Lehigh University
Lehigh Preserve

Theses and Dissertations

1-1-1980

A study of fault-detection in array logic.

Suk-In Yoo

A STUDY OF FAULT-DETECTION IN ARRAY LOGIC

By

Suk-In Yoo

A Thesis
Presented to the Graduate Committee
of Lehigh University
in Candidacy for the Degree of
Master of Science
in
Electrical Engineering

Lehigh University
1980

CERTIFICATE OF APPROVAL

This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering.

August 8, 1980
(date)

Professor in Charge

Chairman of Department

Acknowledgements

The author wishes to express his appreciation to Professor Alfred K. Susskind for his guidance and stimulation, and also thanks to Lehigh University for the teaching assistantship given to him during the graduate program for the degree of Master of Science.

Finally the author wishes to thank his wife, Hae-gung, for her encouragement and indulgence in allowing him the interrupted time to complete this project.

Table of Contents

Table of Contents Con'd

List of Figures

List of Figures Con'd

Abstract

The effects of all single and various multiple fault models in programmable logic arrays (PLA's), associative logic matrices (ALM's), and programmable storage/logic arrays (SLA's) are examined for the purpose of testing.

As one of the testing schemes for PLA's and ALM's, an exhaustive testing method is considered, which is very simple and detects all single and many multiple faults in PLA's, but not all of them in ALM's. Conventional testing schemes need tedious computations to detect some of the faults in PLA's and ALM's, but the number of tests required by these schemes is relatively small.

Conventional testing schemes for finite state machines (FSM's) are briefly reviewed and an appropriate testing scheme for Moore-type FSM's implemented by programmable storage/logic arrays (SLA's) is discussed. A new scheme for making an FSM more readily testable is given. It is directly applicable to Moore-type FSM, whereas other means for improving testability need conversions between Moore and Mealy forms. The previous methods may increase the number of states in the Moore machine to achieve testability, which is not desirable. The new scheme does not increase the number of states, i.e., the number of states in the Moore machine modified for testing is the same as that in the original Moore machine. However, the length of the test sequence will be greater.

1. Introduction

Programmable Logic Arrays (PLA's) provide an economical way of realizing combinational switching functions [1]. The PLA, which is simply two-level logic, becomes attractive in LSI due to its memory-like regular structure, as will be explained in Section 2. To achieve combinational functions with more than two levels of logic, the Associative Logic Matrices (ALM's) [9] may well have an advantage over PLA's.

As with any other logic circuit, PLA's and ALM's should be tested to insure that they operate correctly. The testing scheme considered here is to apply all possible input vectors to the array and check to see if the sum of all responses is correct, which will be called "checking C_0". This scheme has the disadvantage of requiring a large number of steps, i.e., with n inputs these are 2^n steps. But because fault models in PLA's or ALM's may well have to be more diverse than in other combinational circuits, because of the way PLA's or ALM's are fabricated, the conventional testing schemes require tedious computations when it is desired to consider the large variety of possible faults. The exhaustive testing scheme eliminates all of these computations, but does so at the cost of long test sequences. Another advantage of exhaustive testing is the simplicity of the test apparatus, both with respect to logic and memory.

Another type of array considered here is the programmable Storage/Logic Array (SLA) [10]. It can realize sequential circuits efficiently, as will be explained in Section 4. We consider the conventional testing scheme for sequential circuits based on distinguishing sequences and show how the SLA proposed in the literature could be modified to make testing by distinguishing sequences feasible as well as more efficient.

2. Testing Simple PLA's

A simplified schematic of the basic PLA array is given in Fig. 1. We show n-p-n transistors which, under the positive-logic convention (parameter representing logical value is larger for logical 1 than for logical 0) and with the parallel, grounded emitter configuration shown mechanize the NOR function. Thus our configuration realizes the NOR-NOR-INVERT or NOR-OR logic form. Since NOR is the product of the inverted inputs (e.g., $\overline{A + B} = \bar{A}\bar{B}$), it follows that the array shown in Fig. 1 realizes functions in the form of a sum of products, where the products consist of the *complements* of the connected inputs. In particular, Fig. 1 shows the following functions

$$f_1 = x_1\bar{x}_3 + x_2\bar{x}_4$$

$$f_2 = x_2\bar{x}_4 + \bar{x}_1\bar{x}_2\bar{x}_3$$

$$f_3 = x_1\bar{x}_3 + x_2x_3x_4$$

The horizontal lines at the output of the first level of gating, which mechanize the individual products, are frequently called the *word lines* and we shall use that term here. The inputs x_i and x_j are sometimes called the decoder outputs because in some PLA applications these inputs are not single literals, but products of more than one variable (typical two). This has been shown to be advantageous in some applications and will be discussed later.

We show below that the basic PLA can be tested by checking the value of C_0 at each output. This method of testing allows us to detect not only the commonly assumed single stuck-at faults, but also a variety of multiple faults, errors in programming, and even shorts.

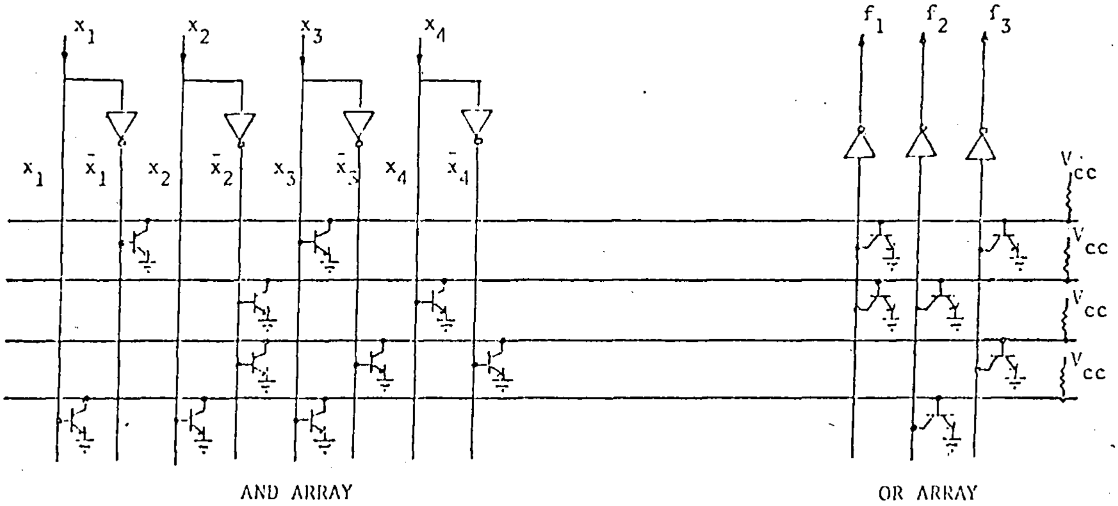


Fig. 1 Basic PLA

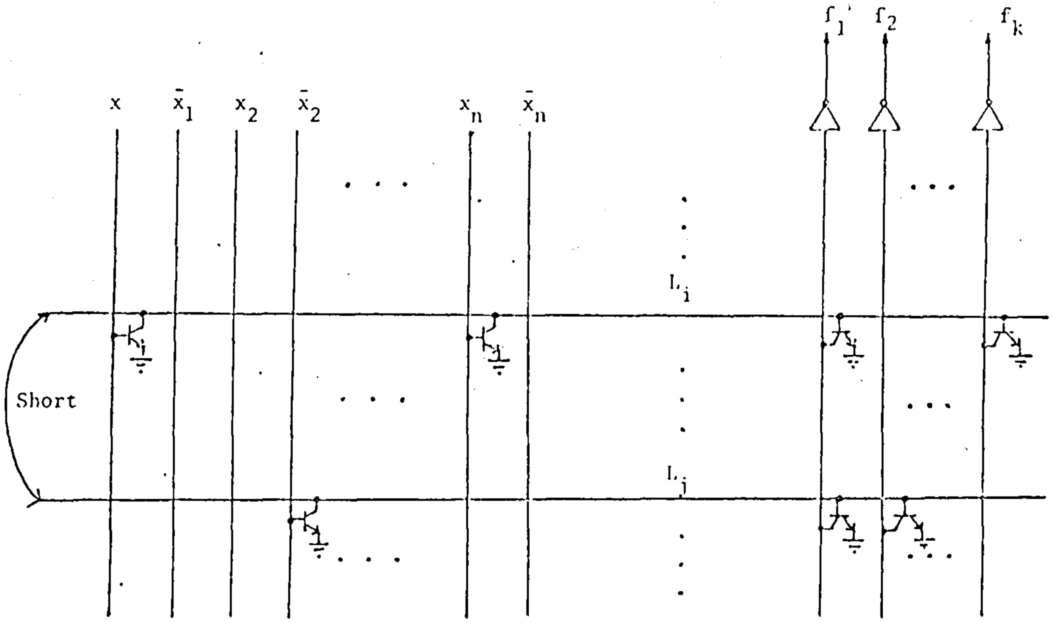


Fig. 2 Typical Pair of Word Lines

2.1 Crosspoint Defects

If in the AND array there is a missing device and the missing device should provide a connection to input line x_i^*, then the logical product mechanized on the word line to which the transistor should have been connected will have the variable $\bar{x}_i^*$ missing. In other words, the product $P = \bar{x}_i^* x_j \cdots x_s$ will become $P' = x_j \cdots x_s$. We say that this fault causes a growth because P' covers a larger subcube (implicant) than P, and so the function to which the word line is connected will have its true body enlarged (providing the connection to x_i^* was not redundant). It is easy to see that when more than one device is missing in the AND array, there results a growth in one or more of the functions realized in the array. In particular, if a word line has no connection in the AND array, then all functions to which that word line is connected in the OR array are set to the logical constant 1.

It follows that checking C_0 will detect any combination of missing devices in the AND array.

If there is a device missing in the OR array and that device should connect row r to function k, f_k, then the product realized on row r is no longer an implicant of f_k. We say that the missing device causes a drop because the true body of f_k has been diminished by the dropping of a product in the sum (unless the missing product was redundant). More than one missing device in the OR array will cause various kinds of drops, and any multiplicity of these will always be detected by checking C_0.

Missing devices in both the AND array and the OR array, however, are not necessarily detected by checking C_0. Consider, for example, the realization of the majority function M = AB + AC + BC, with word lines L_1 through L_3 realizing the products AB, AC, and BC, respectively. If there is a device missing between input line $\bar{A}$ and L_1 as well as devices missing where connections to lines L_2 and L_3 should be made, the faulty function realized is $M' = B$. Because M' has four ones as does M, checking C_0 will fail to reveal the assumed multiple fault pattern.

An extra device in the AND array connected to input x_j^* will result in the product realized on the corresponding word line having $\bar{x}_j^*$ added. We will call this a shrinkage, because the augmented product will cover a smaller subcube (implicant), and so the function to which the word line with the extra device is connected will have its true body diminished. Unless there is redundancy, the effect of the extra device will be detected by checking C_0. Similarly, multiple extra devices in the AND array will also be detected. For the special case where the extra device connects to x_j^* and the word line also has a connection to $\bar{x}_j^*$, the result is a drop of the nominal product, and C_0 will detect this case as well.

One or more extra devices in the OR array add extra products to the function(s) realized. These faults are detected by checking C_0.

No general statement can be made about the effectiveness of checking C_0 in the presence of extra devices in both the AND array and the OR array.

In summary, we have shown that checking C_0 detects all single crosspoint defects as well as a variety of multiple defects, but not all possible combinations.

2.2 Stuck Lines

An input line stuck in 1 causes all word lines connected to that input line to be set to logical zero. This causes one or more drops and is detected by checking C_0. An input line x_i^* stuck in 0 causes every product with nominal $\bar{x}_i^*$ to become independent of $\bar{x}_i^*$ ($P = \bar{x}_i^* x_j \cdots x_k$ becomes $P' = x_j \cdots x_k$). Hence this fault causes one or more growths and these are detected by checking C_0.

It is easy to see that output lines stuck are detected by checking C_0.

A word line stuck in 1(0) is the extreme case of a growth (drop) i.e., the corresponding product has grown to the logical constant 1(0).

2.3 Shorts

Consider Fig. 2 and let there be a short between word lines L_i and L_j. This short has an effect only when the inputs are such that the nominal product on L_i is true (false) and that on L_j is false (true). In the first case, L_i is nominally high (low) while L_j is nominally low (high). The short, however, makes both lines low in both cases, so that the word lines act as though there were devices in all places where there are input-line connections to L_i or L_j. In other words, the behavior due to the short is equivalent to extra devices in the AND array. As was pointed out above, any combination of extra devices in the AND array is detected by checking C_0, and thus shorted word lines in any combination are detected.

Next, we consider those lines that feed the inputs to the inverters; we will call these lines "function lines". Since shorts between lines cause the lower line to dominate, both shorted lines will carry logical 0 when *either* has a nominal 0. In the case of shorted function lines, this is equivalent to both lines having devices in *all* the places of the OR array where *either* has a device. As was shown before, extra devices in the OR array are detected by checking C_0, and therefore so are shorts between function lines.

A short between a word line L_i and a function line that nominally is not connected to L_i will result in lowering the function line when one or more of the transistors connected to L_i conducts. Since the function realized on L_i is a product P_i (of the *complements* of the variables connected to transistors on that word line), the word line is low whenever one or more of the connected variables is true, and so the output f_j in the presence of the short becomes $f_j' = f_j + \bar{P}_i$. Thus the true body of the faulty output is enlarged, and verifying C_0 for output f_j does check for the assumed type of short.

To illustrate, suppose there is a short in the OR array of Fig. 1 between function line 2 and the topmost word line. Then f_2 becomes one whenever $\bar{x}_1$ is high *or* x_3 is high. Thus we get the faulty function $f_2' = x_2\bar{x}_4 + \bar{x}_1\bar{x}_2\bar{x}_3 + \bar{x}_1 + x_3 = x_2\bar{x}_4 + \bar{x}_1 + x_3$, which has 13 ones, whereas the fault-free output f_2 has only six ones.

In the case where the short involves a function line that is connected to the word line L_i in the fault-free circuit, the effect is that of a short between base and collector of the transistor that makes the connection. This causes the function line to become stuck in 0, i.e., the output will be stuck in 1. This is detected by checking C_0.

When an input line connected to x_i^* is shorted to word line L_i, then L_i is low whenever x_i^* is low <u>or</u> one or more of the other inputs x_j^* through x_k^* to L_i is low. Thus instead of realizing the product $P_i = \bar{x}_j^* \cdots \bar{x}_k^*$, the word line realizes $P_i' = x_i^* \cdot P_i$, which is called a <u>shrinkage</u>, if the fault-free circuit has no connection to x_i^*; if it does have a connection to x_i^*, then there is a short between base and collector of the connecting transistor. This makes L_i stuck in 0, i.e., it causes a <u>drop</u>. Checking C_0 on the corresponding output will detect the fault in both cases.

Since shorts between lines cause the lower line to dominate, both of the shorted lines will be low when either is nominally low. If both of the shorted lines, x_i^* and x_j^*, are connected to word line L_m, then the faulty product on L_m would be $P_m' = (\bar{x}_i^* + \bar{x}_j^*) \cdot x_p^* \cdots x_q^*$ instead of the nominal product $P_m = \bar{x}_i^* \cdot \bar{x}_j^* \cdot x_p^* \cdots x_q^*$. If one of the lines, x_i^* (x_j^*), is connected to word line L_n, then $P_n' = P_n + P_n^0 \cdot \bar{x}_j^*$ $(\bar{x}_i^*)$, where the nominal product is $P_n = P_n^0 \cdot \bar{x}_i^*(\bar{x}_j^*)$. In either case, the true body of the product is enlarged, and so is that of the corresponding output f_i. It is apparent that checking C_0 for the output f_i always detects shorts between input lines.

Similarly to before, when an input line connected to x_i^* and a function line are shorted, then the function line will mechanize $\bar{f}_j' = \bar{f}_j \cdot x_i^*$ $(f_j' = f_j + \bar{x}_i^*)$, because the lower line always dominates the higher one. (Whenever x_i^* is low, the function line will be low; the function line is still low regardless of the x_i^* value whenever the nominal function line is low). The true body of output j is thus enlarged. This short can also be detected on some other output. If the input line x_i^* is connected to word line L_k, then the product realized on word line L_k in the presence of the short will become $P_k' = P_k + P_k^0 \cdot f_j$, where the nominal product is $P_k = P_k^0 \cdot \bar{x}_i^*$. This growth in P_k can be detected on any output that is connected to L_k. Since in both cases the true body of the output is enlarged, it is easy to see that the shorts assumed here are also detected by checking C_0.

2.4 More General Decoder Form and Application of C_0

Sometimes it is advantageous to use more than one literal (typically two) as the inputs to the PLA's. For simplicity, consider Fig. 3, where two literals are used as the inputs to the AND array and n-p-n transistors are used in the NOR-OR logic form in the PLA. From Fig. 3, where the decoders are of the form shown in Fig. 4, we have

P_1 = Product mechanized on L_1 = $(x_1+\bar{x}_2)(\bar{x}_1+x_2)(\bar{x}_3+\bar{x}_4)(x_5+\bar{x}_6)$

P_2 = Product mechanized on L_2 = $(x_1+x_2)(x_3+\bar{x}_4)(x_5+\bar{x}_6)(\bar{x}_5+\bar{x}_6)$

P_3 = Product mechanized on L_3 = $(x_1+\bar{x}_2)(\bar{x}_1+\bar{x}_2)(\bar{x}_3+x_4)(\bar{x}_3+\bar{x}_4)(x_5+x_6)$

p_4 = Product mechanized on L_4 = $(x_1+x_2)(\bar{x}_1+x_2)(x_3+x_4)(\bar{x}_5+x_6)$

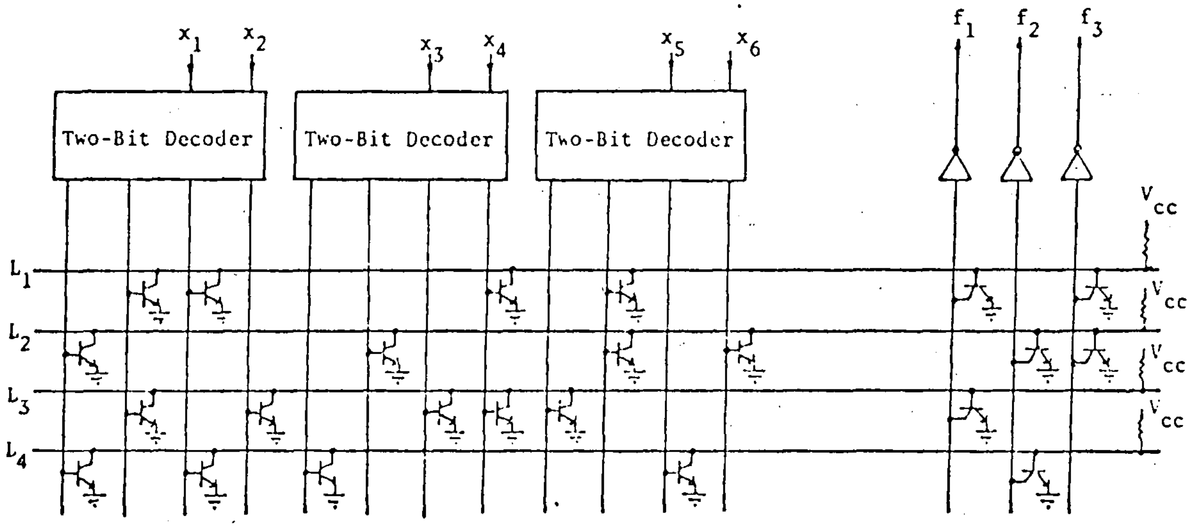


Fig. 3 PLA with Two-Bit Decoders

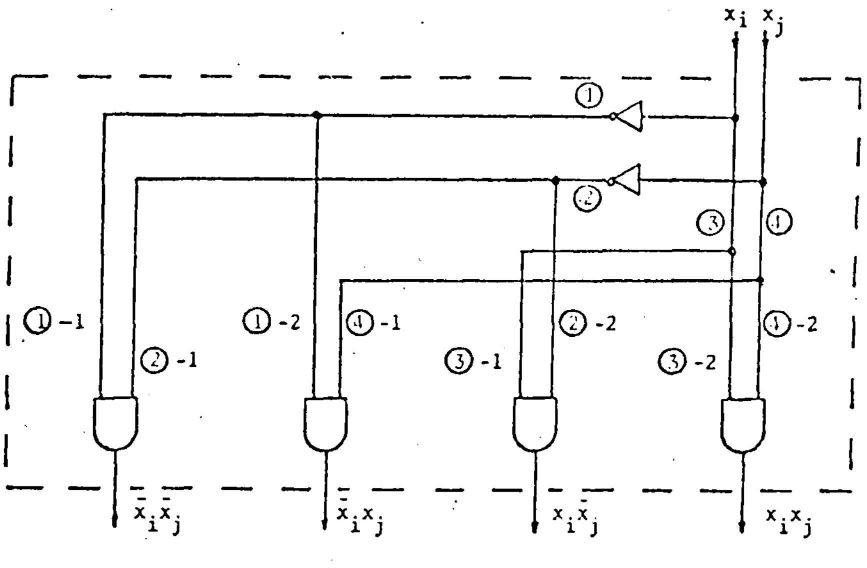


Fig. 4 Two-Bit Decoder

$$f_1 = P_1 + P_3$$

$$f_2 = P_2 + P_4$$

$$f_3 = P_1 + P_2$$

In general, the product P_m realized on the word line L_m is

$$P_m = (x_i^* + x_j^*)(x_k^* + x_\ell^*) \cdots (x_m^* + x_n^*), \text{ where } i \neq j,\ k \neq \ell, \text{ and } m \neq n$$

2.41 Single or Multiple Missing Devices

If the missing device is in the AND array and the missing device should provide a connection to the input line $x_i^*x_j^*$, then the product $P = (\bar{x}_i^* + \bar{x}_j^*)(x_k^* + x_\ell^*)\cdots(x_m^* + x_n^*)$ will become $P' = (x_k^* + x_\ell^*)\cdots(x_m^* + x_n^*)$ so that this fault causes a growth. Similarly, more than one device missing in the AND array results in a growth in one or more of the functions realized in the array.

If the missing device is in the OR array and the device should connect row r to function f, then the function $f = P_r + P_i + \cdots + P_j$ will become $f' = P_i + \cdots + P_j$, so that this fault causes a drop. It is evident that more than one device missing in the OR array results in various kinds of drops. In all cases C_0 verification can serve as a test.

2.42 Single or Multiple Extra Devices

An extra device in the AND array connected to input $x_i^*x_j^*$ will result in the product realized on the corresponding word line having $(\bar{x}_i^* + \bar{x}_j^*)$ added, which causes a shrinkage. (The nominal product $P = (x_\ell^* + x_m^*)\cdots(x_p^* + x_q^*)$ will become $P' = (\bar{x}_i^* + \bar{x}_j^*)(x_\ell^* + x_m^*)\cdots(x_p^* + x_q^*)$ due to the fault). Similarly, multiple extra devices in the AND array result in a shrinkage in one or more of the functions and this is detected by checking C_0 in the array.

If an extra device in the OR array connects row r to function f, then the function $f = P_i + \cdots + P_j$ will become $f' = f + P_r$, so that the true body is enlarged by the fault. Multiple extra devices in the OR array enlarge the true body of one or more of the corresponding outputs. Clearly these faults are all detected by checking C_0.

2.43 Stuck Lines in the PLA

As explained in Section 2.2 most of the stuck lines are equivalent to single (multiple) missing devices or single (multiple) extra devices, and it is not difficult to see that stuck lines not equivalent to missing or extra devices are also detected by checking C_0.

2.44 Shorts in the PLA

In all cases of shorted lines, the same arguments as given under shorts in Section 2.3 are applicable here, except that the input literals in the products are not x_i^* but $(x_i^* + x_j^*)$ for $i \neq j$. Therefore, we can say that the effectiveness of testing by verifying C_0 is not reduced when a two-bit decoder is used.

2.45 Stuck Lines in the Decoders

Refer to Fig. 4. Assuming that all pins are fault-free, input lines to the AND gates in the decoder stuck in 0 are equivalent to output lines from the AND gates stuck in 0, which are equivalent to input lines to the AND array of the PLA stuck in 0.

If the fan-out line x_i^* to the AND gate (i.e., $\textcircled{i}$ - 1 or $\textcircled{i}$ - 2 in Fig. 4.) is stuck in 1, then the output from the corresponding AND gate becomes x_j^*, while nominally it is $x_i^* x_j^*$, so that the word line L_m connected to the output from the decoder realizes $P_m' = \bar{x}_j^*(x_\ell^* + x_m^*) \cdots (x_p^* + x_q^*)$ as its product, whereas its nominal product is $P_m = (\bar{x}_i^* + \bar{x}_j^*)(x_\ell^* + x_m^*) \cdots (x_p^* + x_q^*)$. If the fan-out point x_i^* to the AND gate (i.e., $\textcircled{i}$ in Fig. 4.) is stuck in 1, then the two outputs from the AND gates become x_j^* and $\bar{x}_j^*$, where the nominal outputs are $x_i^* x_j^*$ and $x_i^* \bar{x}_j^*$, respectively. This causes the word line L_m to realize $P_m' = \bar{x}_j^* \cdot x_j^* \cdot (x_\ell^* + x_m^*) \cdots (x_p^* + x_q^*) = 0$, if both outputs from the decoder are connected to the word line L_m. Otherwise (i.e., only one output connected), the faulty product becomes $P'_m = \bar{x}^*_j(x^*_\ell + x^*_m)\cdots(x^*_p + x^*_q)$ or $P'_m = x^*_j(x^*_\ell + x^*_m)\cdots(x^*_p + x^*_q)$. Since in either case the fault causes a drop, this is detected by checking C_0.

3. Testing the Associative Logic Matrix

Greer's Associative Logic Matrix [9] makes possible the efficient realization of multiple output, multiple level, combinational and sequential networks by means of the regular interconnection structure of read-only memory and programmable logic arrays. For the implementation of complex multiple-output Boolean functions, which frequently can be expressed efficiently in more than two levels of logic, Associative Logic Matrices (ALM's) may well be advantageous over Programmable Logic Arrays (PLA's), which are typically restricted to two-level logic.

The ability to implement networks involving more than two levels of logic is achieved in the ALM through the use of "internal function logic". This logic involves additional bit lines which serve the dual role of forming logical sums (or products) and providing the resulting signals as inputs in the formation of subsequent functions. For simplicity, we restrict the realization of associative logic to four-level combinational circuits. All connections in the array are "wired-NOR"ed by means of n-p-n transistors. In Fig. 5, the internal function g is $g = x_1x_2 + x_3$, one output is $f_1 = x_4g + \bar{x}_1\bar{g} + x_1\bar{x}_2\bar{x}_4$, and another output is $f_2 = x_4g + \bar{x}_2\bar{g}$.

The structure of the ALM differs from that of the PLA in the addition of the G-array, which realizes the internal functions. Hence the ALM consists of the AND array, the OR array, and the G-array, as shown in Fig. 5. The rightmost bit line of the G-array will be called the "collector line of the G-array" and the other two lines, which are used as inputs in the formation of the output functions, will be called the "output line g of the G-array" and the "output line $\bar{g}$ of the G-array", respectively, as shown in Fig. 5.

To ease fault-detection in the ALM's, we will add extra logic. It consists of one extra output, f_e, which is fed by all word lines connected to the output line $\bar{g}$ of the G-array. Thus f_e is of the form $f_e = gx_i^*\cdots x_j^* + \cdots + gx_k^*\cdots x_\ell^*$. (In Fig. 5, the extra output is $f_e = gx_4$).

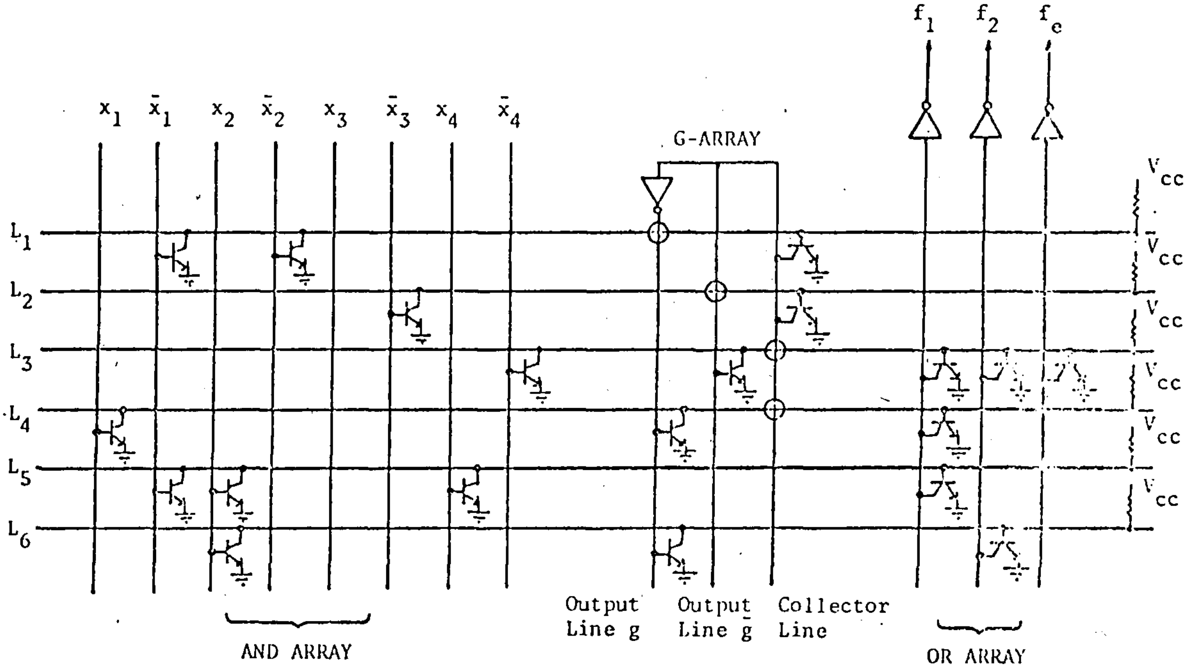


Fig. 5 Associative Logic Matrix

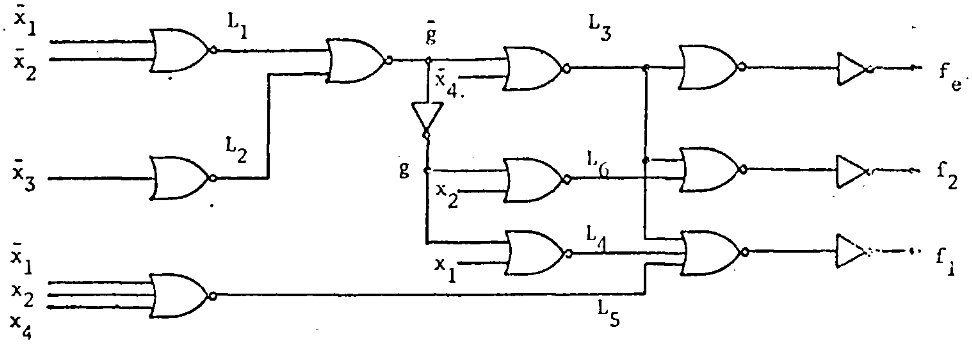


Fig. 6 ALM of Fig. 3.5 Redrawn

To avoid duplicating previous explanations in Section 2, we list those faults that have the same effect in ALM's as in PLA's and omit further discussion of these:

1. Single or multiple missing (extra) devices in the AND array, where the corresponding word lines do not feed the internal function (i.e., the corresponding word lines are not connected to the collector line of the G-array).
2. Single or multiple missing (extra) devices in the OR array.
3. Stuck lines not in the G-array.
4. Shorts between word lines, unless one or both of the shorted lines feed the internal function.
5. Shorts between input lines in the AND array that have no path to the collector line of the G-array.
6. Shorts between output lines in the OR array.
7. Shorts between a word line and an output line in the OR array.
8. Shorts between a word line and an input line in the AND array.
9. Shorts between an input line in the AND array and an output line in the OR array.

3.1 Effect of Other Faults in the ALM's

3.11 Single or Multiple Missing Devices

If a missing device in the AND array should provide a connection to the input line x_i^* <u>and</u> the word line L_k to which the device should have been connected is one of the word lines feeding the internal function g, then the product realized on L_k will become $P_k' = x_m^* \cdots x_n^*$ instead of $P_k = \bar{x}_i^* \cdot P_k'$, which we have called a <u>growth</u>. This fault will enlarge the true body of the internal function g and also that of the extra output f_e. Checking C_0 on f_e will detect the fault. It is easy to see that multiple missing devices in the AND array will be detected by checking C_0 on f_e if at least one of the corresponding word lines feeds the internal function g.

If a missing device in the G-array should provide a connection to the output line g*, then the product realized on its corresponding word line L_m will become $P'_m = x^*_\ell \cdots x^*_m$ instead of $P_m = \bar{g}^* \cdot P'_m$, and hence there is a growth. Since this will enlarge the true body of the output f_i fed by the word line L_m, checking C_0 on f_i will detect it.

A missing device in the G-array which should provide a connection to the collector line drops the product on its corresponding word line, and so the true body of the extra output f_e is reduced, which will be detected by checking C_0 on f_e. Moreover, either multiple missing devices in the AND array *and* the output line g* of the G-array or missing devices in the OR array *and* the collector line of the G-array will be surely detected by checking C_0. But multiple missing devices in both the output line g* and the collector line of the G-array are not necessarily detected by checking C_0.

3.12 Single or Multiple Extra Devices

If an extra device in the AND array connects the word line L_k to the input x^*_i *and* the word line L_k does feed the internal function g, then the product on L_k will become $P'_k = \bar{x}^*_i \cdot P_k$, which we have called a *shrinkage*. This will reduce the true body of the internal function g and also that of the extra output f_e. Checking C_0 of f_e will detect this fault. Multiple extra devices in the AND array will be surely detected by checking C_0 of the extra output f_e, if at least one of the corresponding word lines feeds the internal function g.

One or more extra devices in the output line g* of the G-array, except for special case A discussed below, will be detected by checking C_0 of the output f_k fed by the corresponding word line, because the fault causes a shrinkage and reduces the true body of the output f_k. It is not difficult to see that multiple extra devices in both the AND array and the output line g* will be detected by checking C_0. One or more extra devices in the collector line of the G-array, except for case B treated below, cause one or more products realized on the corresponding word lines to become additional implicants of the internal function g, and so the true body of the extra output f_e is enlarged, which will be surely detected by checking C_0 of f_e. Multiple extra devices in both the OR array and the collector line of the G-array will also be detected by checking C_0.

Case A

If one or more extra devices are connected to the output line g* of the G-array and at least one of the corresponding word lines nominally feeds the internal function g, then this fault will cause feedback. Consider Fig. 5 and Fig. 6 , which is a conventional representation of Fig. 5 . If an extra device connects the output line g to the word line L_1, then the extra device will cause feedback, as shown in Fig. 7 . (This situation is illustrated by the top circle in Fig. 5 .) Suppose x_1, x_2, and x_3 are, respectively, 1, 1, and 0, so that nominally $L_1 = 1$, $L_2 = 0$ and g = 1. With the fault, however, if g were 1, the three inversions around the closed loop would complement g, so the value of g could not remain 1 and in fact the value of g would oscillate. This is the kind of fault that a static type of test just cannot detect; only waveform observation will be sure to result in detection.

If an extra device connects the output line $\bar{g}$ to a word line L_2, then it will cause feedback as shown in Fig. 8. (This situation is illustrated by the circle on Line L_2 of Fig. 5.) This feedback over an even number of inversions can be detected by a sequence of two tests. The first makes both L_1 and L_2 low, so that $\bar{g} = 1$. This is a stable condition in the presence of the feedback. The second test makes L_2 nominally high while keeping L_1 low, so that nominally $\bar{g} = 0$. If feedback is present, however, the second test will leave $\bar{g}$ unchanged, i.e., it remains 1. While the assumed fault is detectable, simply checking C_0 will not always work.

Case B

If one or more extra devices connect word lines to the collector line of the G-array and at least one of the corresponding word lines is nominally connected to the output line g*, then this will cause

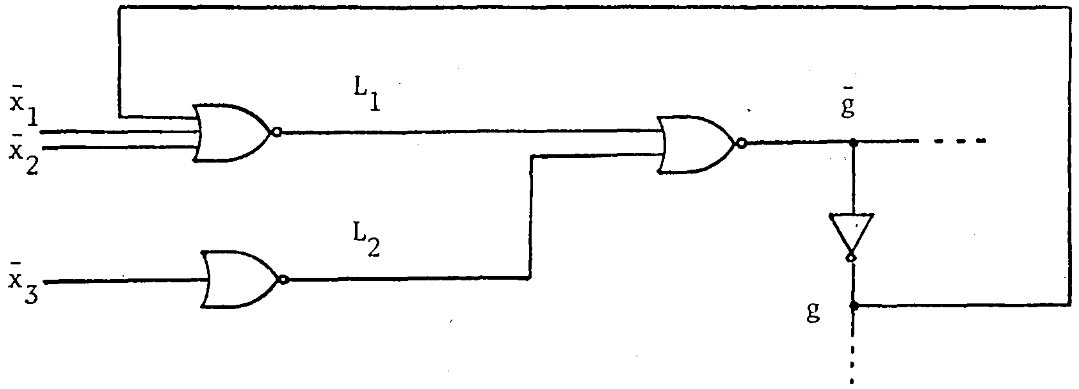


Fig. 7 Feedback from g Due to Extra Device

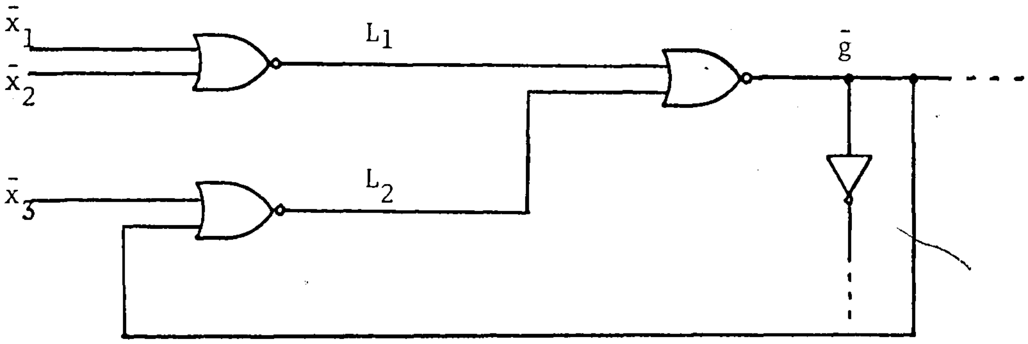


Fig. 8 Feedback from $\bar{g}$ Due to Extra Device

feedback. Refer to Figs. 5 and 6. If an extra device connects the word line L_3 to the collector line of the G-array, then the extra device will cause feedback as shown in Fig. 9. (This situation is illustrated by the circle on the third line of Fig. 5.) As before, this feedback can be detected by a sequence of two tests. The first makes both $\bar{g}$ and $\bar{x}_4$ low, so that $L_3 = 1$, which is a stable condition in the presence of the feedback. The second test makes $\bar{g}$ nominally high while keeping $\bar{x}_4$ low, so that nominally $L_3 = 0$. If feedback is present, then the second test will leave L_3 unchanged, i.e., it remains 1. However, simply checking C_0 will not necessarily detect this fault.

If an extra device connects the word line L_4 to the collector line of the G-array, then the extra device will cause feedback as shown in Fig. 10. (This is illustrated by the circle on the fourth line of Fig. 5.) As before, when an input combination which makes all of L_1, L_2, and x_1 low is given, the value of L_4 will oscillate due to the feedback. A static type of test cannot detect this fault; only waveform observation can.

3.13 Stuck Lines in the G-array

Since an output line g* stuck in the G-array is equivalent to one or more missing devices or one or more extra devices in the G-array, this will be easily detected by checking C_0.

If the collector line of the G-array is stuck at 1(0), then the fault will be equivalent to both the output line g stuck at 0(1) and the output line $\bar{g}$ stuck at 1 (0). The output line $\bar{g}$ stuck at 1(0) reduces (enlarges) the true body of the extra output f_e, so that checking C_0 of f_e will detect the collector line stuck. If a word line L_k is stuck and L_k is one of the word lines feeding the internal function g, then the fault will surely enlarge (reduce) the true body of the internal function g and also that of the extra output f_e. This fault will be detected by checking C_0 of f_e.

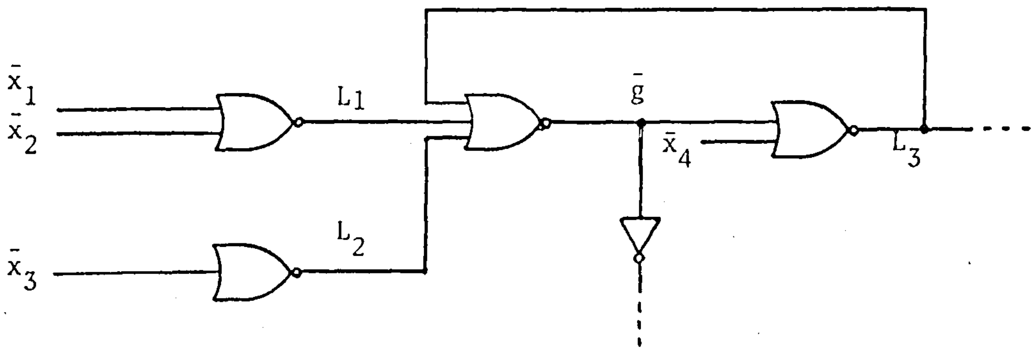


Fig. 9 Feedback Due to Extra Device on Collector Line

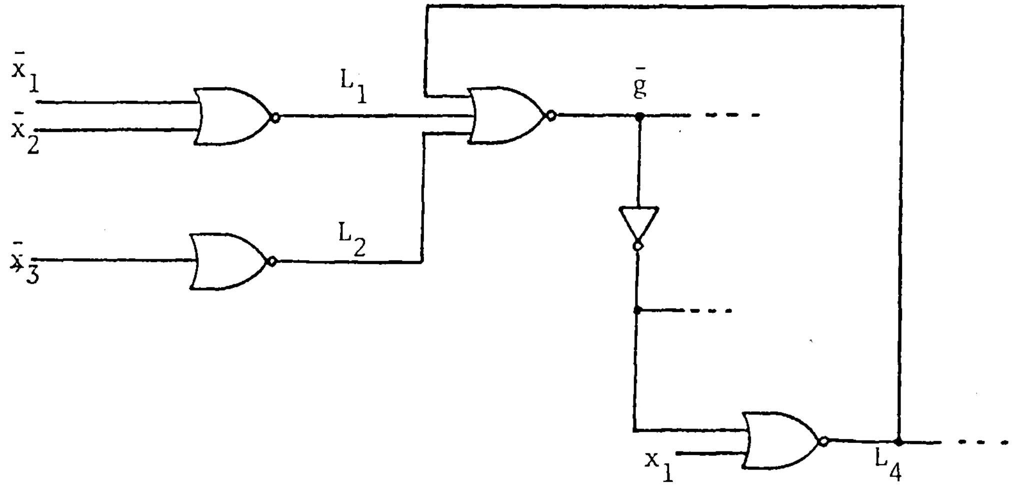


Fig. 10 Feedback that Can Cause Oscillation

3.14 Shorts Between Word Lines

If both of the shorted word lines feed the internal function, then the short between lines is equivalent to extra devices in both word lines. Extra devices in both lines will reduce the true body of the internal function g and also that of the extra output f_e, so that checking C_0 of f_e will surely detect the fault.

Similarly, if one of the shorted lines feeds the internal function g and the other, L_k, is not connected to the output line g* of the G-array, then both the true body of the internal function and that of the output fed by the word line L_k will be reduced, because the short is equivalent to extra devices in both lines. This will also be detected by checking C_0.

However, if one of the lines feeds the internal function g and the other is connected to the output line $\bar{g}$ (g) of the G-array, then the fault will (not) always be detected by checking C_0. Refer to Fig. 5 and Fig. 6. If the word line L_1 is shorted to the word line L_4, which is connected to the output line g of the G-array, then the circuit will be changed to that shown in Fig. 11, where the signals on L_1 and L_4 will oscillate. This is not detected by a static type of test. If the word line L_1 is shorted to the word line L_3 connected to the output line $\bar{g}$ of the G-array, then this will be detected by checking C_0 on the extra output f_e. Refer to Fig. 12. Once L_1 becomes low both L_1 and L_3 will be stuck at 0 due to the short, so that the true body of f_e will be reduced.

3.15 Shorts Between Input Lines in the AND Array

As explained in the discussion of the PLA, shorts between input lines enlarge the true body of the products realized on all word lines connected to those input lines. If one or more word lines connected to those input lines feed the internal function g, then checking C_0 on the extra output f_e will detect this fault, because the fault will enlarge the true body of the internal function g as well as the extra output f_e.

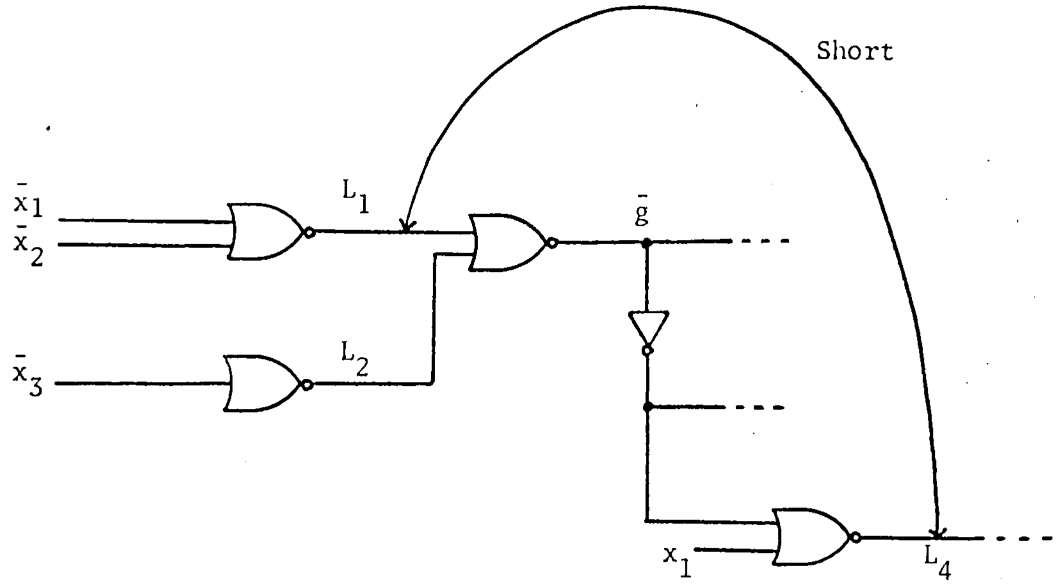


Fig. 11 Feedback Due to Short I

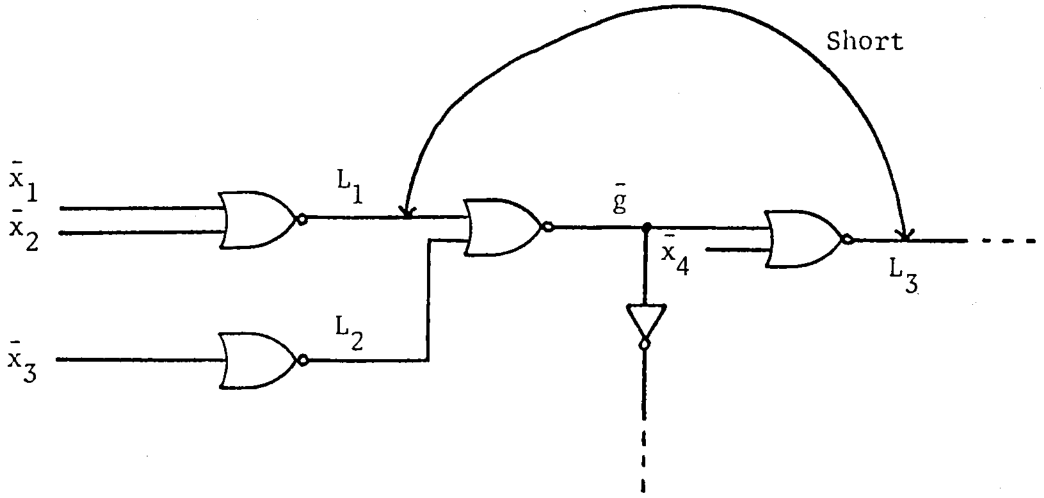


Fig. 12 Feedback Due to Short II

3.16 Shorts Between a Word Line and an Input Line in the AND Array

If a word line L_k is not connected to the input line x_i^* in the fault-free circuit and L_k is one of the word lines feeding the internal function g, then the product realized on L_k will become $P_k' = P_k + \bar{x}_i^*$ due to the short, because the short between lines makes the lower value dominate. This enlarges the true body of the internal function g, and also that of the extra output f_e, which is readily detected by checking C_0 of f_e. Otherwise (e.g., L_k is nominally connected to x_i^* and L_k is one of the word lines feeding the internal function g), L_k will become stuck at 0 due to the short, which connects base and collector of the transistor nominally driven by x_i^*. This will be also detected by checking C_0 of f_e.

3.17 Shorts Between a Word Line and the Collector Line of the G-array

If the shorted word line L_k is not connected to the output line g* of the G-array and L_k does not feed the internal function g in the fault-free circuit, then the short between L_k and the collector line will result in the faulty internal function $g' = \bar{P}_k + g$, so that the true body of the extra output f_e will be enlarged. Checking C_0 of f_e will detect the fault. If L_k feeds the internal function g in the fault-free circuit, then there is a base-to-collector short and the short will result in the collector line stuck at 0. As discussed in 3.213, this is detected by checking C_0 of f .

When the word line L_k is nominally connected to the output line g of the G-array and L_k is shorted to the collector line of the G-array, then checking C_0 on the extra output f_e will detect this fault. Consider Figs. 5 and 6. If the word line L_4 and the collector line of the G-array are shorted, then the circuit will be changed to that shown in Fig. 13. In Fig. 13, since the circuit locks up with both $\bar{g}$ and L_4 stuck at 0 once $\bar{g}$ becomes low, the short will enlarge the true bodies of the internal function g and the extra output f_e. Hence, checking C_0 of f_e will detect the fault.

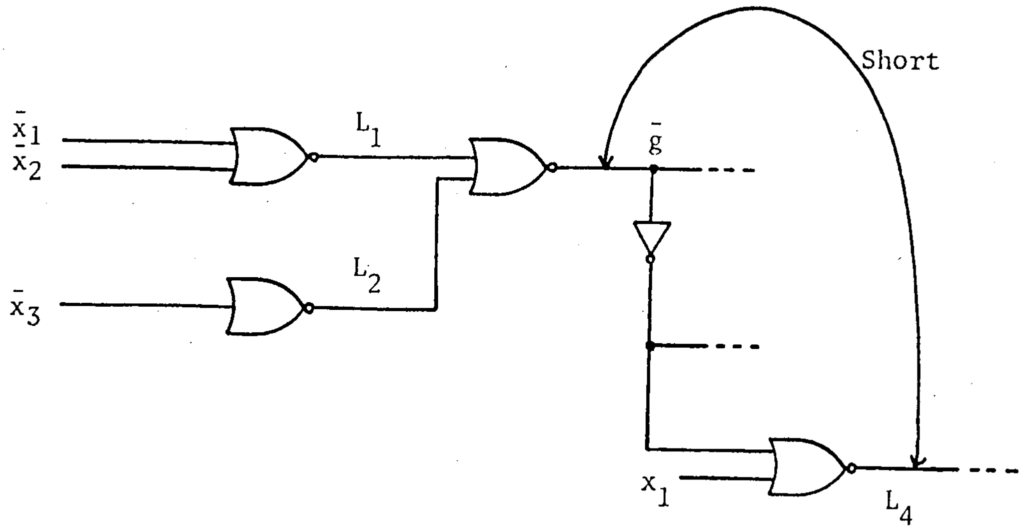


Fig. 13 Feedback Due to Short III

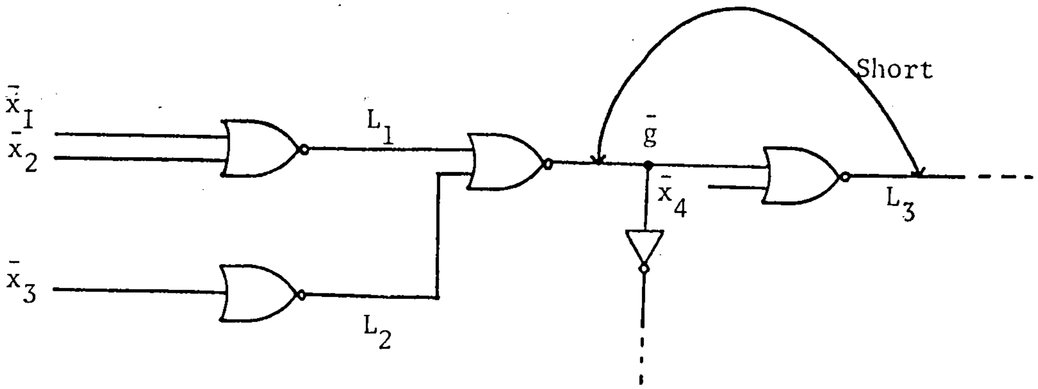


Fig. 14 Feedback Due to Short IV

However, if the word line L_k is nominally connected to the output line $\bar{g}$ of the G-array and it is shorted to the collector line of the G-array, then simply checking C_0 will not detect the fault. Consider Figs. 5 and 6. If the word line L_3 and the collector line of the G-array are shorted, then the circuit will be changed to that shown in Fig. 14. The short will make the values of $\bar{g}$ and L_3 oscillate.

3.18 Shorts Between a Word Line and an Output Line g* of the G-array

If the word line L_k is not connected to the output line g* and L_k does not feed the internal function g in the fault-free circuit, then the short between L_k and the output line g* of the G-array will result in $P_k' = P_k \cdot g^*$, which will reduce the true body of the output f_i fed by the word line L_k. Checking C_0 on f_i will surely detect the fault. But if L_k is connected to the output line g* in the fault-free circuit, then L_k will become stuck at 0 due to the fault, so that the true body of the output f_i fed by L_k will be reduced. The fault can be detected by checking C_0 of f_i.

When the word line L_k does feed the internal function g in the fault-free circuit, the short between L_k and the output line g of the G-array will be always detected by checking C_0. Consider Figs. 5 and Fig. 6. If the word line L_1 and the output line g of the G-array are shorted, then the circuit will be changed to that shown in Fig. 15. As before, once L_1 becomes low, both L_1 and g will be stuck at 0. This will reduce the true body of the internal function g, so that checking C_0 of f_e will detect the fault.

However, if L_k does feed the internal function g in the fault-free circuit, the short berween L_k and the output line $\bar{g}$ of the G-array will not be detected by simply checking C_0. Consider Fig. 5 and Fig. 6. If the word line L_1 and the output line $\bar{g}$ of the G-array are shorted, then the circuit will be changed to that shown in Fig. 16. Since the values of L_1 and $\bar{g}$ oscillate due to the short, checking C_0 will no longer detect the fault.

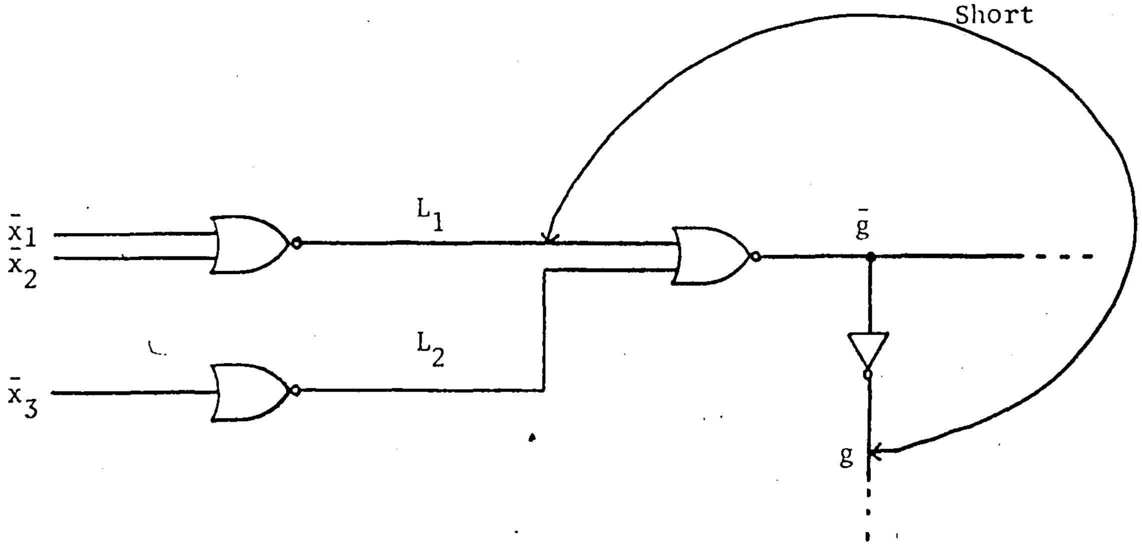


Fig. 15 Feedback Due to Short V

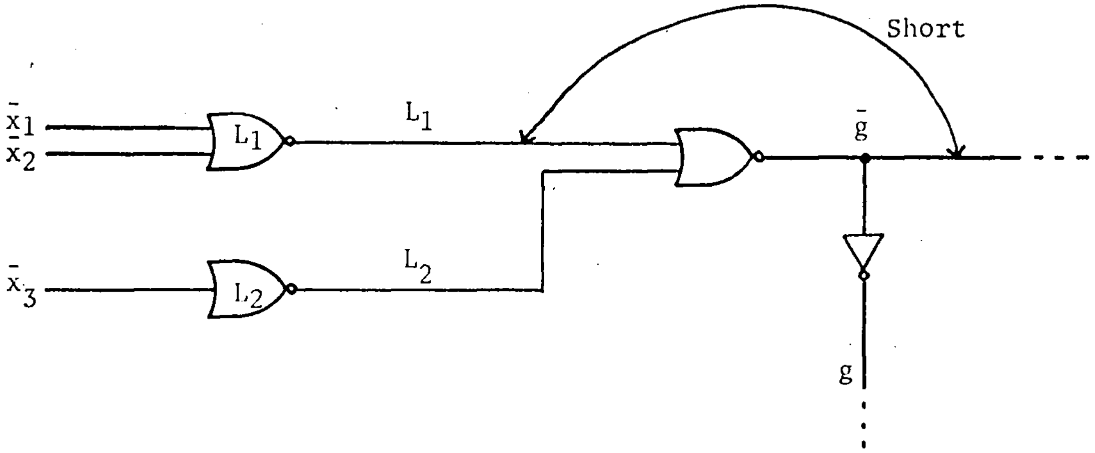


Fig. 16 Feedback Due to Short VI

3.2 Discussion

While the ALM, because of the addition of the G-array, is not limited to the realization of two-level combinational logic, faults in the G-array are not easily detected. In particular, some faults cause oscillation and cannot be detected by a static type of test, but only by waveform observation. Thus we conclude that testing of PLA's is easier than testing of ALM's.

4. Fault-Detection in Programmable Storage/Logic Arrays

Patil and Welch's programmable Storage/Logic Array (SLA) [10] is a form of PLA which contains flip-flops distributed throughout the array. Because in some computer designs purely combinational PLA's are difficult to use extensively due to pin limitations, some PLA's with flip-flops providing internal feedback from the outputs back to the inputs, as shown in Fig. 17, have been proposed [1], [2]. SLA's differ from previously described PLA's in that the AND and OR arrays are folded together so that input lines and output lines are alternated within a single array (see Fig. 18). As described in [10], "This has two important effects: (1) substantially more flip-flops can be added without the need for excess input-output routing space, and (2) rows of the array...can be subdivided into multiple independent segments which can represent independent variables over smaller portions of the array." Furthermore, the columns can also be subdivided into segments carrying independent variables with localized access by adding more flip-flops at the intervals along the columns.

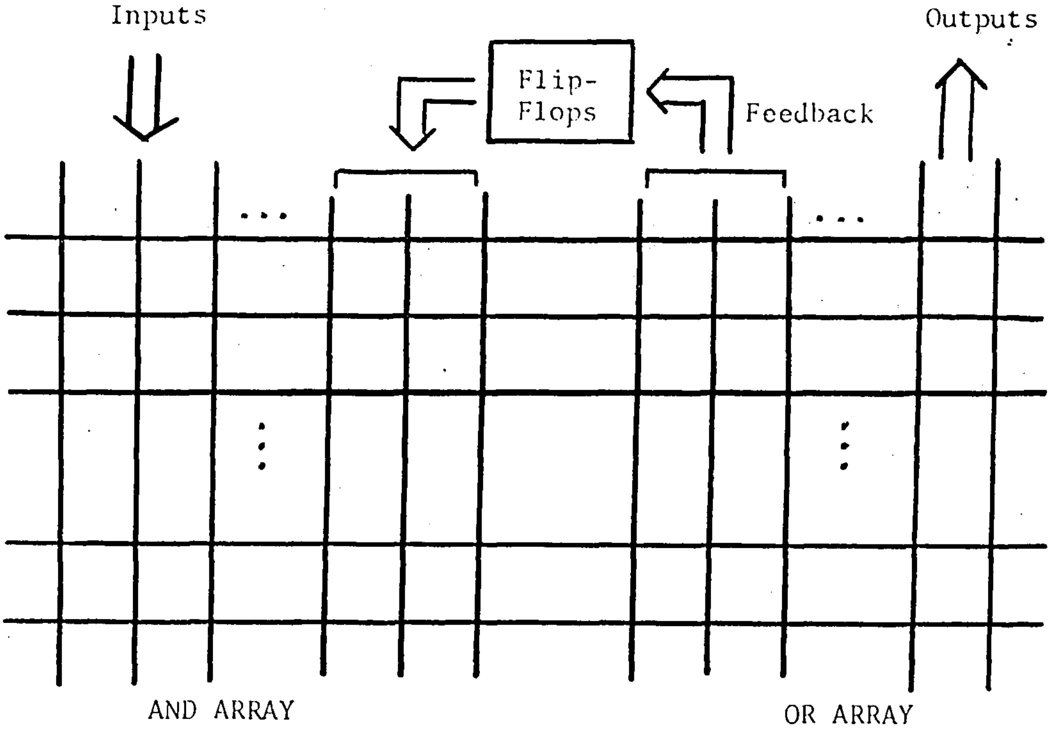


Fig. 17 PLA with Feedback

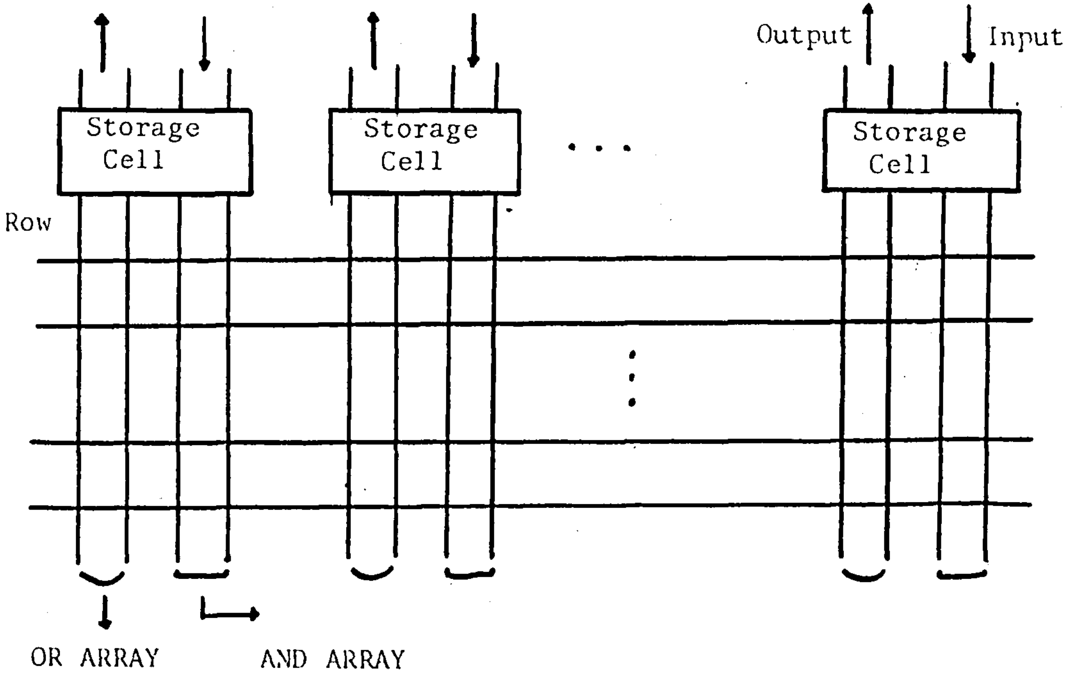


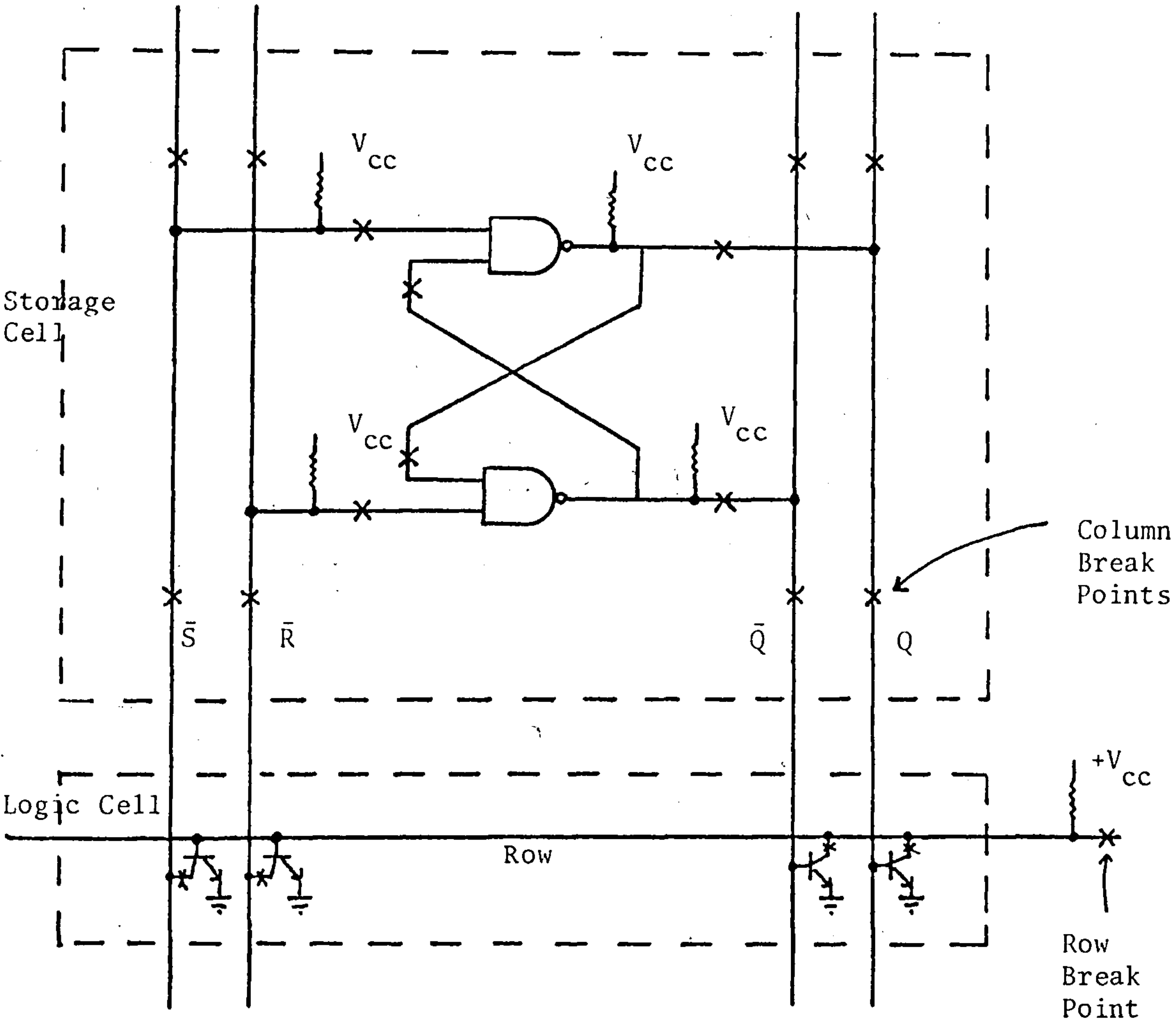
Fig. 18 Storage/Logic Array (SLA)

In the SLA circuit shown in Fig. 19, row-column connections are made by transistors with collectors that are selectively connected in a wired-NOR structure, and "storage cells" consist of cross-coupled NAND gates with complemented inputs $\bar{S}$ and $\bar{R}$ (i.e., set-reset flip-flops are used), so that the two outputs from the NAND gates, Q and $\bar{Q}$, will be 0(1) and 1(0), respectively, if the two inputs $\bar{S}$ and $\bar{R}$ are 1(0), 0(1), respectively. Outputs Q^+ and $\bar{Q}^+$ maintain the previous values if $\bar{S}$ and $\bar{R}$ are both 1, and Q^+ and $\bar{Q}^+$ are unpredictable if $\bar{S}$ and $\bar{R}$ are both 0. (Here and later the superscript + denotes the signal shortly after set and/or reset values have been established.)

Since the leads in the storage cell contain breakpoints, it can be used, by opening the breakpoints, for purposes other than the flip-flop described above. For example, the feedback loop can be broken so that the outputs of the cell are simple combinational functions of the inputs.

As one example of SLA logic, we consider here finite-state machines (FSM's). Refer to Fig. 20. If the machine has m states, n bivalued inputs, and k bivalued outputs, then the total number of cells required in the SLA will be $(k + n + 3 + \lceil \log_2 m \rceil)$ where $\lceil p \rceil$ is the integer equal to or just larger than p. Of this number, n cells are used for the n inputs, one for the reset input, and one for the clock-pulse input. These cells are buffers obtained by breaking the feedback loops. A total of $\lceil \log_2 m \rceil$ flip-flop cells are used for the storage of the state variables, q_i, and k flip-flops for the storage of the k outputs, Z_i. One flip-flop, F_α, is used for determining the proper time duration of the clock-pulse. This flip-flop will be called the <u>clock-pulse modifier</u>.

There is one row in the array for each possible state transition and the corresponding outputs. Thus if under some input state S_i can go to S_j, under another input S_i can go to S_k, and under the third input S_i stays unchanged, then there is a total of two rows involving state S_i. Given m states and p different inputs, there can be at <u>most</u> mp rows. The transition is made and the associated output is established



x : Programmable Break Points

Fig. 19 Basic Cells of an SLA

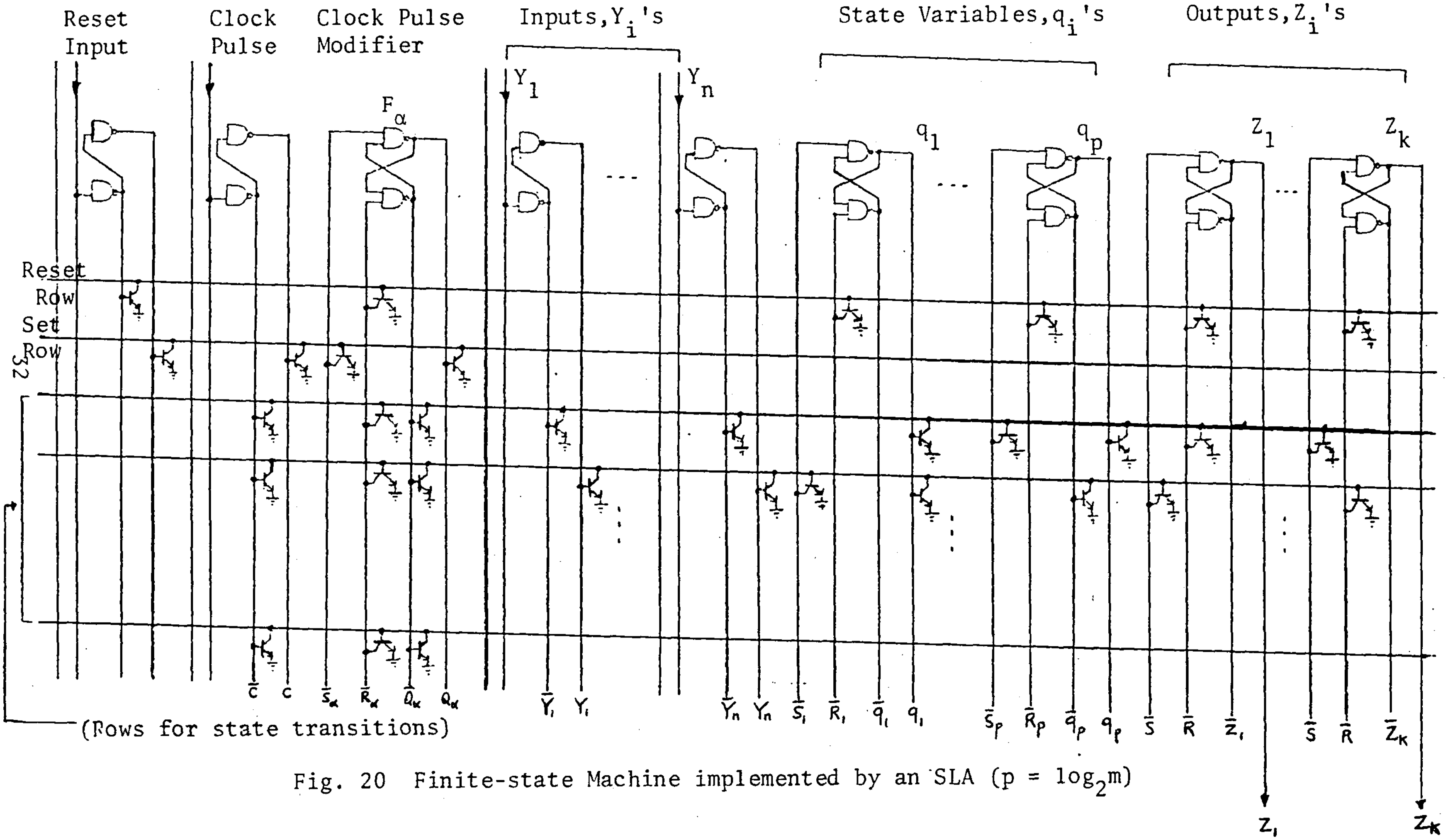


Fig. 20 Finite-state Machine implemented by an SLA ($p = \log_2 m$)

when the corresponding row is activated, i.e., made high. There are also two extra rows. One will reset the machine; the other will set the machine into the initial state when activated. For proper operation the machine is first reset and then becomes set by activating the corresponding row. The row to reset the circuit, which will be called the reset row, is driven by the negated reset input, and each input line $\bar{S}_i(\bar{R}_i)$ to the state flip-flops and output flip-flops is connected to the reset row, so that the initial values of the state variables and outputs are inserted when the reset row is activated. The input line $\bar{R}_\alpha$ to the flip-flop F_α is also connected to the reset row. When the reset input is high, the reset row is activated, and because $\bar{Q}_\alpha = 1$, all other rows are made low, so that the initial conditions and outputs will be stored in the flip-flops in accordance with their connections to the reset row. (In Fig. 20, the initial state-variable values and outputs will all be zero.) The row to set the machine, which will be called the set row, is connected to $\bar{S}_\alpha$ and Q_α of the flip-flop F_α. The set row is also connected to the reset input and the clock-pulse input. When both reset input and clock pulse are low (note that $Q_\alpha(\bar{Q}_\alpha)$ became low (high) when the circuit was first reset), then the set row will be activated. Thus $Q_\alpha(\bar{Q}_\alpha)$ is changed to high (low), which will cause the set row to go back to low, but $Q_\alpha(\bar{Q}_\alpha)$ still remains high (low). This makes the circuit ready for state transitions, because all rows for state transitions are connected to $\bar{Q}_\alpha$.

Each row for implementing a state transition is connected to the state variables q_i^* at the outputs of the state flip-flops and the input variables Y_i^* applied from outside the array. The input lines $\bar{S}_i, \bar{R}_i$ to the state flip-flops and the input lines to the output flip-flops are connected to the appropriate rows. All rows for state transitions are connected to the negated clock-pulse input from the outside, so that a row (no row if the present state is expected to be unchanged) is activated for a state transition only when the present state and inputs are appropriate and the clock pulse becomes high.

All rows for state transitions are also connected to the output line $\bar{Q}_\alpha$ from flip-flop f_α, which serves to protect the circuit from improper clock-pulse length. The input line $\bar{R}_\alpha$ to flip-flop F_α is also connected to each of these rows. If one of the rows for state transitions is then activated when the clock pulse is high, the corresponding next state variables q_i's and outputs Z_i's will be stored in the state and output flip-flops, and at the same time $Q_\alpha(\bar{Q}_\alpha)$ from the flip-flop F_α will become low (high). This value of $Q_\alpha(\bar{Q}_\alpha)$, which is not changed unless the clock pulse becomes low, will make all rows for state transitions low. In other words, more than one state transition for one clock pulse is not allowed, even if the length of a clock pulse is excessive. (This scheme achieves the effect of edge triggering.)

As a simple example, consider machine M_1 with the flow table shown in Fig. 21. It has four states, one output, one input, and seven state transitions, so that there will be seven cells $(1 + 1 + 3 + \log_2 4 = 7)$. The cell for the input, the one for the reset input, and the one for the clock-pulse input do not have feedback loops. Nine rows (a reset row, a set row, and seven rows for state transitions) in the SLA are shown in Fig. 22. When the reset input is made high, row r_1 is activated, so that $Q_\alpha(\bar{Q}_\alpha)$ will become low (high), and both state variables q_1 and q_2 will become low. This represents the initial state A, and the output Z of the initial state A will be low. If next the reset input is changed to low and the clock pulse is made low, then row r_2 will be activated, and so $Q_\alpha(\bar{Q}_\alpha)$ will be changed to high (low), but state variables q_1, q_2 and the output Z are unchanged. There are two rows, r_3 and r_4, which recognize the initial value of the state variables $(q_1 q_2 = 0\ 0)$, so that if the input Y is made low and the clock pulse goes high, then row r_3 will be activated and so q_1, q_2, and Z become, respectively, low, high, and low, which represents the next state $B(q_1 q_2 = 01)$ and its output $(Z = 0)$. Similarly, when the next clock pulse occurs, one of the two rows r_5 and r_6 will be activated for the corresponding state transition and output.

q_1q_2		Y = 0	Y = 1	Output
00	A	B	C	0
01	B	C	D	0
10	C	D	C	1
11	D	A	B	0

Fig. 21 Flow Table of M_1

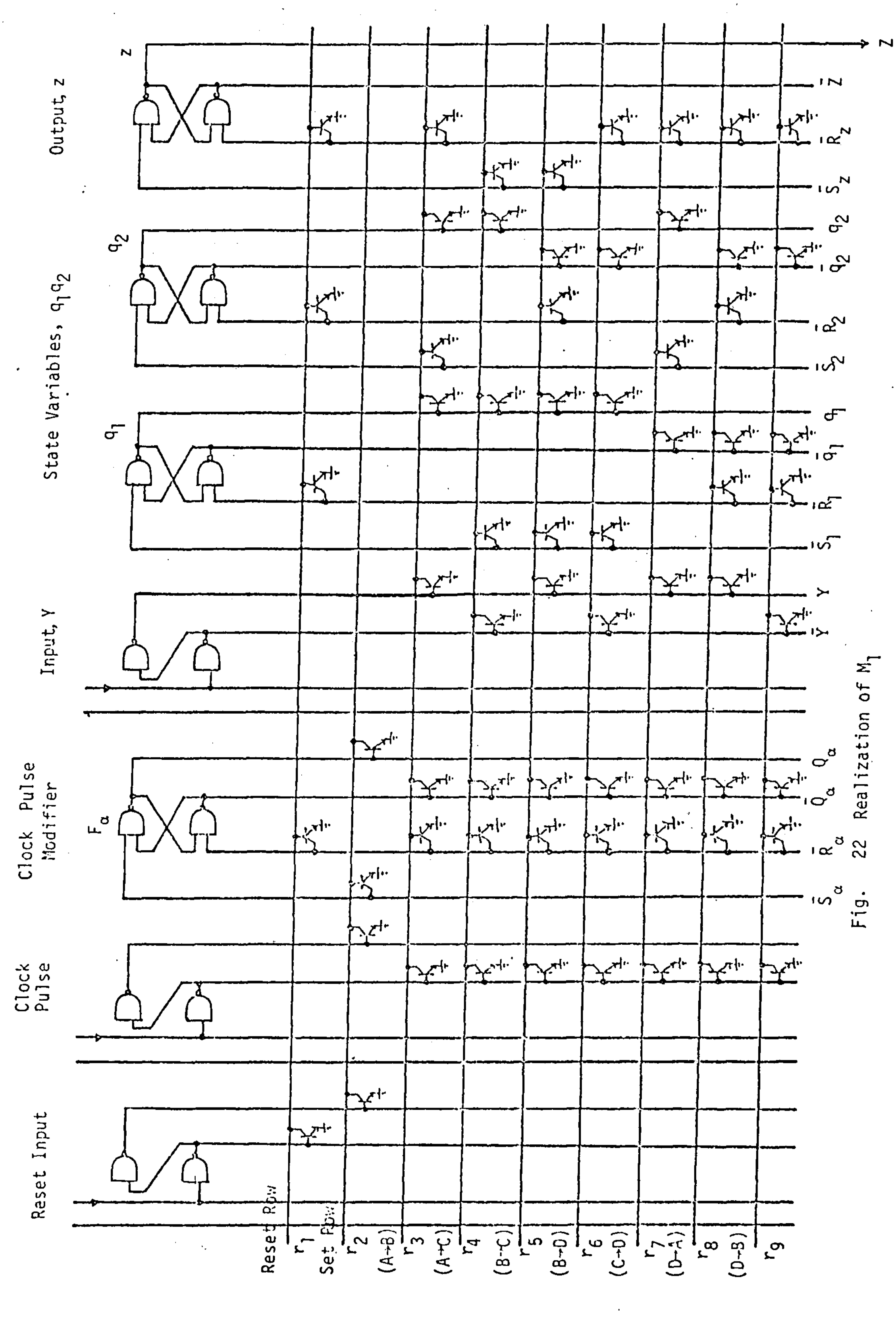


Fig. 22 Realization of M_1

4.1 Effects of Faults

4.11 Stuck Lines

Refer to Fig. 20. Input line $\bar{S}$ to the flip-flop stuck at 1(0) will cause the output Q to be reset (or it becomes unpredictable, as explained later in case C) if the other, $\bar{R}$, is nominally low. Otherwise (i.e., $\bar{R}$ nominally high), the output Q will be unchanged (set) due to the fault. Similarly, input line $\bar{R}$ stuck at 1(0) will cause the output Q to be set (unpredictable) if $\bar{S}$ is nominally low. Otherwise, the fault will cause Q to be unchanged (reset). Thus the fault may result in the incorrect next state, but it will leave the outputs correct if the flip-flop F is one of the state flip-flops. The possible malfunction will be called an incorrect state transition (IST). If the flip-flop involved is one of the output flip-flops, an incorrect output, denoted IO, may occur. Except for case A, if the input line $\bar{R}_\alpha$ to the flip-flop F_α for the clock-pulse modifier is stuck at 1(0), then the circuit will no longer be synchronous, unless the fault is redundant, so that the next state and the outputs may be incorrect, which will also be called an incorrect state transition (IST). The input line $\bar{S}_\alpha$ stuck at 1(0) (except for case C) will make all the rows connected to $\bar{Q}_\alpha$ low, so that no state transitions (NST) will occur.

The output line $Q_j^*(Q_\alpha^*)$ from a state flip-flop F_j (the flip-flop F_α) stuck-at-1 keeps all the rows connected to $Q_j^*(Q_\alpha^*)$ from being activated, so that the corresponding state transitions cannot occur. This will result in NST. If the output line $Q_j^*(Q_\alpha^*)$ is stuck at 0, then a row connected to $Q_j^*(Q_\alpha^*)$ may be activated for a state transition when it should not be, so that the next state and the output may be incorrect. Hence we have an IST. If the output line Q_k^* from the output flip-flop F_k is stuck at 1(0), then it will clearly cause the IO.

An input line Y_i^* stuck-at-1 keeps all the rows connected to Y_1^* from being activated, i.e., it results in NST. If Y_i^* is stuck at 0, then one of the rows connected to Y_i^* may be activated when it should not be, so that it will result in an IST.

The reset-input line stuck at 1(0) will keep the circuit from being set (reset), so that this will result in NST (IST).

The clock-pulse input line stuck-at-1 keeps all rows connected to it from being activated, which results in NST, while the clock-pulse input line stuck-at-0 causes the circuit to be no longer synchronous, so that it could result in an IST. It is easy to see that row r_k stuck at 1(0) will cause an IST (NST).

Case C

If both inputs $\bar{S}$ and $\bar{R}$ to the flip-flop F become low due to the fault, then the outputs Q^+ and $\bar{Q}^+$ from F will be unpredictable.

4.12 Single or Multiple Missing (Extra) Devices at the Crosspoints

If a missing device disconnects the input line $\bar{S}(\bar{R})$ from the row r_k, then the output Q^+ from the flip-flop F remains unchanged when it should be changed. Thus the fault will cause an IST, but the outputs will be correct, if the flip-flop F is one of the state flip-flops, or an IO will result if the flip-flop F is one of the output flip-flops. A missing device in the output line Q_i^* causes the corresponding row to be activated when it should not be activated, so that the next state and the outputs may be incorrect; hence there results in an IST. Similarly, multiple missing devices will cause IST's and/or IO's.

An extra device in the input line $\bar{S}(\bar{R})$, except for case C, may cause the output Q to be different from the nominal value, so that the next state may be incorrect, hence the result is an IST. If an extra device connects the output line Q_i^* to row r_k, then row r_k will not be activated when it should be, so that the present state and the outputs will be unchanged, hence the result is NST. Similarly, multiple extra devices will cause IST's and/or NST's.

A missing device in the input line Y_i^* which is nominally connected to row r_k causes the row r_k to be activated when row r_k should not be, which results in an IST. If an extra device connects the input line Y_i^* to row r_k, then row r_k will not be activated due to the fault when it should be, so the result is NST. It is not difficult to see that multiple missing (extra) devices in the input lines will cause IST's and/or NST's.

Similarly, one or more missing (extra) devices in the reset-input line and the clock-pulse line will cause IST's and/or NST's.

4.13 Shorts

Here, as before, we assume that a short between lines makes the lower value dominate. In the SLA, if two rows, r_m and r_n, are shorted, then they will always remain low (i.e., be never activated) because at least one of any pair of rows is always low. The result of the short is NST.

Except for case C explained previously, consider a short between the input line $\bar{S}_i(\bar{R}_i)$ and the output line Q_j^* from flip-flop F_j. The input line $\bar{S}_i(\bar{R}_i)$ becomes high when the clock pulse is low. If the output line Q_j^* is nominally low, then the short between Q_j^* and $\bar{S}_i(\bar{R}_i)$ will make $\bar{S}_i(\bar{R}_i)$ low, so that the output Q_i^* from the flip-flop F_i will become 1(0). This may cause an IST if F_i is one of the state flip-flops, an IO if F_i is one of the output flip-flops, and an IST or NST if F_i is the clock-pulse modifier.

A short between $\bar{S}_i$ and $\bar{Q}_i$ (or $\bar{R}_i$ and Q_i) will not affect the normal values of Q_i and $\bar{Q}_i$. If there is a short between $\bar{S}_i$ and Q_i (or $\bar{R}_i$ and $\bar{Q}_i$), then the value of $Q_i(\bar{Q}_i)$ may oscillate between 1 and 0, so that the fault is not easily modelled.

Given the input Y_j^* low, the short between $\bar{S}_i(\bar{R}_i)$ and Y_j^* will make $\bar{S}_i(\bar{R}_i)$ low when $\bar{S}_i(\bar{R}_i)$ should be high. Similarly to the short between Q_j^* and $\bar{S}_i(\bar{R}_i)$, this will cause one of the following: an IST, an IO, or NST, depending on what the flip-flop F_i is used for. It is not difficult to realize that a short between two columns other than those considered above will cause either an IST, an IO, or NST.

Next we consider shorts between a row and a column. When the clock pulse is low, all rows in the array become low, and so all input lines $\bar{S}_i(\bar{R}_i)$ become high. The short between row r_k and the input line $\bar{S}_i(\bar{R}_i)$ will cause $\bar{S}_i(\bar{R}_i)$ to be low, so that, as explained in the previous case, this will cause either an IST, an IO, or NST. Consider a short between row r_k and the output line Q_i^* from the flip-flop. If r_k and Q_i^* are nominally high and low, respectively, then the short will cause r_k to be low. In other words, r_k will not be activated, hence NST. If r_k and Q_i^* are nominally low and high, respectively, then Q_i^* will become low due to the short, so that it will cause an IST (an IO) if the flip-flop F_i is one of the state (output) flip-flops. Similarly, if the input line Y_i^* and the row r_k are nominally low (high) and high (low), respectively, then the short between Y_i^* and r_k will cause NST (an IST). It is easy to see that a short between a row and the reset-input line (the clock-pulse input line) will cause either NST or an IST.

To summarize, our exhaustive examination has shown that the effect of faults in the SLA will be one of the following:

1. No State Transition (NST)
2. Incorrect State Transition (IST)
3. Incorrect Output (IO)

4.2 Change in the Number of States Due to Faults

In preparation for later discussion of the proper method of fault-detection in an SLA, we examine first the effect of faults on the number of states.

4.21 NST's

Since the fault does not allow the present state S_i to be changed to the next state S_j, the total number of states in the machine may be decreased. In other words, once the machine reaches the state S_i, it cannot reach the state S_j, so that the state S_j will no longer occur if S_i is the only state through which the machine can reach S_j.

For example, assume that q_1 is stuck-at-1 in Fig. 22. Then rows r_3, r_4, r_5, and r_6 will never be activated, which will keep state transitions (A→B), (A→C), (B→C), and (B→D) from occurring. Thus, if the initial state is A, then the machine will always remain in state A, so that it ceases being an FSM.

4.22 IO's

In the SLA as described, when a clock pulse occurs, the next state and the new output are uniquely determined by the present state and the inputs. If there are faults in one or more of the output flip-flops, then these faults may cause incorrect outputs, and furthermore, some of the faults may make the outputs of the next state no longer depend only on the present state and the inputs. The fault that makes an output constant reduces the number of states. On the other hand, if a fault makes the outputs of the next state depend on more than the present state and the inputs, as will be explained below, then the number of states will be increased over the nominal number.

Theorem 1. In a Moore-machine implemented by means of an SLA, the number of states will be increased due to some faults if the machine has at least two distinct states, S_i and S_j, the outputs of which are different, and one or more of the next states (successors) of one state, S_i, is the same as one or more of the next states of the other state, S_j.

Proof of Theorem 1

For simplicity, assume that only one output flip-flop, F_z, is used in the SLA. Either $\bar{S}_z$ or $\bar{R}_z$ is nominally connected to each row for implementing a state transition, so that, when the row is activated, the proper output will be stored at the output line Q_z. Consider three rows, r_i, r_j, and r_k which implement the state transitions, respectively, $(S_i \to S_k)$, $(S_j \to S_k)$, and $(S_k \to S_n)$, and let the outputs of states S_i and S_j be respectively, 0 and 1. If the fault (due to missing devices) disconnects the three rows from the input lines $\bar{S}_z$ and $\bar{R}_z$, then the output of state S_k will be the output of the previous state S_i when row r_i is activated, but it will be the output of the previous state S_j when row r_j is activated. Thus the state S_k will have two different outputs, 0 and 1, associated with it. When the row r_k is activated, the output of state S_n will remain unchanged and so it will be the output of the previous state S_k, which has two different outputs depending on the state previous to S_k. Since the output of the next state is not uniquely determined by the present state and the inputs, one or more extra states have been effectively added. Q.E.D.

For example, consider machine M_1, which is strongly connected. Refer to Figs. 21 and 22. Since there are two states B and C satisfying the condition assumed in Theorem 1, the number of states could be increased due to some faults. If the two devices at the crosspoints between r_7 and $\bar{R}_z$ and between r_8 and $\bar{R}_z$ are missing, then the output of state D will be 1 when row r_7 is activated, but it will be 0 when row r_6 is activated, so that the output of state A when row r_8 is activated will not be uniquely determined, because the output of state A is the output of the previous state D. This is shown in the flow-table of Fig. 23. The state D has two different outputs, so that by adding one extra state, E, we can complete the flow table of the faulty machine, which is shown in Fig. 24. (This is a Mealy-machine.)

As a second example, consider machine M_2 which is not strongly connected, but satisfies the condition assumed in Theorem 1. Its flow table is shown in Fig. 25. States A and B have different outputs and the same successor D. Assuming that the row for the state-transition (A→B) is represented by r_{AB}, if all devices between $\bar{S}_z(\bar{R}_z)$ and rows r_{AD}, r_{BD}, and r_{DA} are missing, then the flow table will be changed to that shown in Figs. 26 through 28. The two different possible output values of state D will cause extra states A_1, D_1 to be added as shown in Fig. 27. The resulting Mealy machine, M_2^c, can be transformed to its Moore equivalent, M_2^o, as shown in Fig. 28.

q_1q_2		Y = 0	Y = 1
00	A	B,0	C,1
01	B	C,1	D,0
10	C	D,1	C,1
11	D	A,○	B,0

the output of the state D

Fig. 23 Faulty Version of M_1

	Y = 0	Y = 1
A	B,0	C,1
B	C,1	D,0
C	E,1	C,1
D	A,0	B,0
E	A,1	B,0

Fig. 24 Result of Fault in M_1

	Y = 0	Y = 1	Output
A	B	D	0
B	D	C	1
C	C	C	0
D	D	A	0

Fig. 25 Machine M_2

	Y = 0	Y = 1
A	B, 1	D, 0
B	D, 1	C, 0
C	C, 0	C, 0
D	D, ○	A, ○

the output of state A

the output of state B

the output of the previous entry leading to D

Fig. 26 Machine M_2 with Faults

	Y = 0	Y = 1
A	B,1	D,0
B	D_1,1	C,0
C	C,0	C,0
D	D,0	A,0
D_1	D_1,1	A_1,1
A_1	B,1	D_1,1

Fig. 27 Resulting Mealy Machine M_2^e .

	Y = 0	Y = 1	Output
A	B	D	0
B	E	C	1
C	C	C	0
D	D	A	0
E	E	F	1
F	B	E	1

Fig. 28 Resulting Moore Machine M_2^o

4.23 IST's

If there is an IST, the number of states will never be increased over the nominal number. This is so because once the state, whether correct or not, and the inputs are given, the output flip-flops will generate a unique output pattern based on that state and the inputs. For example, if the input line Y is stuck at 0 in Fig. 22, then rows r_3, r_5, r_7, and r_8 become independent of the Y-value, so that both rows r_3 and r_4 will be activated when only row r_4 should be, and similarly both r_5 and r_6 rather than only r_6 will be activated, both r_8 and r_9 instead of only r_9, and row r_7 will be activated when it should not be. The flow table will be changed to that shown in Fig. 29. Note that the number of states is unchanged.

4.3 Modification of the Circuits

As explained in Section 4.22, some faults in the output flip-flops may cause the number of states to be increased over the nominal number. This effect makes it difficult to apply conventional testing methods for sequential circuits [11], [12] to an SLA because these methods are successful in circuits where faults cannot increase the number of states. We propose here that SLA circuits be modified so that faults cannot increase the number of states. It will be shown that this can be achieved by generating the outputs by means of combinational logic circuits rather than output flip-flops.

The added combinational logic circuits consist of extra rows and cells with feedback loops that are all broken. See Fig. 31, where there is one output cell at the right. (The number of output cells will be $\left\lceil \frac{k}{2} \right\rceil$ if the number of output flip-flops that they replace in the original SLA is k.) Each extra row is connected to the proper state variables q_i^* to satisfy the outputs of all the states of the Moore machine and then each input line to the output cells is connected to one or more extra rows to generate the outputs of the machine. (See

	Y = 0	Y = 1
A	B,0	D,x
B	C,1	C,x
C	D,0	D,0
D	A,0	A,0

x : Unpredictable output

Fig. 29 Machine M_1 with Faults

q_1q_2		Y = 0	Y = 1	Outputs (Z_1Z_2)
00	A	B	C	00
01	B	C	A	10
10	C	A	D	11
11	D	D	B	01

Fig. 30 Machine M_3.

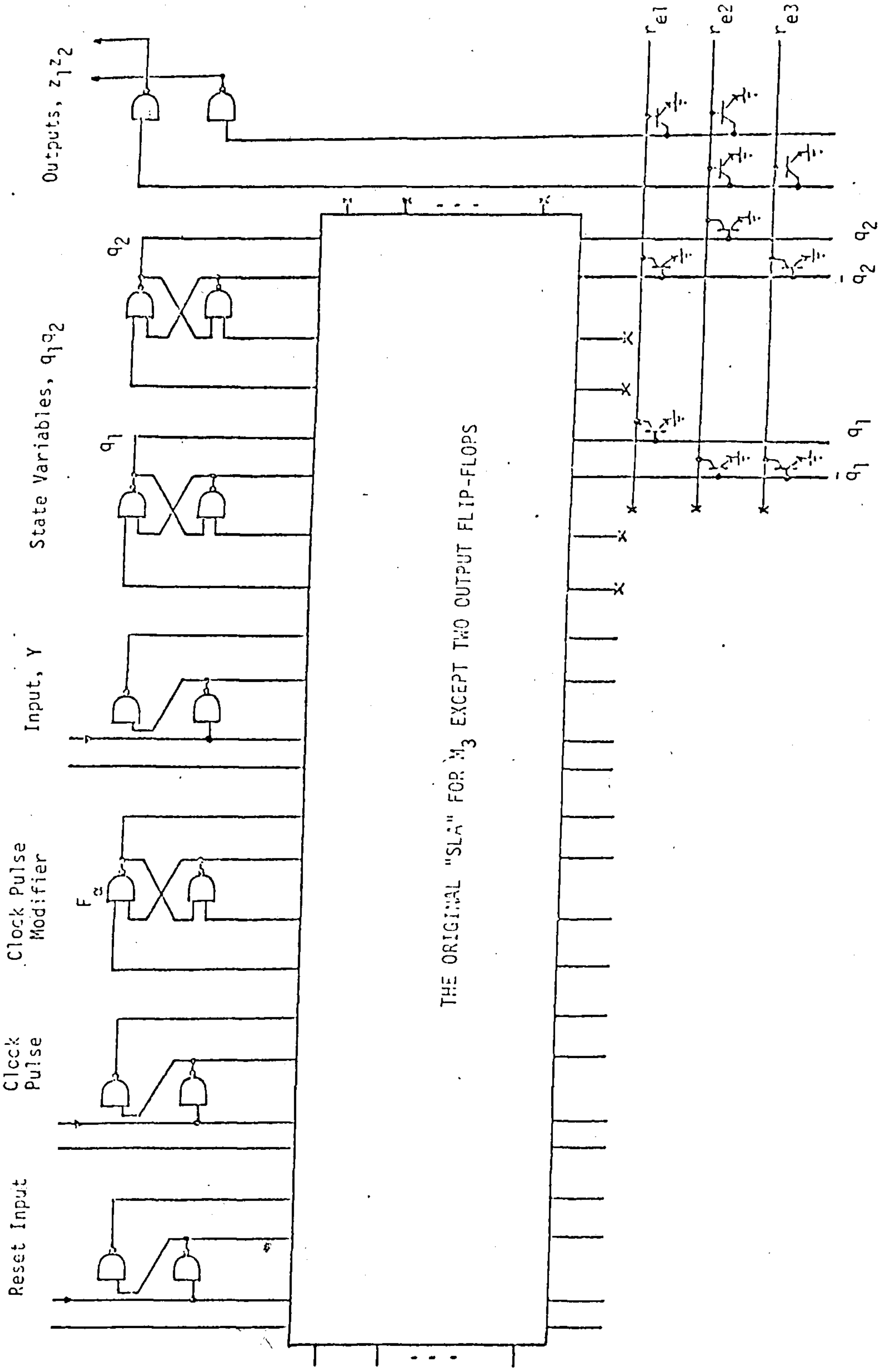


Fig. 31 Modified SLA for M_3

the example below.) It is easy to see that the structure of the combinational circuits described above is exactly that of the two-level PLA, where the inputs to the AND array are the state variables q_i^* and the outputs from the OR array are the outputs from the cells. Since no fault in the PLA can cause the number of states to be increased (i.e., a PLA is not changed to a sequential circuit due to faults as discussed in Section 2), no fault in the modified SLA will cause the number of states to be increased over the nominal number. Thus conventional testing methods can be applied to the modified SLA.

For example, consider machine M_3 with the flow table shown in Fig. 30. The combinational circuits for the outputs (Z_1Z_2) in the modified SLA consist of three extra rows (r_{e1}, r_{e2}, and r_{e3}) and one cell. The latter has all feedback loops broken, so that the two output lines from that cell will carry outputs Z_1 and Z_2. (Note that two flip-flop cells are required for two outputs Z_1 and Z_2 in the original SLA.) As shown in Fig. 31, the products realized on the extra rows r_{e1}, r_{e2}, and r_{e3} are, respectively, $P_{e1} = \bar{q}_1q_2$, $P_{e2} = q_1\bar{q}_2$, and $P_{e3} = q_1q_2$. The outputs Z_1 and Z_2 from the cell will be $Z_1 = P_{e1} + P_{e2}$ and $Z_2 = P_{e2} + P_{e3}$.

4.4 Testing Methods for Sequential Circuits

We will say that a FSM is diagnosible if it has a distinguishing sequence. Techniques for making FSM's diagnosible will first be briefly reviewed and then a novel technique, more appropriate for SLA application, will be given.

4.41 C. R. Kime's Technique [12]

Consider the flow-table of a Mealy machine. We shall say that the machine M_1 contains the machine M_2 if deleting some of M_1's columns creates M_2, which has no equivalent states. If M_2 is diagnosible, then M_1 will also be diagnosible. If M_1 does not contain such a machine, Kime suggested appending to it a single column which is an irreducible machine M_2 that has a distinguishing sequence. Adding this column by means of one extra input symbol will make any machine diagnosible. As Ref. [12] summarizes it "The column Kime adds is a "divide-by-two column." In other words, state S_i, with binary assignment b_i, maps to the state with assignment $\lfloor \frac{1}{2} b_i \rfloor$ ($\lfloor\ \rfloor$ signifies integer part of). The output for the state S_i is the rightmost bit of its state assignment (e.g., for the assignment 01, the output would be 1)."

For an example of this procedure, Ref. [12] gives the machine M_4 shown in Fig. 32. The divide-by-two column is added, resulting in the machine shown in Fig. 33.

The effect of the added column is to shift the state assignment one digit to the right and introduce a zero as the new leftmost digit. It follows that the added column has a distinguishing sequence of length k when there are k bits in the state assignment. In the four-state example, the distinguishing sequence will generate an output sequence which consists of the state variables q_2 and q_1 of each state.

4.42 R. L. Martin's Technique [12]

Martin's technique for making FSM's diagnosible is based on "feedback-shift-register (FSR) realization" of the machine. Fig. 34 shows the typical FSR circuit, where the state variable q_i of the next state in the machine is the variable q_{i-1} of the present state. If a machine is modified for FSR realization, then it has a distinguishing sequence of length k when there are k bits in the state assignment. As described in [12], "We propose adding a cycle of length 2^k column with outputs (added) so that any sequence of k inputs of this added column generates an output sequence which is the state assignment of the initial state... Further, since adding the cycle column to any SM makes it strongly connected, the rather unfortunate constraint of strongly connectedness usually assumed in diagnosing techniques can be discarded." For example, consider machine M_4 given previously in 4.41. As shown in Fig. 35, a distinguishing sequence of length 2 exists due to the cycle column under input 2, and the output sequence will be the state variables q_1 and q_2 of each state.

q_1q_2		0	1
00	A	A/0	C/0
01	B	A/0	D/1
10	C	B/1	A/1
11	D	B/1	C/0

Fig. 32 Machine M_4

q_1q_2		0	1	2
00	A	A/0	C/0	A/0
01	B	A/0	D/1	A/1
10	C	B/1	A/1	B/0
11	D	B/1	C/0	B/1

Fig. 33 Kime's Augmentation of M_4

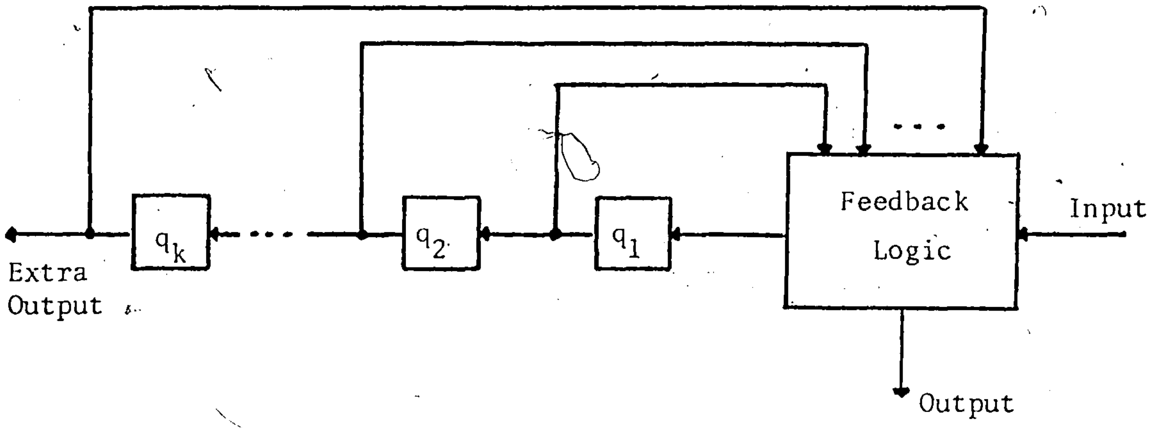


Fig. 34 FSR Realization, where "$q_k q_{k-1} \cdots q_2 q_1$" is the Binary State Assignment.

q_1q_2		0	1	2
00	A	A/0	C/0	B/0
01	B	A/0	D/1	D/0
10	C	B/1	A/1	A/1
11	D	B/1	C/0	C/1

Fig. 35 Machine M_4 Modified in Accordance with Martin's Scheme

4.5 Application of Testing Methods to the FSM in the SLA

Because Kime's technique for making FSM's diagnosible fails to make the machine strongly connected, it leads to a realization that it is not necessarily simple to check. Hence we prefer Martin's technique for achieving testability.

If the Moore machine contained in an SLA is directly modified in accordance with Martin's scheme, then in most cases it will become a Mealy machine due to the output requirements in the added column. Thus the number of states is likely to be increased over the original number, which is an undesirable result, when the modified machine (Mealy machine) is converted to the Moore equivalent for SLA-implementation.

An alternative approach for applying Martin's technique would be to apply conversion twice. First, the original machine is converted to its Mealy equivalent with a minimum number of states. Then after modification in accordance with Martin's scheme the machine is converted to its Moore equivalent. But still the modified Moore machine may have more states than the original Moore machine, as is illustrated in the following example.

Consider Moore machine M_5 with the flow table shown in Fig. 36. When this machine is converted to the Mealy machine shown in Fig. 37, the number of states is not changed. As a next step, we add the extra column under new input 2. If the modified Mealy machine of Fig. 38 is converted back to a Moore machine, then the number of states is increased by one over that of the original machine M_5. This is shown in Fig. 39.

We suggest here an alternative way of adding a column which is directly applicable to Moore machines and does not increase the number of states. Assume a machine in which m states have the output 0 and n states have the output 1. If the flow table of the machine is arranged so that states having the output 0 are placed in the first m rows and the others, having the output 1, are placed in n rows following the first m rows, we get a flow table like that shown in Fig. 40. In the

	0	1	Output
A	B	C	0
B	C	A	0
C	D	B	1
D	A	D	0

Fig. 36 Machine M_5.

	0	1
A	B/0	C/1
B	C/1	A/0
C	D/0	B/0
D	A/0	D/0

Fig. 37 Mealy Equivalent of M_5

q_1q_2		0	1	2
00	A	B/0	C/1	B/0
01	B	C/1	A/0	D/0
10	C	D/0	B/0	A/1
11	D	A/0	D/0	C/1

Fig. 38 Mealy Equivalent Modified

	0	1	2	Output
A	B	C	B	0
B	C	A	D	0
C	D	B	E	1
D	A	D	C	0
E	B	C	B	1

Fig. 39 Moore Equivalent Converted

		0	1	Output
m rows	S_1	.	.	0
		.	.	
	S_2	.	.	0
	.	.	.	.
	.	.	.	.
	.			.
	.			
	S_m			0
n rows	S_{m+1}			1
	.			.
	.			.
	.			.
	S_{m+n}			1

Fig. 40 General Form of Moore-Machine Arranged for Application of a New Testing Scheme.

	0	1	2	Output
S_1	.	.	S_2	0
S_2	.	.	S_3	0
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
	.	.		
S_m			S_{m+1}	0
S_{m+1}			S_{m+2}	1
.			.	.
.			.	.
.			.	.
S_{m+n}			S_1	1

Fig. 41 General Form of Moore-Machine Modified for Testing.

added column, the successor state to state S_i will be S_{i+1} for $i = 1,2,\ldots,m + n - 1$, and for present state S_{m+n}, the next state will be S_1, as shown in Fig. 41. The modified machine will be diagnosible because the added column has a distinguishing sequence of length m-1(n-1) if m(n) is bigger than n(m). The property of strongly connectedness is assured since all states are in a single cycle. As an example, we return to machine M_5 of Fig. 36. By adding the one column shown in Fig. 42, the machine has a distinguishing sequence of length 2, and the number of states is not changed.

	0	1	2	Output
A	B	C	B	0
B	C	A	D	0
D	A	D	C	0
C	D	B	A	1

Fig. 42 Moore Machine M_5 Directly Modified

References

[1] H. Fleisher and L.I. Maissel, "An introduction to array logic," IBM J. Res. Develop., vol. 19, pp. 98-109, March 1975.

[2] J.E. Lagne, et al., "Hardware implementation of a small system in programmable logic arrays," IBM J. Res. Develop., pp. 110-119, March 1975.

[3] E.B. Eichelberger and E. Lindbloom, "A heuristic test-pattern generator for programmable logic arrays," IBM J. Res. Develop., vol. 24, pp. 15-22, January 1980.

[4] C.W. Cha, "A testing strategy for PLA's," in 15th Design Auto. Conf. Proc., 1978, pp. 326-331.

[5] D.L. Ostapko and S.J. Hong, "Fault analysis and test generation for programmable logic arrays," IEEE Trans. Comput., vol C-28, pp. 617-627, September 1979.

[6] J.E. Smith, "Detection of faults in programmable logic arrays," IEEE Trans. Comput., vol. C-28, pp. 845-853, November 1979.

[7] Y. Kambayashi, "Logic design of programmable logic arrays," IEEE Trans. Comput., vol. C-28, pp. 609-617, September 1979.

[8] E.L. Muehldorf and T.W. Williams, "Optimized stuck fault test pattern generation for PLA macros," in Dig. Semiconductor Test Symp., Cherry Hill, N.J., October 25-27, 1977, pp. 88-101, IEEE catalog no. 77ch-12f-7c.

[9] D.L. Greer, "An associative logic matrix," IEEE J. Solid-State Circuits, vol. SC-11, pp. 679-691, October 1976.

[10] S.S. Patil and T.A. Welch, "A programmable logic approach for VSLI," IEEE Trans. Comput., vol. C-28, pp. 594-601, September, 1979.

[11] C.R. Kime, A Failure Detection Method of Sequential Circuits, Department of Electrical Engineering, University of Iowa Technical Report, 66-130 (January 1966).

[12] R.L. Martin, Studies in Feedback-Shift-Register Synthesis of Sequential Machines, Research Monograph No. 50, The M.I.T. Press, Cambridge, Mass.

[13] F.C. Hennie, Finite-State Models for Logical Machines, John Wiley & Sons, Inc., New York, 1968.

[14] Z. Kohavi, Switching and Finite Automata Theory, Second Edition, McGraw-Hill Book Co., New York 1970.