E. B. Eichelberger
E. Lindbloom

A Heuristic Test-Pattern Generator for Programmable Logic Arrays

This paper describes a heuristic method for generating test patterns for Programmable Logic Arrays (PLAs). Exploiting the regular structure of PLAs, both random and deterministic test-pattern generation techniques are combined to achieve coverage of crosspoint defects. Patterns to select or deselect product terms are generated through direct inspection of an array; test paths to an observable output are established by successive, rapidly converging assignments of primary input values. Results obtained with a PL/I program implementation of the method are described; these results demonstrate that the method developed is both effective and computationally inexpensive.

Introduction

Previous work in test-pattern generation [1-5] has shown that random patterns can be used to easily and efficiently achieve stuck fault test coverage in excess of 90% for most combinational logic networks. Unfortunately, random patterns have proved to be ineffective for testing faults in Programmable Logic Arrays (PLAs) [5]. In this paper, a heuristic method will be described that exploits the concepts of random test patterns and extends their application to generating tests for PLAs. This method is called PLA/TG, an acronym for programmable logic array/test generator.

Random test patterns do not give high test coverage for PLAs mainly because the AND array in a PLA normally has a relatively large number of *used crosspoints* in each product term (Fig. 1). The probability of detecting a missing crosspoint with a random pattern is no better than $1/2^n$, where n is the number of used crosspoints in the product term. Since n is frequently greater than 10, the test coverage using random tests is quite low. The problem is solved in the PLA/TG procedure by deterministically generating embryonic tests for each used crosspoint in the AND array. These tests are then combined using a procedure that exploits the PLA structure and utilizes random input values wherever possible.

The following concepts are employed by PLA/TG to achieve further efficiencies in test generation and fault evaluation.

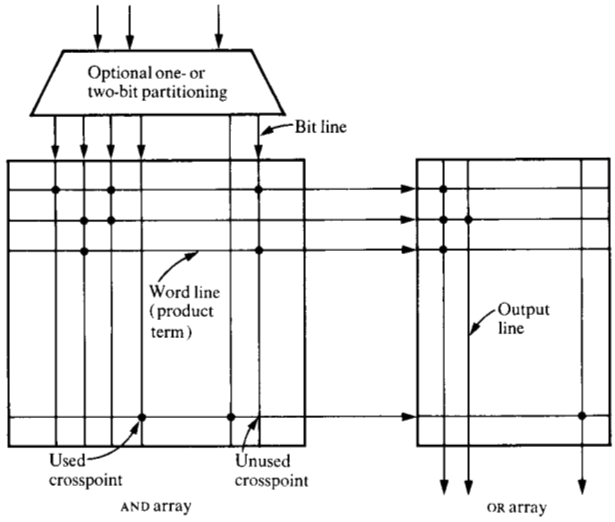

Figure 1 PLA logic schematic and terminology.

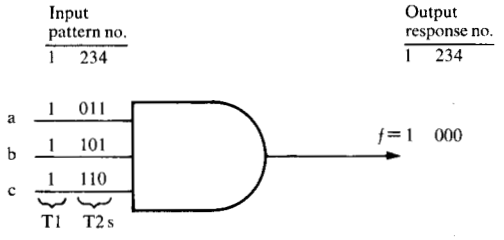

Figure 2 Embryonic test patterns for an AND block.

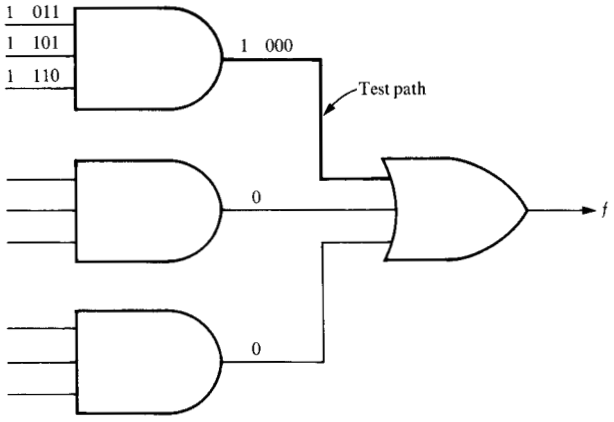

Figure 3 Embryonic test patterns for an AND block together with a sensitized test path.

1. Direct (nonexpanded) representation of the PLA: a matrix representation is used for test generation without expansion into equivalent logic blocks.
2. Reduced fault assumptions: one stuck condition is modeled at each personalized crosspoint in the AND and OR arrays. Through the use of an additional closing routine, all testable stuck conditions in the bit partitioning circuitry are covered.
3. Pattern subsumption: the embryonic patterns are subsumed as long as such subsumption is effective (*i.e.*, additional test coverage is gained), thereby producing a nearly minimum test set.
4. Elimination of the need for conventional fault simulation: given the PLA structure, a simple analytic check can determine if a candidate pattern detects a given fault, thus obviating the primary need for the most costly part of test pattern preparation in an LSI environment, fault simulation.

This paper presents a detailed description of the PLA/TG method. First the test objectives and fault assumptions are discussed. This is followed by descriptions of the test-generation procedures and of special closing routines required for complete test coverage. Then, in the sections following, the assumptions behind the fault modeling utilized in the method are discussed. Finally, the results obtained with a PL/I implementation of PLA/TG are described.

Test objectives and fault assumptions

Working with a digitized description of information for a particular PLA, such as that contained in Fig. 1, the generator creates a set of patterns that consists of the following two types of embryonic tests.

1. *Test Type 1 (T1):* All used crosspoints on a given word line are set to logic *one*.
2. *Test Type 2 (T2):* One selected used crosspoint on a given word line is set to *zero*; the rest of the used crosspoints are set to *one*.

These two test types are precisely the familiar embryonic patterns for an AND block in stuck-fault practice as shown in Fig. 2. The total number of Type 1 tests is equal to the number of product terms and the total number of Type 2 tests is equal to the number of used crosspoints in the AND array.

For either of the two types of tests to be successful there must also exist a sensitized test path through the OR array as illustrated in Fig. 3.

Probably the major difference between PLA/TG and other PLA test approaches is the method by which paths are sensitized through the OR array. Normally this is done in a deterministic manner that results in decision trees that can have relatively long program run time. In PLA/TG, a set of primary input values is randomly assigned and then a check is made to determine whether a path has been sensitized. If the path is not sensitized the process is repeated until all tests have been established or a maximum number of iterations has been reached.

It is worth noting that the PLA characteristic causing random test patterns to be ineffective (*i.e.*, many used crosspoints per product term) is also the characteristic that renders the procedure of combining random patterns with T1 and T2 tests effective for sensitizing test paths. That is, if the n inputs to an AND gate are randomly assigned, then the probability of the output being *zero* is $(2^n - 1)/2^n$. Thus, randomly assigning other inputs not specified by a T1 or T2 test will usually result in a sensitized path.

Although both T1 and T2 tests are sensitized the same way, as illustrated in Fig. 3, they do have different requirements. The T1 test will test the input to the OR block "stuck-at-*zero*." Thus *all* OR blocks fed by the AND block

must be sensitized for some T1 test. The T2 tests will test the inputs to the AND block "stuck-at-*one*." Thus it is sufficient to sensitize the path to any OR block fed by the AND block.

These facts suggest the following approach to combining the embryonic T1 and T2 tests. First, two or more T1 tests should *not* be combined since they might both feed the same OR block and combining them would prevent the testing of their inputs to this common OR block. On the other hand, combining a T2 test with a T1 test is acceptable since the T2 test will usually help sensitize the T1 test path, and the associated product term of the T2 test will have a high probability of feeding at least one OR gate not fed by the T1 product term. Also, two or more T2 tests can usually be combined without causing any path-sensitizing problem.

This suggests the following simple procedure that is fast and results in relatively few test patterns.

Step 1 Specify and assign each T1 embryonic test to a unique test *pattern*.

Step 2 Specify each T2 embryonic test, one at a time, and try to combine it with one of the test patterns. It will combine if no specified bit value is different from the corresponding specified bit value in the test. If it will not combine with any existing test pattern, then assign it to a new test pattern.

This procedure will be described in more detail in the next section.

It is the objective of the generator to see that there is a valid T1 test for each use of each word line in the OR array and one T2 test for each use of a bit line in the AND array. This objective can be viewed as equivalent to modeling a single stuck-fault at each used crosspoint. The PLA/TG procedure does just that; its fault list is equal in length to the number of used crosspoints in the entire PLA. Figure 4 illustrates, with a portion of the two-level PLA viewed as conventional AND-OR logic, which stuck-faults are assumed. Coverage of this limited set of stuck-faults will ensure that all gates in the AND or OR arrays operate properly. A complete set of T1 and T2 patterns will detect any missing-crosspoint defect.

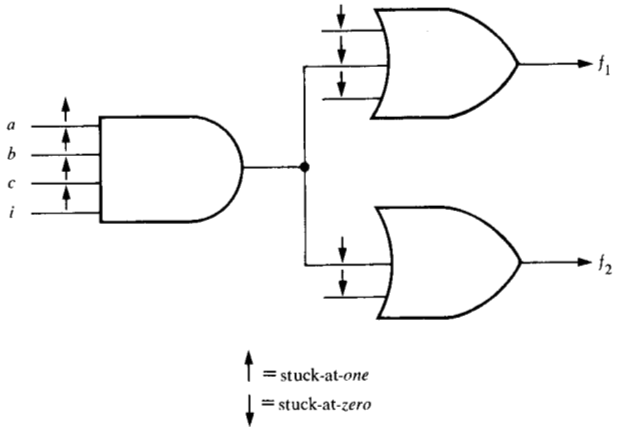

Figure 4 Missing-crosspoint faults viewed as conventional logic stuck-faults.

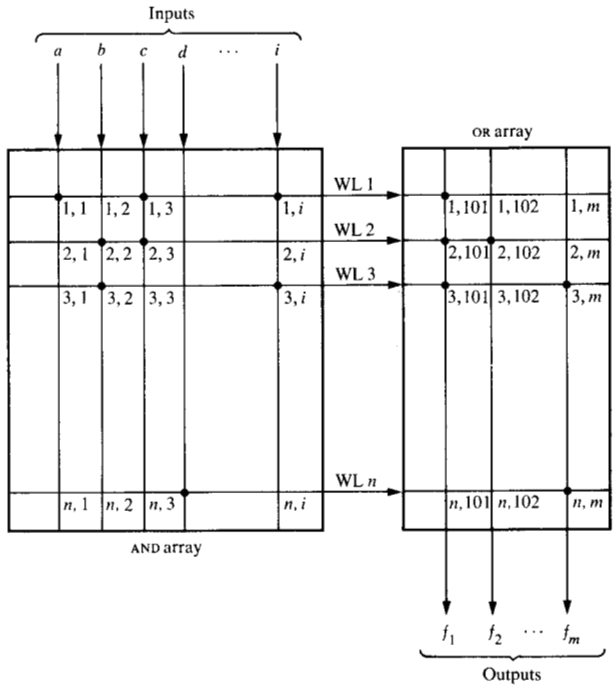

Figure 5 PLA logic schematic with labeled crosspoints.

Test generation procedure

The generation procedure begins by specifying a set (PT1) of T1 patterns. A T1 test consists of all the primary input (PI) values required to "turn *on*"—set to logic *one*—a given word line (see Fig. 5). For word line 1 in Fig. 5 this would be

a	*b*	*c*	*d*	···	*i*
1	U	1	U	···	1,

where "U" stands for "unspecified." This is the embryonic condition required to turn *on* WL1 via its used crosspoints at coordinates (1, 1), (1, 3), and (1, *i*) and to test the potential stuck-at-*zero* condition at (1, 101) in the OR array. This partial pattern does nothing, however, to ensure a sensitized path to an observable output. Next, a set of T2 patterns is specified (PT2). The subset for WL1 in Fig. 5 would be

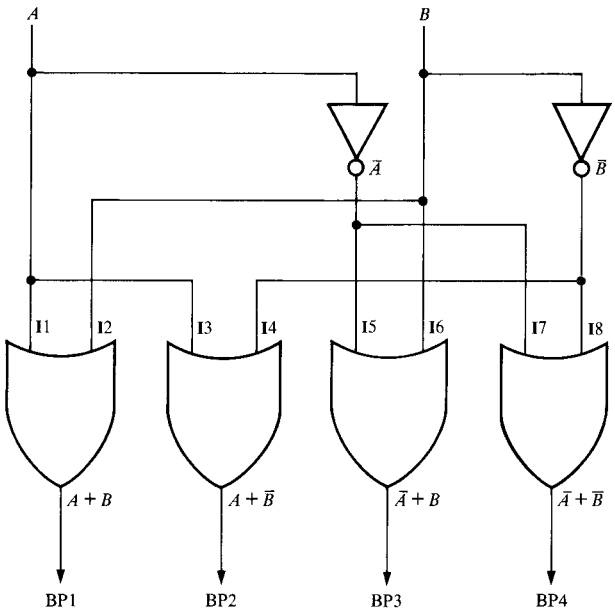

Figure 6 Two-bit partitioning (two-to-four decode) circuitry.

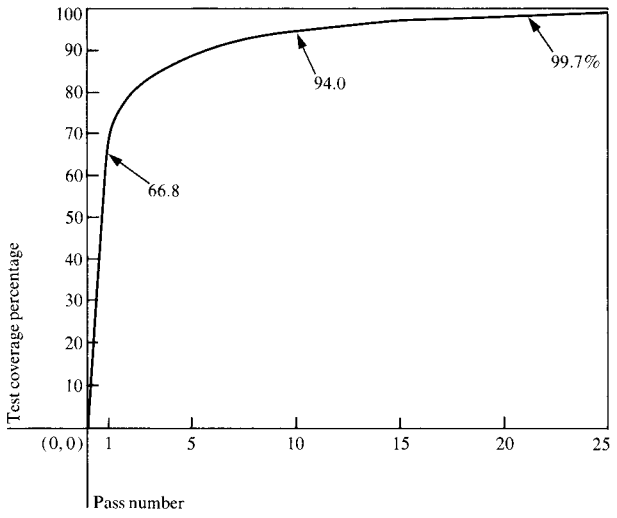

Figure 7 Average test coverage *versus* number of passes through PLA/TG for the 19 PLAs listed in Table 1.

a	b	c	d	$\cdots$	i
0	U	1	U	$\cdots$	1
1	U	0	U	$\cdots$	1
1	U	1	U	$\cdots$	0.

These constitute a necessary condition for testing the stuck-at-*one* conditions at coordinates (1, 1), (1, 3), and (1, i). In specifying either of the partial test types, if the bit line in question is driven by a two-bit partitioning (2BP) circuit [6] (Fig. 6) rather than a PI, then a reverse transform to PI values is performed via a table. When a logic *zero* is desired on any 2BP output, a unique set of PI values is required. In the case of a logic *one* on a 2BP output, there may be one, two, or three PI choices that will satisfy this. The choice is made at random in PLA/TG to increase the likelihood of reaching a valid detection test and to increase the chance for stuck-fault coverage in the 2BP logic itself, a topic which will be discussed in a later section.

In the interest of final test pattern economy, a subsume operation is now performed with PT1 and PT2. First all patterns in PT1 are placed directly in a new set, P. Then, for each pattern in PT2, the question is asked, "Can it be subsumed under any pattern in P?" In general, pattern B will subsume under pattern A if for every PI position one of the following conditions is true:

$$\begin{Bmatrix} A \\ B \end{Bmatrix} = \begin{Bmatrix} 1 \\ 1 \end{Bmatrix}, \begin{Bmatrix} 0 \\ 0 \end{Bmatrix}, \begin{Bmatrix} \mathrm{U} \\ \mathrm{U} \end{Bmatrix}, \begin{Bmatrix} 1 \\ \mathrm{U} \end{Bmatrix}, \begin{Bmatrix} 0 \\ \mathrm{U} \end{Bmatrix}, \begin{Bmatrix} \mathrm{U} \\ 1 \end{Bmatrix}, \begin{Bmatrix} \mathrm{U} \\ 0 \end{Bmatrix}.$$

If a pattern B from PT2 will subsume under a pattern A in P, then any U in A that has a logic *one* or *zero*, in a corresponding PI position in B, is changed to that value; the pointer of the fault for which B was designed is changed to refer to A. If a partial pattern from PT2 will not subsume under any pattern in P, it is added, without change, to P.

When subsumption is complete, the next step is to randomly assign values (1, 0) to all unassigned PIs in the patterns in P. This is done in lieu of the forward drive operations in deterministic test generation, where, having set the required PI values to obtain an embryonic test condition, choices are made on the remaining PIs to ensure a detection path to an observation point. The PLA/TG procedure makes a random attempt to establish such a path.

At the end of the random assignment step, what has been generated is a fully specified set of *candidate* patterns P which have the following two properties.

1. They contain the necessary embryonic conditions to test all modeled faults in the AND and OR arrays.
2. They contain arbitrary attempts at sensitizing detection paths for these test conditions.

The next step is *pattern evaluation; i.e.*, does a pattern detect the fault(s) for which it was designed after random assignment of unspecified PIs or has detection been blocked? For either T1 or T2 patterns, this amounts to exactly the same question: Are all word lines that are ORed with the word line *under test* set to the non-

Table 1 PLA/TG results for 19 custom PLAs.

PLA number	*Product terms*	*Inputs*	*Outputs*	*Used crosspoints*	*Test coverage (%)*	*Number of patterns*	*Faults untested*	*Time 370/168 CPU* (s)
1	16	15	9	145	100	77		1.0
2	20	13	9	168	100	64		1.0
3	21	18	4	147	100	108		2.7
4	27	21	9	108	100	92		1.0
5	16	15	7	123	100	59		0.9
6	30	15	4	222	100	112		2.0
7	17	14	3	130	100	67		1.2
8	30	16	8	226	100	107		2.8
9	21	17	6	137	100	70		1.3
10	35	17	21	245	100	85		1.5
11	36	15	7	296	98.9	150	3	5.2
12	17	15	6	179	99.4	125	1	3.1
13	21	16	4	168	100	109		2.3
14	54	14	51	987	99.4	252	5	23.0
15	74	23	24	547	99.6	152	2	17.2
16	19	21	4	133	97.7	51	3	1.8
17	11	12	5	52	100	45		6.1
18	32	21	17	228	100	70		4.9
19	34	13	34	244	100	41		0.9
Average	28	16	12	238	99.7	97		4.1

Table 2 PLA/TG results for 12 fixed-size-module PLAs.

Part number	*Product terms*	*Inputs*	*Outputs*	*Used crosspoints*	*Test coverage (%)*	*Number of patterns*	*Faults untested*	*Time 370/168 CPU* (min:s)
1	62	40	41	1257	97.5	342	31	:57
2	107	52	39	1317	96.5	389	46	1:16
3	98	50	35	819	92.5	388	61	:53
4	66	46	24	733	97.2	186	20	:28
5	73	51	26	622	98.3	304	10	:59
6	104	52	25	1691	98.2	408	29	1:16
7	95	47	41	1315	99.0	301	12	:45
8	92	49	45	1446	97.9	548	30	1:41
9	100	52	32	783	96.6	156	26	:23
10	73	51	29	702	98.7	192	9	:17
11	84	51	28	489	100.0	123	0	:08
12	66	47	35	825	99.6	216	3	:25
Average	85	49	33	999	97.2	296	23	:47

controlling value, *i.e.*, logic *zero*? In proceeding through the fault list, a single analytic question determines if the candidate pattern is valid: Is at least one bit line "turned *off*" in each of the other ANDs ORed with the AND under test? If the answer is *yes*, the corresponding fault is marked as having been detected. At the end of the pattern evaluation operation any pattern in P that does not uniquely detect a fault is deleted.

If any faults remain undetected, passes are taken through the procedure in the following manner.

1. The partial tests required for the untested faults are retrieved from PT1 and PT2.
2. Subsumption, as defined, is performed on this subset.
3. Unspecified PIs are randomly assigned values.
4. Pattern evaluation is performed.

Test coverage climbs quickly on these successive passes. Figure 7 presents fault-coverage percentages *versus* the number of passes through the PLA/TG procedure for 19 PLAs used in an actual product design (discussed in a subsequent section and detailed in Table 1). The average

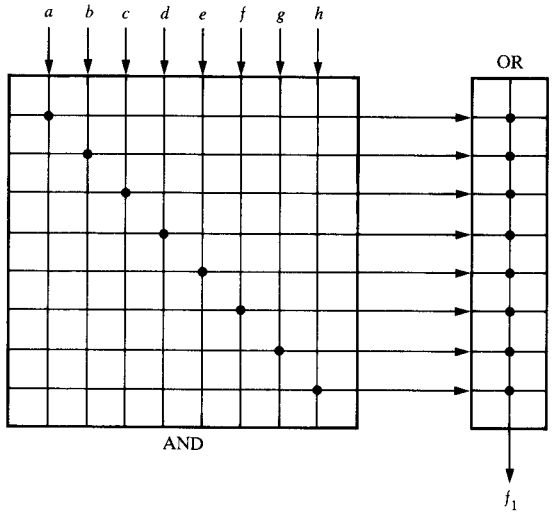

Figure 8 Example of a PLA personality not susceptible to random test path sensitization.

number of passes required to generate a complete set of patterns was 16 for this group of PLAs; the largest number was 45.

In the interest of breaking possible detection "blocks," the following two variations are introduced on these additional passes.

1. If 2BP circuitry is used in the PLA and if 100% coverage is not obtained by the tenth pass, then new partial tests are generated rather than retrieving them from PT1 or PT2. Since there may be more than one way to set up a particular T1 or T2 through the partitioning logic, another random choice may aid fault detection.
2. If on any additional pass no more faults are detected, then the subsume step is dropped on the next pass. It is possible that two patterns "deadlock" so long as one subsumes the other.

In the manner described, PLA/TG continues until all faults are detected or until T_{max} additional passes result in 0% added fault coverage, where T_{max} is set by the user. For the results presented in Tables 1 and 2, T_{max} was set equal to 25.

Special closing routines

Although the previously described heuristic procedure works very well for most PLA personalizations, for a small percentage of personalizations it may not work as well. An example of this is shown in Fig. 8. For these particular personalizations, a special closing routine is utilized at the end of pattern generation. In this case, the generator sets up one bit and then randomly searches for the single, unique sensitizing pattern. This results in a success probability of only 1/128 for the configuration illustrated in Fig. 8.

In general, if a large number of product terms having relatively few used crosspoints in the AND array all feed a single OR line, the probability of obtaining a test is low.

The problem is essentially the same for both T1 and T2 tests. That is, it is necessary to force all other product terms feeding the OR line to *zero*. This can be done on the residual untesteds by selecting each of the related product terms whose output is not *zero* and setting one of its unspecified inputs to *zero*. Since setting one product term to *zero* can result in another product term being forced to a *one*, the closing algorithm must also iterate n times, using random selection techniques, where n is a user-specified variable.

It should be noted that the program used to obtain the results described in this paper employed a much less general closing algorithm. In fact, the basic algorithm has almost always obtained a test for all nonredundant crosspoints.

The other question to be considered at the end of pattern generation is the following: If two-bit partitioning is used, are all stuck-faults in the partitioning circuitry covered? If all modeled faults in the AND or OR arrays are covered, then the only exposure is the input stuck-at-*zero* fault on the lines labeled I1, I2, I3, · · ·, I8 in Fig. 6. We know that BP1 has been set to *one* in a T1 pattern for each word line it is used in, but unless both choices

(I1, I2) = (1, 0) and (0, 1)

have been employed, we are not sure that each node stuck at *zero* is covered. The more frequently the bit line is used, the more likely, given our random choice of input A and B values, that we have tested both faults. But, to guarantee coverage, the following function is performed on any PLA with two-bit partitioning. Pattern set P is searched for the T1 tests for all the uses of a given 2BP output line. The question asked is this: Have both of the required sets of PI values for A and B been used? If *yes*, go to the next partitioned output. If *no*, generate one (two) new T1 test(s) according to the standard procedure using the alternate A, B choice(s).

Remarks on fault modeling

It has understandably been argued that, for completeness, a PLA test generator should fault-model all cross-

point defects, *extra* as well as missing devices [7-10]. We did not do this in PLA/TG, primarily for the following two reasons.

1. To associate a stuck-fault with each intended crosspoint as we have done, and thereby to ensure detection of any missing device, is effectively the same fault model that would have been used had the circuit been designed in traditional logic. Both manufacturing experience and engineering analysis have shown that coverage of this model gives acceptable coverage of multiple stuck-faults and net-to-net shorts. Without experience to the contrary, we did not want to inflate the fault list by something more than a factor of two. In PLA Number 15 in Table 1 there were 547 *used* crosspoints and, in its logical matrix representation, 2750 *unused* crosspoints.
2. In the custom physical design of PLAs, many unused crosspoints are dropped. To avoid generation of useless patterns for nonexistent, "unused" crosspoints, test generations would have to be fed a description of layout, which is not normally a requirement for the missing-crosspoints model.

If, however, experience dictated the need, the extra crosspoint defect could easily be accommodated by PLA/TG.

A valid T1 test through one or more outputs can be used to test for *extra* personalized crosspoints in *both* the AND and the OR arrays.

Consider Fig. 9. If the AND gate has a T1 test and the bit line for an unused crosspoint, u_i, is *zero*, then the value of the product term will also be *zero* if the crosspoint u_i is incorrectly personalized. The presence of this extra personalization is detected at f_i. The same test will also detect an extra personalization in the OR array at v_j. This is detected on the output f_j.

Thus, for a sensitized T1 test, the unused crosspoints in that "row" will be tested wherever the input bit line is *zero* and wherever the output bit line is *zero*. In most cases, *all* unused crosspoints in a given row can be tested by adding at most three additional T1 tests, one for each other possible input value combination of the two-bit partitioning circuits.

Results

The PLA/TG program was run on all 19 custom PLAs used in an actual product design. Initially, 3% of the modeled faults were untested; on analysis all the corresponding gates (used crosspoints) proved to be redundant. As a result all but 14 of these gates were removed prior to design freeze, and the program was run again. Results are

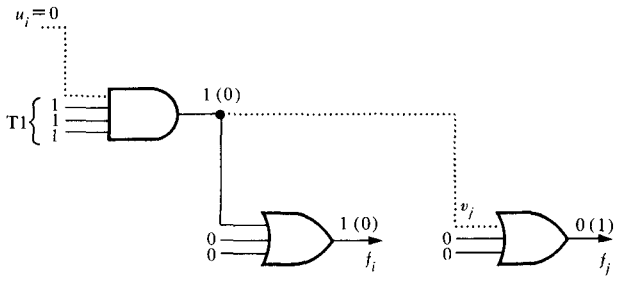

Figure 9 One "row" of a PLA with extra crosspoint connections viewed as conventional AND-OR logic.

given in Table 1. The times given are total IBM 370/Model 168 CPU seconds, measured from the access of the personality description from the matrix-format data base through to the completion of the PLA/TG procedure.

To gain perspective on these results, these same 19 PLAs were converted to conventional logic and then submitted to a well-established, conventional logic test-pattern generation program implemented in assembly language. The run time required for the conventional technique was 6.5 times greater than for PLA/TG. If, however, the times required to convert the PLAs to conventional logic and to create the files for the test generator are included, then this ratio becomes 20-to-1.

Additional results for the larger, fixed-size arrays used in another design are tabulated in Table 2, where again each untested crosspoint was, on inspection, shown to be redundant.

Conclusions

The PLA/TG procedure addresses a subset of the combinatorial logic test generation problem, *i.e.*, programmable logic arrays. Crosspoint defect detection patterns are generated through direct inspection of the array personality plus a random sensitizing of the required bias conditions. The advantage of this procedure over classical combinatorial methods or other PLA test-pattern generators lies in its computational speed.

A valuable by-product of the procedure is its use as a design aid in the elimination of logic redundancies in PLAs. In practice, PLA/TG quickly enumerates all unnecessary crosspoints.

As described in this paper, PLA/TG is limited to PLAs with optional input bit partitioning. A natural extension of the work would include PLA offerings that included input and output latching.

References

1. P. Agrawal and V. D. Agrawal, "Probabilistic Analysis of Random Test Generation Method of Irredundant Combinational Logic Networks," *IEEE Trans. Computers* **C-24,** 691-694 (1975).
2. H. D. Schnurmann, E. Lindbloom, and R. G. Carpenter, "The Weighted Random Test-Pattern Generator," *IEEE Trans. Computers* **C-24,** 695-700 (1975).
3. J. J. Shedletsky, "A Rationale for the Random Testing of Combinational Digital Circuits," *Proc. Comp. Conf. 1975 Fall* (Washington, DC, September 1975) **11,** 5-8 (1975).
4. K. P. Parker, "Adaptive Random Test Generation," *J. Design Aut. Fault-Tol. Comp.* **1,** 62-83 (1976).
5. T. W. Williams and E. B. Eichelberger, "Random Patterns Within a Structured Sequential Logic Design," *Digest of 1977 Semiconductor Test Symposium*, Cherry Hill, NJ, October 25-27, 1977, pp. 19-28.
6. H. Fleisher and L. I. Maissel, "An Introduction to Array Logic," *IBM J. Res. Develop.* **19,** 98-109 (1975).
7. E. I. Muehldorf and T. W. Williams, "Optimized Stuck Fault Test Pattern Generation for PLA Macros," *Digest of 1977 Semiconductor Test Symposium*, Cherry Hill, NJ, October 25-27, 1977, pp. 89-101.
8. C. W. Cha, "A Testing Strategy for PLAs," *Proceedings of the 15th Design Automation Conference*, Las Vegas, NV, June 19-21, 1978.
9. D. L. Ostapko and S. J. Hong, "Fault Analysis and Test Generation for Programmable Logic Arrays," *Proceedings of the 15th Annual International Conference on Fault Tolerant Computing*, Toulouse, France, June 21-23, 1978, pp. 83-89.
10. V. K. Agarwal, "Multiple Fault Detection in PLA's," *Proceedings of the 9th Annual International Conference on Fault Tolerant Computing*, Madison, WI, June 20-22, 1979, pp. 227-234.

Received April 17, 1979; revised June 25, 1979

The authors are located at the IBM System Communications Division laboratory, Kingston, New York 12401.