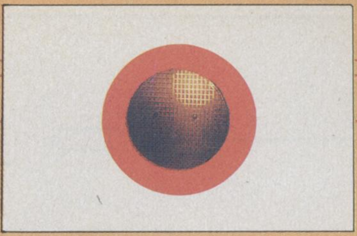
This article is translated in revised form from *Monographs of the Design Automation Working Group of the Information Processing Society of Japan,* Vol. 19, No. 3, Nov. 1983, and is reprinted with permission. © 1983 IPSJ



Design for Testability for Complete Test Coverage

Very large scale circuits present great obstacles to complete testing. A pattern generation algorithm offers a direct approach to the problem.

Akira Motohara and Hideo Fujiwara, Osaka University

The problem of reliability is gaining increasing importance with the rapid advance of semiconductor technology toward very large scale integration of logic circuits. The very large scale of the circuits, however, makes test pattern generation extremely difficult, and when test patterns cannot be obtained within the allowed computation time, aborted faults make it difficult to achieve a high rate of test coverage.

Designing for testability has been offered as a solution of this problem (see Williams and Parker[1]). Methods to reduce the complexity of testing for sequential circuits to the level for combinational circuits have been proposed and achieved.[2-4] However, for combinational circuits of large scale, it is still extremely difficult to achieve 100 percent test coverage.

Among the most promising DFT methods are those aimed at complete test coverage through the addition of a few hardware elements to the circuit. This redundant hardware is called *test points.*[5,6] The test point strategies we are considering here fall into four categories:

(1) Insert AND, OR gates.

(2) Change NOT gates to NOR, NAND, gates.

(3) Add primary input points for control.

(4) Add primary output points for observation.

Summary

Some design-for-testability techniques, such as level-sensitive scan design, scan path, and scan/set, reduce test pattern generation of sequential circuits to that of combinational circuits by enhancing the controllability and/or observability of all the memory elements. However, even for combinational circuits, 100 percent test coverage of large-scale circuits is generally very difficult to achieve. This article presents DFT methods aimed at achieving total coverage. Two methods are compared: One, based on testability analysis, involves the addition of test points to improve testability before test pattern generation. The other method employs a test pattern generation algorithm (the FAN algorithm). Results show that 100 percent coverage within the allowed limits is difficult with the former approach. The latter, however, enables us to generate a test pattern for any detectable fault within the allowed time limits, and 100 percent test coverage is possible.

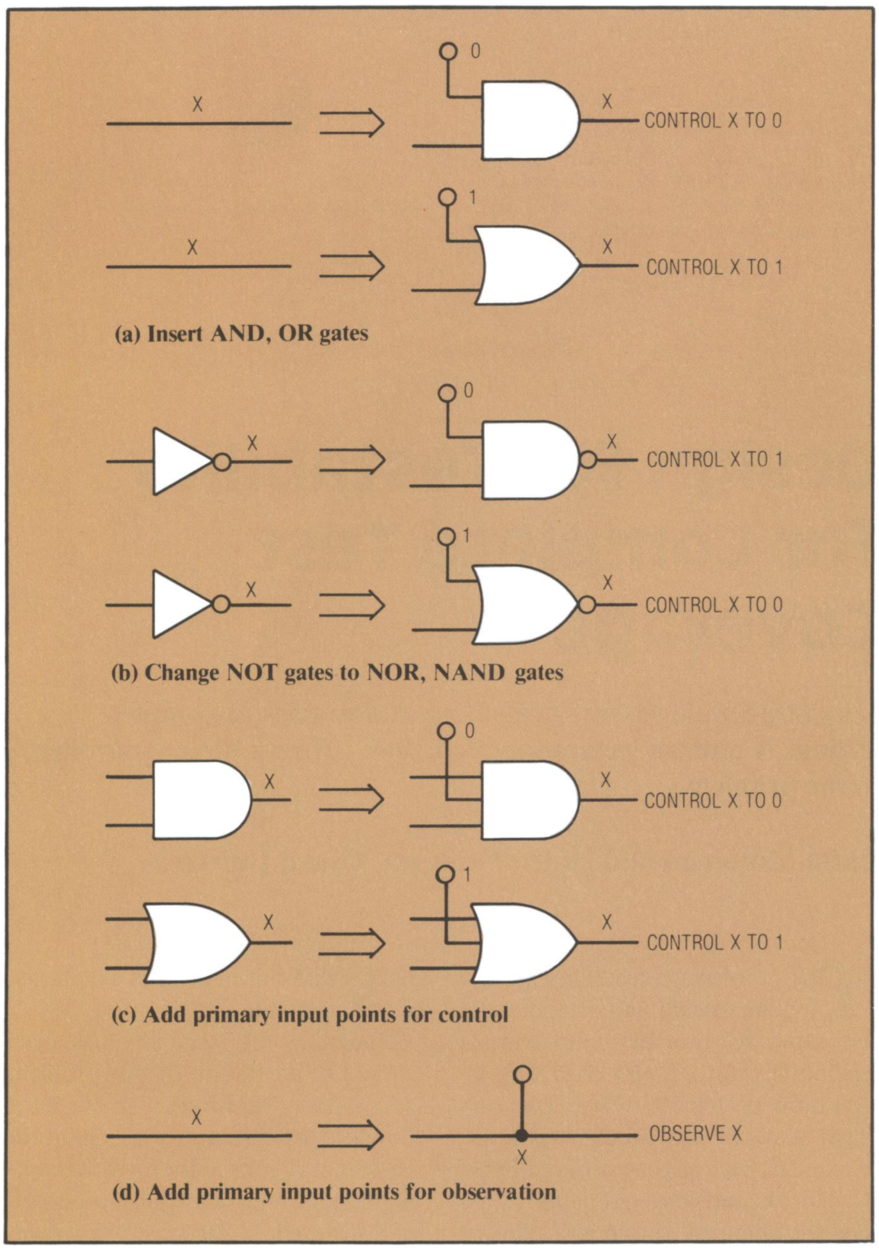


Figure 1. Types of test points.

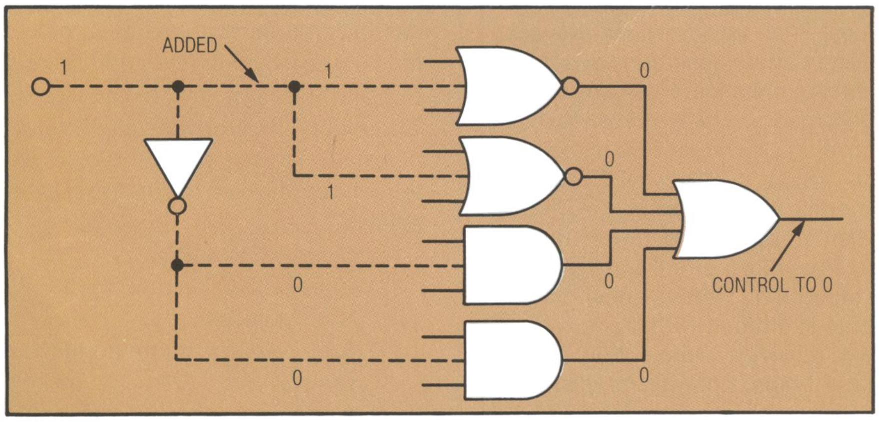


Figure 2. Addition of test points.

These strategies are illustrated in Figure 1.

When two or more of the control test points (Figure 1, a-c) are to be used at the same time, they are merged into one primary input to reduce the number of additional input points (Figure 2).

Moreover, if scan-path techniques, as shown in Figure 3, are used, one test point will then correspond to one flip-flop, and the increase in the number of external points can be avoided.

We have investigated two methods of altering circuits to facilitate testability through the addition of test points. The first method is based on testability analysis and involves the addition of test points to improve testability before a test pattern is generated. Obtained results show that 100 percent test coverage within the allowed time limits is difficult to achieve with this method. The second method employs a test pattern generation algorithm, enabling us to generate a test pattern for any detectable fault within the allowed time limits, and making 100 percent test coverage possible. We programmed the two methods, and after evaluating some circuits with several thousand gates, we were able to obtain very favorable results with the second method. In this article we examine and compare these methods.

Circuits considered here are combinational circuits consisting of AND, OR, NOT, NAND, and NOR elements, and faults are assumed to be single stuck-at faults.

DFT through testability analysis

To increase the effectiveness of test pattern generation, testability measures, which express the ease or difficulty of testing, are used in two algorithms: Podem[7] and Fan,[8] short for path-oriented decision-making and fanout-oriented test generation algorithm, respectively.

Here we draw attention to the measurement of testability, and we describe methods of design modification that facilitate testing through the improvement of testability. The overall flow of these DFT methods is shown in

Figure 4. Although the entire process could be automated, we have implemented the system in interactive form to improve efficiency by enabling the designer to input the choice of signal lines needing improvement and the decision as to whether the desired degree of testability has been achieved.

Testability analysis. Various measures have been proposed as measures of testability.[9-11] Here we use Goldstein's measures,[9] which express the costs of controlling and observing. Thus, to maximize testability, these costs must be minimized. Goldstein proposed six measures in all, three of which apply to combinational circuits:

- [CC^1 (L)] The smallest number of signal lines which must have their logical values set in order to set the logical value of signal line L to 1; called signal line L's "1" controllability cost.
- [CC^0 (L)] Defined in the same way as [CC^1 (L)]; called signal line L's "0" controllability cost.
- [CO (L)] The smallest number of signal lines which must have their logical values set in order to propagate the value of signal line L to primary output; called the observability cost of signal line L.

Each of these values can be found for any given circuit through simple calculations.

Selection of signal lines needing improvement. At this point the designer can see from the system the greatest controllability and observability cost values. Then he decides whether to improve the controllability or the observability and sets the threshold values. Values exceeding threshold values are then improved, starting from those nearest the primary input signal line.

Test point insertion location search. Inserting test points as shown in Figure 1, a-c, markedly improves the controllability for the signal lines on the output side of the insertion point, leaving the input side unchanged. Accordingly, test points are inserted several gates to the input side of signal lines

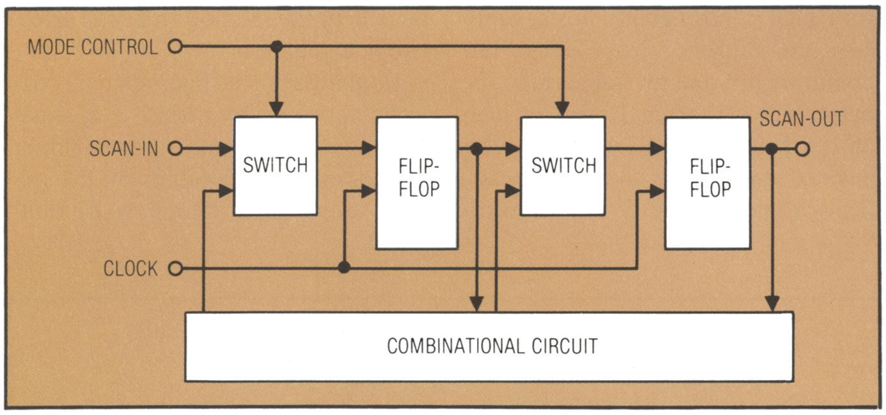


Figure 3. Scan path method.

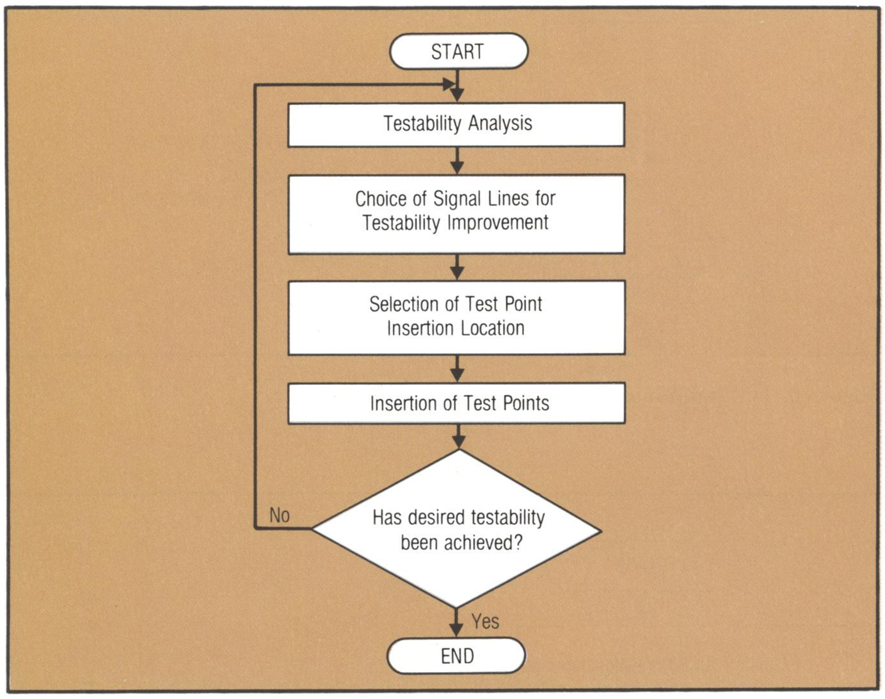

START
Testability Analysis
Choice of Signal Lines for Testability Improvement
Selection of Test Point Insertion Location
Insertion of Test Points
Has desired testability been achieved?
No
Yes
END

Figure 4. Work sequence in testability analysis DFT method.

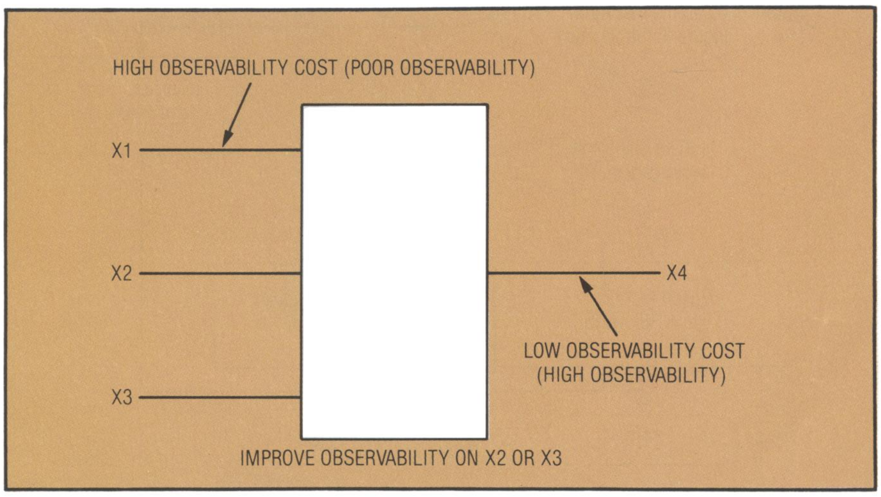

HIGH OBSERVABILITY COST (POOR OBSERVABILITY)
X1
X2
X3
X4
LOW OBSERVABILITY COST (HIGH OBSERVABILITY)
IMPROVE OBSERVABILITY ON X2 OR X3

Figure 5. Test point insertion location search.

with high controllability costs. Similarly, to improve observability, test points are inserted several gates to the output side of the signal lines with high observability costs. The exact placement of test points is an extremely difficult problem, however, and we introduce here the method that was effective in actual tests.

Looking at signal line with high controllability cost, we find two situations. In the first, one of several input lines dominates the others. In the second, several input lines have simultaneously high controllability cost. In the former case, it is best to insert a test point a bit further toward the input side. In the latter case, insertion of the test point is best on the signal line itself. Space does not permit us to go into the standards used in deciding whether we are faced with the first or second situation, but we have found it generally difficult to determine the most suitable standard.

When a signal line has high controllability cost, we find that either the observability cost drops rapidly, or it does not. With a rapid drop, we assume a problem in controllability, and we then search for a proper point of test-point insertion to lower the controllability cost (see Figure 5). If the observability cost does not drop rapidly, the next step is to analyze output. If no sudden drop in observability cost can be seen all the way through to the primary output, then a test point is inserted on the original signal line a few gates toward the output side, as in Figure 1d. Insertion so that the observability cost is one half of that for the original signal line was the most effective.

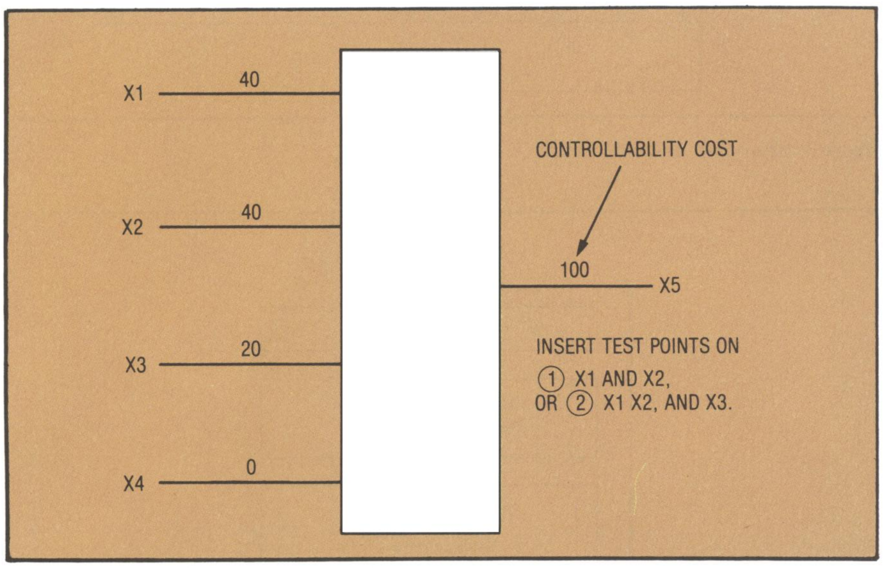


Figure 6. Test point insertion.

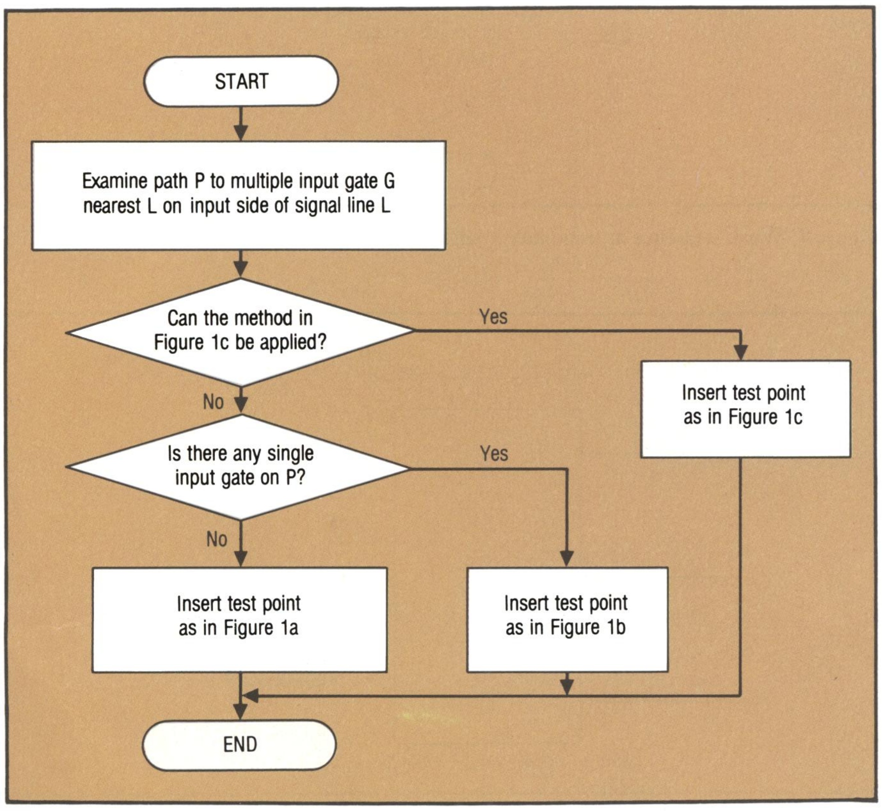

START
Examine path P to multiple input gate G nearest L on input side of signal line L
Can the method in Figure 1c be applied?
Yes
No
Insert test point as in Figure 1c
Is there any single input gate on P?
Yes
No
Insert test point as in Figure 1a
Insert test point as in Figure 1b
END

Figure 7. Selection of test point types.

Test point insertion methods. Figure 6 shows one example of an insertion location. The numbers appearing above each line express the controllability costs. In this example, there is no need to insert test points on input lines X1 through X4 to lower the controllability cost on line X5. Insertion of two test points, one on each of the lines X1 and X2 with poor controllability, or three test points on lines X1, X2, and X3, would be sufficient. This would spare us the problem of the standards to be applied when inserting the test point on the input line.

For the test point, a gate could be added as shown in Figure 1a. However, the addition of a gate would cause delay in a circuit designed for high-speed operation, requiring a redesign of the circuit with the new test point. It is preferable, if possible, to use one of the methods shown in Figure 1, b and c. When adding a gate to signal line *L*, we follow the flow chart in Figure 7 to determine whether method 1b or 1c can be used.

Experimental results. After programming the design method described above, we applied it to some circuits of several thousand gates and surveyed how the test coverage changed with the insertion of test points. The computer used for our calculations was the large computer at Osaka University, an NEC System 1000 (computation speed: 15 MIPS). We used Fortran for programming, and for the test pattern generation algorithm, we used the FAN algorithm.[8] Aborted faults were the faults that remained undetected after more than 100 backtrack operations.

The characteristics of the circuits before designing for testability are described in Table 1. Test coverage was defined as follows:

$$\text{Test comprehensiveness} = \frac{\text{Number of detected faults}}{\text{Number of detected faults} + \text{aborted faults}}$$

Our findings are shown in the graph in Figure 8. Although there are cases, as shown by circuit 3, where the insertion of test points causes a clear drop in aborted faults, there are also cases, as with circuit 2, where the addition of test points makes the testing more difficult.

Taking these results as evidence, it is clear that the measures of testability do not always reflect the ease or difficulty of testing accurately. Also, although the insertion of test points clearly improved testability, it would be impossible to say that in all cases the circuit had been designed for testability.

DFT through test pattern generation algorithms

As shown above, it is difficult to obtain complete test coverage with methods based on testability analysis. In fact, for the number of test points added, the test coverage did not effectively improve. Here, therefore, we expand upon a method that ensures generation of test patterns for all detectable faults within the allowed computation time—in other words, a method of adding test points that will result in 100 percent test coverage for all detectable faults. This method pays special attention to aborted faults occurring after the test pattern generation algorithm is applied, and it involves the subsequent placement of test points to make such faults easier to test.

Types of aborted faults. With a test pattern generation algorithm that sends a signal whose value varies with the existence or absence of a fault (called a "faulty signal") along a signal line on which a fault is inserted (called a "faulty line"), we try propagating a faulty signal from the faulty line to a primary output. When we start a backtrack cycle of a specific number (say 100 times) of backtracks, t, the test pattern generation is aborted, indicating an aborted fault. We can assume that an aborted fault is difficult to test for. Below, a fault that is not aborted—i.e., that does not appear after a limited number of backtracks, t—is called a *t-easily tested fault;* when a fault is aborted, it is called a *t-difficult-to-test-for fault.* When no particular number of backtracks is specified, t may be omitted.

From the perspective of test pattern generation, aborted faults can be divided into three types:

Fault type 1. An aborted fault caused by difficulty in producing a faulty signal.

Fault type 2. A fault for which faulty signal production is easy, but propagation of the faulty signal to the next gate is difficult.

Fault type 3. A fault for which observation of the faulty signal at primary output is difficult, even though both the production and propagation of the faulty signal for more than one gate were accomplished without difficulty.

We can determine into which of these categories the fault falls by running a test pattern generation algorithm.

Table 1.
Characteristics of circuits before designing for testability.

Circuit Number	Gates	Defined Faults	Aborted Faults	Test Coverage (%)
#1	1165	2747	45	98.31
#2	2348	5888	24	99.57
#3	2592	6348	30	99.57

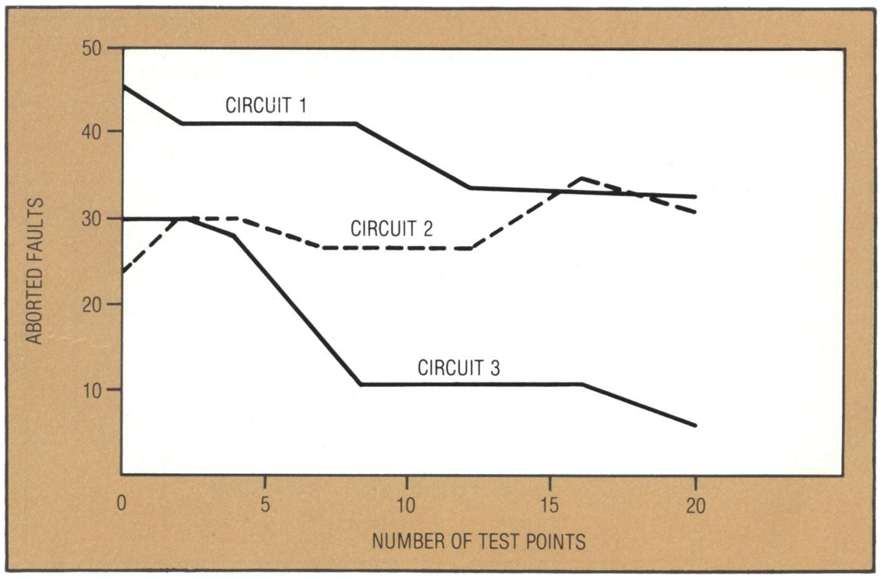


Figure 8. Experimental results of testability analysis method.

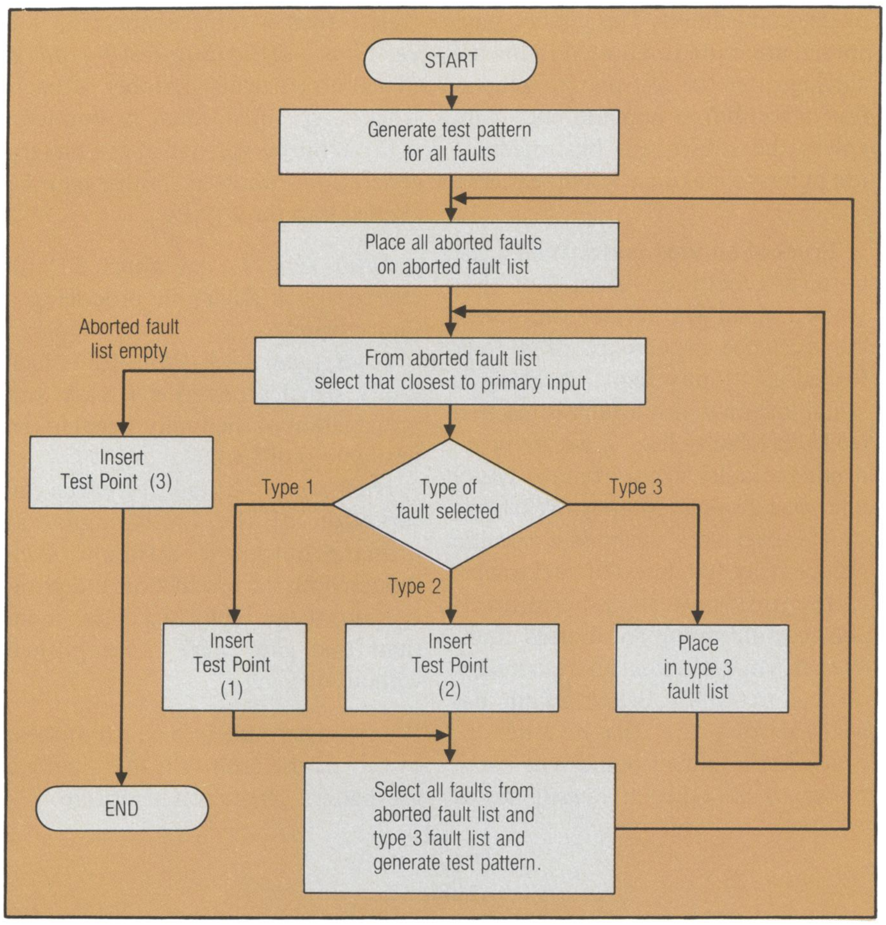

START
Generate test pattern for all faults
Place all aborted faults on aborted fault list
Aborted fault list empty
From aborted fault list select that closest to primary input
Insert Test Point (3)
Type 1
Type of fault selected
Type 3
Type 2
Insert Test Point (1)
Insert Test Point (2)
Place in type 3 fault list
END
Select all faults from aborted fault list and type 3 fault list and generate test pattern.

Figure 9. Overall flow of test point insertion.

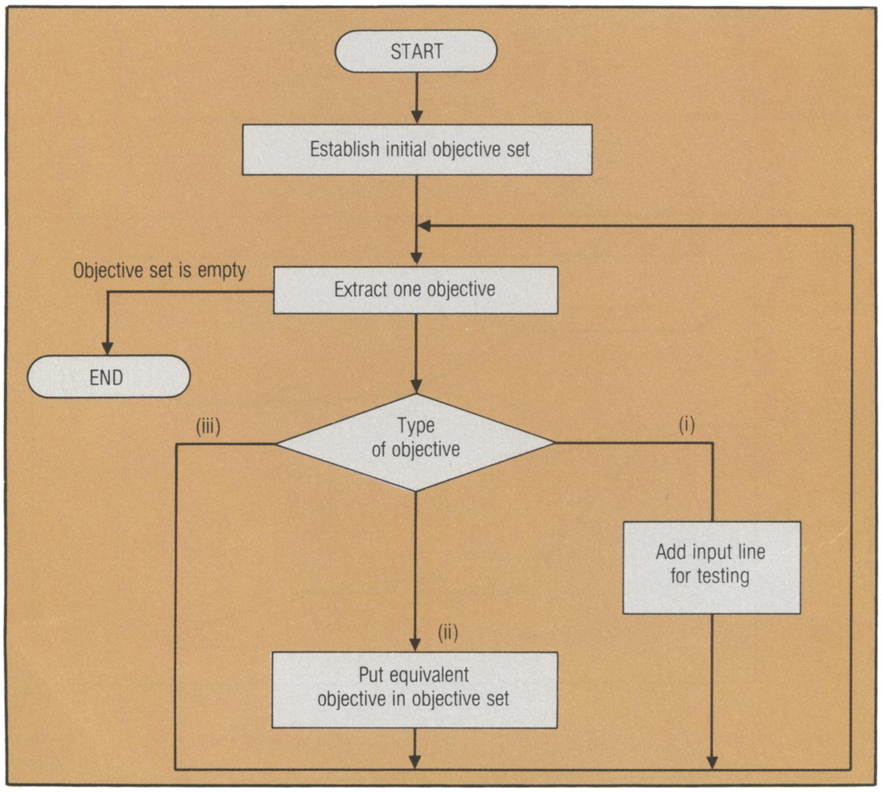

START
Establish initial objective set
Objective set is empty
Extract one objective
END
(iii)
Type of objective
(i)
Add input line for testing
(ii)
Put equivalent objective in objective set

Figure 10. Test point insertion for type 1 and type 2 faults.

Insertion of test points. The overall flow is shown in Figure 9. The test points are of two types, shown in Figure 1, c and d. The following discussion covers the methods of test point insertion based on the type of aborted fault:

Fault type 1. Test point insertion follows the flow chart in Figure 10, insuring that a faulty signal can be produced without backtracking.

When we are trying to assign a certain value to a certain signal line, the value and the signal line are together called an "objective." Several objectives held simultaneously are called a "set of objectives." When signal line L experiences a stuck-at fault at either 0 or 1 in a type 1 fault, the initial objective is $(0, L)$ or $(1, L)$.

Looking at the "type of objective" frame in Figure 10, we can see that for type (i), we can meet the objective by inserting a test point of the type seen in Figure 1c. For type (ii), the objective cannot be met. For type (iii), the signal line is a primary input, and no test point is needed. With type (ii), we need to see what type of value on the input would enable the objective to be achieved. That value would be called the "equivalent objective" (Figure 11).

After several test points have been inserted in accordance with the flow chart in Figure 10, the objective set will become empty and the insertion of the test points will be complete. For testing, these input lines will become one test point according to the circuit construction in Figure 2.

Fault type 2. Test point insertion for a type 2 fault is also carried out according to Figure 10. However, the initial objective set is determined in a different manner from that for a type 1 fault. When, as in Figure 12, we have a stuck-at 0 fault on signal line L, the initial objective set is

$$\{(1,L),\ (0,M),\ (0,N)\}$$

Fault type 3. Test point insertion for a type 3 fault is carried out only after test point insertion has been completed for fault types 1 and 2. The test point used is shown in Figure 1d, and the following procedures are followed to minimize the number of test points to be inserted:

First, for each fault, make a search for possible locations that ensure that test point insertion facilitates detection. Once located, a faulty signal should be produced and propagation attempted, in the same manner as for classifying aborted faults.

Next, choose the smallest number of test point insertion locations that will enable all faults to be detected.

If we are to eliminate all aborted faults by the above method, any fault occurring in the newly added test points must be easily testable. Following the flow chart in Figure 9, we can avoid a difficult-to-detect fault in the test points in the manner shown briefly below:

First, for the test point insertion shown in Figure 1c, used with fault types 1 and 2, consider the example shown in Figure 13. A stuck-at 0 fault at test point *X* is equivalent to a stuck-at 0 fault at *Y* and is thus easy to test for. A stuck-at 1 fault at *X* becomes redundant if the stuck-at 0 fault at *Y* is redundant, and becomes testable if the stuck-at 0 fault at *Y* is testable, because it can be tested by the same test pattern with *X* = 0. Thus, the stuck-at 1 fault at *X* can be seen as the equivalent of the stuck-at 0 fault at *Y*. Accordingly, the number of difficult-to-test-for faults does not increase.

Second, to use the method shown in Figure 1d for type 3 faults, consider the example shown in Figure 14. This type of test point insertion is carried out after types 1 and 2 have been eliminated from the circuit. Accordingly, it is easy to set signal line *Y* at either 0 or 1, making it easy to test signal line *X*.

Experimental results. The results we obtained from programming the DFT method incorporating a test pattern generation algorithm are shown in Table 2. The computer, the language, and the applied circuits we used were all the same as those we used for the testability analysis method. As shown in Table 2, with the test pattern genera-

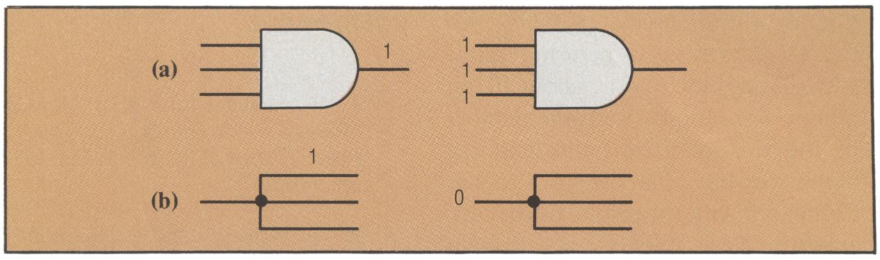


Figure 11. Equivalent objectives.

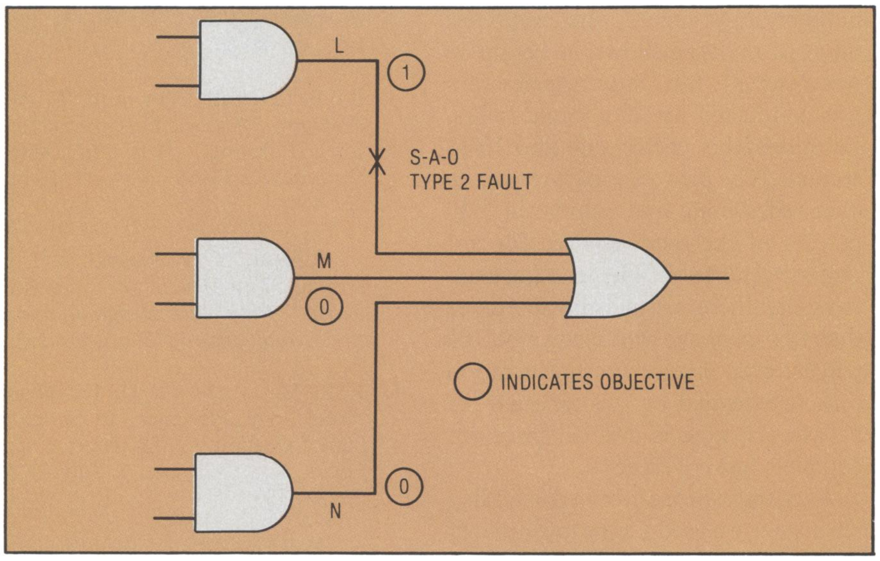


Figure 12. Initial objective for type 2 fault.

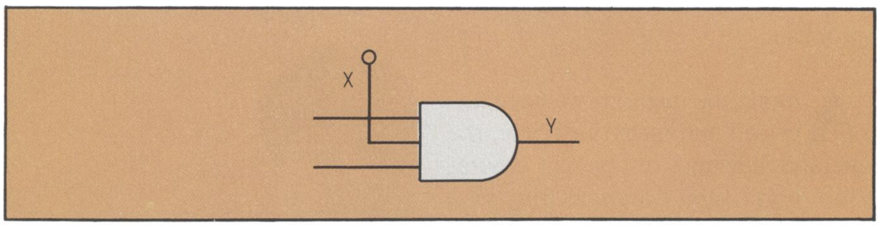


Figure 13. Example of test points for fault types 1 and 2.

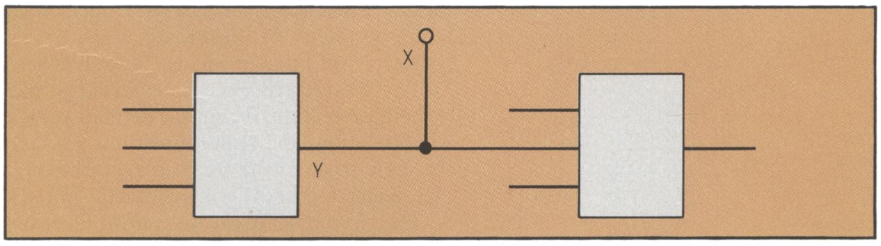


Figure 14. Example of test points for fault type 3.

Table 2.
Results of design for testability with test pattern generation algorithm.

Circuit Number	Added Input	Added Output	Computation Time (sec.)	Test Coverage (%)
#1	3	8	13.7	100.0
#2	1	6	11.3	100.0
#3	4	4	14.5	100.0

tion algorithm, circuits of from 1000 to 3000 gates were 100 percent testable with the addition of only about 10 test points. The ratio between the number of the first aborted faults and the number of test points was 3.4 to 4.1. Compared with the results of the testability analysis method, insertion of test points was highly effective.

Computation time. To lessen the number of test points, whenever a test point of the type shown in Figure 1c was inserted, test pattern generation was conducted for all aborted faults. Because of this, each circuit took 10-15 seconds. The process took precedence over redesigning and comprised 7-30 percent of the time required for test pattern generation. However, considering that there is usually no guarantee that test coverage will always be 100 percent even if test pattern generation time is extended by this amount, we believe the time is a small price for complete test coverage.

As to the expense involved in adding hardware, we used a very low number of test points and only those which could be realized at a minimum cost; thus we believe that cost is not a great problem.

Among the methods we have discussed and mentioned here, the method utilizing a test pattern generation algorithm obtained the best results by far. We attribute this to our use of a direct solution to the problem of difficult-to-test-for faults. □

Acknowledgments

We wish to thank Hiroshi Ozaki, professor emeritus at Osaka University, for his support and encouragement of this work. We also thank the members of the sixth chair of the Department of Electronic Engineering, Osaka Univeristy, for their valuable discussions. Special thanks to Kenjiro Sasaoka (presently with MITI) for the testability programming as well as the results of the test pattern generation experiment.

References

1. T. W. Williams and K. P. Parker, "Design for Testability—a Survey," *Proc. IEEE,* Vol. 71, No. 1, Jan. 1983, pp. 98-112.

2. M. J. Williams and J. B. Angell, "Enhancing Testability of Large Scale Integrated Circuits via Test Point and Additional Logic," *IEEE Trans. Computers,* Vol. C-22, No. 1, Jan. 1973, pp. 46-60.

3. E. B. Eichelberger and T. W. Williams, "A Logic Design Structure for LSI Testing," *Proc. 14th Design Automation Conf.,* June 1977, pp. 462-468.

4. S. Funatsu, N. Wakatsuki, and T. Arima, "Test Generation Systems in Japan," *Proc. 12th Design Automation Symp.,* June 1975, pp. 114-122.

5. J. P. Hayes and A. D. Friedman, "Test Point Placement to Simplify Fault Detection," *IEEE Trans. Computers,* Vol. C-23, No. 7, July 1974, pp. 727-735.

6. W. Coy and A. Vogel, "Inserting Test Points for Simple Test Generation," *Proc. Fourth Int'l Conf. Fault-Tolerant Systems and Diagnostics,* 1981, pp. 180-186.

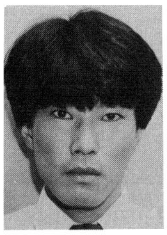
7. P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," *IEEE Trans. Computers,* Vol. C-30, No. 3, Mar. 1981, pp. 215-222.

8. H. Fujiwara and T. Shimono, "On the Acceleration of Test Generation Algorithms," *Proc. 13th Int'l Conf. Fault Tolerant Computing,* June 1983, pp. 98-105; also *IEEE Trans. Computers,* Vol. C-32, No. 12, Dec. 1983, pp. 1137-1144.

9. L. H. Goldstein, "Controllability/Observability Analysis of Digital Circuits," *IEEE Trans. Circuits and Systems,* Vol. CAS-26, No. 9, Sept. 1979, pp. 685-693.

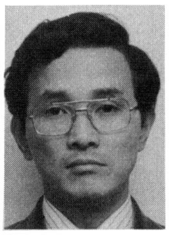
10. P. G. Kovijanic, "Testability Analysis," *Digest of Papers, 1979 Test Conf.,* pp. 310-316.

11. H. Fujiwara and H. Ozaki, "A New Measure for Test Generation by a Heuristic Method," IECE monograph, EC80-38, Oct. 1980, pp. 1-8.



Akira Motohara is a graduate student in the Department of Electronic Engineering, Faculty of Engineering, at Osaka University. His research interests include test pattern generation, design for testability, logic simulation, and fault simulation.

He received the BE degree in electronic engineering from Osaka University in 1983. He is a member of the Institute of Electronics and Communication Engineers of Japan.



Hideo Fujiwara has been with the Department of Electronic Engineering of Osaka University since 1974. In 1981 he was a visting assistant professor at the University of Waterloo, and, in 1984, a visiting associate professor at McGill University, Canada. His research interests include switching theory, design for testability, test pattern generation, fault simulation, built-in self-test, and fault-tolerant systems.

Fujiwara received the BE, ME, and PhD degrees in electronic engineering from Osaka University in 1969, 1971, and 1974 respectively. He was member of the Technical Program Committee of the 1982 International Symposium on Fault-Tolerant Computing. He received an Institute of Electronics and Communication Engineers Young Engineer Award in 1977. He is a senior member of the IEEE, a member of the IECE of Japan, and a member of the Information Processing Society of Japan.

The authors' address is Department of Electronic Engineering, Faculty of Engineering, Osaka University, 2-1 Yamada-Oka, Suita City, Osaka 565, Japan.