Among these are: 1) efficient algorithms and strategies for failure *location;* 2) test-point insertion to enhance detectability and diagnosability; 3) tradeoffs involved in extending size of the macroblocks used in test generation to encompass shift registers, counters, etc.; 4) use of static test algorithms to detect dynamic failures. This is only a small list hopefully reflecting the scope of work to be done in the future.

Acknowledgment

An approach similar to that in the iterative test generator has been independently developed by R. A. Rasmussen, IBM, Endicott, N.Y.

References

[1] J. P. Roth, "Diagnosis of automata failures: A calculus and a method," *IBM J. Res. Develop.*, vol. 10, pp. 278–291, July 1966.

[2] J. P. Roth, W. G. Bouricius, and P. R. Schneider, "Programmed algorithms to compute tests to detect and distinguish between failures in logic circuits," *IEEE Trans. Electron. Comput.*, vol. EC-16, pp. 567–580, Oct. 1967.

[3] G. R. Putzolu and J. P. Roth, "An algorithm and a program for generation of test patterns for sequential circuits," *Proc. 3rd Hawaii Int. Conf. Syst. Sci.*, pp. 64–67, Jan. 1970.

[4] M. Correia, D. Cossman, G. Putzolu, and T. Snethen, "Minimizing the problem of logic testing by the interaction of a design group with user-oriented facilities," *Proc. 7th Annu. Design Automation Workshop*, pp. 100–107, June 1970.

[5] D. B. Armstrong, "On finding a nearly minimal set of fault detection tests for combinational logic nets," *IEEE Trans. Electron. Comput.*, vol. EC-15, pp. 66–73, Feb. 1966.

[6] F. Sellers, M. Hsiao, and L. Bearnson, "Analyzing errors with the Boolean difference," *Ist Annu. IEEE Comput. Conf. Dig.*, pp. 6–9, Sept. 1967.

[7] P. R. Schneider, "The necessity to examine *D*-chains in diagnostic test generation—An example," *IBM J. Res. Develop.*, vol. 11, pp. 114, Jan. 1967.

[8] E. B. Eichelberger, "Hazard detection in combinational and sequential switching circuits," *IBM J. Res. Develop.*, vol. 9, pp. 90–99, Mar. 1965.

Derivation of Minimum Test Sets for Unate Logical Circuits

RODOLFO BETANCOURT, MEMBER, IEEE

Abstract—A derivation of test sets S_0 and S_1 for irredundant unate logical circuits is presented. It is shown that these sets (S_0 and S_1, respectively) detect all stuck-at-0 and stuck-at-1 faults in all realizations with no internal inverters of a given unate function. They can be obtained easily from the minimum sum and minimum product forms, from a Karnaugh map, or from a Hasse diagram of the function. These sets are minimum in the sense that there is no set with a smaller number of elements that detects all faults in the class of realizations of a logical function. In particular, it is found that a two-level AND–OR (OR–AND) network needs all the tests in S_0 (S_1).

Index Terms—Fault detection, fault diagnosis, logic test generation, reliability, unate functions.

Introduction

THE CLASS of faults considered in this paper comprises the so-called "stuck-at-1" and "stuck-at-0" faults. They are of a more or less permanent nature and arise because of open or grounded wires, burned-out diodes or transistors, etc. Their logical effect in the circuit is to impress a logical 0 or a logical 1 on one or more of the lines, irrespective of the input signals applied to the network. As an example, in Fig. 1 a simple circuit that realizes the function $f=x_1x_2+x_3x_4$ is shown. If line 1 is stuck at 1, then the network realizes the function $f_{1/1}=x_2+x_3x_4$. To detect such a fault, we can use the input combinations that give an output in the faulty circuit different from the output in the correct one. From the Karnaugh map in Fig. 2 the tests may be determined to be 0100, 0101, and 0110. Similarly, if line 1 is stuck at 0, the circuit realizes the function $f_{1/0}=x_3x_4$ as indicated in Fig. 3. The tests that detect the fault "line 1 stuck at 0" are 1100, 1101, and 0110.

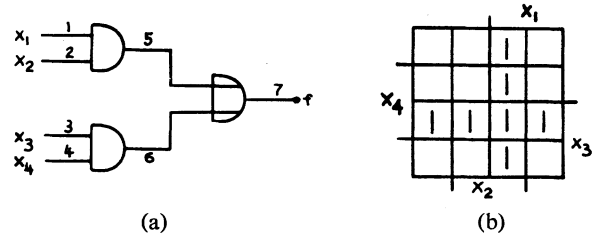
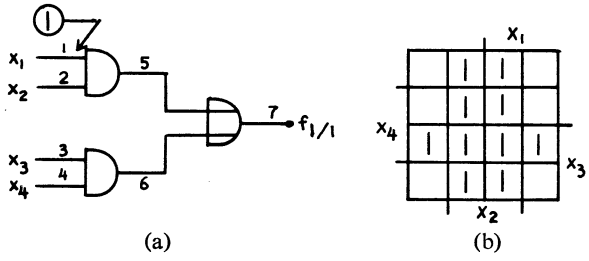


Fig. 1. (a) Circuit for $f=x_1x_2+x_3x_4$. (b) Karnaugh map for f.

To detect any possible fault whose effect is to give an erroneous output, we can apply sequentially 2^n different input combinations (for a circuit with n variables) until a discrepancy appears between the actual and the expected outputs or until the last one of the 2^n input combinations used indicates that the circuit is logically correct. This ex-

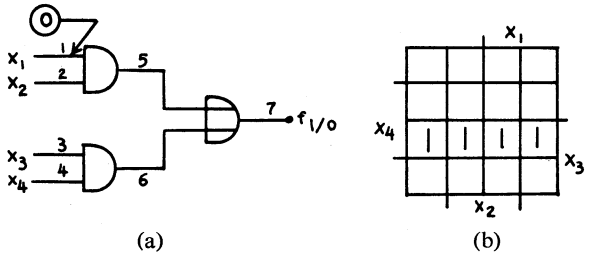
Manuscript received March 1, 1971; revised June 14, 1971.

The author was with the Digital Systems Laboratory, Stanford Electronics Laboratories, Stanford University, Stanford, Calif. He is now with the Instituto Tecnologico of Monterrey, Monterrey, Mexico.



(a) (b)

Fig. 2. (a) Circuit for $f_{1/1}=x_2=x_3x_4$. (b) Karnaugh map for $f_{1/1}$.



(a) (b)

Fig. 3. (a) Circuit for $f_{1/0}=x_3x_4$. (b) Karnaugh map for $f_{1/0}$.

haustive method is too long and inefficient, since it is possible in almost all the cases to detect all stuck-at-1 and stuck-at-0 faults using a much smaller number of input combinations, as has been described elsewhere by Galey, Norby, and Roth [1], Armstrong [2], Poage [3], and Roth [4].

Fault Detection in Frontal Networks

To begin our study of fault detection, we restrict ourselves to frontal functions, since they are more mathematically tractable and they render elegant and promising results.

For our purposes we define a frontal function as a logic function that has only uncomplemented variables in its minimum sum (or minimum product) formula. The designation "frontal" comes from the fact that they can be realized using only front contacts in a contact network. Also, a logic network with only AND and OR gates will realize a frontal function and vice versa.

Frontal functions have the following property: if $t_i=(e_{i1}e_{i2}e_{i3}\cdots e_{in})>t_j=(e_{j1}e_{j2}e_{j3}\cdots e_{jn})$, $e_{qk}\in\{0,\ 1\}$, then $f(t_i)\geq f(t_j)$, where f is a frontal function and $t_i>t_j$ means that t_i has a 1 in every position where t_j does. Thus, for example, 01101>01001 and 01111>01000.

It has been shown by Sellers, Hsiao, and Benson [5] that the Boolean difference can be used for error analysis in logical networks. Let $f(x)=f(x_1, x_2, \cdots, x_n)$ be a logic function of n variables. We then define the Boolean difference of $f(X)$ with respect to x_1 as

$$H^i(X) = df(X)/dx_i = f(x_1, \cdots, x_i, \cdots, x_n) \oplus f(x_1, \cdots, \bar{x}_i, \cdots, x_n). \quad (1)$$

Example:

$$f(X) = f(x_1, x_2, x_3) = x_2(x_1 + x_3).$$

then

$$H^1(X) = x_2(x_1 + x_3) \oplus x_2(\bar{x}_1 + x_3) = x_2\bar{x}_3.$$

For analytical purposes we want to distinguish between stuck-at-0 and stuck-at-1 faults; therefore we define the functions h_0^i and h_1^i as

$$h_0^i = f(x_1, \cdots, x_{i-1}, x_i, \cdots, x_n) \oplus f(x_1, \cdots, x_{i-1}, 0, \cdots, x_n) = f(X) \oplus f_0^i \quad (2a)$$

and

$$h_1^i = f(x_1, \cdots, x_{i-1}, x_i, \cdots, x_n) \oplus f(x_1, \cdots, x_{i-1}, 1, \cdots, x_n) = f(X) \oplus f_1^i \quad (2b)$$

where

$$f_0^i = f(x_1, \cdots, x_{i-1}, 0, \cdots, x_n)$$

and

$$f_1^i = f(x_1, \cdots, x_{i-1}, 1, \cdots, x_n).$$

If the primary input x_i of a logical circuit that realizes $f(x)$ is stuck at 0, then the output of the network is $f(x_1, \cdots, x_{i-1}, 0, \cdots, x_n)=f_0^i$, and this fault can be detected by any input combination t that makes $f(t)=\bar{f}_0^i(t)$, i.e., an input combination t for which we have $f(t)\oplus f_0^i(t)=h_0^i(t)=1$. Similarly, a faulty primary input x_i stuck at 1 can be detected by any input combination t that makes $h_1^i(t)=1$.

Thus the set $\{t\,|\,h_0^i(t)=1\}$ is the set of tests that detect primary input x_i stuck at 0 and $\{t\,|\,h_1^i(t)=1\}$ is the set of tests that detect primary input x_i stuck at 1.

Example:

$$f(X) = f(x_1, x_2, x_3) = x_2(x_1 + x_3).$$

Then

$$h_0^1 = f(X) \oplus f_0^1 = x_2(x_1 + x_3) \oplus x_2x_3 = x_1x_2\bar{x}_3$$

and

$$h_1^1 = f(X) \oplus f_1^1 = x_2(x_1 + x_3) \oplus x_2 = \bar{x}_1x_2\bar{x}_3.$$

The input combination 110 will detect the faulty input line x_1 stuck at 0 since $h_0^1(110)=1$, and 010 will detect the input line x_1 stuck at 1 since $h_1^1(010)=1$.

From the definition of H^i, h_0^i, and h_1^i we get

$$\begin{aligned} H^i &= f(x_1, \cdots, x_i, \cdots, x_n) \oplus f(x_1, \cdots, \bar{x}_i, \cdots, x_n) \\ &= (x_if_1^i + \bar{x}_if_0^i) \oplus (\bar{x}_if^i + x_if_0^i) \\ &= \bar{f}_0^if_1^i + f_0^i\bar{f}_1^i = f_0^i \oplus f_1^i. \end{aligned} \quad (3)$$

Since x_i does not appear in either f_0^i or f_1^i, H^i does not depend on x_i. We also have

$$h_0^i = f \oplus f_0^i = (x_if_1^i + \bar{x}_if_0^i) \oplus f_0^i = x(f_0^i \oplus f_1^i) = x_iH^i \quad (4a)$$

$$h_1^i = f \oplus f_1^i = (x_if_1^i + \bar{x}_if_0^i) \oplus f_1^i = \bar{x}_i(f_1^i \oplus f_1^i) = \bar{x}_iH^i \quad (4b)$$

and

$$h_0^i + h_1^i = x_iH^i + \bar{x}_iH^i = H^i. \quad (5)$$

Sets τ_0 and τ_1

Definition: τ_0^i is the set of input combinations that detects the primary input x_i stuck at 0. It is given by

$$\tau_0^i = \{t \mid h_0^i(t) = 1\} = \text{ON-SET of } h_0^i. \tag{6a}$$

Definition: τ_0^i is the set of input combinations that detects the primary input x_i stuck at 1. It is given by

$$\tau_1^i = \{t \mid h_1^i(t) = 1\} = \text{ON-SET of } h_1^i. \tag{6b}$$

For frontal functions we have $f_0^i < f_1^i$ and $f_0^i \bar{f}_1^i = 0$, and $h_0^i = x_i \bar{f}_0^i f_1^i + x_i f_0^i \bar{f}_1^i$ reduces to $x_i \bar{f}_0^i f_1^i$; thus

$$\tau_0^i = (x_i \bar{f}_0^i f_1^i)^{-1}(1). \tag{7a}$$

In the same way $h_1^i = \bar{x}_i \bar{f}_0^i f_1^i$ and

$$\tau_1^i = (\bar{x}_i \bar{f}_0^i f_1^i)^{-1}(1). \tag{7b}$$

Now f_0^i and f_1^i are independent of x_i and we draw the following conclusions.

Properties of the Sets τ_0 and τ_1

1) If $t = (e_1 e_2 \cdots e_i \cdots e_n) \in \tau_0^i$, $e_j \in \{0, 1\}$, then $e_i = 1$. This follows immediately from the expression $h_0^i = x_i \bar{f}_0^i f_1^i$, since $h_0^i(t) = 1$ only if $x_i = 1$.

2) If $t = (e_1 e_2 \cdots e_i \cdots e_n) \in \tau_1^i$, then $e_i = 0$, since $h_1^i = \bar{x}_i \bar{f}_0^i f_1^i$ and $h_1^i(t) = 1$ only if $\bar{x}_i = 1$, that is, if $x_i = 0$.

3) $t \in \tau_0^i \Leftrightarrow t^* \in \tau_1^i$ where t^* is obtained complementing the ith position of t.

4) From 3) the set $(\bar{f}_0^i f_1^i)^{-1}$ (1) has an even number of elements; that is, τ_0^i and τ_1^i have the same number of elements.

5) Since $f(t) = 1$ if $t \in \tau_0$ and $f(t) = 0$ if $t \in \tau_1$, then from properties 1), 2), and 3) we find the following algorithm to determine the sets τ_0 and τ_1.

Algorithm to Find τ_0 and τ_1

Take the ON-SET of f and find those vertices t that have a 1 in the ith coordinate. If when changing this 1 to 0 (i.e., finding t^*) we have that $f(t^*) = 0$, then $t \in \tau_0^i$ and $t^* \in \tau_1^i$.

Proof: The necessity follows directly from the properties stated above. The condition that t have a 1 in the ith position is included in the conditions $f(t) = 1$, $f(t^*) = 0$; that is, $[f(t) = 1, f(t^*) = 0]$ implies t has a 1 in the ith position because f is frontal, and we have $f(t) > f(t^*) \Rightarrow t > t^*$. The sufficiency part of the algorithm is immediately obvious since we have $f(t) = 1$ and $f(t^*) = 0$; that is, the output changes when the ith input changes: it is the fundamental condition for an input combination to be a test. Analytically, let t be such that $f(t) = 1$ and $f(t^*) = 0$ when complementing the ith position; then, since f can be expressed as $f = x_1 f_1^i + \bar{x}_i f_0^i$, and $f_1^i > f_0^i$, f_1^i takes the value 1 in the vertex t, and f_0^i takes the value 0 in the vertex t (or $\bar{f}_0^i$ takes the value 1 in t), f_0^i and f_1^i are independent of x_i and have the same value in t and in t^*. Consequently, $\bar{f}_0^i f_1^i$ must take the value 1 in t. This is precisely the function that determines τ_0 and τ_1.

TABLE I

ON-OFF SET TABLE FOR $f(x_1, x_2, x_3, x_4) = \sum(3, 6, 7, 11, 12, 13, 15, 16, 17)_8$

OFF-SET \ ON-SET		3 0011	6 0110	7 0111	11 1001	12 1010	13 1011	15 1101	16 1110	17 1111
0	0000									
1	0001	3			1					
2	0010	4	2			1				
4	0100		3							
5	0101			3				1		
10	1000				4	3				
14	1100							4	3	

Example:

$$f = x_1 x_3 + x_1 x_4 + x_2 x_3 + x_3 x_4$$
$$= \sum (3, 6, 7, 11, 12, 13, 15, 16, 17)_8.$$

The procedure to find τ_0 and τ_1 is carried out in Table I.

Explanation of Table I

If, when changing a 1 to 0 in the kth position of t, an entry in the ON-SET, the resulting number t^* belongs to the OFF-SET, we put down the number k on the cell corresponding to the intersection of t and t^*. This procedure is carried for every 1 bit in every entry of the ON-SET. In Table I the entries $k = 1, 2, 3, 4$ represent the input variables. If in a cell of the table there appears an entry k, then the input combination specified by the corresponding column heading will test the kth input variable stuck at 0, while the input combination specified by the corresponding row heading will test the kth variable stuck at 1.

For the example above the tests are

$$\tau_0^1 = \{11, 12, 15\} \qquad \tau_1^1 = \{1, 2, 5\}$$
$$\tau_0^2 = \{6\} \qquad \tau_1^2 = \{2\}$$
$$\tau_0^3 = \{3, 6, 7, 12, 16\} \qquad \tau_1^3 = \{1, 4, 5, 10, 14\}$$
$$\tau_0^4 = \{3, 11, 15\} \qquad \tau_1^4 = \{2, 10, 14\}.$$

SUFFICIENT SET OF TESTS

Later in this paper it is shown that the unions of tests

$$\bigcup_{j=1}^{n} \tau_0^j \quad \text{and} \quad \bigcup_{j=1}^{n} \tau_0^j$$

will detect any fault of the stuck-at-0 or stuck-at-1 type in the internal lines as well as in the input terminal in any irredundant frontal network that realizes f. But, as we will show presently, there is a smaller set of tests that will perform the same job.

Definition: A *zero-bias function* is the function realized by a logical network when a line is stuck at 0.

Definition: A *one-bias function* is the function realized by a logical network when a line is stuck at 1.

Theorem 1: In a nonredundant frontal network that realizes $f=f(x_1, x_2, \cdots, x_n)$, the zero-bias functions are frontal and are strictly included in f.

Proof: Assume line j is stuck at 0. Remove the part of the network that feeds the stuck line. Call g the function realized by the pruned-out part. Expressing f in normal form as a sum of products and factoring g, we get $f=gF_1+F_2$ where g, F_1, and F_2 are obviously frontal functions (only AND and OR gates are present). Then

$$f\,|_{\text{line } j \text{ stuck at } 0} = g\,|_{=0} F_1 + F_2 = F_2 \subseteq f.$$

Since the network is nonredundant, the fault is detectable and $f \neq F_2$; then $F_2 \subset f$.

Theorem 2: In a nonredundant frontal network that realizes $f=f(x_1, x_2, \cdots, x_n)$, the one-bias functions are frontal and strictly include f.

Proof: This is similar to Theorem 1 except that f is expressed in normal form as a product of sums.

Theorem 3: Let $f=f(x_1, x_2, \cdots, x_n)$ be a frontal function, p^i a vertex of the n cube, and

$$S_0 = \{p^i \mid f(p^i) = 1 \text{ and } p^i > p^j \Rightarrow f(p^j) = 0\}. \quad (8a)$$

Then S_0 is a sufficient set of input combinations to detect all single stuck-at-0 faults in any nonredundant frontal network that realizes f.

Proof: Suppose $p^j \notin S_0$ detects a stuck-at-0 fault and g_0 is the zero-bias function corresponding to that fault. By Theorem 1, $g_0 \subset f$ and consequently $f(p^j)=1$ and $g_0(p^j)=0$. Since $f(p^j)=1$ and $p^j \notin S_0$, there exists a $p^k \in S_0$, $p^k < p^j$ such that $f(p^k)=1$. Now g_0 is frontal; thus $g_0(p^j)=0$ implies $g_0(p^k)=0$. Hence $p^k \in S_0$ is also a test for that failure.

Theorem 4: Let $f=f(x_1, x_2, \cdots, x_n)$ be a frontal function, p^i a vertex of the n cube, and

$$S_1 = \{p^i \mid f(p^i) = 0 \text{ and } p^j > p^i \Rightarrow f(p^j) = 1\}. \quad (8b)$$

Then S_1 is a sufficient set of input combinations to detect all single stuck-at-1 faults in any nonredundant frontal network that realizes f.

Proof: Suppose $p^j \notin S_1$ detects a stuck-at-1 fault and g_1 is the one-bias function corresponding to that fault. By Theorem 2, $g_1 \supset f$ and consequently $f(p^j)=0$ and $g_1(p^j)=1$. Since $f(p^j)=0$ and $p^j \notin S_1$, there exists a $p^k \in S_1$, $p^k > p^j$ such that $f(p^k)=0$. Now, g_1 is frontal; thus $g_1(p^j)=1$ implies $g_1(p^k)=1$, and hence $p^k \in S_1$ is also a test for that failure.

In a frontal function all the prime implicants are essential, and to determine uniquely a frontal function, it is only necessary to know the smallest vertex (closest to $000\cdots00$) in each prime implicant. This follows from the fact that if $p^j > p^k$ and $f(p^k)=1$, then $f(p^j)=1$. These smallest vertices correspond to the elements of S_0 and therefore we have that to each prime implicant there corresponds one and only one element of S_0.

Similarly, in every prime implicate the largest vertex (the farthest from $000\cdots0$) corresponds to an element of S_1.

The sets S_0 and S_1 can be found directly from the Karnaugh map, or a Hasse diagram, or the minimal sum and minimal product Boolean expression.

To obtain S_0 by the last method, substitute any present variable in each prime implicant by a 1 and substitute a 0 for every variable not present. To obtain S_1, in each prime implicate substitute any present variable by a 0 and any variable not present by a 1.

Define the sets

$$T_0^i = \tau_0^i \cap S_0$$

$$T_1^i = \tau_1^i \cap S_1.$$

Then T_0^i (T_1^i) is the set of input combinations that detect the primary input x_i stuck at 0 (stuck at 1).

Since $S_0 \subseteq$ ON-SET of f and $S_1 \subseteq$ OFF-SET of f, then the sets T_0 and T_1 can be found directly using the following method: $t_0 \in S_0$ is also an element of T_0^i if and only if t_0 has a 1 in the ith position. Similarly, $t_1 \in S_1$ is also an element of T_1^i if and only if t_1 has a 0 in the ith position.

Example:

$$\begin{aligned} f &= x_1x_3 + x_1x_4 + x_2x_3 + x_3x_4 \\ &= (x_1 + x_3)(x_3 + x_4)(x_1 + x_2 + x_4) \\ &= \sum (3, 6, 7, 11, 12, 13, 15, 16, 17)_8. \end{aligned}$$

Then

$$S_0 = \{3, 6, 11, 12\} \qquad S_1 = \{2, 5, 14\}$$

and

$$T_0^1 = \{11, 12\} \quad T_0^2 = \{6\} \quad T_0^3 = \{3, 6, 12\} \quad T_0^4 = \{3, 11\}$$

$$T_1^1 = \{2, 5\} \quad T_1^2 = \{2\} \quad T_1^3 = \{5, 14\} \quad T_1^4 = \{2, 14\}.$$

It has been shown above that S_0 and S_1 are sufficient to detect all single faults, but there are networks that can be tested completely using subsets of $S_0 \cup S_1$, while other realizations need all the tests given by S_0 and S_1. In particular, we have the following theorem.

Theorem 5: All the elements of $S_0(S_1)$ are necessary to detect all stuck-at-0 (stuck-at-1) faults in a two-level irredundant AND–OR (OR–AND) realization of a function f. So the sets S_0 and S_1 are minimal for the class of frontal realization of f.

Proof: It has been shown above that every prime implicant is determined uniquely by an element of S_0; that is, each element of S_0 is covered by one and only one prime implicant. An input combination, say s_0, cannot be a test for single stuck-at-0 faults in lines feeding different gates A_i and A_j in the first level of an AND–OR network: the outputs of A_1 and A_j for such s_0 would be 1 if the gates were working properly, and, if any of the input lines to A_i are stuck at 0, the output of A_j, and consequently the output of the entire network, is still 1, and s_0 cannot be a test for A_i.

Similar reasoning applies for a two-level OR–AND network.

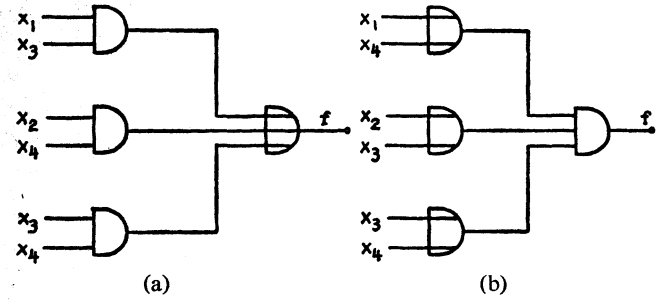


Fig. 4. Networks for $f=x_1x_3+x_2x_4+x_3x_4=(x_1+x_4)(x_2+x_3)(x_3+x_4)$. $S_0=\{3, 5, 12\}$; $S_1=\{6, 11, 14\}$. (a) AND–OR circuit—minimum sets of tests: stuck at 0: $\{3, 5, 12\}=S_0$; stuck at 1: $\{6, 11\}\subset S_1$. (b) OR–AND circuit—minimum sets of tests: stuck at 0: $\{5, 12\}\subset S_0$; stuck at 1: $\{6, 11, 14\}=S_1$.

A two-level AND–OR (OR–AND) network may not need all the S_1 (S_0) tests to detect all stuck-at-1 (stuck-at-0) faults, as shown in Fig. 4; but still $S_0\cup S_1$ is minimum for the class of realizations of f in the sense that there is no set Q with fewer elements than $S_0\cup S_1$ that detects all single stuck faults in all irredundant frontal realizations of a function f.

Effects of Network Form

To study the dependence between the structural form of a network and the number of tests necessary to detect all faults, we relate the tests in the internal lines to those for the primary inputs. To explain this point, let us consider the network in Fig. 5 that realizes the function $f=x_1x_3+x_1x_4x_2x_3+x_3x_4$, for which we found previously that the input combinations $S_0=\{3, 6, 11, 12\}$ and $S_1=\{2, 5, 14\}$ will detect any fault. To find tests that detect a fault, say in line a, we first show that any test that detects a fault in a line after branching must be a test for the line before the branching happens. To prove this last assertion, we introduce a new variable for every branch, in our case x_a and x_e for the two branches of input x_1; then a fault in x_1 will be equivalent to a double fault in x_a and x_e. The relationship between single and double faults is given by the following theorem.

Theorem 6: Let $f=f(x_1, x_2, \cdots, x_n)$ be a frontal function; then

$$\tau_0^i \subseteq \tau_{00}^{ij} \quad \text{and} \quad \tau_1^i \subseteq \tau_{11}^{ij}$$

where $\tau_{00}^{ij}=f\subset f_{00}^{ij}$ and $\tau_{11}^{ij}=f\oplus f_{11}^{ij}$ are the sets of input combinations that detect the double faults "x_i and x_j stuck at 0" and "x_i and x_j stuck at 1."

Proof:

$$f_{00}^{ij} \subseteq f_0^i \text{ because } f \text{ is frontal}$$

$$\bar{f}_0^i \subseteq \bar{f}_{00}^{ij} \text{ complementing}$$

$$f\bar{f}_0^i \subseteq f\bar{f}_{00}^{ij}$$

$$\bar{f}f_0^i + f\bar{f}_0^i \subseteq \bar{f}f_{00}^{ij} + f\bar{f}_{00}^{ij}.$$

Since $f_0^i\subset f$ and $f_{00}^{ij}\subset f$, then $\bar{f}f_0^i=\bar{f}f_{00}^{ij}=0$. Finally

$$f \oplus f_0^i \subseteq f \oplus f_{00}^{ij}$$

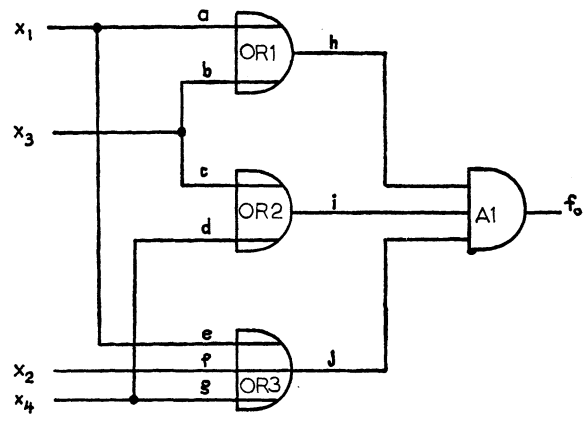


Fig. 5. Network for $f=(x_1+x_3)(x_3+x_4)(x_1+x_2+x_4)$.

or

$$\tau_0^i \subseteq \tau_{00}^{ij} \tag{9a}$$

in the same way

$$\tau_1^i \subseteq \tau_{11}^{ij}. \tag{9b}$$

Since f is frontal, so is f_0^i and $f_{00}^{ij}=f_0^i$ if and only if f_0^i is independent of x_j. For example, if $f(x_1, x_2, x_3)=x_1x_2+x_3$, then $f_0^1=x_3$ (x_2 is not essential in f_0^1) and $f_{00}^{12}=f_0^1=x_3$. In the situations where we will apply Theorem 6, x_i and x_j correspond to different names given to the same variable feeding different gates, and, since we are dealing with irredundant networks, we have the following corollary.

Corollary 1: Let $f=f(x_1, x_2, \cdots, x_n)$ be a frontal function and let input x_i branch out into lines y_p and y_q; then

$$\tau_0^p \subset \tau_{00}^{pq} = \tau_0^i \quad \text{and} \quad \tau_1^p \subset \tau_{11}^{pq} = \tau_1^i.$$

Since $T_0^p\subseteq\tau_0^p$ and $T_1^p\subseteq\tau_1^p$, then we have that

$$T_0^p \subset T_{00}^{pq} = T_0^i \tag{10a}$$

and

$$T_1^p \subset T_{11}^{pq} = T_1^i. \tag{10b}$$

The splitting apart of set T^{pq} into sets T^p and T^q must obey the following rules.

1) If the two branches reunite in an OR gate, then T_0^p and T_0^q are disjoint; otherwise (under the principle of path sensitizing and the assumed frontality of the network) any common test will inject two 1's in the merging OR gate blocking the sensitized paths. Similarly, two branches coming together into an AND gate cannot share a common test for stuck-at-1 faults.

2) If lines $p, r, s, \cdots$ feed an OR gate, then $T_0^p, T_0^r, T_0^s, \cdots$ are pairwise disjoint and $T_1^p=T_1^r=T_1^s\cdots$.

3) If lines $p, r, s, \cdots$ feed an AND gate, then $T_0^p=T_0^r=T_0^s\cdots$ and $T_1^p, T_1^r, T_1^s, \cdots$ are pairwise disjoint.

Example: In the network in Fig. 5, input x_1 fans out into lines a and e, input x_3 into lines b and c, and x_4 into lines d and g.

Any input combination that detects the fault "line a stuck at 0" must have a 1 in position 1 (line a is a branched-out line of x_1) and a 0 in position 3 (x_3 feeds the gate OR1 also), and any input combination that detects "line b stuck at 0" must have a 1 in position 3 and a 0 in position 1, and so the sets T_0^a and T_0^b are disjoint; then

$$T_0^a \subseteq T_0^1 - (T_0^1 \cap T_0^3) = \{11\}$$

and

$$T_0^b \subseteq T_0^3 - (T_0^1 \cap T_0^3) = \{3, 6\}.$$

The intersection $T_0^1 \cap T_0^3$ is the set of input combinations that has a 1 in positions 1 and 3 and distinguishes the double fault "lines a and b stuck at 0" from single faults tying down either line a or line b.

The test set for in output line of OR1 is equal to the union of the test sets for input lines a and b and the tests for multiple faults given by the pairwise intersection $T_0^1 \cap T_0^3$:

$$T_0^h = T_0^a \cup T_0^b \cup (T_0^1 \cap T_0^3) = \{3, 6, 11, 12\}.$$

Any input combination that detects the fault line a stuck at 1 must have a 0 in position 1 and a 0 in position 3 (x_3 feeds gate OR1 also). The faults in lines a, b, and h are equivalent and so the tests must be the same:

$$T_1^a = T_1^b = T_1^h = T_1^1 \cap T_1^3 = \{5\}.$$

Fault Detection in Backal Networks

A logical function is *backal* if in any minimal formula all the variables appear only complemented.

Example:

$$f = \bar{x}_1\bar{x}_2 + \bar{x}_1\bar{x}_4 + \bar{x}_2\bar{x}_3 + \bar{x}_2\bar{x}_4 + \bar{x}_3\bar{x}_4.$$

The theory developed so far for frontal functions applies also to backal ones with the following modifications:

$$h_0^i = \bar{x}_i f_0^i \bar{f}_1^i$$

$$h_1^i = x_i f_0^i \bar{f}_1^i$$

$$S_0 = \{p^i \mid f(p^i) = 1 \text{ and } p^j > p^i \Rightarrow f(p^j) = 0\}$$

$$S_1 = \{p^i \mid f(p^i) = 0 \text{ and } p^j < p^i \Rightarrow f(p^j) = 1\}.$$

Fault Detection in Unate Functions

A logical function that can be represented by normal form in which no variable appears both complemented and uncomplemented is called *unate*.

Let $f = f(x_1, x_2 \cdots, x_s, y_1, y_2, \cdots, y_{n-s})$ be a unate function where f is frontal in the set $X = \{x_1, x_2, \cdots, x_s\}$ and backal in the set $Y = \{y_1, \cdots, y_{n-s}\}$ ($x_1, \cdots, x_s$ appear only uncomplemented and $y_1, \cdots y_{n-s}$ appear only complemented in a minimal sum formula); then Theorems 1–6 apply to the set X and similar theorems apply to the set Y. In particular, to find the sets S_0 and S_1, the following procedure is used. In each prime implicant of the minimal sum every x_i present is substituted by 1 and the missing x_i by 0, and every $\bar{y}_i$ present by 0 and the missing y_i by 1. The binary vectors so obtained are the elements of S_0. Similarly, in each prime implicate of the minimal product every x_i present is substituted by a 0 and the missing x_i by 1, and every $\bar{y}_i$ present is substituted by a 1 and the missing y_i by 0, giving the elements of S_1.

Example:

$$f = \sum (1, 2, 3, 5, 6, 7, 12, 13)_8$$
$$= \bar{x}_1x_4 + \bar{x}_1x_3 + \bar{x}_2x_3 = (\bar{x}_1 + \bar{x}_2)(x_3 + x_4)(\bar{x}_1 + x_3).$$

In this example $X = \{x_3, x_4\}$ and $Y = \{x_1, x_2\}$. Then

$$S_0 = \{\overset{\bar{x}_1}{0}\ 1\ 0\ \overset{x_4}{1}, \quad \overset{\bar{x}_1}{0}\ 1\ \overset{x_3}{1}\ 0, \quad + \quad 1\ \overset{\bar{x}_2}{0}\ \overset{x_3}{1}\ 0\}$$
$$= \{5, \ 6, \ 12\}$$

$$S_1 = \{\overset{(\bar{x}_1+\bar{x}_2\quad)}{1\ 1\ 1\ 1}, \quad \overset{(\quad x_3+x_4)}{0\ 0\ 0\ 0}, \quad \overset{(\bar{x}_1\ +\ x_3\ )}{1\ 0\ 0\ 1}\}$$
$$= \{0, \ 11, \ 17\}.$$

Theorem 7: The sets S_0 and S_1 defined above are sufficient sets of tests to detect any stuck-at-0 or stuck-at-1 fault in any AND–OR realization of the unage function f.

Proof: If f is frontal, this theorem is equivalent to Theorems 3 and 4, and similar arguments hold if f is backal. For a general unate function we introduce the relationship $<:$. We say that $t_1 <: t_2$ if t_1 has a 0 in every position corresponding to a frontal variable where t_2 has a 0, and t_1 has a 1 in every position corresponding to a backal variable where t_2 has a 1. For example, suppose f is frontal in $X = \{u_1, u_2\}$ and backal in $Y = \{u_3, u_4\}$; then $0010 <: 1010$ and $1001 <: 1100$. Then $f(t_1) < f(t_2) \rightarrow t_1 <: t_2$.

Call g_0 the function realized by a network when a line is stuck at 0 and $t \notin S_0$ an input combination that detects such a fault; then clearly $f(t) = 1$ and $g_0(t) = 0$. But in the prime implicant that includes t there is another point $s \in S_0$, $s <: t$ such that $f(s) = 1$. Now, since the network consists only of AND and OR gates, $g_0(s) = 0$ and $s \in S_0$ is also a test for that fault. Similarly, let g_1 be the function realized by a network when a line is stuck at 1 and $t \notin S_1$ an input combination that detects such a fault; then $f(t) = 0$ and $g_1(t) = 1$. But in the prime implicate that includes t there is also another point $s \in S_1$, $s :> t$ such that $f(s) = 0$. Now, since the network consists only of AND and OR gates, $g_1(s) = 1$ and $s \in S_1$ is also a test for that fault.

References

[1] J. M. Galey, R. E. Norby, and J. P. Roth, "Techniques for the diagnosis of switching circuit failures," *IEEE Trans. Commun. Electron.*, vol. 83, Jan. 1964, pp. 509–514.

[2] D. B. Armstrong, "On finding a nearly minimal set of fault detection tests for combinatorial logic nets," *IEEE Trans. Electron. Comput.*, vol. EC-15, Feb. 1966, pp. 66–73.

[3] J. F. Poage, "Derivation of optimum tests to detect faults in combinatorial circuits," in *Proc. Symp. Mathematical Theory of Automata* (Polytechnic Inst. Brooklyn). Brooklyn, N. Y.: Polytechnic Press, 1963, pp. 483–528.

[4] J. P. Roth, "Diagnosis of automata failures: A calculus and a method," *IBM J.*, vol. 10, no. 4, 1966, pp. 278–291.

[5] F. F. Sellers, Jr., M. Y. Hsiao, and L. W. Bearnson, "Analyzing errors with the Boolean difference," *IEEE Trans. Comput.*, vol. C-17, July 1968, pp. 676–683.