M.Tech (Computer Science) Dissertation Series

General False Path Problem In Timing Analysis of Combinational Circuits

a dissertation submitted in partial fulfilment of the requirements for the M. Tech (Computer Science) degree of the Indian Statistical Institute

[redacted]

By

J. V. S. Mani

under the supervision of

Prof. Bhargab B. Bhattacharya, Electronics Unit

INDIAN STATISTICAL INSTITUTE
203, Barrackpore Trunk Road
Calcutta – 700 035

2nd August, 1997

Certificate of Approval

This is to certify that the dissertation titled **General False Path Problem In Timing Analysis of Combinational Circuits** *submitted by* **Mr. J. V. S. Mani** *to the* **Electronics Unit, Indian Statistical Institute, Calcutta,** *is a bonafide record of the work and investigation carried out by him under my supervision and guidance.*

Date: 2nd August, 1997
Indian Statistical Institute
Calcutta

Bhargab B. Bhattacharya 2-8-97

(Bhargab B. Bhattacharya)

Professor,
Electronics Unit
Indian Statistical Institute
Calcutta

Contents

GENERAL FALSE PATH PROBLEM IN TIMING ANALYSIS OF COMBINATIONAL CIRCUITS

The False Path Problem or the problem of detecting the paths which do not contribute to the in combinational circuits has been discussed here. A precise definition of the False Path has been attempted. Some of our observations on the False Paths have been recorded here. The general false path problem is to detect whether a given path (not necessarily the longest one) is a false path. An efficient algorithm for solving the general false path problem has been proposed here. This algorithm can be also extended to generate all the possible *sensitizable paths* with the delays greater than threshold T. Scope for related future work exists in demonstrating the efficiency and effectiveness of the proposed algorithm experimentally.

INTRODUCTION

Timing verification can be performed by simulation or by timing analysis. Simulation is performed by a simulator which generates the output signals of each component and compute the delay of each component according to some delay model and the input signals to the component. Due to the huge computing time required by the simulator, the timing analysis, which takes an input vector-independent approach, is preferred. The timing analysis ignores the operating conditions and functionalities of the components in the design.

Based on the connection information and the delay models of components, an acyclic graph is constructed to model the design. The vertices and the edges of the graph represent the components and the connection between the components in the design respectively. The weight associated with a vertex (an edge) is the delay of the corresponding component (the medium delay of the corresponding connection).

The delay of a path is represented by the sum of the weights of all vertices and edges involved. Timing analysis is to check the delays of all paths in the graph and to report the paths which violate some timing constraints. Therefore, path selection algorithms which report long paths (referred to as the *critical paths* and short paths are crucial to the timing analysis approach.

But due to the ignoring of operating conditions and functionalities of components in the design, a path being reported by timing analysis may be a false path. A false path is a path which can never be activated by any input vector. Reporting false paths to the designer provides no useful information for the designer to correct the timing violations. Hence, the timing verifier (the task which implements the timing analysis approach) can detect the *long false paths*(a *false path* with a delay greater than a certain threshold T) and report only the *long sensitizable paths*(a *long path* which is not a *false path*) to the designer.

In order to avoid reporting the *false paths* to the designer, some of the previously proposed approaches have tried to use input-vector independent approaches which report the *false paths* to the basic *timing analysis algorithm*. However, these algorithms report either a superset or a subset of the *actual false path* set (We shall discuss this later). Also, these algorithms cannot perform the *timing analysis* also simultaneously.

The major objective of this work is to design and develop efficient and effective algorithms which report all the possible long sensitizable paths and can also be extended to perform the *timing analysis* simultaneously. A *false path* is simply a path which is never activated by any input vector. The problem of identifying whether a path (may not be the longest one) is a false path will be referred to as the *general false path problem*.

PREVIOUS WORK

Most of the previously proposed methods for detecting a *false path* are based on the D-Algorithm approach. This algorithm takes an input-vector independent approach. But when the stability of the signals are taken into account, a path identified to be a false path by the D-Algorithm approach may not necessarily be a false path [YEN89].

REVIEW OF PREVIOUS WORK DONE RELATED TO FALSE PATHS

In Figure 1, we consider the path P = B-1-D-2-E-3-X_1-4-Y. In order to allow a signal go through path P from the primary input B to the primary output Y, we need to set all the other gates feeding to gates along P to be *non-control values.* The *non-control value* for the various gates are listed under Table 1. The formal definition of *Control Value, Non-Control Value* may be found in [YEN89]. In this example, we will set C_1=1, A=0 and C_2=0 respectively. Then the D-Algrithm approach is used to propagate these setting values and induce new setting values from these setting values. If any signal becomes *inconsistent*, we claim this path is a *false path.* A signal set to more than one values is considered to be *inconsistent.* Since C_1 and C_2 in this example belong to the same signal C, the setting values of C_1=1 and C_2=0 will cause C to be inconsistent. Thus, the path P is claimed to be a *false path.*

This means that the path P is not *sensitized* by any input vector. Let us now take the stability of signals into account and see if the path P still remains *unsensitized* by all possible input vector combinations. Consider that all the inputs are stabilised initially (at 0 ns). Assume gate delay of all gates in the circuit to be 1 ns and the signal delay to be 0 ns. Let us enumerate the various *path systems* to the output Y from different inputs, along with their path delays as follows:-

(i) A-3-X_2-Y 2 ns

(ii) B-1-D-2-E-3-X_2-4-Y 4 ns (Path P)

(iii) C-C_1-2-E-3-X_2-4-Y 3 ns

(iv) C-C_2-4-Y 1 ns

The Input Vector B=0,1, C=0 sensitizes path sysem (i) such that the Output Y gets stabilized to Y=1 at T=2 ns when A=1 and sensitizes path system (iii) such that the Output Y gets stabilized to Y=0 at T=3 ns when A=0. The path system (iv) gets sensitized whenever C is set to 1. Thus, we see that none of the input vectors sensitizes path system (ii) which represents Path P. Hence P is a *false path.* This is of extreme importance in *timing analysis*, as we find that although path P can cause the longest delay in the circuit, since it is not sensitized at all, the delay due to it can be ignored completely and hence, the worst case delay in the circuit is 3 ns and not 4 ns. This is an example of an *actual false path.* We shall refer to this example in many of our future discussions later in this paper.

Whenever a path is detected to be a *false path* by the D-Algorithm approach, we need to further apply the signal stability criterion to finally confirm that the path is indeed *false.* In the above example, it is merely a coincidence that the D-Algorithm approach has been able to detect the *actual false path* alone accurately. But this is not always the case. We shall now consider the example in Figure 2, in which we shall show that the paths detected to be *false paths* by the D-Algorithm approach are not the *actual false paths* (to be defined later).

As before, all the gate delays are 1 ns, signal delays are 0 ns and the inputs are stabilised at 0 ns. Consider the two paths $P_1 = A_1$-1-D-2-F-3-G and $P_2 = B_1$-1-D-2-F-3-G in Figure 2. Both P_1 and P_2 are the longest paths with the same delay 3 ns. In order to allow the signals to propagate along path P_1 from A_1 to G according to the D-Algorithm approach, we set signals $B_1 = 1$, C = 0 and E = 1 respectively. Then we use the D-Algorithm approach to induce new setting

values of signals from these signals and check to see whether any signal became inconsistent. Since $B_1 = 1 = B = B_2$, we set B_2 to be 1. And because gate 4 is a NOR gate and one of the input signals B_2 to gate 4 has value 1, we can induce that the output E of gate 4 must have value 0. Because the initial setting value of E is 1, the new setting value of E = 0 makes E become *inconsistent*. Thus, path P_1 is a *false path*. By a similar argument, path P_2 in Figure 2 is also a *false path*.

Because the two longest paths in Figure 2 are both *false paths*, they will not contribute their 3 ns delay to the output G. The worst case delay of G should be 2 ns. But when the input vector (A = 0, B = 0, C = 1) is applied, we can get D = 0 and E = 1 at 1 ns, F = 1 at 2 ns and G = 1 at 3 ns. Since E = 1 which is stabilized at 1 ns is a non-control value to gate 3, the output G of gate 3 cannot be decided until F is stabilized and has its logical value. Since F is stabilized at 2 ns, G can only be stabilized at 3 ns. This condition shows that the worst delay of this circuit is 3 ns. But using the D-Algorithm approach, we claim both P_1 and P_2 are *false paths*. This contradicts the result given by the D-Algorithm approach. Hence, the two paths P_1 and P_2 are not *actual false paths* although the D-Algorithm approach says that both of them are *false paths*.

Also, we see in the above example that both the paths P_1 and P_2 are getting equally and simultaneously sensitized, when the input vector is A=0, B=0 and C=1, and the Primary Output G gets stabilized at T=3 ns. We shall call such paths (two or more in number) which simultaneously get sensitized on the application of one or more input vectors and also contribute the same delay to the circuit all the time (the delays will be the same for all the input vectors in case more than one such input vector exist) as *Equally and Simultaneously Sensitizable Paths*. We shall refer to such special paths later on in our literature.

If in a situation, two or more paths apparently seem to get simultaneously sensitized, yet do not contribute the same delay to the circuit, then we regard only that path which contributes the minimum delay to the circuit as the *Singly Sensitizable Path*. In case, a set of such paths that equally contribute the same minimum delay to the circuit, we still call all the paths in that set of paths as

Equally and Simultaneously Sensitizable Paths. On the other hand, among the remaining paths, which contribute a greater delay to the circuit than that contributed by the *Singly Sensitizable Path* or the *Equally and Simultaneously Sensitizable Path Set*, if there exist one or more paths that are not *sensitizable* by any other input vector, all such paths are also *Actual False Paths* while otherwise, all such paths are also *Sensitizable Paths.*

There is still another possibility, in which case, we may have two *Sensitizable Paths*, which contribute the same delay to the circuit, but not simultaneously, i.e., only on the application of different input vectors. We shall call all such paths (two or more in number), contributing the same delay to the circuit as an *Only Equally Sensitizable Path Set.* There can be several such *Equally Sensitizable Path Sets* in a given circuit, the path delay of each such set being unique.

FALSE_PATH_CHECKING ALGORITHM PROPOSED IN [YEN89]

The Algorithm *false_path_checking(P,false_path)* given in **[YEN89]** also takes an input-vector independent D-Algorithm like approach, but does not guarantee whether all the paths other than those detected to be *false paths* are indeed *sensitizable.*

We shall first try to discuss the **General False Path Problem** and our observations on the *actual false paths.* Thereafter, we shall discuss the method that we have adopted to represent Combinational Circuits and the relevant data structures. Later we shall present an input vector independent graph theoretic approach for the detection of *false paths* and discuss the properties of the *associated graph* equivalent of a given combinational circuit and its *dual.*

We shall also discuss the similar approaches made so far, in dealing with the false path problem and their shortcomings.

Before going ahead with our algorithm, we shall first review a few concepts that have already been discussed in **[YEN89]**.

Theorem 1: A path P = $s_0,g_0,s_1,g_1,s_2,g_2,\cdots,g_{k-1},s_k$, is a sensitizable path if and only if there exist at least one input vector I such that all the signals s_i, $0 \leq i \leq k$, along path P satisfies the following two conditions:-

(1) Early-arrive-signal$(s_i, I, T_I) \cap \mathbf{C}(g_i, I) = \phi$, **and 2**

(2) If Late-arrive-signal$(s_i, I, T_I) \neq \phi$, $s_i \in \mathbf{C}(g_i, I)$,

where signal s_i is an input signal to gate g_i)

A path which is not a sensitizable path is a *false path.* This is the definition of the *false path* that we shall adhere to in our discussion on this subject.

GENERAL FALSE PATH PROBLEM

The general *false path* problem can be formulated as follows:

For a given combinational circuit C, let **I** denote the primary inputs set, **O** denote the primary outputs set and P = s_0,g_0, s_1,g_1,...,s_{k-1},g_{k-1},s_k be a path. The general *false path* problem is to detect whether some path P is a *false path.*

Below, we give some of the key terms with their definitions, which we shall use in our descriptions and algorithms.

The **Stable Time of Signals**, **Control Value** and **Non-Control Value** have already been explained in [**YEN89**].

We define *OFF-PATHS* to be the paths, the signals along which are set to the Non-Control values of the respective gates to which they are input while trying to verify if a given path is *false* or not by the D-Algorithm approach.

We define *ACTUAL FALSE PATHS* as the paths which are not only *false paths* by the D-Algorithm, but can be proved to be *false paths* by **Theorem 1** also.

ASSUMPTION:

For simplicity, we assume here that the propagation delay of every *Component*

(Gate) is 1 ns.

SOME OBSERVATIONS ON ACTUAL FALSE PATHS:

1. If a sub-path of a path is false, the entire path by itself becomes false. For, if there does not exist any input vector that can sensitize a portion of the path, it is obvious that the entire path containing it cannot be sensitized at all.

But the Converse is not necessarily true. i.e., a sub-path of a false path need not necessarily be a false path.

A false-path is specific to a primary input-primary output pair. A portion of a false path from the same primary input to a different primary output, which lies on the original false-path itself, can be sensitizable, i.e., this path need not necessarily be another false path. Hence, a sub-path of a false path need not necessarily be a false path.

This can be shown by referring back to the example in **Figure 1**.

In figure 1, the path B-1-D-2-E-3-X_2-4-Y is a false path, with respect to the Primary Output Y. But the sub-path B-1-D-2-E-3-X_1 of the same path B-1-D-2-E-3-X_2-4-Y is not a false path, with respect to the Primary Output X_1. This is shown by the observation that when A=0, C=1, the path B-1-D-2-E-3-X_1 gets sensitized, and the Output X_1 gets stabilized to 0 or 1 respectively as the Primary Input B is set to 0 or 1.

2. The *false path* is a path from a primary input to a primary output. Even if the minimum sum of products form expression of a primary output signal does not contain one or more primary inputs, and if there exists a path between the corresponding primary input- primary output pair, still this path need not necessarily be an false path. The Example is shown in Figure 3.

In this example, **Y = A + C**. Also note that this circuit is logically equivalent to the circuit shown in Figure 1. Let us consider the various path systems in this circuit in the same way as we had earlier done to the circuit shown in Figure 1. The different path systems in this circuit are as follows:-

(i) A-3-X_2-Y 2 ns

(ii) B-2-E-3-X_2-4-Y 3 ns

(iii) C-C_1-1-D-2-E-3-X_2-4-Y 4 ns

(iv) C-C_2-4-Y 1 ns

The Input Vector B=0,1, C=0 sensitizes path sysem (i) such that the Output Y gets stabilized to Y=1 at T=2 ns when A=1. Also, A=0, B=1, C=0 sensitizes path system (ii) such that the Output Y gets stabilized to Y=0 at T=3 ns. Similarly, A=0, B=0, C=0 sensitizes path system (iii) such that the Output Y gets stabilized to Y=0 at T=4 ns. Also, the path system (iv) gets sensitized whenever C is set to 1. Thus, we see that all the 4 path systems here, are sensitizable.

The *KEY OBSERVATION* here is that although the minimum sum of products form expression of Y (Y = A + C) does not contain the primary input B, still the path system B-2-E-3-X_2-4-Y is *sensitizable*, meaning it is not a *false path*.

Also, even if the minimum sum of products form expression of a primary output contains all the primary inputs, still all the paths from the primary inputs to that primary output need not necessarily be sensitizable.

This is clear from the example in Figure 4. Figure 4 is, in fact, a slightly modified circuit of the circuit shown in Figure 3. The modification is that there is a direct input signal line from the Primary Input B to the gate 4, whose output is the Primary Output Y. Hence, the minimum sum of products form expression of Y becomes **Y = A + B + C**. Let us now consider the various path systems in this circuit in the same way as we had earlier done to the circuits shown in Figures

1 and 3. The different path systems in this circuit are enumerated as follows:-

(i) A-3-X_2-Y 2 ns

(ii) B-B_2-2-E-3-X_2-4-Y 3 ns (Path P)

(iii) C-C_1-1-D-2-E-3-X_2-4-Y 4 ns

(iv) C-C_2-4-Y 1 ns

(v) B-B_1-4-Y 1 ns

The Input Vector B=1, C=0, A=1,0 sensitizes path sysem (v) such that the Output Y gets stabilized to Y=1 at T=1 ns. Similarly, C=1, B=0, A=0,1 sensitizes path system (iv) such that the Output Y gets stabilized to Y=1 at T=1 ns. Also, A=1, B=0, C=0 sensitizes path system (i) such that the Output Y gets stabilized to Y=1 at T=2 ns. Also, C=1, B=1, A=0,1 sensitizes both the path systems (iv) and (v) equally such that the Output Y gets stabilized to Y=1 at T=1 ns. The last mentioned is another classic example of a special case in which two or more path systems simulataneously get sensitized equally on the application of atleast one input vector (The earlier case was in shown in Figure 2, in which, both the path systems P_1 and P_2 get sensitized equally and simultaneously on the application of the input vector A=0, B=0, C=1, resulting in the Primary Output G getting stabilized at T=3 ns). Also, A=0, B=0, C=0 sensitizes path system (iv) such that the Output Y gets stabilized to Y=0 at 4 ns.

Thus, we find that, the path system (ii) in Figure 4 is not sensitized by any input vector. Hence, the path P, (B-B_2-2-E-3-X_2-4-Y) is a **false path**.

Thus, we see that although B appears in the minimum sum of products form expression of Y, the path P from B to Y via path system (ii) is a *false path*. Hence, our observation.

3. D-Algorithm detects all those paths as possible false paths whose corresponding off-paths contain atleast one signal such that this signal is an input signal to two gates with different non-control values lying on those paths.

Theorem 2 : A path which is not a *false path* by the D-Algorithm can never be a *false path* by **Theorem 1** also. The proof is as follows:-

Proof:

The D-Algorithm approach does not take the *stableness of signals* into account. Hence, the D-Algorithm approach assumes that all the signals are stabilized earlier than the signals along the path that we are considering [YEN 89]. In other words, as per the definitions given in YEN89, all the Signals along all the off-paths are Early-Arrive signals(s_i,I,T_I), where, s_i is an *input signal to gate* g_i *along the path considered*, I is an *input vector assignment* and T_I is the *stable time of input vector I.*

Since we set all the signals along the off-paths to the respective non-control values of the gates to which they are input, the first condition under **Theorem 1** gets satisfied.

Again, since all the signals along all the off-paths are Early-arrive-signals(s_i, I, T_i), Late-arrive-signals(s_i, I, T_i) = ϕ. Hence, we need not necessarily set s_i to C(g_i, I), where C(g_i, I) is the control value of the gate g_i.

Hence, both the conditions in **Theorem 1** are taken care of by the D-Algorithm approach, although it does not take the stable time of signals into account.

Conclusion:

Thus, we may conclude that the *ACTUAL FALSE PATH SET* is a **PROPER SUBSET** of the *FALSE PATH SET* detected by the D-Algorithm Approach. Also, the Algorithm given in [**YEN89**] detects a **SUBSET** of the *ACTUAL FALSE PATH SET.*

We have given a diagrammatic illustration of our above conclusion in Figure

5.

4. We define *FALSE SUB-PATH* as the path which contains the signal input to the gate, while an inconsistency occurs in any of the remaining signals to that gate along an off-path.

FALSE SUB-PATH is specific in relation to a primary output, i.e., a path which is a *FALSE SUB-PATH* with respect to one of the primary outputs need not necessarily be false with respect to any of the remaining primary outputs.

The above statement is a direct corollary of our Observation 1. A *false sub-path* is certainly a sub-path of a *false path* and a sub-path of a *false path* need not necessarily be false. Hence, a *false sub-path* is also primary-output specific. In other words, A path which is a *false sub-path* with respect to one Primary Output need not be a *false sub-path* with respect to another Primary Output which shoots out from a sub-path of the original *false path*. This can be shown by referring back to the example in **Figure 1**.

In figure 1, the path B-1-D is a *false sub-path*, with respect to the Primary Output Y, since the path B-1-D-2-E-3-X_2-4-Y is a *false path*. But the same path B-1-D is not a *false sub-path*, with respect to the Primary Output X_1, since the path B-1-D-2-E-3-X_1 is not a *false path*. Hence, our Observation.

5. We define *FALSE SUB-PORTION* as that portion of the circuit containing the *false sub-path*, the gate to which it gets input and the remaining input signals to that gate. Similar to the *FALSE PATH* and the *FALSE SUB-PATH*, the *FALSE SUB-PORTION* is also primary-output specific.

According to the D-Algorithm approach, the *false sub-paths* are **independent** of the hardware-changes that may be incorporated in the *false sub-portions* without affecting the rest of the circuit in any way, i.e., irrespective of the hardware changes made in the false sub-portion of the circuit, without affecting the logical design of the rest of the circuit, the corresponding *false path* still remains unsensitizable.

The above fact is illustrated by the example shown in Figures 1 and 3. It is easy to verify that the circuit shown in Figure 3 is logically equivalent to the circuit shown in Figure 1. The modification has been made only in the *false sub-portion* of the circuit in Figure 1 to get the circuit in Figure 3. There is a minor change in the delay that could be caused by the path from B to Y. While in the Circuit 1 shown in Figure 1, the maximum delay of 4 ns could be caused due to the path from B to Y and the next maximum delay of 3 ns could be caused due to the path C-C_1-2-E-3-X_2-4-Y, provided such paths can be sensitized, in the Circuit 3 shown in Figure 3, the maximum delay of 4 ns could be caused due to the path C-C_1-1-D-2-E-3-X_2-4-Y and the next maximum delay of 3 ns could be caused due to the path from B to Y, provided such paths can be sensitized.

We have already shown that the path from B to Y in circuit 1 is a false path by the D-Algorithm approach. On exactly similar lines, the corresponding path from B to Y in circuit 3 still remains a false path.

However, while the path from B to Y in circuit 1 is an actual false path, its corresponding path in circuit 3 is not an actual false path. We have already shown this in our observation 2, by taking the stable time of signals into account. Also, in this case, no new actual false paths have been introduced in the circuit. Hence, this is a typical example of a situation in which the actual false paths get eliminated by suitable hardware changes in the false sub-portion of a circuit, without affecting the output logic. It is possible to eliminate the actual false paths by changing the overall design and not merely concentrating only on the false sub-portion in the circuit. An example of eliminating the false paths in circuit 1 this way has been shown in circuit 6. It should, however, be noted that such hardware changes are most likely to cause the elimination of the *intermediate signals*, keeping intact only the primary inputs and the primary outputs.

6. When we consider the elimination of *false paths*, we should also determine whether they really serve any purpose from the *timing analysis* point of view. For example, the worst case delay of 4 ns in circuit 1 is caused by the path from B to Y, which is an *actual false path*. Hence the actual delay caused by circuit 1

is 3 ns and not 4 ns, while the actual delay in circuit 3 is 4 ns since all the paths in circuit 3 are sensitizable. Hence, even though the two circuits are logically equivalent, performance-wise, circuit 1 is preferred to circuit 3. However, circuit 1 has an *actual false path*, while circuit 3 is devoid of any false path.

Hence, while designing a circuit, a designer is not necessarily working with the risk of introducing *false paths* in the circuit. But, however, introducing false paths complicates the work of the *timing analyzer*. Timing analyzer either needs efficient false path detection algorithms in order to eliminate the *long false paths* from its analysis or else spends a huge amount of time wastefully, analyzing all the *false paths* as well. Experimental results of running existing false path detection algorithms on Bench Mark Circuits have revealed that as the circuit complexity increases, the number of long false paths also increases [YEN89]. In [YEN89], the authors have reported that on running their false path detection algorithm with the timing analyzer, they discovered that more than 60even when the *threshold value* of the *timing analyzer* T was set at around 90of the time delay contributed to that circuit by its *longest sensitizable path*. Also, the same authors have mentioned that they do not report all the *actual false paths*. Hence, some of the paths not reported as *false paths* by their algorithm may still be false. Hence, the total number of *actual false paths* may be still larger. Hence, it is a real trade-off for the designer, as whether to design faster circuits containing greater number of *false paths* or else, refrain from introducing *false paths* in the circuit at the designing stage itself.

This observation tells us that by designing more efficient *false path detection algorithms*, we free the designer from the fear of introducing a greater number of *false paths*. However, a more useful work will be to evolve a *design methodology*, that prevents the introduction of *false paths* during the *design phase* itself.

GRAPH-THEORETIC APPROACH TO THE FALSE PATH PROBLEM

In all the papers discussing the *false path problem* and the timing analysis approach to find all the sensitizable paths responsible for the delay beyond a maximum threshold T in a given combinational circuit, a suitable graphical representation of a digital circuit has not been discussed. Here, we present a graphical representation of an *acyclic combinational circuit*

ASSUMPTION: For simplicity, we assume here that the propagation delay of every *Component (Gate)* is unity.

GRAPHICAL REPRESENTATION OF COMBINATIONAL CIRCUITS

A suitable *data-structure* has been evolved to represent the acycic combinational circuits.Basically, a combinational circuit consists of three different *elements*, namely, *Primary Inputs*, *Components* and *Primay Outputs*, represented by *vertices* and the interconnections between them, which are represented by *edges* of the corresponding graph
G=(V,E). From this basic graph, which wholly represents the given combinational circuit, we algorithmically construct the associated graph and its dual, which form the basis for our analysis of *false paths*.

BASIC GRAPH:

DATA STRUCTURE OF AN ELEMENT:

All the three *elements*: *Primary Inputs*, *Components* and *Primary Outputs* are represented by *nodes*, the structure of which contains the following fields:-

Identification No., Static Level No., Dynamic Level No., Type, Val, Status Register, Predecessor List and Successor List.

Identification No. uniquely identifies the Primary Inputs, Components and Primary Outputs. Option may be provided to let the dynamic assignment of the *Identification No.* by the running algorithm, when the circuit is input by the user, or the user himself may assign the *Identification No.* to all the *elements* input by him.

The **Predecessor List** of a given element X is the list of all those *elements* whose outputs serve as inputs to X. Clearly, the **Predecessor List** of any *Primary Input* is empty. The **Predecessor List** of any *Component* may have one or more elements while the **Predecessor List** of any *Primary Output* contains exactly one *element*.

Associated Data Structure: In the actual implementation, the structure representing element X points to the root of a *Linked List* formed from the *predecessor elements* by using their **Identification nos.** as the *key value*. The nodes of this *Linked List* point to the corresponding *elements* that are the *predecessors* of X.

The **Successor List** of a given element X is the list of all those *elements* whose inputs are outputs from X. Clearly, the **Successor List** of a *Primary Output* is empty. The **Successor List** of any *Component* and any *Primary Input* is always non-empty and may have one or more *elements*.

Associated Data Structure: In the actual implementation, the structure representing element X points to the root of a *Linked List* formed from the suc-

cessor elements by using their **Identification nos.** as the *key value*. The nodes of this *Linked List* point to the corresponding *elements* that are the *successors* of *X*.

The **Static Level** of all *primary inputs* is taken to be 0 while the **Static Level** of any other *element* in the circuit under static conditions, i.e, **when the** *primary inputs* in the circuit have not been fed their *logical values* is one more than the *maximum* of the *levels* of all the *elements* in its **Predecessor List.**

The **Dynamic Level** of all *primary inputs* is taken to be 0 while the **Dynamic Level** of an element *X* (*Primary Output* or *Component*), (under dynamic conditions) is one more than the minimum of the dynamic levels of those elements in its Predecessor List, that have the Control Value of *X* and in case none of the elements in its Predecessor List contains the Control Value, the **Dynamic Level of the element** *X* is one more than the maximum of the dynamic levels of the elements in its Predecessor List. Thus, the dynamic level is a dynamically changing variable and it depends on the logical values to which the *primary inputs* have been set.

The **Dynamic Level** of an *element* directly corresponds to the time at which the *signal* coming out from it becomes *stable*, on the application of an input vector, assuming that all the *primary inputs* are **stabilized** at $\mathbf{t = 0}$.

Type defines whether the *element* is a *Primary Input*, *Primary Output* or one of the following *Components*: **NOT, OR, AND, NOR, NAND, XOR** and **XNOR** by a unique type no.

Value is the Logical Value of the output from a Component. In the case of Primary Inputs, Val is the Logical Value input by the user and in the case of Primary Outputs, Val is the Logical Value of the Primary Output for the given input vector.

Status is used to determine whether the Value has been determined or not for the given input vector.

The addresses of all the elements are stored in an *AVL Tree* by using their identification nos as the key value. Using this *AVL Tree*, any element can be accessed by fetching its address. This also facilitates elementary operations such as addition, deletion and modification of any element in the circuit.

The above Data Structure Description completely specifies the combinational circuit that has been input.

Based on this data structure, algorithms have been designed to prepare a net list, determine the fan-in and fan-out, static level and dynamic level (depending upon the input vector) of the various elements, evaluate the outputs at the various components and the primary outputs for a given input vector, enumerate the various paths from a given primary input to a given primary output and construct the associated graph with respect to each of the primary outputs and their corresponding duals for the purpose of analysis of the false paths and the sensitizable paths.

Given a combinational circuit, its associated graph (or its dual associated graph) with respect to a primary output is constructed as follows:-

1. The Primary Output is the Start Node of the graph.

2. Set the Node to this Start Node.

3. Initialize 3 Queues: *Leaf Node Queue*, *Terminal Node Queue* and *Daughter Node Queue*

4. The Set_Val(Start Node) is arbitrary.
If it is '1' for the associated graph, it is '0' for its dual associated graph and if it is '0' for the associated graph, it is '1' for its dual associated graph.

5. Call Const_assgraph(Node,Set_Val(Node)).

6. End.

Procedure Const_assgraph(Node,Set_Val(Node))

begin

If Node = Primary Input, Add Primary Input to *Leaf Node Queue* and stop.
Else
begin
Set Node to parent node.
If Set_Val(parent node) = control value of its
predecessor,
begin
(i) Fork at this node such that it has as many
daughter nodes as the number of inputs to its predecessor.
Each of the daughter nodes corresponds to one input
to the parent node. Add all the daughter nodes to the
Daughter Node Queue.
(ii) Set the Set_Val(Node) of each of the daughter nodes to
the value that will set the Set_Val(parent node) to
the control value of the parent node.

While the *Daughter Node Queue* $\neq \phi$,
begin
(a) Visit the first daughter node.
(b) Set it to Temp.
(c) Call Const_assgraph(Temp,Set_Val(Temp)).

(d) Dequeue Temp from the Daughter Node Queue.

end while

Else,

begin

(i) Add all its inputs to the *Daughter Node Queue.*

(ii) Set the parent node to previous node.

While any of its inputs in *Daughter Node Queue* ϵ

Primary Input Set,

begin

(a) Connect this in series with the previous node.

(b) Set the value of Set_Val of this node to the value

that is necessary to set the Set_Val(parent node) to

the non-control value of the parent node.

(c) Set this input node to previous node.

(d) If no. of nodes in Daughter Node Queue is 1,

add this to *Leaf Node Queue.*

(e) Dequeue the input from the *Daughter Node Queue.*

end While

While the *Daughter Node Queue* $\neq \phi$,

begin

(a) Visit the first node in *Daughter Node Queue.*

(b) Set it to Temp.

(c) Connect this in series with all the nodes in the

Leaf Node Queue.

(d) Set the value of Set_Val of this node to the value

that is necessary to2set the Set_Val(parent node) to

the non-control value of the parent node.

(e) Call Const_assgraph(Temp,Set_Val(Temp)).

(d) If no. of nodes in Daughter Node Queue is 1,

add this to *Leaf Node Queue.*

(e) Dequeue the input from the *Daughter Node Queue.*

end While

end

end Add all the leaf nodes to the *Terminal Node Queue.* end **Procedure**

Our approach has been to determine the static level of all the elements in the circuit, which would enable us to test the static sensitization conditions by the D-Algorithm Approach. The static level of an element in a circuit does not change, irrespective of the input vector. The static level finding algorithm has been described as follows:-

```
    Procedure LEVEL (element)


    begin
if type(element) = primary input
level(element) = 0;
    else
begin
for(pres:=pred(element);pres;pres:=next(pres))
LEVEL(pres);
if level(element) < level(pres)+delay(element)
level(element) = level(pres)+delay(element);
endfor;
end;
endif
end.
```

But in our algorithm, we make use of dynamic sensitization. This calls for the determination of the dynamic level of an element in the circuit. As already explained, the dynamic level of an element is dependent upon the earliest stable input signal having its control value and if none of its input signals has its control

value, then its dynamic level is dependent upon the last stable input signal having its non-control value. Hence, in the case of dynamic sensitization, we consider an input vector and dynamic level of the same component may vary with the input vector. This is the essential difference between the static level and the dynamic level of an element in a circuit. The dynamic level finding algorithm has been described as follows:-

```
    Procedure DLEVEL (element)


    begin
if type(element) = primary input
dlevel(element) = 0;
else
begin
flag:=0;
for(pres:=pred(element);pres;pres:=next(pres))
if(val(pres)=NCV(element))
begin
if(flag=0)
begin
DLEVEL(pres);
if(dlevel(element) < dlevel(pres)+delay(element))
dlevel(element) = dlevel(pres)+delay(element);
end;
endif;
end;
else
begin
DLEVEL(pres);
if(dlevel(element) > dlevel(pres)+delay(element))
begin
```

```
dlevel(element) = dlevel(pres)+delay(element);
flag=1;
end
else
begin
if(flag=0)
begin
dlevel(element) = dlevel(pres)+delay(element);
flag=1;
end;
endif;
end;
endif;
end;
endif;
endfor;
end;
endif
end.
```

SOME OBSERVATIONS ON ASSOCIATED GRAPHS AND THEIR CORRESPONDING DUALS

1. Each one of the graphs can be directly constructed from the other(We shall outline this algorithm later), provided there is no impossible path in either of the graphs.

2. The *logical values* to which the various signals are set in one *graph* are exactly the *complements* of the *respective logical values* in its corresponding *dual graph*

3. By studying the different path systems, given an input vector, we can always find the logical value of the output.

4. The graph can be converted into a weighted graph for our analysis, where weights on the edges of the graph correspond to the delay of the component between the two signals, as in the associated graph and its dual, an edge represents a component while the vertices represent the signals along the path, in which the component lies. Weights along the path from a primary input to the primary output correspond to the delay along the path, if the path gets activated by the corresponding input vector. This enables us to find the sensitizable paths that contribute the minimum delay to the circuit.

CONCLUSION

An efficient algorithm to detect all or most of the false paths in a circuit has to be designed based on the associated graph models. Since we have tried to incorporate the sensitization condition mentioned in [YEN89], it is hoped that our proposed model performs atleast as good as the *false_path_checking* algorithm presented in [YEN89].

BIBLIOGRAPHY

[YEN89] David H.C.Du, Steve H.C.Yen, S.Ghanta. "On the General False Path Problem in Timing Analysis" Proceedings of DAC, 555-560, 1989.

[BRAY89] Patrick C. McGeer, Robert K. Brayton. "Efficient Algorithms for Computing the Longest Viable Path in a Combinational Network" Proceedings of DAC, 561-567, 1989.

[PERR89] S.Perremans, L.Claesen, H. De Man. "Static Timing Analysis of Dynamically Sensitizable Paths" Proceedings of DAC, 568-573, 1989.

Appendix

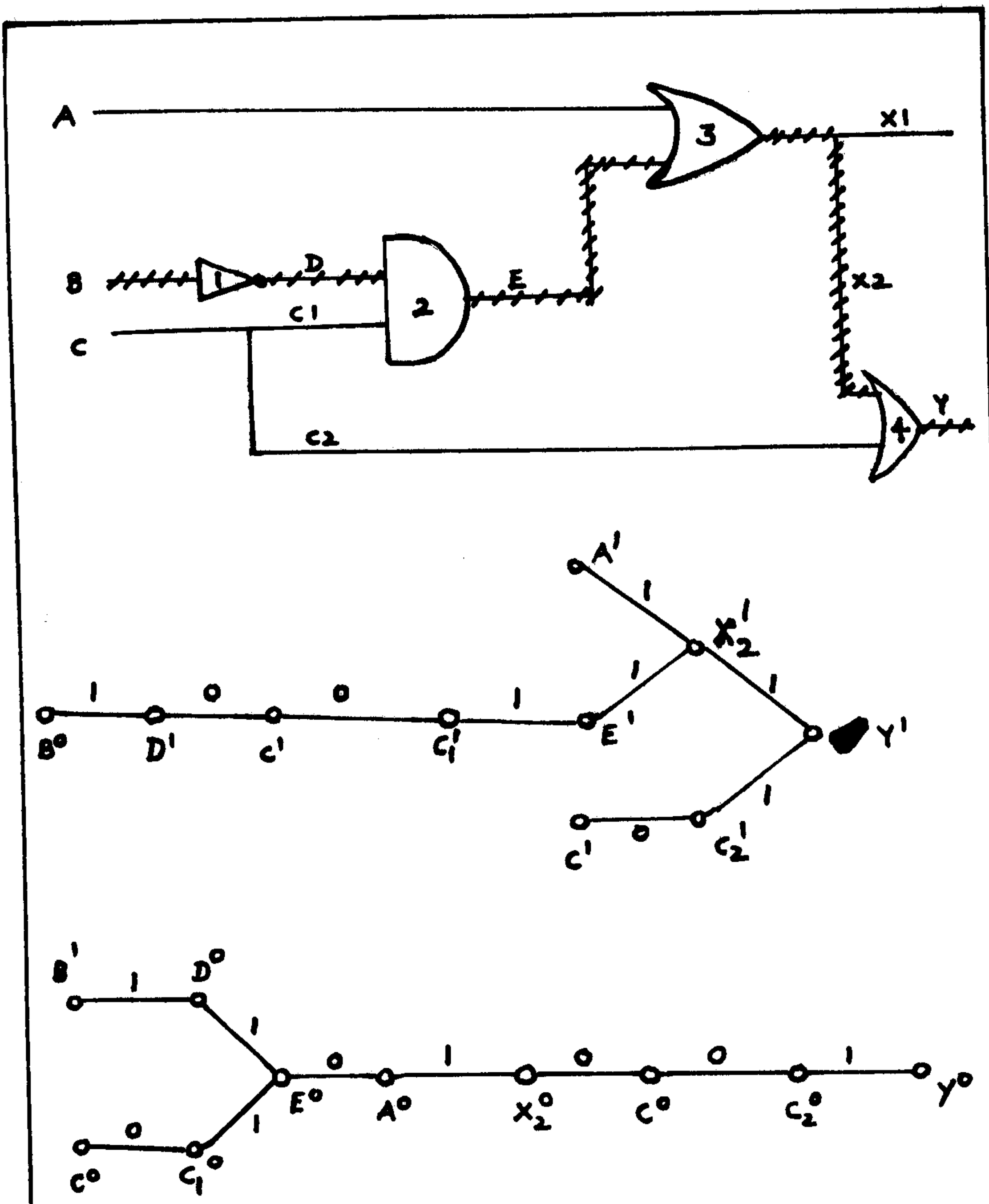

FIGURE 1. CIRCUIT 1 WITH ASSOCIATED GRAPH & ITS DUAL

~~////~~ SHOWS AN ACTUAL FALSE PATH

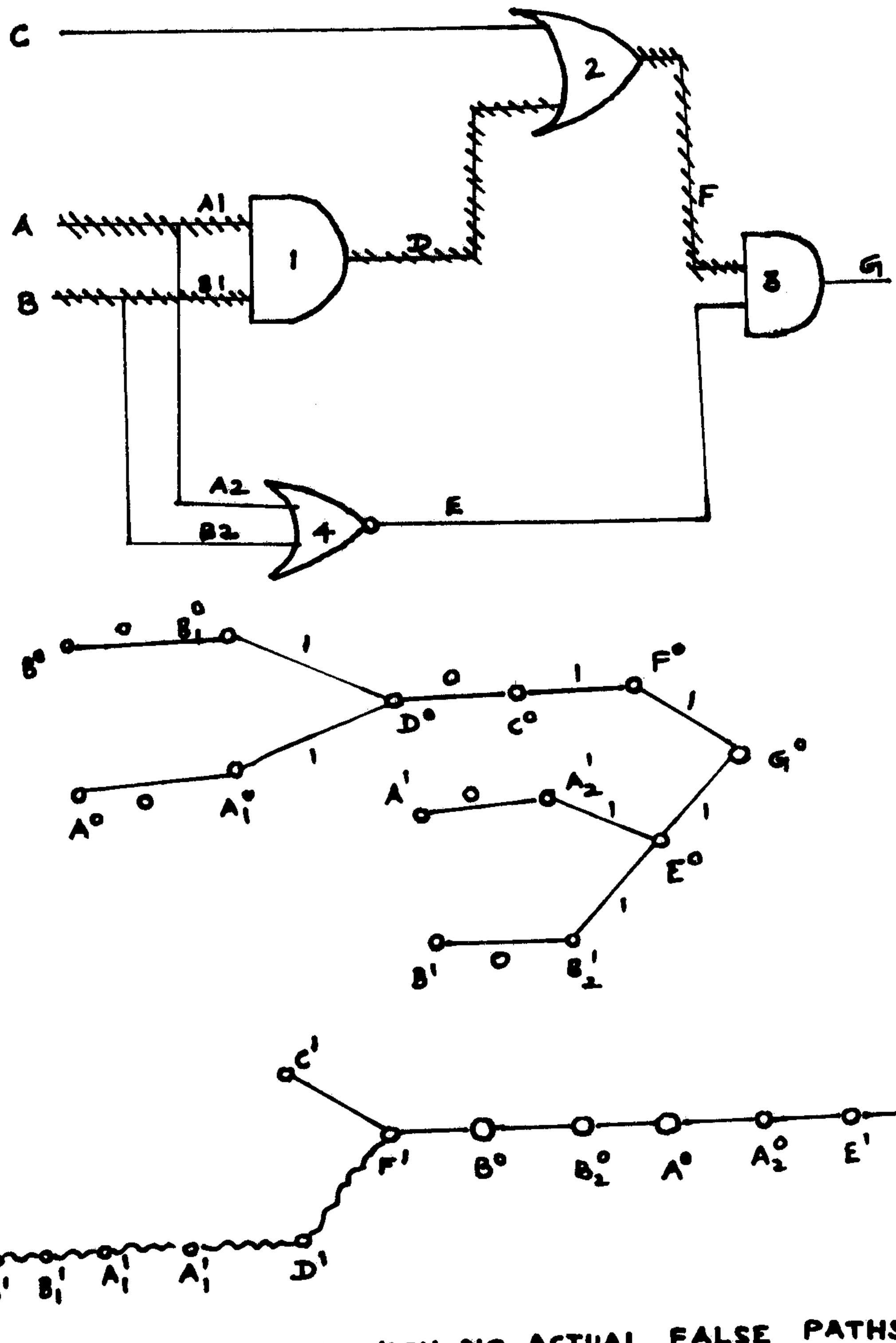

FIGURE 2 : CIRCUIT WITH NO ACTUAL FALSE PATHS

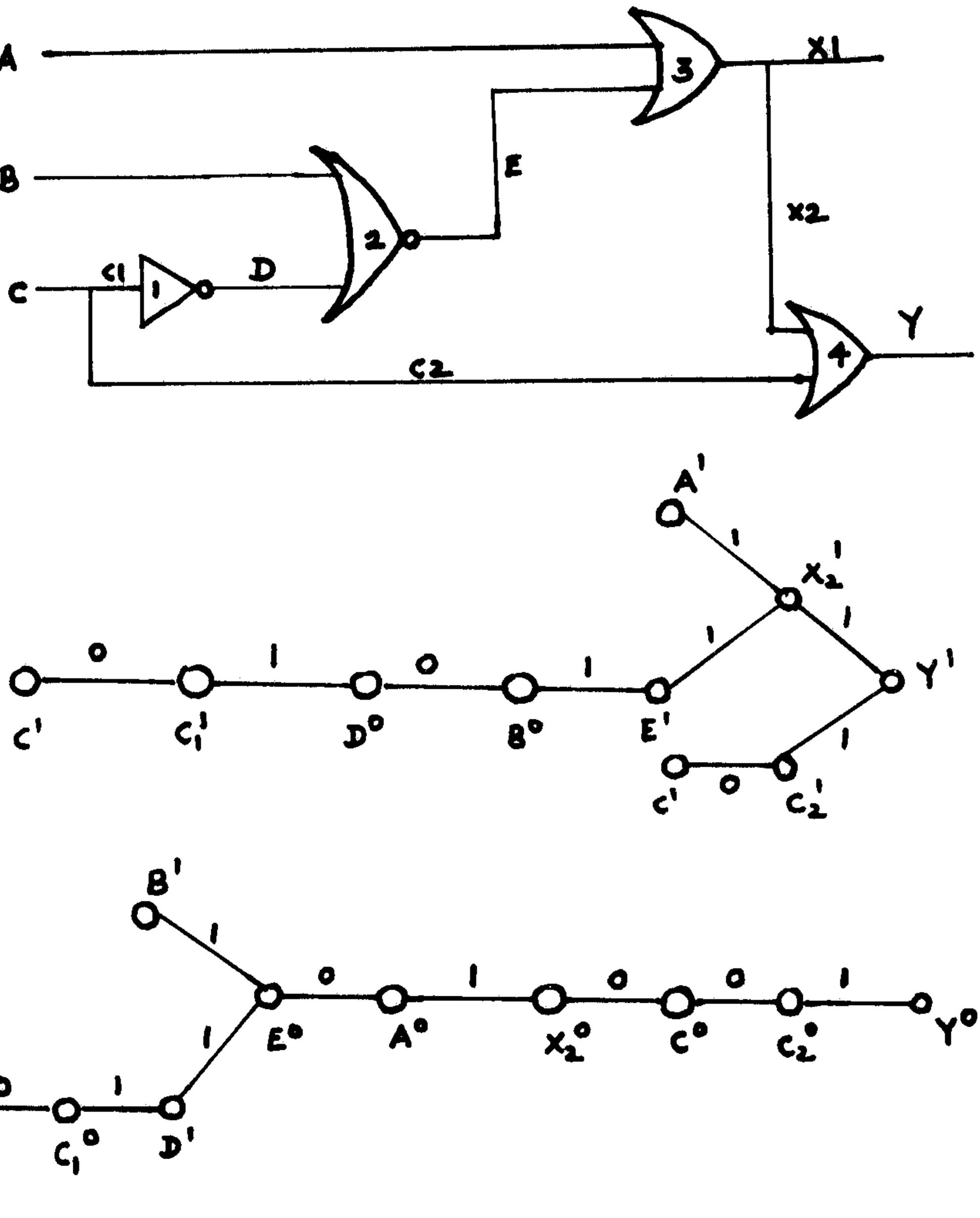

FIGURE 3 : CIRCUIT 3 WITH NO ACTUAL FALSE PATHS, LOGICALLY EQUIVALENT TO CIRCUIT 1, BUT CONTRIBUTES GREATER DELAY

BELOW GIVEN ARE ASSOCIATED GRAPH WITH DUAL

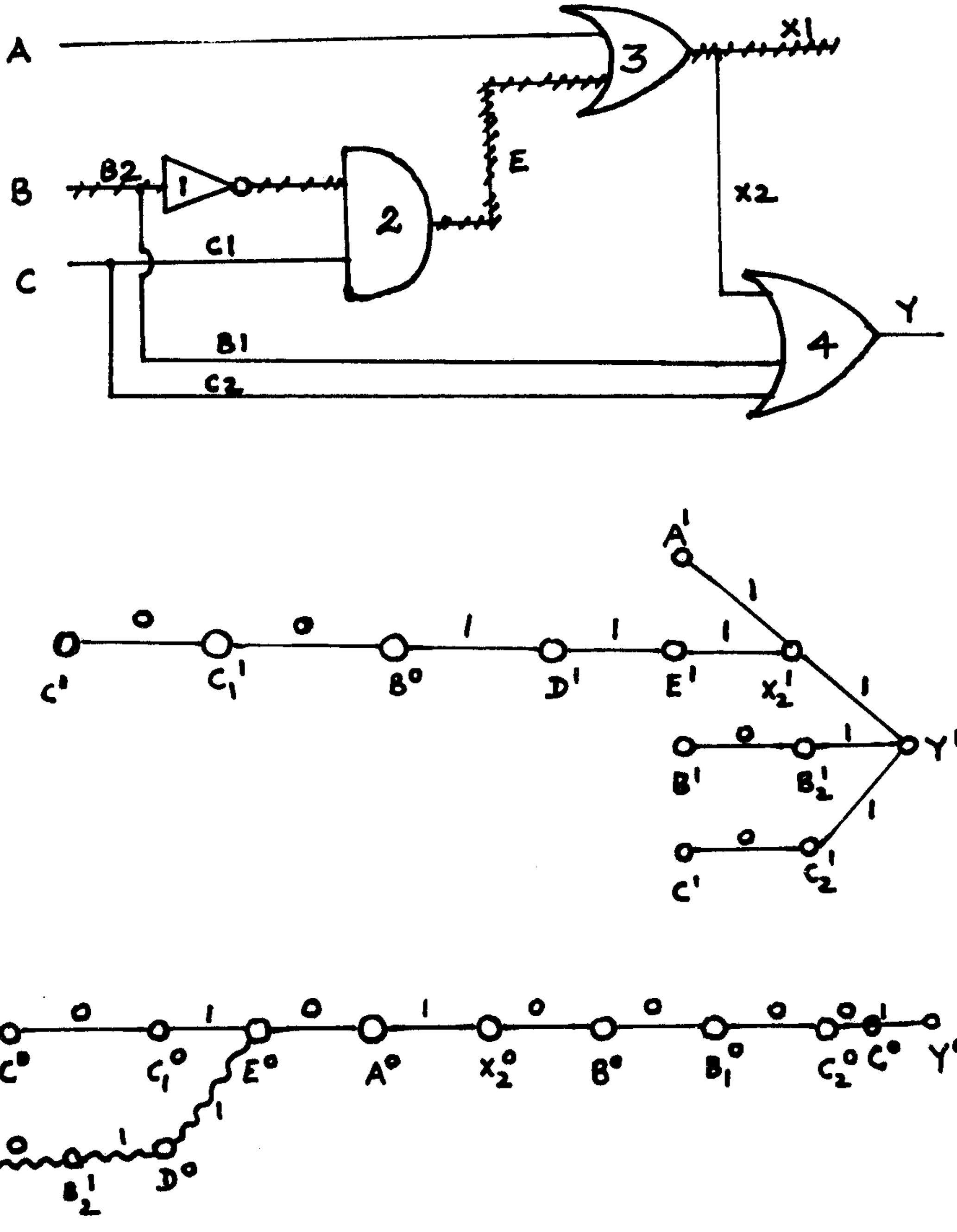

FIGURE 4 SHOWS CIRCUIT 4 WITH AN ACTUAL FALSE PATH FROM B TO Y ALTHOUGH B APPEARS IN THE MINIMUM SUM OF PRODUCTS EXPRESSION OF Y = A+B+C

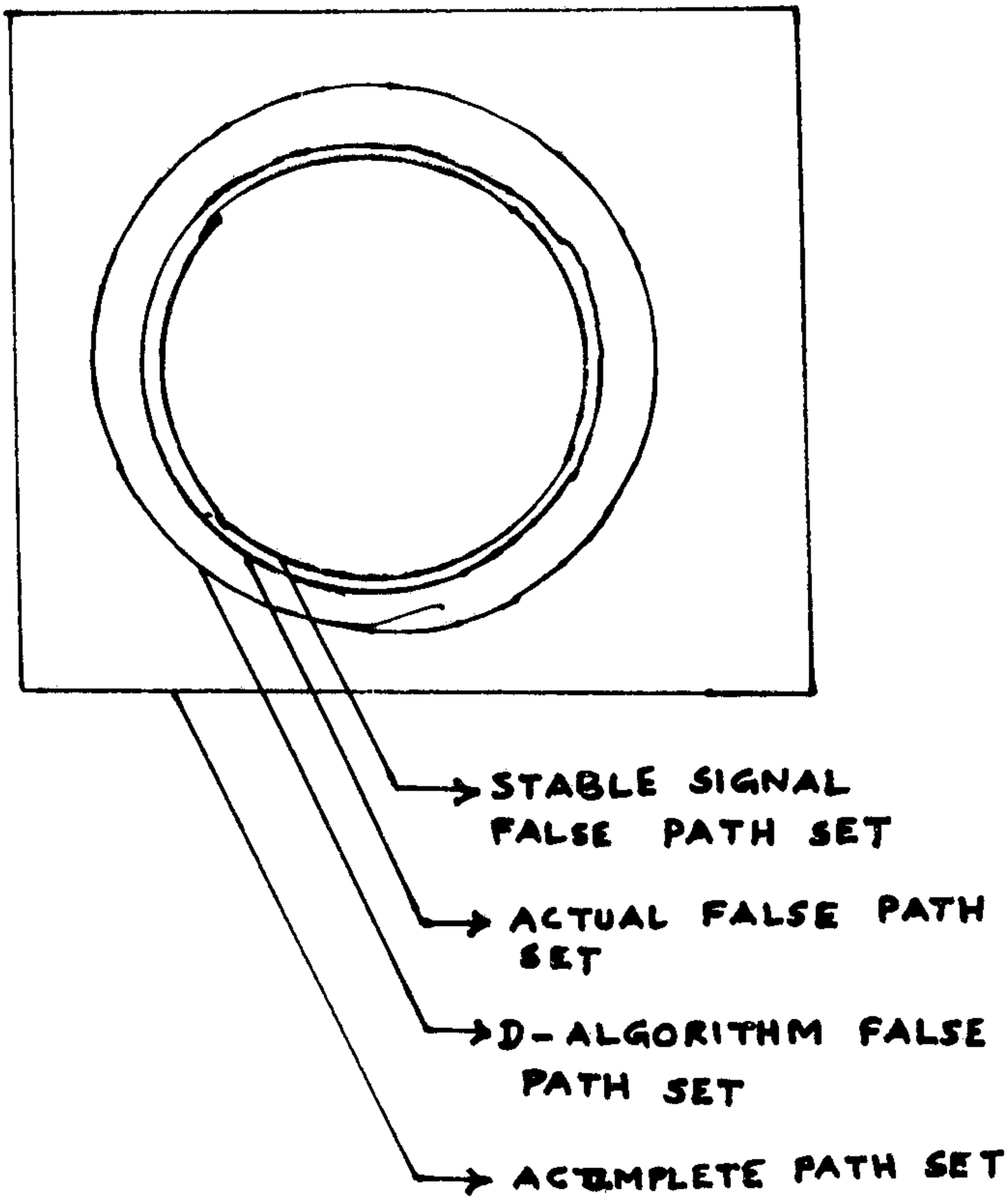

FIGURE 5 SHOWS AN ILLUSTRATION OF FALSE PATH SETS DETERMINED BY DIFFERENT APPROACHES

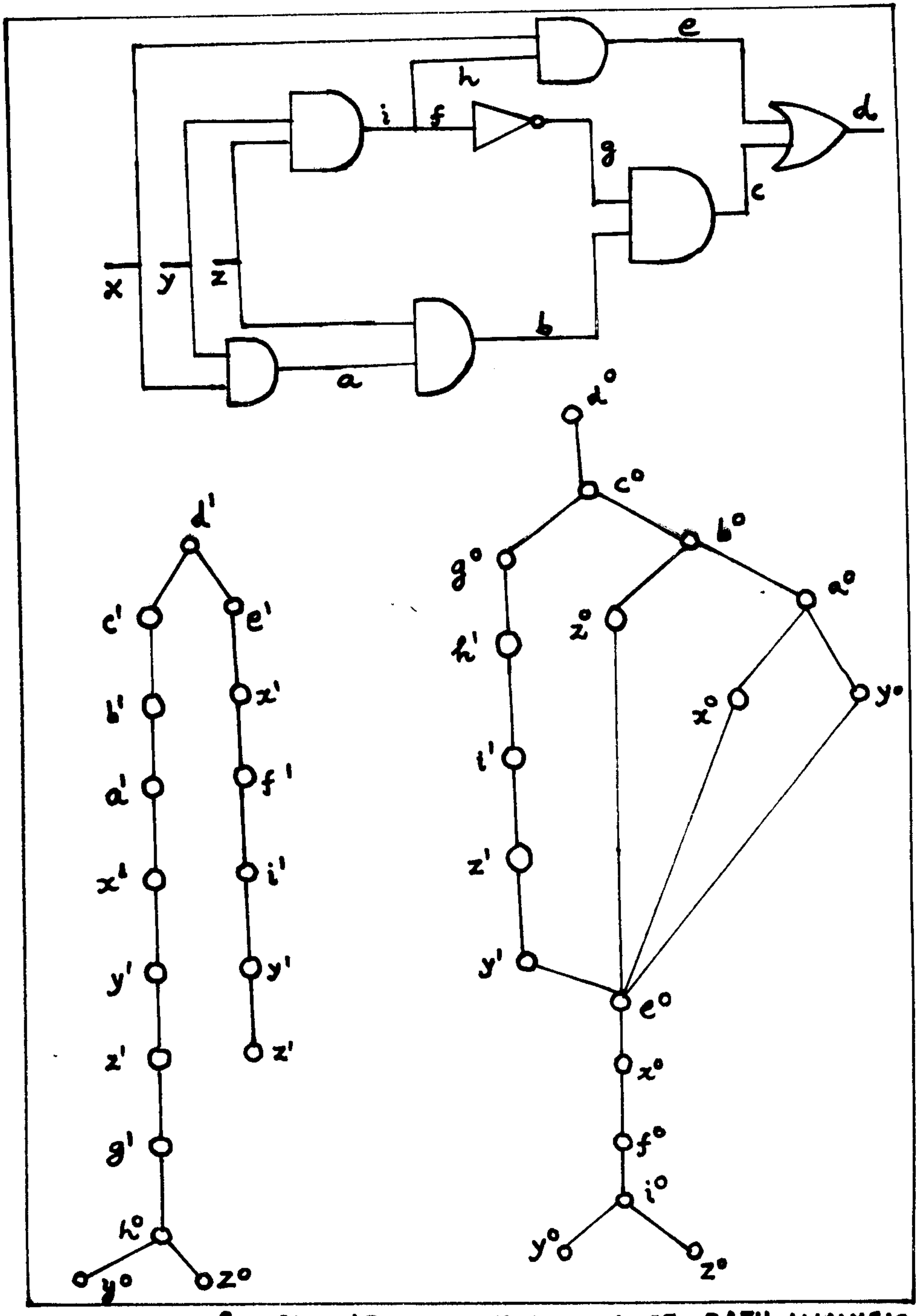

FIGURE 6 SHOWS COMPLICATION OF PATH ANALYSIS

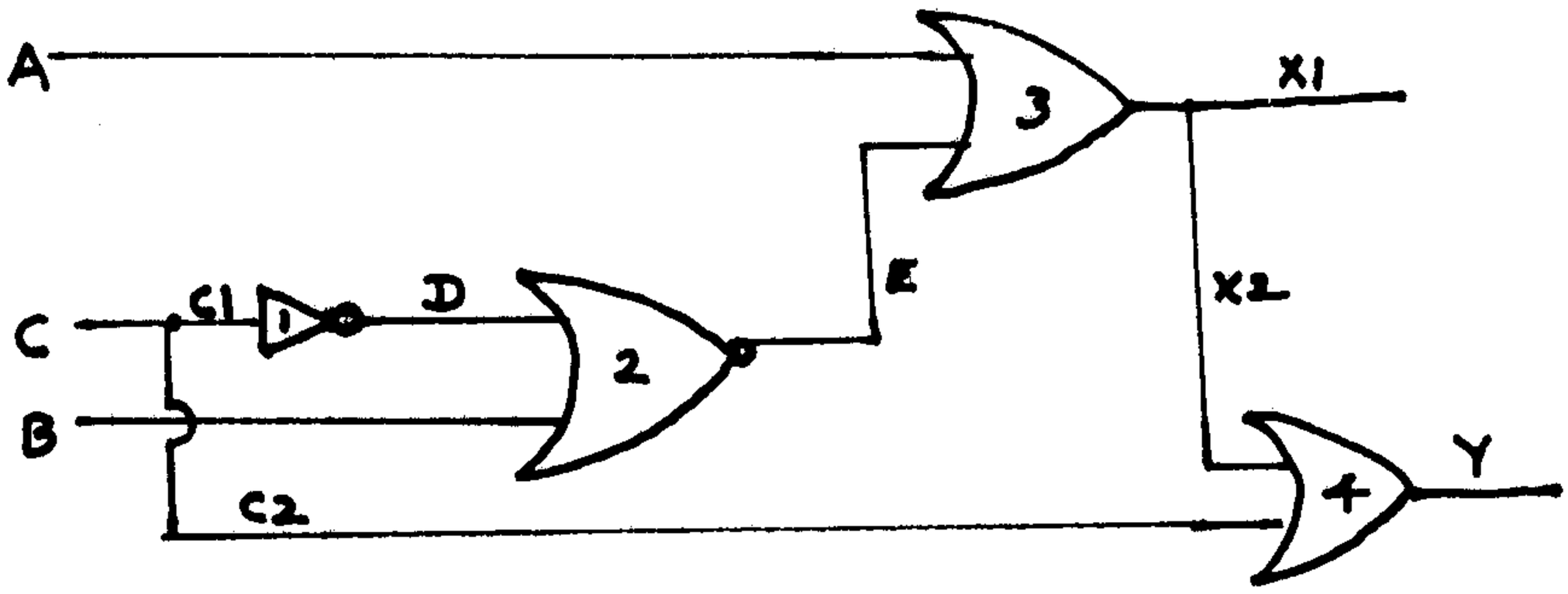

FIGURE 7 (a). HARDWARE CHANGE IN FIG. 1 ELIMINATES FALSE PATH BUT RESULTS IN INCREASED WORST CASE DELAY

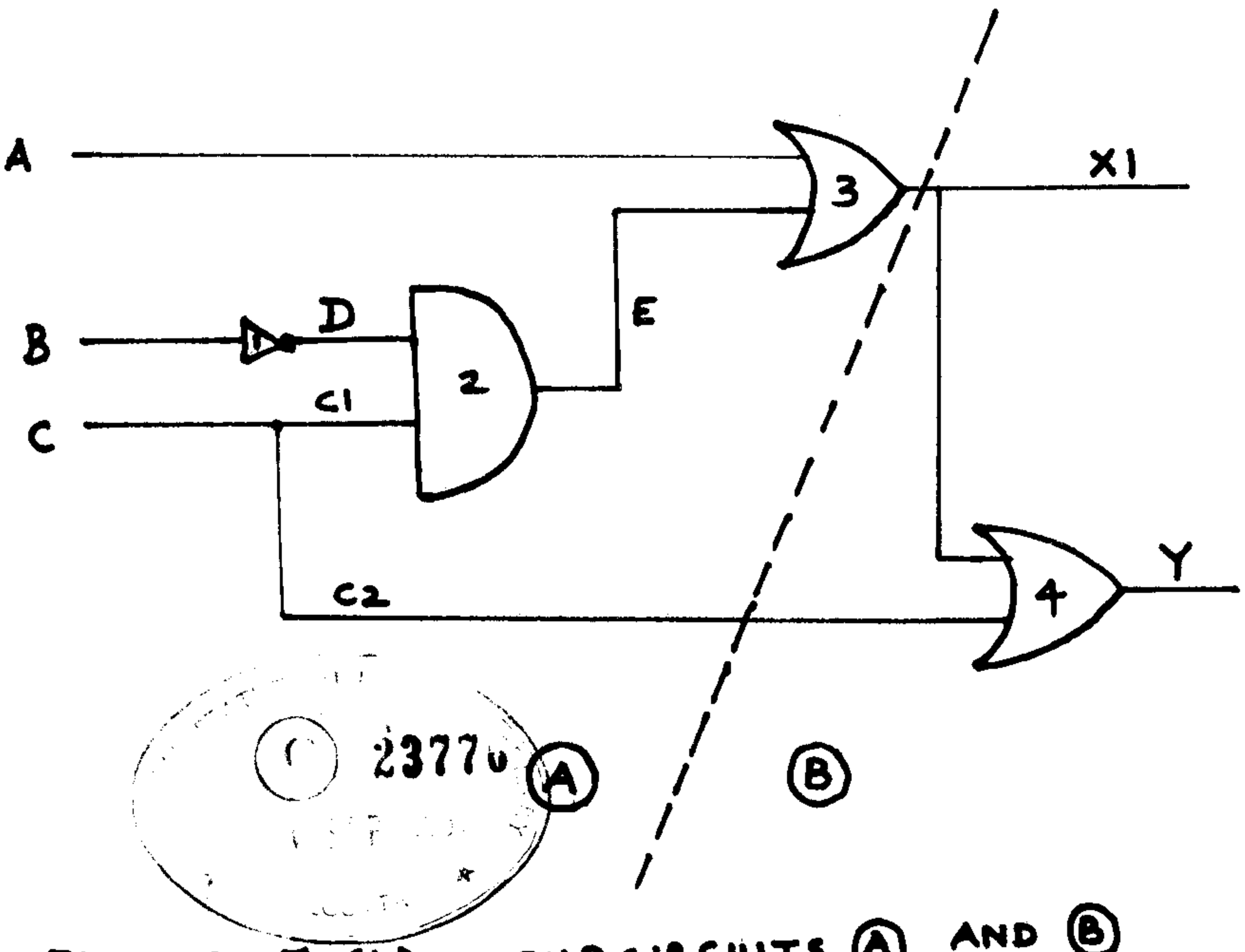

FIGURE 7 (b). SUBCIRCUITS (A) AND (B) DO NOT CONTAIN ANY FALSE PATH THOUGH CIRCUIT CONTAINS FALSE PATH : INEFFECTIVE PARTITIONING