TEST-DETECT

Kapil Gore
Electrical & Computer Engineering Department
Auburn University, Auburn AL 36830
(Peer Reviewer: Mr. Nikhil Mathur)

Abstract **– The TEST-DETECT algorithm is presented in this paper which is used to ascertain all faults detected by a given test. It is based upon the utilization of a "calculus of D-cubes" that provides the means for effectively performing the necessary computations for very large logic circuits. The D-calculus is initially reviewed here followed by the TEST-DETECT algorithm which can be applied in parallel as seen in the concluding section of this paper.**

I. Introduction

TEST-DETECT is a new algorithm developed by J. P. Roth in 1967 to ascertain all failures detected by a given test, consisting of signal values on the primary inputs, of an acyclic logic circuit. It is an offshoot of the D-calculus invented during the development of the first D-Algorithm. These algorithms could be used as a basis for the generation of a set of tests to detect any single failure in a logic circuit in a method that was more efficient than any based on previously known techniques.

Although the D- Algorithm and TEST-DETECT can easily be extended to handle other failures and logic, the programmed versions described here are limited to single failures of the stuck-at-1 and stuck-at-0 faults in circuits composed of AND, OR, NAND, NOR, and XOR logic blocks. When describing such circuits it is possible to assign the same number or name to both the logic block and its output line since all the blocks are single output. This convention will be used here so that the terms "block i" and "line i" can be used interchangeably. Also, it will be assumed that the n lines or blocks in the circuit are assigned unique number labels starting from integers 1,2, …,n subject to the constraints that each line shall have a higher number than its predecessors, and the primary inputs shall be assigned successive numbers starting with 1. With this convention the term backward is defined to mean in the direction of decreasing line numbers and forward to mean in the direction of increasing line numbers.

II. Summary of D-calculus

A "calculus of D-cubes" was developed by Roth that allowed one to describe analytically the behavior of, and to compute diagnostic tests for, failing automata.

The functional transformation for each logic block in the circuit is given in terms of its *singular cover*, a kind of truth table description for the block's function with extra columns present in the table to account for circuit variables other than those associated directly with the given block. Each row in the singular cover is termed a *singular cube*. Such cubes are called *prime* when no larger singular cube of the function can exist which contains the given cube. The set of so called primitive D-cubes, or pdc, for a NOR gate are shown in Fig. 1. These D-cubes have the following interpretation: the symbol D may assume only the two values 0 and 1, and the assignment must be held fixed for all the D's in a given D-cube. The symbol $\overline{D}$ denotes the complement of the value assigned to D.

Thus the D-cube $D0\overline{D}$ represents the two cubes 001 and 100. In effect, this D-cube specifies that when the NOR gate is functioning properly and a

0 is placed on line 12, the output must be the complement of the signal on line 11. The two dual sets of D-cubes are provided since the assignment of the D or $\overline{D}$ to the inputs is, at this stage, arbitrary.

11 — NOR — 13
12

(a)

11	12	13
0	0	1
	1	0
1		0

(b)

11	12	13
D	0	$\overline{D}$
0	D	$\overline{D}$
D	D	$\overline{D}$
D	$\overline{D}$	0

11	12	13
$\overline{D}$	0	D
0	$\overline{D}$	D
$\overline{D}$	$\overline{D}$	D
$\overline{D}$	D	0

(c)

Fig. 1 Singular cover and primitive D-cubes for a 2-input NOR gate

A *primitive D-cube of failure*, or pdcf, is used to specify how to exhibit a given failure at the site where it occurs. The pdcf construction procedure is limited to the simple stuck-at-1 or stuck-at-0 faults considered in this paper. When a line or a block is to be tested for a stuck-at-1 fault, it is necessary to place a 0 on the line. Thus the first step in the construction is to select for the given block associated with this line a prime singular cube with a 0 at its output coordinate position. Then the output 0 is replaced by a $\overline{D}$. For the NOR gate of Fig. 1, the stuck-at-1 pdcf's are $x1\overline{D}$ and $1x\overline{D}$ while the stuck-at-0 pdcf is 00D. For primary input lines, the pdcf's are merely a D or $\overline{D}$ at their respective coordinate position.

III. TEST-DETECT

Given a test T, TEST-DETECT computes the set of all failures detected by T. Thus it is a kind of converse of the D-algorithm. A method for ascertaining all failures F detected by T would be to construct c(T, F) for each F and check whether or not any primary output coordinate had value D or $\overline{D}$.

If TC denotes the vector of signals assigned by the test T to each line of the logic circuit, and if TC_i has the value α =1 or 0, T cannot test for the failure "line i stuck-at-α" since the good and failing circuits would appear identical. Thus if F_i denotes the failure line i stuck-at-$\overline{\alpha}$, we should just investigate such F_i. Further, the set of failures detected by T can now be uniquely specified by a vector FD of lines on which T detects a failure: if $i \in FD$ and $TC_i=\alpha$, then T detects the failure F_i.

A method will now be described for effectively computing c(T, F_i), and hence for computing whether or not i can be placed in FD. This is done by constructing the D-chain emanating for line i, similar to the D-algorithm. The activity vector A and its use in producing an orderly level-by-level march to the primary outputs is once again similar to D-algorithm. However, all line signal values are fixed so that we are only concerned with whether or not the D-chain propagates through the given block. For the sake of storage efficiency, c(T, F_i) is never explicitly generated. Instead it is recorded that a line in TC has a D or $\overline{D}$ on it by placing the line's number in a list DL. Thus if $j \in DL$ and TC_i=1, then coordinate j in c(T, F_i) is D; otherwise TC and c(T, F_i) agree. Since we are only interested in those lines for which active computation is being carried out, the definition of DL will be modified so as to exclude those lines which have already been processed and do not have successors in A.

When the D-chain reaches a primary output, $DL \cap PO \neq \phi$, it can be concluded that T detects F_i and stops the generation of c(T, F_i). If the D-chain dies, A=ϕ, before this occurs, T does not detect F_i. These are the two basic

criteria for stopping computation. If the lines are examined for inclusion in FD in numerical order starting with the highest numbered one, the status of line j has been determined before that of line i (j>i). Then the following lemma can be used to develop a third criterion for stopping that truncates the formation of c(T, F_i) and greatly increases the efficiency of TEST-DETECT.

Lemma A: If at any stage in the computation of c(T, F_i) there is only one line j in DL, then I is in FD if and only if $j \in FD$.

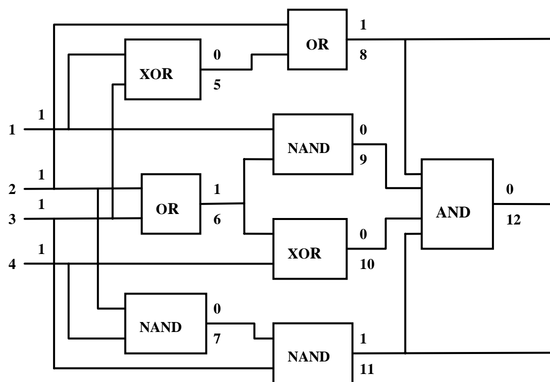

Fig. 2 Example used to illustrate TEST-DETECT

As an illustration of TEST-DETECT, let us examine the circuit in Fig. 2 wherein the test T assigns the value 1 to all the primary inputs. For all lines, the appropriate entry in the TC is shown as the number above the line. When the first two lines, 11 and 12, are processed, placing their number in DL automatically satisfies the test $DL \cap PO \neq \phi$, i.e. if $i \in PO$ it can immediately be placed in FD. Line 10 has only one successor, 12, and when this entry in A is examined it is found that no pdc for block 12 exists that has a 0 at coordinate 9 and a D or $\overline{D}$ at coordinate 12. Hence A becomes empty and 10 cannot be placed in FD. Similarly, line 9 is found not to be in FD while line 8 is. When line 7 is processed, DL=7 and A=11. A pdc with the required values exists for block 11. Thus DL=11 and A=12. Since DL has been reduced to a single entry that is in FD, we conclude, using Lemma A, that 7 must be in FD.

Investigating line 6 means that initially DL=6 and A=9, 10. Since appropriate pdc's exist for both 9 and 10, two iterations in the D-chain extension will produce DL=9, 10 and A=12. The cube 1DD1D is a pdc for block 12, so the D-chain can be extended through 12. Now DL=12 and Lemma A can again be applied to show line 6 is in FD. Proceeding in a similar manner, lines 4 and 2 are found to be in FD, while lines 5, 3 and 1 are not.

IV. Applying TEST-DETECT principle in parallel

As with the original TEST-DETECT algorithm, it is crucial to process the faults in levelized gate order. In parallel-patterns fault simulation, many patterns are being simulated at the same time, so the notion of D-frontier becomes somewhat unclear. For convenience, we consider the entire stack of gates waiting to be evaluated to be the D-frontier. If this stack reduces to a single gate, called a dominator of the faulty gate, the test detect principle applies: the complete packet of fault detection results for the current fault can be deduced from the current faulty dominator gate output values and results of the earlier insertion of a fault on the dominator gate.

This "early cutoff" of fault propagation may also be applied in cases where there are many gates in the stack, provided that the gate under consideration has an output cone that is disjoint from the output cones of all other gates remaining to be processed. In that case, the successors of the gate are not placed in the stack, but the results of an earlier fault insertion and propagation are used as was the case with a dominator gate. In the case of a non-dominator

gate with a disjoint output cone, however, the complete fault detection results cannot be determined from the effects of that gate alone. Instead, the fault detection packet for that gate must be ORed with the packets of all other such gates, so we must accumulate a packet of fault detection results.

There are three versions of parallel pattern test detect approaches-

(1) Parallel Test-Detect, which calculates stopping points by testing for cone-disjoint gates during the fault simulation,
(2) Count test-Detect, which identifies stopping points by noting that the evaluation stack has reduced to a single gate, and
(3) Dominator Test-Detect, which identifies stopping points during preprocessing as a by-product of calculating dominator gates.

V. Conclusion

In TEST-DETECT algorithm, J. P. Roth made use of the relationship between faults, a relationship not exploited in parallel-patterns fault simulation. Roth processed the faults in reverse levelized order, so that when a fault at gate i is considered, all faults at gates occurring later in the order have already been treated. As soon as the D-frontier reduces to a single gate, fault simulation can be terminated, since a fault at that gate has earlier been inserted and propagated towards the primary outputs. It is sufficient to use the results of the earlier propagation, which eliminates the need to evaluate all gates lying between the single D-frontier gate and the circuit outputs. It is this elimination of previously performed gate evaluations that gives TEST-DETECT its power.

References:

[1] J. P. Roth, W. G. Bouricius & P. R. Schneider, "Programmed algorithms to compute tests to detect and distinguish between failures in logic circuits", IEEE Transactions on Electronics & Computers, June 1967.

[2] J. P. Roth, "Diagnosis of automata failures: a calculus and a method", IBM J. Research Develop., vol. 10, pp. 278-291, July 1966

[3] B. Underwood & J. Ferguson, "The parallel test-detect fault simulation algorithm", 1989 IEEE International Test Conference