

II-331 FAN: A Fast Test Generation System for VLSI Circuits

Hideo FUJIWARA*

Abstract

A new test generation algorithm named FAN (fanout-oriented test generation algorithm) is presented. In the FAN algorithm, several techniques are adopted to reduce the number of backtracks and to accelerate test generation. The efficiency of the FAN is compared with the PODEM algorithm, combining each algorithm with fault simulation. The experimental results show that the FAN algorithm is faster and more efficient than the PODEM algorithm.

1. Introduction

The very large scale of logic circuits makes test pattern generation extremely difficult. Recent work has established that the problem of test generation, even for monotone circuits, is NP-complete¹⁾. Hence, it appears that the computation is, for the worst case, exponential with the size of the circuit. Designing for testability has been offered as a solution of this problem¹⁾. The techniques using shift registers such as LSSD²⁾, Scan Path³⁾, etc., can reduce the complexity of testing for sequential circuits to the level for combinational circuits. Hence, for these LSSD-type circuits, it is sufficient to develop a fast and efficient test generation algorithm only for combinational circuits. Many test generation algorithms have been proposed over the years^{4~8, 11)}. The most widely used is the D-algorithm reported by Roth⁵⁾. The PODEM algorithm developed by Goel⁷⁾ was shown to be faster than the D-algorithm.

Since all these algorithms, D-algorithm, PODEM, and FAN, are complete, given enough time, they can generate test patterns for each testable fault, i.e., 100% fault coverage can be achieved. However, being limited in computing time, we have to give up continuing test generation for some faults, e.g., faults for which the number of backtracks exceed some value, say 10 or 1,000. Such faults, called **aborted faults**, make it difficult to achieve a high rate of fault coverage. In this paper, we first present the FAN algorithm and then consider the relationship among limitation of backtracks, fault coverage and computation time. In FAN, several techniques are adopted to reduce the number of backtracks in the algorithm and to accelerate test generation. The efficiency of FAN is compared with PODEM, combining each algorithm with fault simulation. The experimental results show that FAN is faster and more efficient than PODEM. It is shown that fault coverage and computation time are both very susceptible to influences from limitation of backtracks. If we set an appropriate limit on the number of backtracks, the ATPG system based on the FAN algorithm can achieve a high fault coverage at a high rate of speed for all combinational

* Department of Electronics and Communications, Faculty of Engineering, Meiji University.

circuits for benchmark.

2. Strategies in the FAN Algorithm

In generating a test, a decision tree is created in which there is more than one choice available at each decision node. The initial choice is arbitrary but it may be necessary during the execution of the algorithm to return and try another possible choice. In order to accelerate the algorithm it is necessary to reduce the number of these backtracks, and to shorten the processing time between backtracks. The reduction of the number of backtracks is particularly important. In order to reduce the number of backtracks, it is important to find the nonexistence of solution as soon as possible. In the **branch and bound** algorithm, when we find that there exists no solution below the current node in the decision tree, we should backtrack immediately to avoid the subsequent unnecessary search.

In the following, we shall present each strategy used in the FAN algorithm.

- In each step of the algorithm, determine as many signal values as possible that can be **uniquely** implied.

To do this we take the implication operation that completely traces such signal determination both forwards and backwards through the circuit. Moreover, we take the following process.

- Assign a faulty signal D or \bar{D} that is **uniquely** determined or implied by the fault in question.

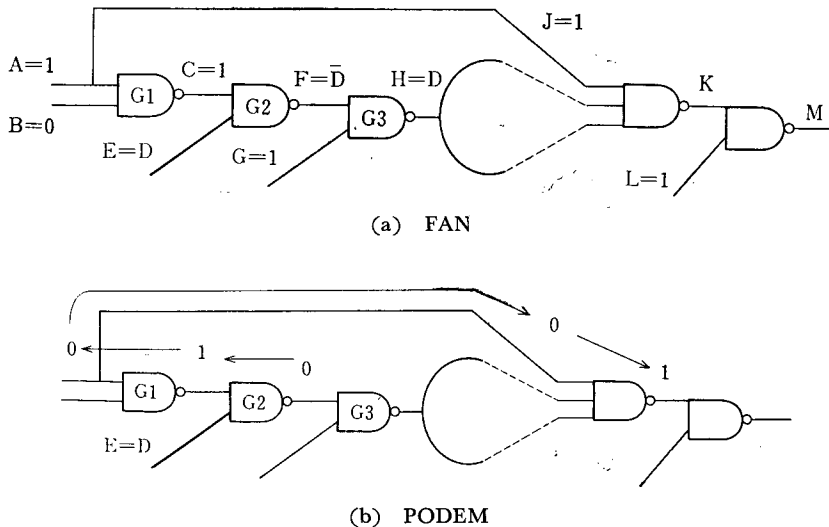


Fig. 1 Effect of unique sensitization

Consider the circuit of Fig. 1(a). Supposing that the D-frontier consists of a single gate, we often have specific paths such that every path from the site of the D-frontier to a primary output always goes through those paths. In this example, every path from gate G_2 to a primary output passes through the paths F-H and K-M. In order to propagate the value D or \bar{D} to a primary output we have to propagate the faulty signal along both F-H

and K-M. Therefore, if there exists a test at this point, paths F-H and K-M should be sensitized. Then we have the assignment $C=1$, $G=1$, $J=1$ and $L=1$ to sensitize them. This partial sensitization which is uniquely determined is called a **unique sensitization**. In Fig. 1(a) after the implication of this assignment we have $A=1$, $B=0$, $F=\bar{D}$ and $H=D$ without backtracking. On the other hand, PODEM sets the initial objective $(0, F)$ to propagate the faulty signal to line F and performs the backtrace procedure. If the backtrace performs along the path as shown in Fig. 1(b), we have the assignment $A=0$ which implies $J=0$ and $K=1$. Though no inconsistency appears at this point, an inconsistency or the disappearance of the D-frontier will occur in the future when the faulty signal propagates from H to K. Although the PODEM algorithm can find such an inconsistency by using X-path check, the backtracking from $A=0$ to $A=1$ is unavoidable.

- When the D-frontier consists of a single gate, apply a unique sensitization.

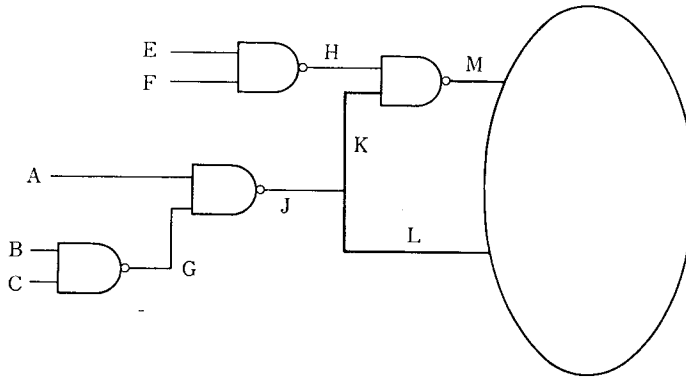


Fig. 2 Head lines.

When a signal line L is reachable from some fanout point, i.e., there exists a path from some fanout point forwards to L, we say that L is **bound**. A signal line that is not bound is said to be **free**. When a free line L is adjacent to some bound line, we say that L is a **head line**. In Fig. 2, lines A, B, C, E, F, G, H, and J are all free, and lines K, L, and M are bound. Among the free lines, J and H are head lines of the circuit since J and H are adjacent to the bound lines L and M, respectively.

The backtrace procedure in PODEM traces a single path backwards to a primary input. However, to avoid useless backtracking, it is better to stop the backtrace operation at a head line and to let its line justification wait until the last stage of test generation. Since sub-circuits composed of only free lines and the associated gates are fanout-free, line justification can always be performed without backtracking.

- Stop the backtrace at a head line, and postpone the line justification for the head line to later.

Performing a unique sensitization, we need to identify paths that would be uniquely sensitized. Also, we need to identify all the head lines in the circuit. These must be identified and this topological information should be stored somehow before the test generation starts. The computation time of these preprocesses can be, however, as small as **negligible** compared with the total computation time for test generation.

- **Multiple backtrace**, i.e., concurrent backtracing of more than one paths is more efficient than the backtrace along a single path.

In the backtrace of PODEM, an objective is defined by a pair of an objective value and an objective line. An objective which will be used in the multiple backtrace in FAN is defined by a triplet:

$$(s, n_0(s), n_1(s))$$

where s is an objective line, $n_0(s)$ is the number of times the object value 0 is required to be set on s , and $n_1(s)$ is the number of times the object value 1 is required to be set on s . The multiple backtrace starts with more than one initial objectives, i.e., a set of **initial objectives**. Beginning with the set of initial objectives, a set of objectives that appear during the procedure is called a set of **current objectives**. A set of objectives that will be obtained at head lines is called a set of **final objectives**.

$$(s, n_0(s), n_1(s)) = (s, 1, 0)$$

and an initial objective required to set 1 to s is

$$(s, n_0(s), n_1(s)) = (s, 0, 1).$$

Working breadth-first from these initial objectives backwards to head lines, we determine the next objectives from the current objectives successively as follows:

- 1) AND gate: Let X be an input that is the easiest to control setting 0. Then

$$n_0(X) = n_0(Y), \quad n_1(X) = n_1(Y)$$

and for other inputs X_i

$$n_0(X_i) = 0, \quad n_1(X_i) = n_1(Y)$$

where Y is the output of the AND gate.

- 2) OR gate: Let X be an input that is the easiest to control setting 1. Then

$$n_0(X) = n_0(Y), \quad n_1(X) = n_1(Y)$$

and for other inputs X_i

$$n_0(X_i) = n_0(Y), \quad n_1(X_i) = 0$$

- 3) NOT gate:

$$n_0(X) = n_1(Y), \quad n_1(X) = n_0(Y).$$

- 4) Fanout-point:

$$n_0(X) = \sum_{i=1}^k n_0(X_i), \quad n_1(X) = \sum_{i=1}^k n_1(X_i)$$

where line X fans out to X_1, X_2, \dots, X_k .

Each current objective is backtraced until a head line, say p , is reached. At that point, the final objective value is determined to be 0 if $n_0(p) > n_1(p)$ or 1 if $n_0(p) < n_1(p)$.

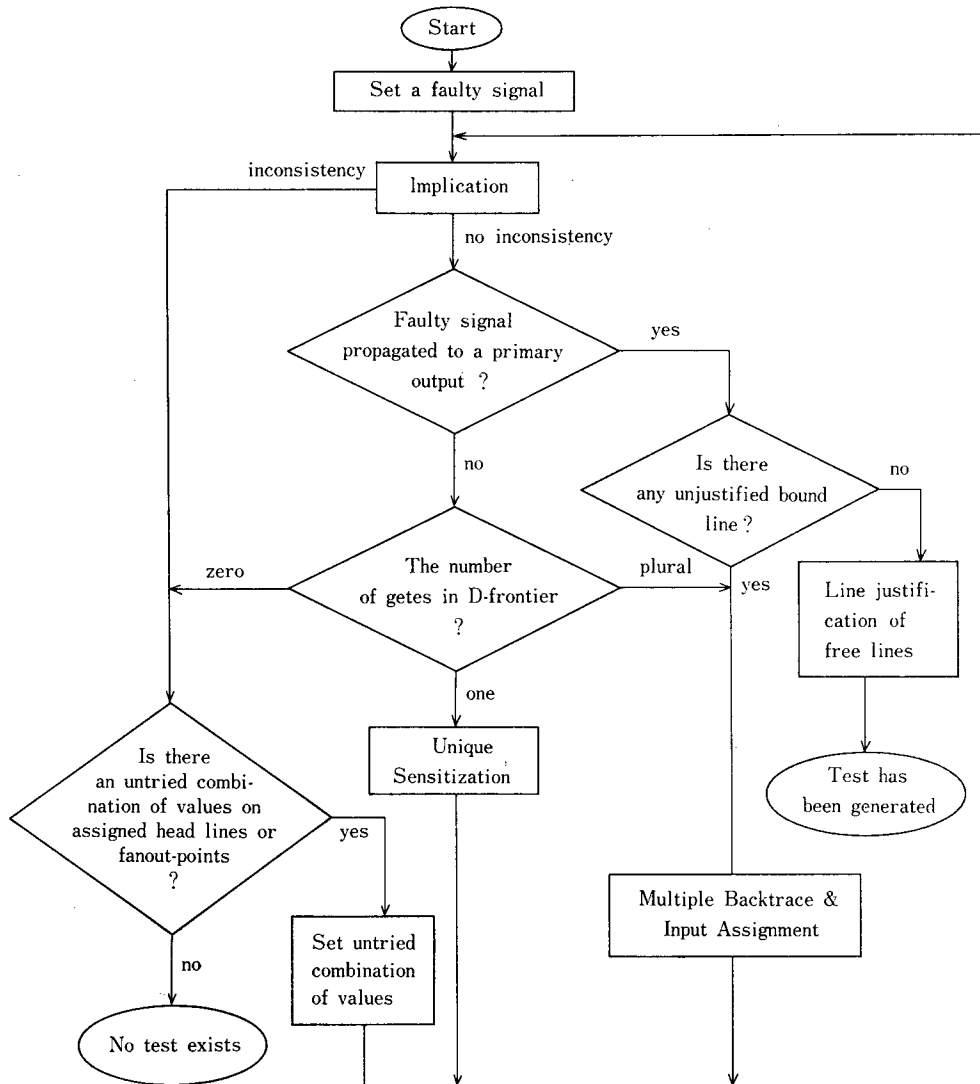


Fig. 3 Flowchart of FAN algorithm

3. Description of the FAN Algorithm

The flowchart of the FAN algorithm is given in Fig. 3.

Multiple Backtrace and Input Assignment: By setting initial objectives (to propagate a faulty signal or to justify unjustified lines), the multiple backtracing finds out final objectives. From among the final objectives, we choose one final objective, say (v, L) , such that the assignment of value v to line L has a good likelihood of helping towards meeting the initial objectives. Then, we assign value v to line L for implication. The remaining final objectives will be used afterward for further implications while those final objectives are effective. The remaining final objectives are defined to be ineffective if the initial objective was to propagate D or \bar{D} and the D-frontier has changed after implication or if the initial objective was to justify unjustified lines and all the unjustified lines has been justified after

implication.

Backtracking: The decision tree is identical to that of PODEM, i.e., an ordered list of nodes with each node identifying a current assignment of either 0 or 1 to one head line, and the ordering reflects the relative sequence in which the current assignments were made. A node is flagged if the initial assignment has been rejected and the alternative is being tried. When both assignment choices at a node are rejected, then it is removed and the predecessor node's current assignment is rejected.

4. Experimental Results

To be of practical use, test pattern generation and fault simulation should interact effectively. Test patterns produced by a test generation algorithm are simulated against the faulted circuits and the fault coverage is evaluated from the results of the fault simulation, which include lists of tested and untested faults. We have implemented two automatic test pattern generation systems, PODEM* and FAN*, based on PODEM and FAN, respectively, with a modified concurrent fault simulator for combinational circuits. Both programs were implemented in FORTRAN on a NEC System ACOS-1000 (15MIPS), and were applied to ten combinational circuits. **Table 1** shows the characteristics of those circuits. The results are given **Tables 2~5**.

To obtain the data, both test generation algorithms, PODEM and FAN, were executed to generate a test for each single stuck-at fault. The number of times a backtrack occurs during the generation of each test pattern was calculated by the programs, and the average number of backtracks is shown in **Table 4**. Since both PODEM and FAN are complete algorithms, given enough time, both will generate tests for each testable faults. However, being limited in computing time, the programs discontinued the test generation for those faults that the numbers of backtracks exceeded some value. Such faults are called **aborted faults** in **Table 2**. In **Tables 2~5**, two cases are shown for the limit of the number of backtracks, i.e., 10 and 1,000. Note that since our PODEM and FAN are executed for the circuits that include only AND, NAND, OR, NOR, or NOT gates, all EXOR gates in C432

Table 1 Characteristics of the circuits

Circuit Name	Total Gates	Total Lines	Input Lines	Output Lines	Fanout Stems	Faults**
C 432	160	432	36	7	89	524
C 499	202	499	41	32	59	758
C 880	383	880	60	26	125	942
C1355*	546	1,355	41	32	259	1,574
C1908	880	1,908	33	25	385	1,879
C2670	1,193	2,670	233	140	454	2,747
C3540	1,669	3,540	50	22	579	3,428
C5315	2,307	5,315	178	123	806	5,350
C6288	2,406	6,288	32	32	1,454	7,744
C7552	3,521	7,552	207	108	1,300	7,550

* Circuit C1355 is functionally equivalent to circuit C499 (all EXOR gates in C499 are expanded into 4-NAND gate equivalents)

** Simply generated equivalent set (need not be minimal)

Table 2 Fault Coverage

Circuit	% Tested Faults				% Aborted Faults			
	PODEM*		FAN*		PODEM*		FAN*	
	10	1,000	10	1,000	10	1,000	10	1,000
C 432	91.5	91.8	93.7	93.7	7.1	6.8	0.5	0.5
C 499	99.4	99.4	97.2	99.4	0.6	0.6	2.2	0
C 880	100	100	100	100	0	0	0	0
C1355	99.5	99.5	97.5	99.5	0.5	0.5	1.9	0.5
C1908	99.5	99.5	99.3	99.5	0.2	0.2	0.4	0.1
C2670	94.6	95.4	95.7	95.7	4.6	2.1	1.1	1.1
C3540	95.5	95.5	95.8	96.0	4.2	2.1	0.5	0.2
C5315	98.3	98.8	98.9	98.9	1.0	0.2	0	0
C6288	99.5	99.5	99.4	99.5	0	0	0.2	0.1
C7552	96.8	97.8	98.2	98.2	2.7	1.4	0.9	0.8

Table 3 Computing Time (Seconds)

Circuit	PODEM* (PODEM)		FAN* (FAN)	
	10	1,000	10	1,000
C 432	1.9(1.3)	47.1(46.4)	1.5(0.8)	3.6(2.9)
C 499	7.9(3.9)	23.9(19.9)	12.6(5.4)	16.2(8.8)
C 880	1.3(0.4)	1.3(0.4)	1.3(0.4)	1.3(0.4)
C1335	9.1(4.3)	23.7(18.8)	9.0(5.0)	13.5(9.4)
C1908	9.2(3.5)	20.9(15.1)	9.4(3.9)	13.5(8.1)
C2670	13.3(8.4)	228.3(223.9)	10.6(5.4)	49.4(44.3)
C3540	27.6(16.0)	301.2(290.0)	21.8(9.9)	42.9(31.3)
C5315	24.0(11.1)	106.6(93.4)	20.1(6.3)	19.7(6.2)
C6288	68.7(4.4)	89.8(25.8)	67.7(4.6)	81.7(18.5)
C7552	73.8(51.8)	1,312.4(1,287.4)	50.8(25.5)	118.6(93.6)

Table 4 Average Number of Backtracks

Circuit	PODEM		FAN	
	10	1,000	10	1,000
C 432	4.4	371.7	0.5	27.3
C 499	1.0	61.9	3.2	17.9
C 880	0	0	0	0
C1355	1.7	54.2	3.2	32.2
C1908	0.4	22.4	0.8	12.8
C2670	4.5	256.0	1.2	110.0
C3540	4.6	271.3	0.6	28.6
C5315	4.0	51.1	0.1	0.2
C6288	2.6	30.1	2.7	112.3
C7552	6.0	299.2	3.2	168.7

and C499 are expanded into 4-NAND gate equivalents before test generation.

Let us suppose that we want to get a high fault coverage such that the ratio of aborted

Table 5 Number of Test Patterns

Circuit	PODEM*		FAN*	
	10	1,000	10	1,000
C 432	62	64	73	74
C 499	122	122	115	131
C 880	83	83	79	79
C1355	141	141	117	123
C1908	170	170	152	155
C2670	157	168	165	165
C3540	209	208	208	204
C5315	188	194	202	202
C6288	38	38	30	30
C7552	238	284	285	285

faults is less than about 1% (see **Table 2**). For FAN*, it is sufficient to set a limit of backtracks to 10, except for C499 and C1355 (both are the same). However, for PODEM*, even 1,000 backtracks are not enough for C432, C2670, C3540, and C7552. Furthermore, PODEM* wastes too long computation time compared with FAN*, in the case of 1,000 backtracks (see **Table 3**). As seen in this example, the fault coverage and the computation time are very susceptible to influences from the limit of the number of backtracks. For FAN*, an appropriate limit on the number of backtracks might be 10 for our circuits. As shown in **Tables 2** and **3**, FAN* can achieve a high fault coverage at a high rate of speed.

5. Acknowledgement

The author would like to thank Mr. T. Shimono of NEC Corp. and Mr. A. Motohara of Matsushita Electric Industrial Co., Ltd. for their contribution to implementing PODEM and FAN.

References

- 1) H. Fujiwara and S. Toida, "The complexity of fault detection: An approach to design for testability", Proc. FTCS-12, pp.101-108, 1982.
- 2) E.B. Eichelberger and T.W. Williams, "A logic design structure for LSI testing", Proc. 14th Design Automation Conf., pp.462-468, 1977.
- 3) S. Funatus, N. Wakatsuki, and T. Arima, "Test generation systems in Japan", Proc. 12th Design Automation Conf., pp.114-122, 1975.
- 4) F.F. Sellers, M.Y. Hsiao, and L.W. Bearnson, "Analyzing errors with the Boolean difference", IEEE Trans. Comput., vol. C-17, no.7, pp.678-683, July 1968.
- 5) J.P.Roth, "Diagnosis of automata failures: A calculus and a method", IBM J. Res. Develop., vol.10, pp.278-291, July 1966.
- 6) C.W. Cha, W.E. Donath, and F. Ozguner, "9-V algorithm for test pattern generation of combinational digital circuits", IEEE Trans. Comput., vol. C-27, no.3, p.193-200, March 1978.
- 7) P. Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits", IEEE Trans. Comput., vol. C-30, pp.215-222, March 1981.
- 8) H. Fujiwara and T. Shimono, "On the acceleration of test generation algorithms", IEEE Trans. Com-

- put., vol. C-32, no.12, pp.1137-1144, Dec. 1983.
- 9) E. G. Ulrich and T. Baker, "The concurrent simulation of nearly identical digital networks", Proc. 10th Design Automation Workshop, pp.145-150, 1973.
 - 10) L. H. Goldstein, "Controllability/observability analysis of digital circuits", IEEE Trans. Circuits and Syst., vol. CAS-26, no.9, pp.685-693, Sept. 1979.
 - 11) H. Fujiwara, Logic Testing and Design for Testability, MIT Press 1985.

Appendix

1. Circuit Description Language

The rules of the circuit description language are as follows.

- Statements are classified as
 - function/connection statements (if the first column is a blank),
 - comment statements (if the first column is "*"), or
 - continuation statements (if the first column is "=").
- All blanks except the first column are ignored.
- A label is a string of at most 8 characters without a blank, ",", and "/".
- Function/connection statements are described as follows:
 - (1) AND, OR, NAND, NOR statements:

AND/inp₁, inp₂,, inp_n/out/

where "inp₁",, "inp_n" are n input labels of the gate, and "out" is the output label of the gate (n ≥ 1).

- (2) XOR statement:

XOR/inp₁,, inp_n/out/

where n ≥ 2.

- (3) NOT statement:

NOT/inp/out/

- (4) FOUT(fanout) statment:

FOUT/inp/out₁,, out_m/

where line "inp" fans out to lines "out₁",, "out_m". Note that "out₁",, "out_m" should not be an input of other fanoutpoints.

- (5) IN, OUT statements:

IN/pin₁,, pin_k/inp₁,, inp_k/

where "pin₁",, "pin_k" are pin-names that are connected to inputs of the circuit, "inp₁",, "inp_k", respectively.

If "pin_i" and "inp_i" are the same label for all i, then we can write as follows:

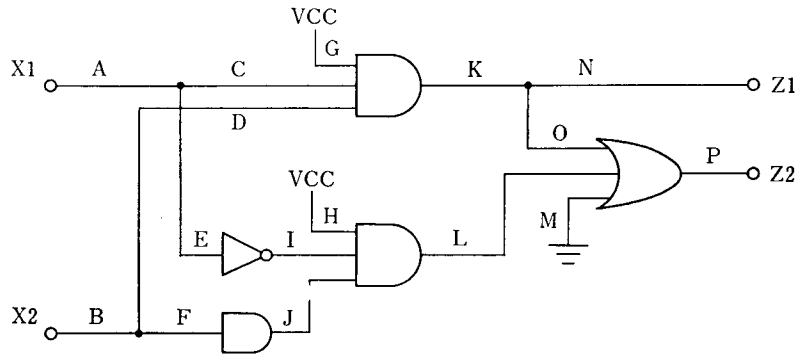
IN/inp₁,, inp_k/

Similarly,

OUT/out₁,, out_m/pin₁,, pin_m/

- (6) VCC, GND statements:

VCC/line₁,, line_k/



***** TEST DATA *****

IN/X1, X2/A, B/
 FOUT/A/C, E/
 FOUT/B/D, F/
 AND/G, C, D/K/
 AND/F/J/
 AND/H, I, J/L/
 OR/O, M, L/P/
 NOT/E/I/
 FOUT/K/O, N/
 OUT/N, P/Z1, Z2/
 VCC/G, H/
 GND/M/

Fig. 4 Example of circuit description

GND/line₁,, line_m/

(7) END statements:

END

An example of circuit description is illustrated in Fig. 4.

Flowchart for transforming a circuit-data described by the above-mentioned language into a table with circuit-information is shown in Fig. 5. A typical example of main routine for the system is as follows:

```
CALL ELMNTB (IERR, 9, 10, 7)
IF (IERR, NE, 0) STOP
CALL EXOR (10)
CALL EXPANS (10)
CALL GCOUNT (10)
CALL REDUCE (10)
CALL SAVTBL (8)
STOP
END
```

where ELMNTB is a subroutine for making a circuit-table, IERR is a flag having 1 when an error occurs, EXOR is a subroutine for transforming every 3-input EXOR gate into two 2-input EXOR gates, EXPANS is a subroutine for converting every 2-input EXOR gate into four NAND equivalents, GCOUNT is a subroutine for saving the information of the circuit-table into a file, REDUCE is a subroutine for eliminating VCC, GND, and redundant gates, and SAVTBL is a subroutine for saving the circuit-table into a file.

Files #7-#10 are assigned as

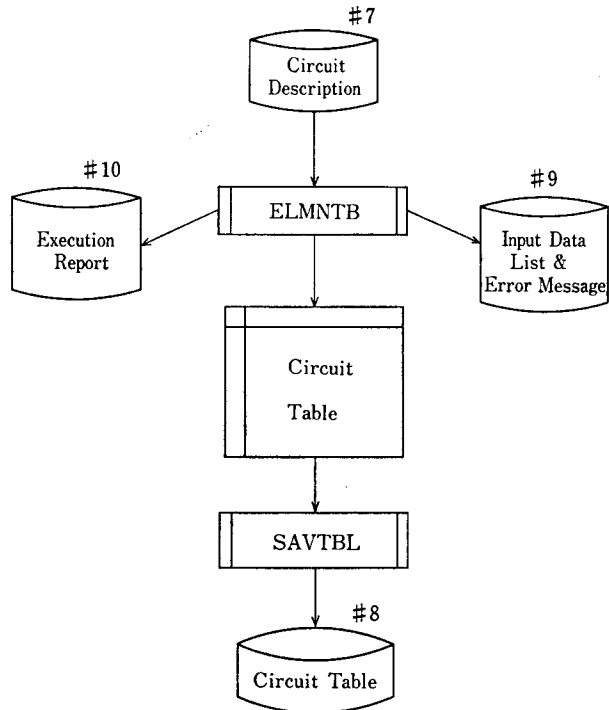


Fig. 5 Flowchart for making circuit-table

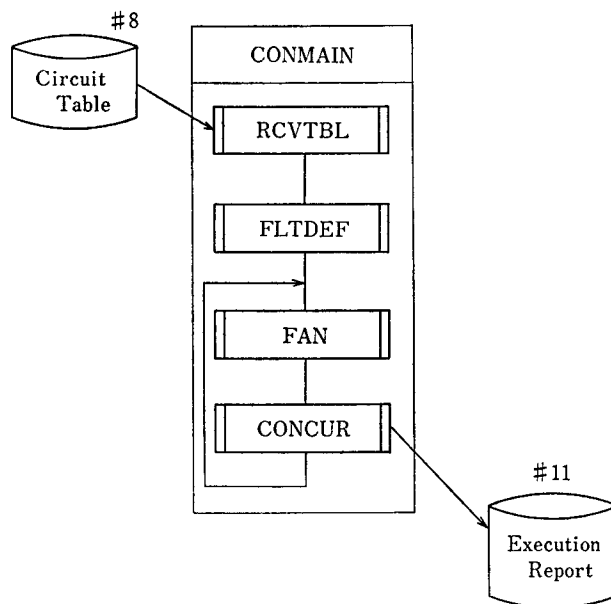


Fig. 6 Flowchart of ATPG System

7=input file for circuit-description-data

8=output file for circuit-table

9=output file for input-data-list and error message

#10=output file for execution report

2. Automatic Test Pattern Generation System

The flowchart for the ATPG system based on FAN and concurrent-like fault simulator is shown in Fig. 6. The main routine is

```
CALL CONMAIN  
STOP  
END
```

where files #8 and #11 are assigned as

```
# 8=input file for circuit-table  
#11=output file for execution report.
```

In Fig. 6, RCVTBL is a subroutine for loading the circuit-table from file #8, FLTDEF is a subroutine for analyzing equivalent faults and making a fault table, FAN is a subroutine for generating a test pattern for a given fault, and CONCUR is a subroutine for fault simulation which find out all faults detectable by a given test pattern.