공학 연구논문집 제24권 제2호 pp. 65~75. 울산대학교
Journal of Engineering Research Vol. 24, No. 2, pp. 65~75, 1993.
University of Ulsan.

VLSI 디지털 회로용 범용 자동 패턴 생성기에 관한 연구

장종권
컴퓨터공학과

〈요 약〉

VLSI 회로망에 사용할 수 있는 범용 자동 시험 패턴 생성기(UATPG)의 기법에 대하여 연구 하였다. UATPG는 기존 ATPG의 용량을 확장하고 CAD 사용자에게 편리한 환경을 제공하는데 초점을 맞추어 설계되었다. 함수적 게이트의 신호선 논리가 확인 및 고장 값 전달을 효과적으로 수행하기 위하여 경험적인 기법을 고안하여 사용하였다. 또한, 시험성을 고려한 설계(DFT)에 사용되는 기억소자(Flip-Flop)가 의사 입출력으로 이용되어 VLSI 디지털 회로망의 시험성을 한층 높여 주었다. 따라서, UATPG는 사용의 용이성과 성능면에서 좋은 결과를 보여 주었다.

A study on a Universal ATPG For VLSI Digital Circuits

Jong-Kwon, Chang
Department of Computer Engineering

〈Abstract〉

In this paper we propose a Universal Automatic Test Pattern Generator (UATPG) for VLSI digital circuits. UATPG is designed to extend the capabilities of the existing ATPG and provide a convenient environment to Computer-Aided Design (CAD) users. We employ heuristic techniques in backtracing and fault propagation for functional gates. In addition, flip-flops with Design For Testability (DFT) [1] are exploited for pseudo-PIs and pseudo-POs to enhance the testabilities of VLSI digital circuits [2]. UATPG shows a good enhancement in convenient usage and performance.

I. INTRODUCTION

Since the emergence of test generation algorithm, the existing ATPG techniques [3,4,5, 6,7] have been developed for efficiency, by adopting better heuristic methods in backtracing and fault propagation, and by organizing the search space and, thus, reducing the number of backtrackings. It is also well known that the existing ATPG techniques have been used for circuit net lists consisted of primitive gates only. However, in practice, circuit designers are tending to use various functional gates (such as mux, decoder, etc.) as well as primitive gates, in designing VLSI digital circuits. Circuit designers usually prefer to design VLSI digital circuits making use of various types of schematic menus provided in the design software package. These functional gates hide gate-level information necessary for test generation. Therefore, we must take a special care of these gates during backtracing and fault propagation. In this paper, we implemented an exhaustive table-driven function for each functional gate and exploit it for test generation.

As the size and logic complexity of VLSI digital circuits have increased, the problem of test generation has surfaced significantly due to their poor controllability and observability [1, 2, 9]. The number of primary inputs (PI) and primary outputs (PO) do not generally increase proportionally to the size of VLSI chips. Thus, the efficiency of the ATPG techniques has been so limited. In this paper, we have attempted to exploit the flip-flops with DFT for pseudo-PIs or pseudo-POs. This ad-hoc technique has been derived from the easy controllability of the flip-flops with DFT by system clock pulse. This ad-hoc technique requires the analysis of the system clock and, thus, two or more test input vectors for a single target fault. However, the number of PI and PO has increased largely due to the addition of pseudo-PI and pseudo-PO and the efficiency of the ATPG has, thus, resulted in significant enhancement as expected.

Finally, this paper consists of seven sections. Section II describes UATPG system in general and section III discusses the preprocessing technique of UATPG system. In section IV table-driven functions are shown for backtracing and fault propagation, for example. We propose an ad-hoc technique which exploits the flip-flops with DFT for pseudo-PIs and pseudo-POs, to enhance the testabilities of VLSI digital circuits. Section VI shows the performance of UATPG system with various digital circuits. Final section summarizes the technique of UATPG system.

II. UATPG System

UATPG system consists of preprocessing, ATPG algorithm and postprocessing as shown in Figure 1. UATPG system is a CAD tool which processes digital circuit net lists described by design language or by schematic menus in the design software package and produces a test input vector for a stuck-at fault assumed in VLSI digital circuits. Note that VLSI digital circuits include various types of functional gates and flip-flops with DFT as well as primitive gates. In particular, a procedure interpreting circuit net lists, from Texas

Description Language (TDL) or schematic menus to Tegas Circuit Files (TCF) [8], is a major preprocessing of UATPG system.

III. Preprocessing

In this section we describe an interpreter which processes digital circuit net lists described by TDL or by CAD system and produce a corresponding input form, i.e., TCF file. The TCF captures a TDL circuit description in a set of tables. Each table contains contiguous entries, one entry per object being described. For example, the PART TABLE has one entry per type of part, the PART OCCURRENCE TABLE has one entry per occurrence of a part, etc. The TCF tables are HEADER TABLE, PART TABLE, PIN NAME TABLE, PART OCCURRENCE TABLE, PIN OCCURRENCE TABLE, and NET TABLE. All pointers in the TCF are relative to the start of the table they point into. For example, if a pointer points to the table entry in a table, it will have the integer value "3". Linked lists of table entries in the TCF are terminated by a "0" pointer value in the last element of the list. This TCF input form is exploited for designing a data structure for test generation. At first we introduce two types of circuit net lists for Figure 2. One type is described by a standard TDL and another is by a Manto Graphics TDL format. TDL' s symbols are treated as tokens in the interpreter algorithm. The flow chart of the interpreter algorithm is illustrated in Figure 3. we use three different types of tokens for reserved word, special symbol and user defined word, respectively. The token (for reserved word) is used to identify TDL' s six sections (such as MODULE, INPUTS, etc.) and gate names. The token (for special symbol) is used to distinguish between tokens and the token (for user defined word) is used to identify input/output pin names, gate names and their pin names which are produced by TDL or CAD system.

The interpreter algorithm produces a TCF form on the basis of the data structures. These data structures are composed of Gate Reference Table (GRT), Gate Occurrence Table (GOT), and String Pool, as shown in Figure 4. GRT holds gate names and their input/output pin' s number in USE section. GOT handles information of DEFINE section such as each gate' s pin number and PI and PO' pin number. Node indicates the location of String Pool to which each gate's pin number is stored. Note that 1* indicates the next node to be connected and 2* the previous node. In String Pool, each pin is stored according to the order of its definition and appearance in GOT. After a TDL file is interpreted, the associated TCF file includes six tables which keep information for circuit net lists.

IV. Functional Gate Test Generation

In real world VLSI circuits are dèsigned using many different types of functional gates which hide gate-level information for test generation. In this section we discuss table-driven functions, for various types of functional gates, which are designed and used for backtracing and fault propagation. Firstly, for example, we illustrate a table-driven function for a half adder (ADHLF). Figure 5 shows ADHLF's switching function and its function table for D-propagation. It can be observed that the outputs (SUM and CARRY) are easily computed on the basis of table-driven function. Secondly, a heuristic technique is employed for backtracing. This heuristic technique is derived from three bases of, the pending logic value to be backtraced, the status of the currently assigned input logic values and the characteristic function, of the pending functional gate. For instance, according to our proposing heuristic technique, we, at first, make enable pin (EN) low when the Decoder is pending for initial backtracing. In other words, 0-logic value at EN should be always selected first to be backtraced, prior to backtracing from any input. For reference, Figure 6 illustrates eight cases of backtracing for a 2 4 Decoder. Note that input selection is tightly related to both the pending output pin number to be backtraced and its logic value. Thirdly, as shown in Figure 7, our proposing heuristic technique also deals with fault propagation efficiently. Our technique determines which output pin should be selected first for better fault propagation, in case of multiple output gates (such as Decoder, Encoder). Finally, for example, the procedure of implication for a 2 4 decoder should be performed for each of its outputs, using five logic values (0, 1, X, D, on the basis of the logic equation of the decoder.

V. Ad-hoc Technique For Digital Circuits With DFT

Most DFT techniques [9] are employed to enhance the testability of VLSI chips and deal with either the resynthesis of an existing design or the addition of extra hardware to the design. In particular, DFT techniques convert sequential networks into combinational networks by scan design. Thus, the testing problem of sequential networks becomes easy to the level of the testing problem of combinational networks. However, the density and size of VLSI chips are growing enormously. Testing problem is still difficult and remains NP-complete, even with DFT techniques [10]. In this section we introduce an ad-hoc testing technique which exploits the flip-flops with DFT for pseudo-PIs and pseudo-POs, to enhance the controllability and observability of VLSI circuits.

V.1. Testing Problem Of Digital Circuits With DFT

Typical types of digital circuits designed with DFT techniques are shown in Figure 8. It can be observed that all flip-flops are chained with a scan path. With this design, we can control a logic value of each flip-flop at will. The previously mentioned functional gate testing technique cannot be directly applied to the digital circuits designed with DFT techniques. We have to handle the flip-flops with DFT in a different way from dealing with general functional gates. Usually, most DFT techniques require the analysis of clocks to propagate faulty signal to P.O. Thus, these techniques have a drawback that requires two or more test input vectors for a single target fault. In addition, with the functional gate testing approach, we have to compute D-propagation on the basis of five-valued algebra and apply the analyzed clocks [3,9]. This adds a big overhead as shown in Table I which illustrates propagation of D-values through a JK flip-flop, to test generation.

Table I. Propagation of D values through a *JK* FF

J	*K*	*q*	*Q*
0	0	D	D
D	0	D	D
-	D'	1	D
0	D'	D	D
D	D'	D	D
1	1	D'	D
D	1	D'	D
D	-	0	D
1	D'	D'	D
D	D'	D'	D

V.2. Ad-hoc Technique

We propose an ad-hoc technique which considers all flip-flops' inputs as pseudo-POs. Firstly, this technique stops executing test generation and regards test generation to be complete as soon as a faulty signal arrives at a P.O. or a pseudo-PO during implication. With a faulty signal arrived to a pseudo-PO, we can then shift out the faulty signal by applying clocks and observe its content at scan out pin. Secondly, our proposing ad-hoc technique considers all flip-flops' outputs as pseudo-PIs. We put all logic values at pseudo-PIs into a decision tree and execute backtracking with the aid of the decision tree [4]. Therefore, a test input vector may consist of a combination of PIs and pseudo-PIs. Finally, our proposing ad-hoc technique enhances the testability of VLSI chips by adopting the flip-flops with DFT as pseudo-PIs and pseudo-POs. Since the number of PIs and POs is actually increased and, thus, the number of backtrackings is significantly decreased. Table II shows technique char acteristics for search space and target circuit. However, we have to pay for this gain during testing application.

Table II. Technique Characteristics

Algorithm	D	PODEM	UATPG
Search Space	All Nodes	All PI's	PODEM + Pseudo PI's
Target Circuit	Combinational Network	Combinational Network	Combinational Network + FFs withDFT

VI. Performance Analysis of UATPG System

UATPG was implemented in C-language on a SUN4 running UNIX. Table III-(a) shows circuit characteristics of six combinational networks. Note that the number of backtrackings is limited to 100 times. The result of UATPG performance for all faults is shown in Table III-(b). In addition, we have UATPG performance for seven digital networks designed with DFT techniques. Table IV-(a) shows circuit characteristics of seven digital networks and Table IV-(b) shows UATPG performance for all faults. Note that the number in parenthesis indicates the result with our proposing ad-hoc technique. Figure 9-(a) & 9-(c) shows the circuit diagrams of S2919 and S1804, respectively. The fault analyses of S2919 and S1804 are illustrated in Figure 9-(b) and 9-(d), respectively. It can be observed that our technique provides a better performance.

Table III. UATPG Performance of Combinational Circuits

NO.	Circuit	Gate	PIs	POs	Pins	fanout	faults
1	C17	6	5	2	25	3	22
2	Cfulladder	5	3	2	20	4	26
3	Cschneider	8	4	3	33	7	38
4	Cdecoder38	21	4	8	68	10	58
5	C74ls181	63	14	8	238	31	247
6	C432	160	36	7	539	31	474

(a) Circuit Characteristics (Combinational Circuit)

No.	Redundant Faults	Aborted Faults	Backtracks	Fault coverage	Generation Ttime
1	0	0	0	100	0.017
2	0	0	0	100	0.017
3	2	0	5	100	0.053
4	0	0	0	100	0.117
5	0	0	21	100	1.383
6	0	20	2701	95.78	19.98

(b) UATPG Performance for All Faults

Table IV. UATPG Performance of DFT Circuit

NO.	Circuit	Gate	PIs	POs	Pins	fanout	faults
1	S12	18	10	4	61	5	36
2	S28	22	10	7	91	8	14
3	S56	29	12	9	119	12	18
4	S114	60	11	25	232	16	94
5	S322	37	68	9	247	12	98
6	S2919	1247	52	25	4115	339	1200
7	S1804	823	12	55	3824	245	1025

(a) Circuit Characteristics (DFT Circuit)

No.	Redundant Faults	Aborted Faults	Backtracks	Fault coverage	Generation Ttime
1	0	0	0	100(100)	0.067(0.067)
2	0	0	0	100(100)	0.033)0.033)
3	2	0	0	100(100)	0.033(0.033)
4	0	0	0	100(100)	0.567(0.567)
5	0	0	0	100(100)	0.400(0.400)
6	96(24)	24(0)	8291(331)	97/073(100)	219.29(78.9)
6	(15)	(0)	(174)	(100)	(89.9)

(b) UATPG Performance for All Faults

VII. Conclusions

UATPG was implemented employing heuristics and an ad-hoc technique. We attempted to extend the capabilities of the existing ATPG for practical usage. Thus, UATPG users can design circuit net lists using either DL or schematic menus and functional gates as well as primitive gates. On the other hand, UATPG employed heuristic techniques in backtracing and fault propagation for functional gates. In addition, UATPG made use of the flip-flops with DFT and, thus, significantly enhanced the testabilities of digital circuits. However, for this gain, we have to pay for the increased number of test input vectors. On the whole, application time of test input vectors becomes longer but test generation time and fault coverage result in faster and better performance, respectively. For further works, one attempt is to develop an algorithm for scan chain ordering to minimize the overall test time. Another direction would be to research an automatic replacement algorithm of the non-scanable flip-flops with the DFT's scanable flip-flops, throughout the VLSI digital circuits.

References

[1] E.B. Eichelberger and T.W. Williams, "A Logic Design Structure for LSI Testability," Journal Design Automation & Fault Tolerant Computing, Vol.2, No. 2, pp. 165-178, May 1978.

[2] L.M. Goldstein and E.L. Thigen, "SCOAP: Sandia Controllability/Observability Analysis Program," Proc. 17th Design Automation Conf., pp. 190-196, June 1980.

[3] J.P. Roth, "Diagnosis of Automata Failures: A Calculus and a Method," IBM Journal of Research and Development, Vol. 10, No. 4, pp. 278-291, July 1966.

[4] P. GOEL, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," IEEE Trans. Compt., vol. C-30, no. 3, pp. 215-222, March 1981.

[5] H. Fujiwara and T. Shimono, "On the Acceleration of Test Generation Algorithms," IEEE Trans. on Computers, Vol. C-32, No. 12, pp. 1137-1144, December 1983.

[6] T. Kirkland and M. Ray Mercer, "A Topological Algorithm For ATPG," 24th Design Automation Conference, pp.502-508, 1987.

[7] M. H. Schulz and E. Auth, "Advanced Automatic Test Pattern Generation and Redundancy Identification Techniques," Poceedings of the 18th Symposium on Fault-Tolerant Computing, pp.30-35, June 1988.

[8] E.W.Thomson and Szygenda, "Digital Logic Simulation in a Time-Based, Table-Driven Environment - Part 2. Parallel Fault Simulation," Computer, Vol.8, No.3, pp.34-49, March 1975.

[9] M. Abramovici, M. Breuer, and A. D. Friedman, Digital Systems Testing and Testable Design, Computer Science Press, Chapter 9, pp. 343-408, 1990.

[10] Ibara, O. H., and S. K. Sahni, "Polynomially Complete Fault Detection Problems," IEEE Trans. on Computers, Vol. C-24, No. 3, pp. 242-249, March 1975.

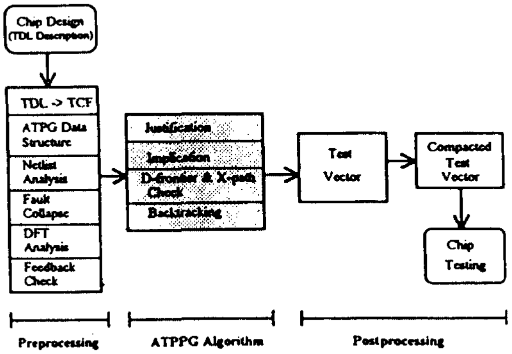
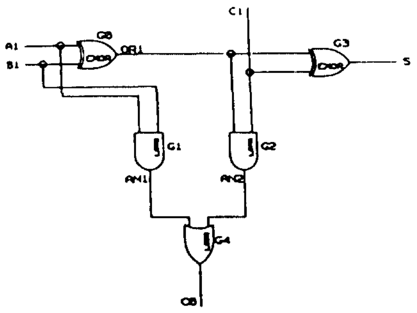


Fig.1 UATPG System Flow Diagram



```
                              "TEGAS Netlister by SEC"
                              "ALIAS / -> :"
                              "ALIAS $ -> -"
                              "DELETE {"
                              "DELETE }"

MODULE: FULLADDER ;           MODULE: FULLADDER ;
INPUTS: A1, B1, C1 ;          INPUTS: A1, B1, C1 ;
OUTPUTS: S, C0 ;              OUTPUTS: S, C0 ;
USE:  ;                       USE: EXOR,
DEFINE:                            AND2,
   G0(OR1)=EXOR(A1, B1);           OR2 ;
   G1(AN1)=AND(A1, B1);       DEFINE:
   G2(AN2)=AND(C1, OR1);         G0(OR1=i)=EXOR(A1=a, B1=b);
   G3(S)=EXOR(OR1, C1);          G1(AN1=a)=AND2(A1=a, B1=b);
   G4(C0)=OR(AN1, AN2);          G2(AN2=a)=AND2(C1=a, OR1=b);
END MODULE ;                     G3(S=a)=EXOR(OR1=a, C1=c);
                                 G4(C0=c)=OR2(AN1=a, AN2=a);
                              END MODULE ;

<Standard TDL format>         <Mantor Graphics TDL format>
```

Fig.2 Two Types of TDL Formats

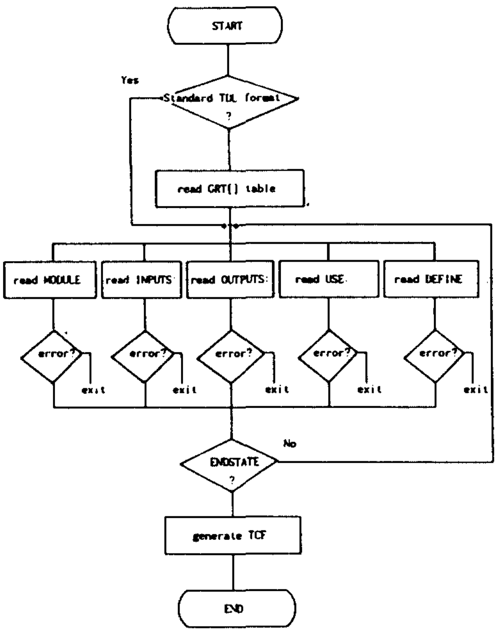
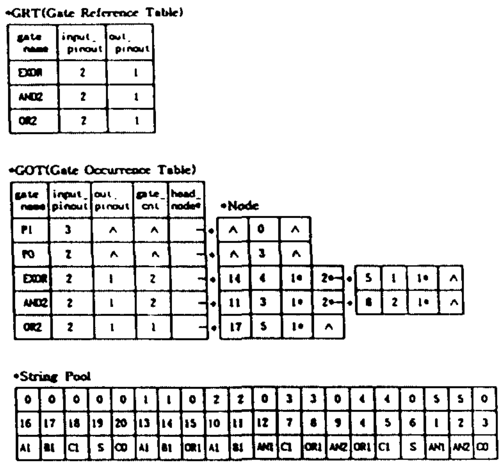


Fig.3 Flow Chart of the Interpreter Algorithm



•GRT(Gate Reference Table)

gate name	input_pinout	out_pinout
EXOR	2	1
AND2	2	1
OR2	2	1

•GOT(Gate Occurrence Table)

gate name	input_pinout	out_pinout	gate_cnt	head_node*
PI	3	^	^	
PO	2	^	^	
EXOR	2	1	2	
AND2	2	1	2	
OR2	2	1	1	

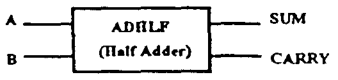
•Node

^	0	^					
^	3	^					
14	4	1•	2•	5	1	1•	^
11	3	1•	2•	8	2	1•	^
17	5	1•	^				

•String Pool

0	0	0	0	0	1	1	0	2	2	0	3	3	0	4	4	0	5	5	0
16	17	18	19	20	13	14	15	10	11	12	7	8	9	4	5	6	1	2	3
A1	B1	C1	S	CO	A1	B1	OR1	A1	B1	AN1	C1	OR1	AN2	OR1	C1	S	AN1	AN2	CO

Fig4. Interpreter Data Structure of Full Adder



(a) Funtional View of ADHLF

SUM

B \ A	0	1	X	D	D'
0	0	1	X	D	D'
1	1	0	X	D'	D
X	X	X	X	X	X
D	D	D'	X	0	1
D'	D'	D	X	1	0

CARRY

B \ A	0	1	X	D	D'
0	0	0	0	0	0
1	0	1	X	D	D'
X	0	X	X	X	X
D	0	D	X	D	0
D'	0	D'	X	0	D'

(b) D-propagation for ADHLF

Fig.5 ADHLF (Half Adder)

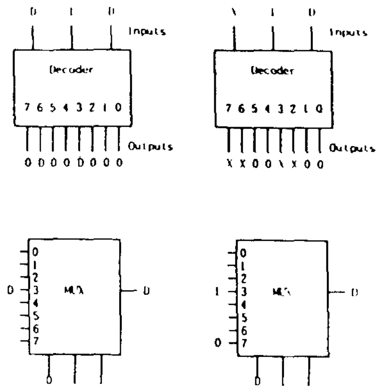


Fig.7 Example of D-Propagation through the Decoder & MUX

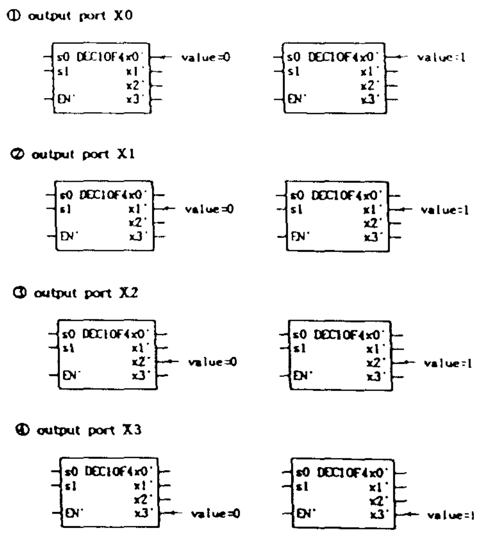


Fig.6 Eight Cases of Backtracing for a 2×4 Decoder

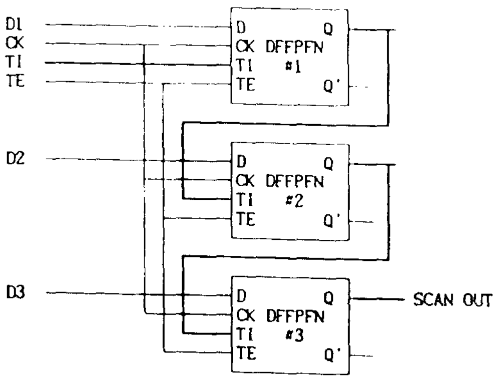


Fig.8 DFT Design

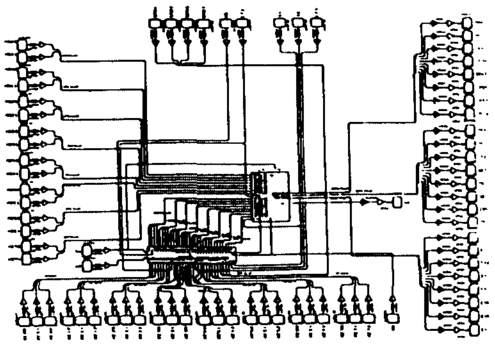


Fig.9-(a) S2919 Circuit

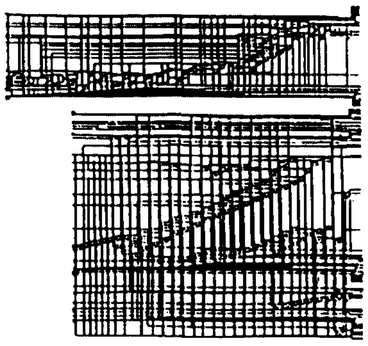


Fig.9-(c) S1804 Circuit

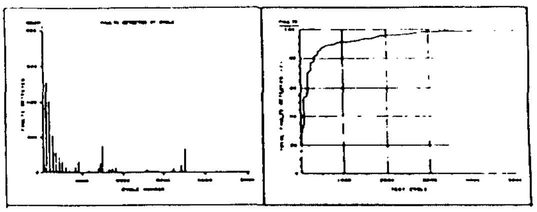


Fig.9-(b) Fault Analysis of S2919

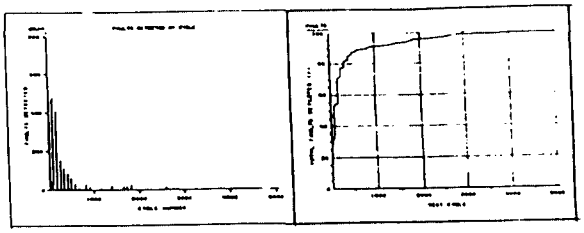


Fig.9-(d) Fault Analysis of S1804