

Based on this new definition fault $m/1$ in Figure 4 remains redundant, but faults $m/1$ and $n/1$ in Figure 5 are now irredundant. Figure 8 shows the soft-detecting pattern for $m/1$. Notice that the clash between 0 and 1 on the bus has generated the symbol u.



A	F
0	1
1	0
u	u

A	B	F
0	-	0
-	0	0
1	1	1
u	1	u
1	u	u
u	u	u

A	B	F
0	-	1
-	0	1
1	1	0
u	1	u
1	u	u
u	u	u

A	B	F
0	0	0
-	1	1
1	-	1
u	0	u
0	u	u
u	u	u

A	B	F
0	0	1
-	1	0
1	-	0
u	0	u
0	u	u
u	u	u

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0
u	-	u
-	u	u

A	E	F
-	0	u
0	1	0
1	1	1
u	-	u
-	u	u

FIGURE 7 Truth table of test primitives using the indeterminate state u.

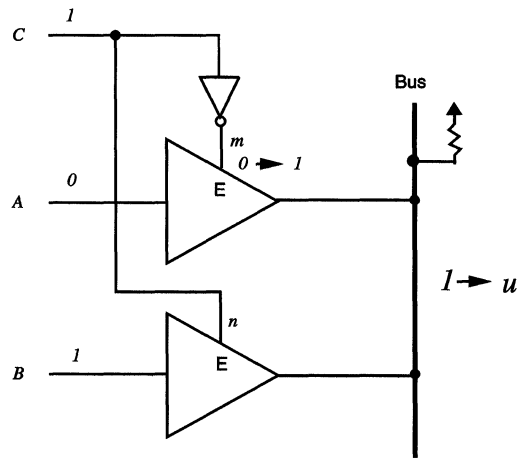


FIGURE 8 A soft-detecting pattern for $m/1$ in Figure 5.

Consider now the circuit of Figure 9(a). The fault $m/1$ has neither a hard nor soft-detecting pattern, even though it drives the enable line. Figure 9(b) shows the same circuit after removal of $m/1$. No further reduction is possible.

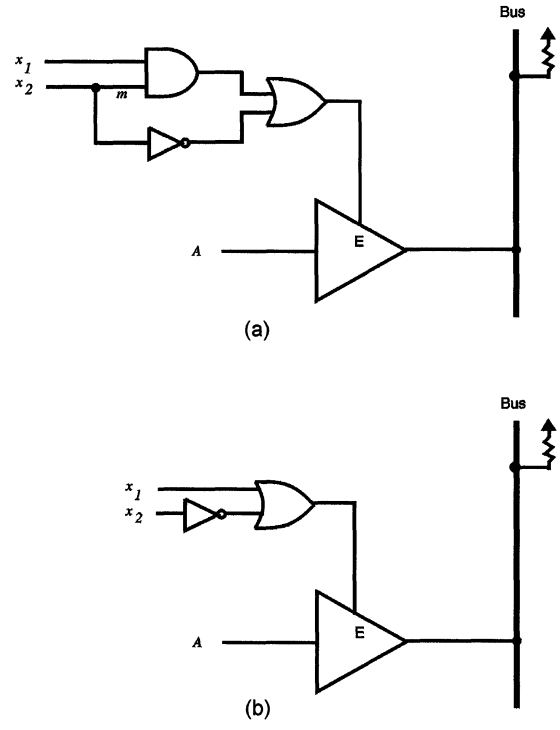


FIGURE 9 (a) Circuit to be analyzed. (b) Resulting circuit after removal of $m/1$.

The conceptual procedure for redundancy identification and circuit simplification is therefore:

1. For every fault that has been proven not to have a hard-detect attempt to compute a soft-detect.
2. List all faults from Step 1 that have neither a hard nor a soft-detect. This is the redundant fault list.
3. To simplify the circuit – sequentially remove the redundant faults [3]. Before attempting to remove a fault re-verify that it is still redundant.

3. EXPERIMENTAL RESULTS

To test how effective the revised definition of redundancy is, we took a D-algorithm-based [9] test generator and modified it accordingly. The program was used in two modes: first with the original definition of redundancy, and then with the revised definition. Our test vehicles were ten large CMOS micro-processor chips that had over one million transistors. All chips constituted a "full-system". The results are shown in Table I.

As seen in Table I the new definition of redundancy was able to reduce the size of the redundancy class tremendously. All faults that have been re-classified from redundant to irredundant were carefully examined to determine if their re-classification was correct. The examination confirmed that all these faults were indeed irredundant. Almost all the initially marked "redundant" faults were situated within the upstream cone of the Enable lines of the tri-state drivers and Muxes. The modified test generator was able to unmark them as "irredundant".

It should be noted that the "Untested" class might have included more redundant faults, but they remained in this class since the test generator was unable to prove that they were, in fact, redundant. This class, in general, will include unproven redundancies and other aborted faults.

TABLE I Experimental results on ten chips

Chip	FC(%)	Untested(%)	Red(%) (old)	Red(%) (new)
1	83	7	10	1
2	89	5	6	0.5
3	91	5	4	1
4	88	4	8	0
5	90	8	2	0.5
6	92	4	4	1
7	87	5	8	0
8	90	7	3	0.5
9	94	3	3	0
10	93	4	3	0

4. TEST GENERATION REVISITED

4.1. An Example

The following example has been created to illustrate the test problems occurring in CMOS designs. This is just an example, and it by no means represents current design styles.

Consider the circuit of Figure 10. Figure 10(a) shows the inability of current test generation algorithms to detect the fault g stuck-at 0 ($g/0$). Figure 10(b) shows a successful generation of the potential test vector for this fault.

The circuit of Figure 10 is composed of two tri-state drivers, TRD1 and TRD2, and one Or-gate, G. Consider the fault $g/0$. Current test generation algorithms [1, 2, 5–9] will assign a value 1 to g and try to "justify" this value by proper assignment to the primary inputs, a and b. The algorithm will also try to sensitize the error condition on line g to the primary output F by further assignment to the primary inputs. In this particular case, the line justification of $g = 1$ requires both a and b to have a value 1. As a result, both values entering gate G are 1, preventing the fault $g/0$ from propagating to the output F. Thus, a traditional test generator will categorize fault $g/0$ as redundant.

Consider now a "non traditional" way of detecting the fault $g/0$. Let us try to compute a test vector that assigns an indeterminate value (u)

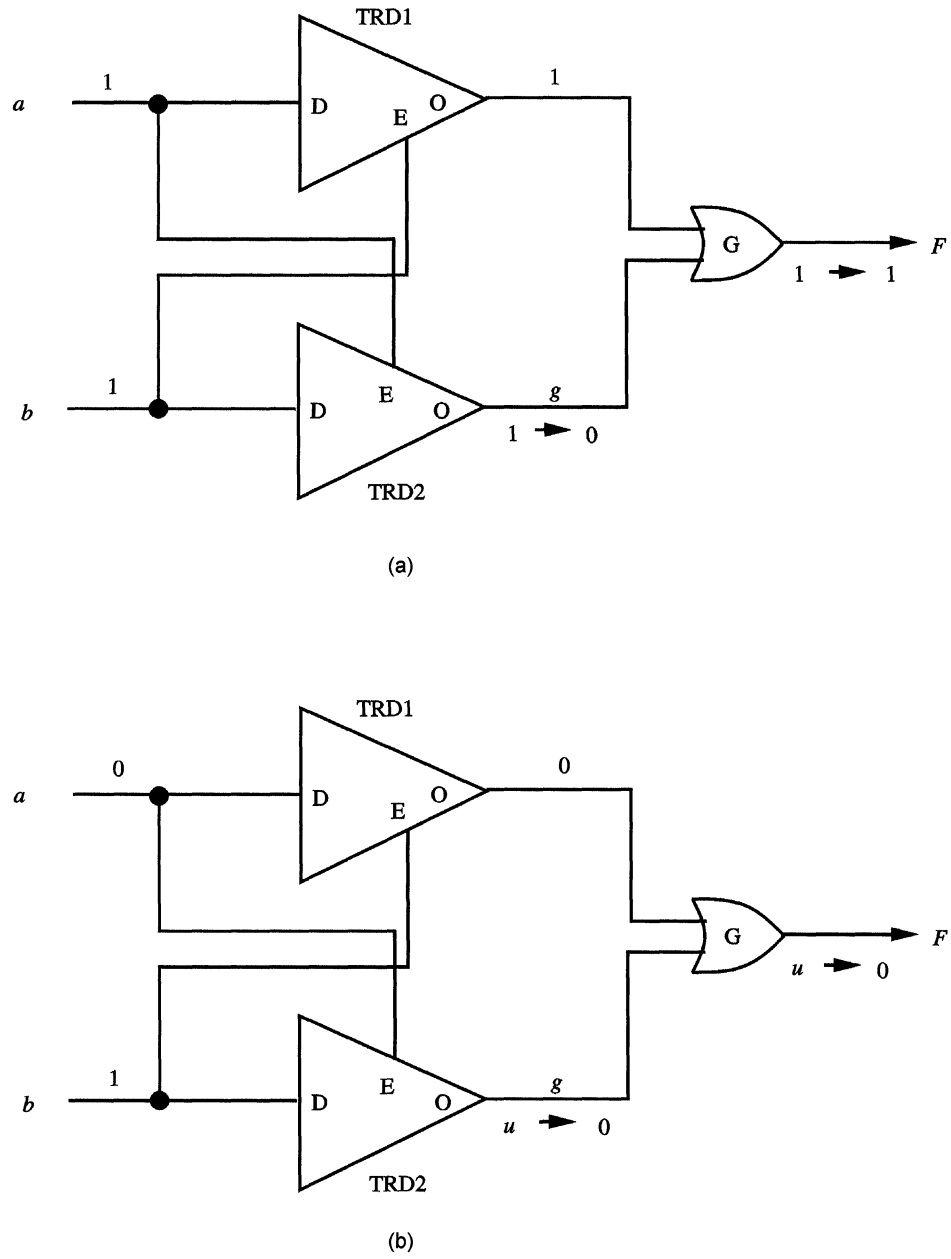


FIGURE 10 An example that illustrates the deficiency of existing test pattern generation algorithms.

to line g. This will require driver TRD2 to be disabled, allowing a to have a value 0. If we assign a value 1 to input b, then driver TRD1 is enabled, passing a value 0 to its output. Thus, the good-machine values entering gate G are 0 and u, and the good-machine response is $F = \mathrm{u}$. Since the output undergoes a $u \rightarrow 0$ change, the fault $g/0$ can be classified as PD, and the test pattern that potentially detects this fault is $\mathrm{a} = 0$, $\mathrm{b} = 1$. Notice that if a pull-up transistor would have been added to output F, the fault $g/0$ would have been detected. The reason for this certain detection is

because the gate G, that in the absence of fault $g/0$ would have responded with a u, would now respond with a 1 (due to the pull-up device). In the presence of $g/0$ both inputs to gate G are 0, and the output is $F=0$. In the bad-machine the pull-up device plays no role since the output value is defined!

As this example shows, the basic premise of test generation algorithms, that it is necessary to provoke a stuck-at 0 fault by injecting a 1, and a stuck-at 1 fault by injecting a 0 prevents the test generator from classifying the fault as a PD. It may, in turn, classify it as redundant, which may be a wrong classification.

Normally test generators are coupled with fault simulators to efficiently mark as many faults as possible as being detected, once a test pattern has been computed for some target fault. It is possible, under this scenario, to uncover some of the PD class, but not all of it. This example can clearly show this point. In this example, the only *good* test pattern is $a=1$, $b=1$. This good (hard) test pattern only detects the fault $F/0$. It cannot serve as a potential test pattern for any other fault. Thus, no test generator coupled with a fault simulator can compute the potential test patterns $a=0$, $b=1$, and $a=1$, $b=0$. In order to uncover these potential test patterns it is necessary to actually target the remaining faults in the "non-traditional" way, as described earlier.

4.2. Modifying the Test Generation Algorithms

In lieu of the example shown earlier several changes are needed. The first change relates to the way a fault is provoked. In order to provoke a stuck-at 1 fault on a line it is necessary to allow for a $u \rightarrow 1$ change, and a $0 \rightarrow u$ change, as well as the traditional $0 \rightarrow 1$ change. Similarly, in order to provoke a stuck-at 0 fault on a line it is necessary to allow for a $u \rightarrow 0$ change, and a $1 \rightarrow u$ change, as well as a $1 \rightarrow 0$ change.

The second change to be made is related to the notion of a sensitized path. Traditionally, a sensitized path is defined as a path along which all values are defined (0 or 1), and that has the property that a change of values in the origin of the path ($0 \rightarrow 1, 1 \rightarrow 0$) propagates to an observable point (primary output or latch). This definition has to be extended to include the case were there are u's along the path, and a change of values in the origin of the path ($u \rightarrow 1, 1 \rightarrow u, u \rightarrow 0, 0 \rightarrow u$) propagates to an observable point, and *changes its response from either* $0 \rightarrow 1$, *or* $1 \rightarrow 0$, *or* $0 \rightarrow u$, *or* $u \rightarrow 0$, *or* $u \rightarrow 1$, *or* $1 \rightarrow u$. A possible scenario for this more general sensitized path is illustrated in Figure 11.

Several comments are in order regarding the implementation of the test generation algorithm. Since the majority of stuck-at faults can be handled by the "traditional" algorithms, it

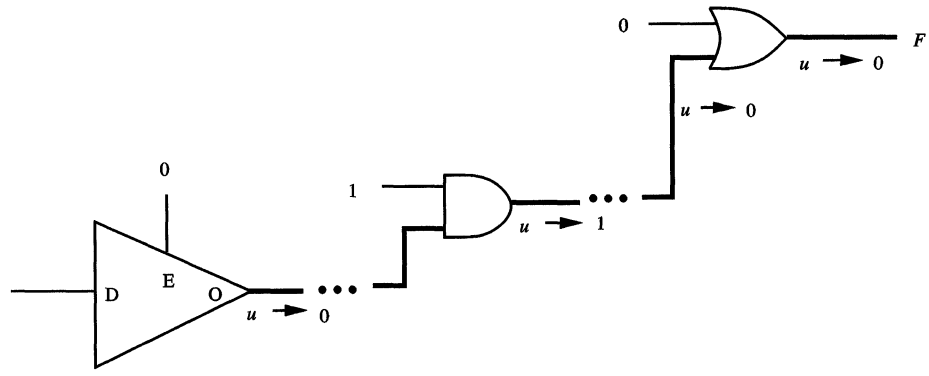


FIGURE 11 A sensitized path having u's that leads to a PD.

should remain its major engine. Experience with "traditional" test generators shows that they can compute vectors for 80% of the stuck faults. It is the other 20% that need special handling.

We propose to design the test generator so that it is composed of a primary and secondary engines. The primary engine is the "traditional" test generator where faults are provoked "the old" way, and paths are sensitized "the old" way. The secondary engine will attempt to compute test vectors the "new way", namely by allowing for the more generalized definition of a sensitized path, as described above.

The test generation process goes as follows. The fault list is passed on to the primary engine with a given abort limit (a limit on how many attempts should be made to compute a vector before the fault is abandoned). The primary engine terminates when either all faults are detected, or when all the untested faults reach their abort limit. Upon termination of the test pattern generation process in the primary engine, the list of the yet untested faults is passed on to the secondary engine with a given abort limit. The test generation process ends with the termination of computation in the secondary engine.

5. RISKS FACED WHEN THE TEST GENERATOR IS NOT MODIFIED

There are several problems associated with using "traditional" test generators in CMOS designs.

One obvious outcome of using a "traditional" test generator in CMOS designs is a reduced fault coverage. This reduced fault coverage will affect the product defect level and the number of field returns.

Another down side associated with the use of "traditional" test generators is reduced diagnosability. This is due to not having all testable faults covered by test patterns (even if they are only potential (soft) test patterns).

A more sophisticated problem occurs when "untestable" faults are used to remove "redundancy" from the design. Most test generators will classify a fault as being redundant when all possible assignments (whether local or global) to the lines affecting the fault have been exhausted leading to no detection. A truly redundant fault may give rise to circuit simplification that removes the fault, and still maintains the same circuit functionality. Since a "traditional" test generator is no longer *complete*, its redundancy list cannot be trusted. It may very well be that a fault listed as "redundant" by the "traditional" test generator is actually potentially testable (*i.e.*, the secondary engine described earlier can compute a potential test pattern for that fault). Any circuit simplification actions taken based on this false redundancy may be detrimental to the circuit functionality. An example to this case is the removal of fault $g/0$ in Figure 10 and faults $m/1$ and $n/1$ in Figure 5.

The notion of redundancy needs to be changed so that it relates to the true mapping of the function, which in general is $F, F' : \{0,1\}^n \rightarrow \{0,1,u\}^m$. In other words, a fault is redundant if and only if $F = F'$ under this more general mapping.

6. CONCLUSIONS

This paper discusses some unique test problems associated with CMOS designs. We have shown that existing test generators may not be able to compute a test for a fault even if one might exist. This problem is due to the use of circuit elements such as pass-gates transistors, and tri-state drivers.

We have shown that in order to detect stuck-at faults in CMOS designs, it is sometimes necessary for the test generator to allow the fault to be provoked by an indeterminate (u) state. We have also shown that in order to detect a fault in CMOS designs, it is sometimes necessary to allow indeterminate states to propagate along a sensitized path. Since these two conditions are outside the scope of existing test pattern generators, they cannot perform a complete and thorough job.

The notion of redundancy, the way we know it, has to be revisited. A fault cannot be classified as being redundant if it cannot be tested by a pattern that assigns a known value to the line (hard-detect). It may very well happen that the fault may be detected by a pattern that assigns an indeterminate value to the line (soft-detect). Again, this is outside the scope of existing test generators.

Test generators may be easily enhanced to remedy the unique problems associated with CMOS designs. We have proposed to have the modified test generator be composed of two parts: a main engine and a secondary engine. The main engine should treat faults the "old way". All the faults left uncovered by the main engine should be passed on to the secondary engine that will try to compute test vectors by allowing the more generalized definition of fault provocation and path sensitization.

Failure to use an enhanced test generator on CMOS designs may lead to a reduced fault coverage, reduced diagnosability, and an increase in product defect level (increase in the number of field returns).

References

[1] *FastScan Reference Manual*, 1993–1994.

[2] Abramovici, M., Breuer, M. A. and Friedman, A. D. (1994). *Digital Systems Testing and Testable Design*. IEEE Press.

[3] Abramovici, M., Breuer, M. A. and Friedman, A. D., *Digital Systems Testing and Testable Design*. IEEE Press, Piscataway, New Jersey, 1994.

[4] Eichelberger, E. B. and Williams, T. W. (1978). A Logic Structure for LSI Testability. *J. Design Automation and Fault-Tolerant Computing*, **2**, 165–178.

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

[6] Fujiwara, H. and Shimono, T., On the acceleration of test generation algorithms. *IEEE Trans. Computers*, **C-32**(12), 1137–1144, December, 1983.

[7] Goel, P., An implicit enumeration algorithm to generate tests for combinational logic circuits. *IEEE Trans. Computers*, **C-30**(3), 215–222, March, 1981.

[8] Goel, P. and Rosales, B. C., PODEM-X: An automatic test generation system for VLSI logic structures. In: *Proc. 18th Design Automation Conf.*, pp. 260–268, June, 1981.

[9] Roth, J. P., Bouricius, W. G. and Schneider, P. R., Programmed algorithms to compute tests to detect and distinguish between failures in logic circuits. *IEEE Trans. on Electronic Computers*, **EC-16**, 567–580, October, 1967.

Author's Biography

Dr. Savir holds a B.Sc. and an M.Sc. degree in Electrical Engineering from the Technion, Israel Institute of Technology, and an M.S. in Statistics and a Ph.D. in Electrical Engineering from Stanford University. He is currently a Distinguished Professor at New Jersey Institute of Technology, where he held the position of Director of computer engineering (1996–2000), and Newark College of Engineering Associate Dean for research (1999–2000). Previously with IBM, Dr. Savir was a Senior Engineer/Scientist at the IBM PowerPC Development Centre in Austin, TX; at IBM Micro electronics Division in Hudson Valley Research Park; at IBM Enterprise System in Poughkeepsie, NY, and a Research Staff Member at the IBM T. J. Watson Research Center, Yorktown Heights, N.Y. He was also an Adjunct Professor of Computer Science and Information System at Pace University, N.Y., and SUNY Purchase, N.Y.

Dr. Savir' research interest lie primarily in the testing field, where he has published numerous papers and coauthor-ed the text "Built-In Test for VLSI: Pseudorandom Techniques" (Wiley, 1987). Other research interest include design automation, design verification, design for testability, statistical methods in design and test, fault simulation, fault diagonosis, and manufacturing quality.

Dr. Savir has received four IBM Invention Achievement Awards, six IBM Publication Achievement Awards, and four IBM Patent application Awards. He is a member of Sigma Xi, and a fellow of the Institute of Electrical and Electronics Engineers.