A Fault Detecting Pattern Generation Procedure for a Sequential Circuit

Michio Murakami* and Yasuaki Ozawa*

1. Introduction

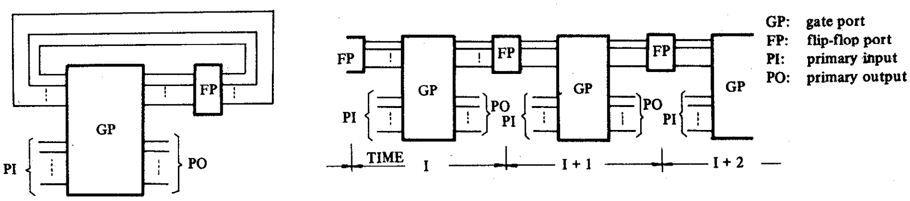
The D-algorithm is a very useful and practical method of generating fault detecting patterns for a combinational circuit [1]. Kubo applied the D-algorithm to a sequential circuit by considering the circuit as a cascade connection of many replicas of a pseudo combinational circuit element as shown in Fig. 1, [2]. But this adaptation has the defect that each FP element is restricted to unit delay, and it cannot be used with other kinds of flip-flops and inhibited conditions.

To test many types of flip-flops such as J-K, R-S, Latch, and their inhibited conditions easily, we formulated a procedure of fault detecting pattern generation for a sequential circuit based on Kubo's model. It differs from the original D-algorithm and Kubo's model mainly in the following points.

(1) A flip-flop is regarded as a macro-gate represented by a Boolean equation with an inhibited condition.

(2) By introducing the concept of a Node and a Branch, we cleared up the D and C-operations. (The terms relating to the D-algorithm such as D-operation, D-intersection, etc. are defined in Ref. [1].)

In this paper, the concept of a Branch is first described, then the procedure of solving the Boolean equation of a flip-flop is covered in detail.



(a) General form (b) Modelled form

Fig. 1 Model of a sequential circuit

2. Extension of the D-algorithm using a Node and a Branch

A Node is defined as a condition in which one of several possible operations must be selected as the next operation during D and C-operation. A Branch is defined as the operation that is selected at a Node. Using the Node and Branch, we cleared up the operation of the D-algorithm and formulated a procedure. It generates fault detecting patterns by creating and deleting Nodes and Branches during D and C-operation. When an inconsistency occurs, it goes back to the last Node and selects another Branch. The selection of a Branch is performed according to the priority condition which is the distance from that point to a primary input or output in Fig. 1 (b), and assists in generating optimum patterns.

This paper first appeared in Japanese in Joho－Shori (Journal of the Information Processing Society of Japan), Vol. 14, No. 5 (1973), pp. 328～333.

* Component Laboratory, Engineering Development Division, OKI Electric Industry Co., Ltd.

A Node is classified into five types.

Type-1: the case of plural elements of activity vector A (see Ref. [1]). This concept is also covered in the original D-algorithm.

Type-2: the case of plural possible combinations to D-intersect a flip-flop, for example; if C is D and the others are all X at the time N for an R-S flip-flop as shown in Fig. 2. There exist three possible combinations to D-intersect, or to get D or $\bar{D}$ on the output line at the time N + 1.

(1) $Q_N = 1$, $R = 0$

(2) $Q_N = 1$, $P = 0$

(3) $S = 1$, $R = 0$, $P = 1$

where Q_N, R, S, P indicate the values on the corresponding lines, and the suffix N indicates time.

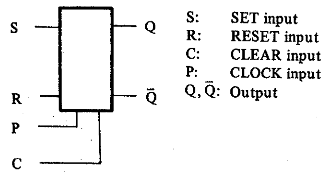


Fig. 2 Model of R-S flip-flop

Type-3: the case where a D-intersected flip-flop has two outputs Q and $\bar{Q}$, since it cannot be defined exactly which pass must be selected.

Type-4: the case of plural possible combinations to C-intersect a gate. This idea also appears in the original D-algorithm.

Type-5: the case of plural possible combinations to C-intersect a flip-flop. For example, if P is 1 and the others are all X at the time N, there exist two possible combinations to force 1 on Q_{N+1};

(1) $C = 0$, $S = 1$, $R = 0$

(2) $C = 0$, $Q_N = 1$, $R = 0$

3. *Procedure for solving the Boolean equation of a flip-flop*

Many types of flip-flop operation calculating procedures have been devised using a Boolean equation in sum-of-product form. We use the equation of an R-S flip-flop as an example throughout this paper, which is:

$$Q_{N+1} = \bar{C} \cdot S \cdot \bar{R} \cdot P + \bar{C} \cdot Q_N \cdot \bar{P} + \bar{C} \cdot Q_N \cdot \bar{R} \tag{1}$$

3.1 *Forward forcing*

This procedure involves forcing implied signals forward toward a primary output during D and C-operation, and is performed simply by substituting into the equation under given conditions.

For example, if Q_N, R and C are 0, P is 1 and S is D, Eq. (1) becomes:

$$Q_{N+1} = (1) \cdot (D) \cdot (1) \cdot (1) + (0) \cdot (0) \cdot (0) \cdot (0) + (1) \cdot (0) \cdot (0) = (D) \tag{2}$$

3.2 *Backward forcing and C-intersection*

The former is forcing an implied signal backward toward a primary input during D and C-operation, and the latter is C-intersection. They are almost identical, and performed in the same routine manner.

The general procedure is:

(1) Substitute the values under given conditions at the time N into the equation.

(2) If Q_{N+1} is 0, calculate all possible combinations which force 0 on all members of the equation. If Q_{N+1} is 1, calculate all possible combinations which force 1 on one member.

(3) Delete some combinations which cause inhibition of a flip-flop (ex. S = 1, R = 1, P = 1 in an R-S flip-flop).

(4) In the case of backward forcing,

(a) if the number of resulting combinations is only one, force the values in the combination.

(b) if there is a plurality of combinations and a certain variable exists whose value in all combinations is identical, force that value on that variable.

In the case of C-intersection, formulate a Node using the resultant combinations.

For example, if Q_{N+1} is 1 and the others are all X at the time N,

$$1 = \bar{C} \cdot S \cdot \bar{R} \cdot P + \bar{C} \cdot Q_N \cdot \bar{P} + \bar{C} \cdot Q_N \cdot \bar{R} \tag{3}$$

There are three combinations.

(1) $C = 0,\ R = 0,\ P = 1$

(2) $C = 0,\ Q_N = 1,\ P = 0$

(3) $C = 0,\ Q_N = 1,\ R = 0$

If it is a C-intersection, these three make a Node.

If it is Backward forcing, there is a plurality of combinations and only C is for all members, and only 1 can be forced on C.

3.3 D-intersection

D-intersection is a procedure to calculate all combinations which force D or $\bar{D}$ on Q_{N+1} under given conditions.

It is generally performed as follows.

(1) Substitute the values at the time N into the equation.

(2) Perform the following two steps for each member which has D ($\bar{D}$).

(a) Apply 1 to all other variables of the member.

(b) Calculate all possible combinations which force 0 on all other members which do not have D ($\bar{D}$). (The value of a member which has both D and D is 0.)

(3) Delete some combinations which cause inhibited conditions.

For example, if S is D, C is 0 and the others are all X at the time N,
Eq. (1) becomes:

$$Q_{N+1} = (D) \cdot \bar{R} \cdot \bar{P} + Q_N \cdot \bar{P} + Q_N \cdot \bar{R} \tag{4}$$

There exists only one member which has D or $\bar{D}$ (in this case D).

The next three conditions must be satisfied to force D on Q_{N+1}.

$\bar{R} \cdot P = 1$	(from 2-a)	(5)
$Q_N \cdot \bar{P} = 0$	(from 2-b)	(6)
$Q_N \cdot \bar{R} = 0$	(from 2-c)	(7)

From Eq. (5), R becomes 0 and P becomes 1, and Q_N becomes 0 from Eq. (7). Finally, there is only one combination to make Q_{N+1} = D, R = 0, P = 1, Q_N = 0.

Using these procedures, we can test any kind of flip-flop.

4. Results and conclusions

This procedure was developed as a part of a fault detecting pattern generating system, and has some system restrictions. The fault detecting ratio of some examples is about 90 ~ 100%. The undetected failures come from cases where a circuit has redundancies or cases where the values of some flip-flops cannot be set to desirable states. Some examples and results of this system are shown in Table 1.

Table 1 Results of pattern generating system

	Example No. 1	Example No. 2	Example No. 3
Circuit type	SL*	S**	S
Gates	40	109	154
Flip-flops	8	13	9
Generated patterns	68	67	104
Undetected failures	0	3	2
Fault detection ratio	100%	99%	99%
Computing time***	40 sec.	70 sec.	180 sec.

* Sequential circuit with feed-back loop
** Sequential circuit without feed-back loop
*** Time using IBM 360/75 or UNIVAC 1108

The system is implemented by FORTRAN, has about 5500 steps, and is used for LSI testing.

References

[1] J.P. Roth, et al.: Programmed algorithm to Compute Tests to Distinguish Between Failures, IEEE trans. on E.C. Vol. EC-16 No. 5 pp. 567-580, 1967

[2] H. Kubo: A Procedure for Generating Test Sequences to Detect Sequential Circuit Failures, NEC Re & Dev. No. 12, pp. 69-78, 1968