E. B. Eichelberger

Hazard Detection in Combinational and Sequential Switching Circuits*

Abstract: This paper is concerned with a unified approach to the detection of hazards in both combinational and sequential circuits through the use of ternary algebra. First, hazards in a combinational network resulting from the simultaneous changing of two or more inputs are discussed. A technique is described that will detect hazards resulting from both single- and multiple-input changes. The various types of hazards connected with gate-type sequential circuits are also discussed, and a general technique is described that will detect any type of hazard or race condition that could result in an incorrect terminal state. This technique could be easily implemented in a computer program which would be capable of detecting hazards in circuits containing hundreds of logic blocks.

Introduction

Whenever the input signals of a combinational or sequential switching circuit are changed, the output signals are predicted by the truth table or flow table to behave in a certain manner. If it is possible for the output signals to behave in a different manner than predicted, the circuit is said to contain a *hazard* for that input transition. The object of this paper is to present a new technique for determining whether or not a switching circuit contains a hazard.

This technique is unique in that:

1. It can be used to evaluate transitions involving the simultaneous changing of two or more input signals.
2. The same technique can be used for both combinational and sequential switching circuits.
3. It can be easily implemented in a computer program which would be capable of analyzing logic networks containing more than 2,000 logic gates.

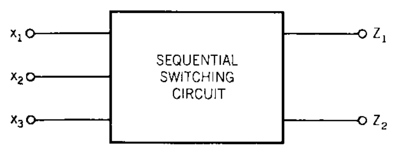
The most significant application of this technique is in evaluating the response of large sequential switching circuits to given input sequences (see Fig. 1), taking into account any malfunctioning due to hazards, races, or oscillations.

Techniques for detecting and eliminating hazards in combinational networks have been described by Huffman[1] and McCluskey.[2] These techniques make use of Boolean algebra and are restricted to hazards resulting from single input variable changes. It has been shown by other authors[3,4] that a suitable ternary algebra can also be used to solve this problem.

In this paper the problem of detecting and eliminating hazards in combinational networks is extended to hazards resulting from the simultaneous changing of two or more input signals. It is shown that two types of hazards can be associated with multiple input changes. The first type, called a *logic hazard*, is similar to and includes static hazards. The second type, called a *function hazard*, is inherent in the Boolean function and cannot be eliminated by modifying the logic network.

A procedure is described for obtaining a ternary function from a binary switching circuit, and some basic properties of this ternary function are defined. It is shown that both logic hazards and function hazards can be detected from the ternary function. These techniques are then applied to

Figure 1 Typical sequential switching circuit illustrating the key problem: to determine the circuit response to a given input sequence, accounting for "hazards," "races," and oscillations.



	INPUT SEQUENCE			
x_1	0	1	0	0
x_2	0	1	0	1
x_3	0	0	1	1

* Presented at the 5th Annual Symposium on Switching Circuit Theory and Logical Design, Princeton University, November 8-14, 1964.

the detection of hazards in sequential switching circuits. A general procedure is given for determining whether a particular input change to a sequential circuit can result in an indeterminant final state.

Assumptions

Although the techniques described here are applicable to contact networks as well as gate-type networks, only the gate-type networks are considered in detail.

It is assumed that each logic gate (whose single output is a binary function of its n inputs) can be approximated by an ideal (no delay) logic gate which has delay elements associated with each input. It is further assumed that each of these input delay elements can have any value between zero and some maximum, d_{max}.

It should be noted that these input delay elements could possibly have different values for every transition; we assume only that the delay will never exceed the upper bound d_{max}.

Multiple input-change hazards

In this section we consider the problem of determining whether or not a combinational (loop-free) network can have a spurious hazard-pulse on its output when two or more inputs are changed at the same time. We are concerned only with hazards for which the output before the input change is equal to the output after the input change and such that during the input change a spurious output pulse may occur. (The problem of determining "dynamic" hazards[1,2] is not considered, since it will be seen that they may be ignored in determining hazards in sequential circuits.)

Since the term *static hazard* has been defined in terms of single-input changes,[4] it is not used for multiple-input changes even though the output character of the hazard is the same. Instead, the term *M-hazard* is used.

Definition 1. A combinational logic network contains an M-hazard for an input change involving the changing of one or more input variables if and only if (1) the output before the change is equal to the output after the change and (2) during the change a spurious pulse may appear on the output.

It should be noted that it is possible for a network to contain an M-hazard for an input change and not generate a spurious output pulse for that input change. However, it is always possible to make it produce a spurious output pulse by inserting delays in certain branches of the network.

There are two different types of M-hazards. The first type, called a *function hazard,* is illustrated in Table 1. For the transition from cell a to cell c it is possible to temporarily enter cell b if the y change slightly precedes the x change. Since a 0 output is specified for cell b it must be possible for a spurious 0 pulse to be present at f for the input change $\bar{w}\bar{x}\bar{y}$ to $\bar{w}xy$.

Consider the p-variable transition from input state A to input state B, where

$$A = (a_1, \cdots a_p, \quad a_{p+1}, \cdots a_n),$$
$$B = (\bar{a}_1, \cdots \bar{a}_p, \quad a_{p+1}, \cdots a_n),$$

and a_i represents some value (1 or 0) for input x_i.

Definition 2. A Boolean function, f, contains a *function hazard* for the input change A to B if and only if

(1) $f(A) = f(B)$ and

(2) there exist both 1's and 0's specified for f within the 2^p cells of the sub-cube $(a_{p+1}, \cdots a_n)$.

It is evident that if a function, f, contains a function hazard for the input change A to B, then there must be some set of values for the changing input variables $x_1 \cdots x_p$ for which f is not equal to $f(A)$ or $f(B)$. Consequently, it must be possible for the delays in the network to be such that these input changes reach the output in a sequence which causes a spurious hazard pulse to be generated. Thus the following theorem is true:

Theorem 1. If a Boolean function, f, contains a function hazard for the input change A to B, then it is impossible to construct a logic gate network realizing f such that the possibility of a hazard pulse occurring for this transition is eliminated.

The second type of M-hazard, called a *logic hazard,* is closely related to static hazards in that both can always be eliminated by properly designing the logic network. In general, if a p-variable transition does not involve a function hazard but may nevertheless result in a hazard pulse on the output of the logic network, the network is said to contain a *p-variable logic hazard* for this transition.

Definition 3. A combinational logic network contains a p-variable logic hazard for the p-variable input change A to B if and only if

(1) $f(A) = f(B)$,
(2) all of the 2^p values specified for f in the sub-cube $(a_{p+1}, \cdots a_n)$ are the same, and
(3) during the input change A to B a spurious hazard pulse may be present on the output.

Table 1 Example of function hazard.

z \ x, y	00	01	11	10
0	1 a	0 b	1 c	1 d
1	1	0	0	0

f

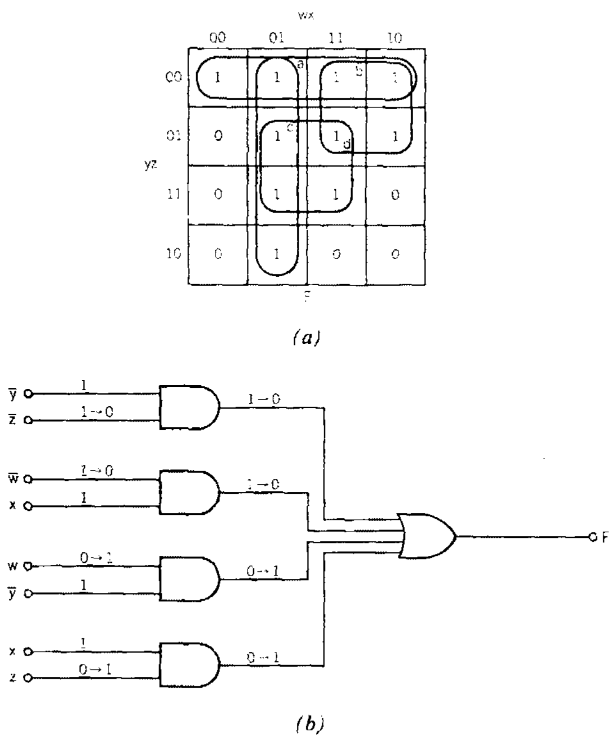
Condition (2) for a p-variable logic hazard is equivalent to stating that the input change does not involve a function hazard. It is also noted that if $p = 1$, conditions (1) and (2) are identical and Definition 3 reduces to the definition for a static hazard. Thus a p-variable logic hazard is a static hazard if $p = 1$.

The relationship between M-hazards, function hazards, and logic hazards still requires some explanation. If a p-variable transition can result in an output hazard pulse, the transition involves an M-hazard. If all the 2^p values specified for f in the sub-cube are the same, the M-hazard is a logic hazard; if not, the M-hazard is a function hazard. If the p-variable transition involves a function hazard, it cannot also involve a p-variable logic hazard. However it may involve a q-variable logic hazard ($q < p$) for one of the q-variable sub-transitions contained within the p-variable transition.

It is also evident from Definition 3 that if a p-variable transition involves a p-variable logic hazard, it may also involve or be the result of one or more q-variable logic hazards ($q < p$). In terms of static hazards ($q = 1$) it means that a network may contain a p-variable logic hazard ($p > 1$) because it contains a static hazard for one of the changing input variables.

However, it is possible for logic networks to be free of static hazards and still contain logic hazards. An example of such a circuit is shown in Fig. 2. It is evident

Figure 2 Example of a network containing logic hazards but no static hazards; (a) map of function; (b) circuit realization.



from the map of the function that the circuit realization is free of static hazards. However, consider the transition from cell a to cell d. It is evident from Fig. 2b that a 0-pulse will appear at the output of the network if $\bar z$ and $\bar w$ both change to 0 before z and w both change to 1. Thus the network contains a logic hazard for the input change from cell a to cell d even though it does not contain a static hazard. The transitions d to a, c to b, and b to c also involve logic hazards.

The logic hazard in this network (Fig. 2b) can be eliminated by adding an AND gate corresponding to the product term $\bar x \bar y$. Since $\bar x \bar y$ is the only prime implicant[5] of F which is not contained in the two-level realization, the question might be asked as to whether or not all prime implicants must always be included in the two-level solution to eliminate all logic hazards. This is indeed the case, as shown by the theorem which follows.

Theorem 2. A sum-of-products realization of F (assuming no product terms with complementary literals) will be free of all logic hazards if and only if the realization contains all the prime implicants of F.

Proof. The absence of complementary literals within a product term eliminates the possibility of a 0-1-0 logic hazard pulse. In order to eliminate all possibility of a 1-0-1 logic hazard pulse, it is necessary and sufficient that at least one of the product terms be 1 for the entire transition.

a. Assume the solution contains all prime implicants. Assume there also exists some p-variable logic hazard. Then there must be a corresponding p-variable transition $(a_1, \cdots a_p, a_{p+1}, \cdots a_n)$ to $(\bar a_1, \cdots \bar a_p, a_{p+1}, \cdots a_n)$ which does not contain a function hazard. Thus $F = 1$ if the literals $a_{p+1}, \cdots a_n$ are all 1, and some subset of these literals must be a prime implicant with a corresponding product term in the solution. But this would prevent the hazard from occurring, so no logic hazard can exist if the solution contains all prime implicants.

b. Assume that some prime implicant $(a_q, \cdots a_n)$ is not included in the realization. Consider the transition $(a_1, \cdots a_{q-1}, a_q, \cdots a_n)$ to $(\bar a_1, \cdots \bar a_{q-1}, a_q, \cdots a_n)$. The only possible product term which can belong to the solution and be 1 for this transition is $(a_q, \cdots a_n)$, so the transition must involve a logic hazard. Thus all prime implicants are required to eliminate all logic hazards.

The dual of Theorem 2 concerning product-of-sums realization is, of course, also true.

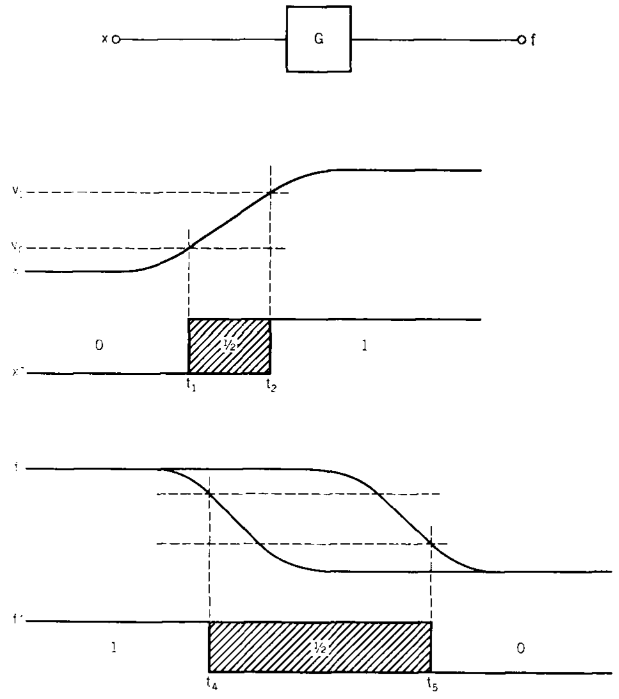
Hazard detection by ternary algebra

The problem of using ternary algebra to detect and eliminate static hazards in combinational switching circuits has been considered extensively by Yoeli and Rinon.[4] In this section we relate the actual switching circuit to a ternary function, and show that the resulting ternary function can be used to detect both function hazards and logic hazards. A technique for generating the ternary function from the binary switching circuit is described and some basic properties of the resulting ternary function are stated and proved.

The relevance of a third value to describe the transient behavior of a gate-type switching network can be seen from Fig. 3. The f curve represents the gate response for the minimum and maximum gate delay as a result of the input x. First consider the input x, which is changing from 0 to 1 ($-$ to $+$). As long as x is below some voltage, v_0, the input is assumed to be a 0, and as long as it is above some voltage, v_1, it is assumed to be a 1. During the time the input is between v_0 and v_1 it is *indeterminate* and may be considered to be either 1 or 0. In fact if a signal is between v_0 and v_1 it can simultaneously affect one logic gate as a 1 and another as a 0, depending upon the electrical characteristics of the two gates. A third value, designated as $\frac{1}{2}$ will be used to represent a signal which may be either 1 or 0. It can be seen from Fig. 3 that the ternary signal, x^*, represents the actual continuous signal, x, very well.

The ternary signal f^*, corresponding to the gate response f, has a $\frac{1}{2}$ value for a longer time period than x^*. This is due to both the transition time and the possible variation in gate delay. It is noted that $f_1 = f_4$ since it is

Figure 3 Ternary approximation to continuous signals.



assumed that the minimum delay through a logic gate is zero.

- *Obtaining the ternary functions of a logic gate*

The ternary function G^*, of a logic gate that realizes the binary function G, can be determined easily by using the fact that the value $\frac{1}{2}$ represents a signal that can be either 1 or 0. Thus G^* can be determined by changing back and forth between 1 and 0 those gate inputs corresponding to values of $\frac{1}{2}$ and then noting whether the gate output, G, changes or remains fixed at a 1 or 0. If G remains fixed at a 1 (0) then $G^* = 1$ (0), and if G changes then $G^* = \frac{1}{2}$.

Hypothesis. $G^*(\frac{1}{2}, \cdots \frac{1}{2}, a_{p+1}, \cdots a_n) = 1$ (0) if and only if it is impossible to change the gate output G from a 1 (0) by changing inputs $x_1, \cdots x_p$ when inputs $x_{p+1}, \cdots x_n$ are fixed at $a_{p+1}, \cdots a_n$, respectively. $G^*(\frac{1}{2}, \cdots \frac{1}{2}, a_{p+1}, \cdots a_n) = \frac{1}{2}$ if and only if it is not equal to 1 or 0.

If it is assumed that the individual logic gates do not contain logic hazards (which is true for all types of logic gates known to the author) then G^* can be determined by examining the map of the gate function G.

Table 2 shows the ternary and binary functions for the EXCLUSIVE-OR gate. The four G^* entries corresponding to xy = 00, 10, 01, and 11 are the same as the G entries. The G^* entry for cell j is determined by the G entries in cells a and b since cell j corresponds to $x = \frac{1}{2}$(1 or 0), and $y = 0$. The entries in cells a and b differ, so the entry in cell j must be $\frac{1}{2}$. (If both entries in cells a and b had been 1, then the entry in cell j would have been 1.) The entry in cell k is determined by examining all four entries for G. Since some are 1 and some are 0, the entry for cell k is $\frac{1}{2}$.

In general, if p of the inputs to an n-input gate are $\frac{1}{2}$, the ternary output, G^*, can be determined by examining the corresponding 2^p entries in the binary truth table for G. If all entries are 1 (0) then $G^* = 1$ (0) and if some entries are 1 and some are 0, then $G^* = \frac{1}{2}$.

Table 2 Binary and ternary functions for EXCLUSIVE OR.

(a) G

y \ x	0	1
0	0 a	1 b
1	1 c	0 d

(b) G*

y \ x	0	$\frac{1}{2}$	1
0	0 i	$\frac{1}{2}$ j	1
$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$ k	$\frac{1}{2}$
1	1	$\frac{1}{2}$	0

Table 3 Ternary functions for AND and OR gates.

AND, $G^* = \text{MIN}(x, y)$

y \ x	0	$\frac{1}{2}$	1
0	0	0	0
$\frac{1}{2}$	0	$\frac{1}{2}$	$\frac{1}{2}$
1	0	$\frac{1}{2}$	1

OR, $G^* = \text{MAX}(x, y)$

y \ x	0	$\frac{1}{2}$	1
0	0	$\frac{1}{2}$	1
$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1
1	1	1	1

Table 4 Possible combinations for G^* (C) and $G^*(D)$.

G^* (D) \ G^* (C)	0	$\frac{1}{2}$	1
0	YES	NO	NO
$\frac{1}{2}$	YES	YES	YES
1	NO	NO	YES

The ternary functions for AND and OR gates are shown in Table 3. The AND and OR correspond respectively to minimum and maximum functions.

Two lemmas concerning the characteristics of ternary gate functions will now be stated and proved.

Lemma 1. If one or more ternary gate inputs are changed from 1 to $\frac{1}{2}$, or 0 to $\frac{1}{2}$, the ternary gate output will either remain unchanged or change to $\frac{1}{2}$.

Lemma 2. If one or more ternary gate inputs are changed from $\frac{1}{2}$ to 1 or 0, the gate output will either remain unchanged or change from $\frac{1}{2}$ to 1 or 0.

Proof. Consider the two ternary input-states C and D where

$$C = (\tfrac{1}{2}, \cdots \tfrac{1}{2}, a_{q+1}, \cdots a_p, a_{p+1}, \cdots a_n)$$
$$D = (\tfrac{1}{2}, \cdots\cdots\cdots\cdots\cdots \tfrac{1}{2}, a_{p+1}, \cdots a_n)$$

and $a_i = 1$ or 0.

Lemma 1 corresponds to a transition from input state C to input state D, and Lemma 2 corresponds to a transition from D to C. By hypothesis, G^* $(D) = 1$ (0) if and only if the gate output cannot be changed by varying inputs $x_1, \cdots x_p$ when $x_{p+1}, \cdots x_n$ equals $a_{p+1}, \cdots a_n$, respectively. Therefore G^* $(D) = 1$ (0) implies that G^* $(C) = 1$ (0) since $x_{p+1}, \cdots x_n = a_{p+1}, \cdots a_n$ for input-state C also. This restriction eliminates four of the nine possible combinations of values for G^* (C) and G^* (D) as shown in Table 4. It is evident from Table 4 that an input change from C to D only results in G^* remaining unchanged or changing from 0 to $\frac{1}{2}$ or 1 to $\frac{1}{2}$. And an input change from D to C can only result in G^* remaining unchanged or changing from $\frac{1}{2}$ to 0 or $\frac{1}{2}$ to 1.

Ternary function characteristics

The ternary function of a combinational logic network is determined by the ternary functions of the logic gates in the network. That is, for any ternary input state the corresponding ternary output is uniquely determined by evaluating the ternary output of each logic gate in the network, starting with those gates whose inputs are also network inputs. Since the ternary functions for the logic gates are determined in the manner described in the previous section, one may now prove certain facts about the resulting network ternary functions.

One characteristic that will be quite useful later is that if one or more inputs to a combinational network are changed from 1 to $\frac{1}{2}$ or 0 to $\frac{1}{2}$, the network output can either remain unchanged, or change to $\frac{1}{2}$, but it cannot change from 1 to 0, from 0 to 1, or from $\frac{1}{2}$ to either 1 or 0. This fact, which is now formally stated and proved, is an extension of Theorem 1 of Yoeli and Rinon.[4]

Theorem 3. If one or more ternary inputs to a combinational logic network changes from 1 to $\frac{1}{2}$ or 0 to $\frac{1}{2}$, then the network output either remains unchanged or changes to $\frac{1}{2}$.

Proof. Consider one of the logic gates G, whose inputs are also network inputs. All inputs to G must either be unchanged or changed to $\frac{1}{2}$, so by Lemma 1, the output of G must also be either unchanged or changed to $\frac{1}{2}$. Thus the inputs and outputs of every gate in the network must either be unchanged or changed to $\frac{1}{2}$. Since the network output is also a gate output the theorem is proved.

Theorem 4. If one or more ternary inputs to a combinational logic network changes from $\frac{1}{2}$ to 1 or $\frac{1}{2}$ to 0, then the network output either remains unchanged or changes from $\frac{1}{2}$ to 1 or $\frac{1}{2}$ to 0.

Proof. The theorem is proved using Lemma 2 and the same argument as for Theorem 3.

Another useful characteristic of a network ternary function is that if the network output f can change in any way during an input change, then the ternary function, f^*, must equal $\frac{1}{2}$, if the changing inputs are assigned values of $\frac{1}{2}$.

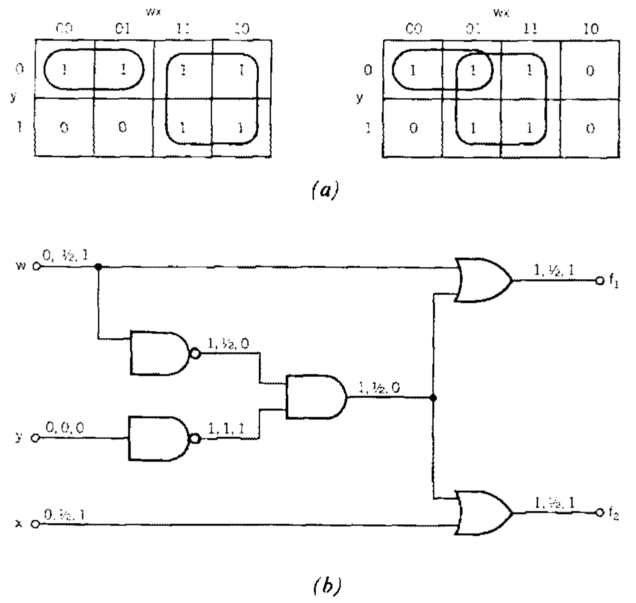


Figure 4 Hazard detection by ternary evaluation. (a) Map of functions; (b) Evaluating input changes for hazards.

Theorem 5. The output, f, of a combinational logic network may change as a result of changing inputs $x_1, \cdots x_p$ (when inputs $x_{p+1}, \cdots x_n = a_{p+1}, \cdots a_n$) if and only if $f^* (\frac{1}{2}, \cdots \frac{1}{2}, a_{p+1}, \cdots a_n) = \frac{1}{2}$.

Proof. Consider those gates in the network whose inputs are also network inputs. If all network inputs that may be changing are assigned the value of $\frac{1}{2}$, then those gate outputs, G, may be changing if and only if $G^* = \frac{1}{2}$ (by hypothesis). If the ternary gate outputs for all gates are determined, then all gate inputs and outputs that may be changing will have the value of $\frac{1}{2}$. Since the network output f is also a gate output, then f may be changing if and only if $f^* = \frac{1}{2}$.

- *Hazard detection*

In this section a theorem is stated and proved which provides the basis for a technique for determining whether or not a network contains a hazard for a particular input change.

Consider the transition from input state A to input state B where

$$\begin{aligned} A &= (a_1 \cdots a_p, \quad a_{p+1}, \cdots a_n), \\ B &= (a_1 \cdots a_p, \quad a_{p+1}, \cdots a_n), \\ A/B &= (\tfrac{1}{2}, \cdots \tfrac{1}{2}, \quad a_{p+1}, \cdots a_n). \end{aligned}$$

Theorem 6. A combinational logic network contains an M-hazard for the input change A to B if and only if

(1) $f^* (A) = f^* (B) \neq \frac{1}{2}$ and

(2) $f^* (A/B) = \frac{1}{2}$.

Proof. By Definition 1, condition (1) must be true. Since $f^* (A) = f^* (B)$ an M-hazard will exist if and only if the network output can change during the input change A to B. But by Theorem 5 this can occur if and only if $f^* (A/B) = \frac{1}{2}$.

In Fig. 4 a logic network is analyzed to determine whether or not it contains a hazard for the input change $\bar{w}\bar{x}\bar{y}$ to $wx\bar{y}$. This is accomplished by determining $f^* (\bar{w}\bar{x}\bar{y})$, $f^* (\frac{1}{2}, \frac{1}{2}, \bar{y})$, and $f^* (w, x, \bar{y})$ by evaluating the ternary functions of each logic gate. It is evident from the resulting values obtained for f_1^* and f_2^* that both outputs contain a hazard for this transition. From the maps of the functions f_1 and f_2 it is evident that f_1 contains a logic hazard and f_2 contains a function hazard. The hazard in f_1 could be corrected by changing the logic network but the hazard in f_2 cannot be eliminated without changing the function. (It is noted that both types of hazards are detected by the same technique of evaluating the ternary function of the network for the transition.)

Hazard detection in sequential circuits

The problem of determining whether or not a sequential circuit will respond to an input change in the manner predicted by the flow table or transition table has been given considerable attention in the past.[6–8] At least three types of problems have been considered:

Static hazards. It has been shown that static hazards in the combinational logic generating the feedback signals can cause the circuit to malfunction and should, in general, be eliminated.

Critical races. When two or more feedback signals are changing together, a race is said to exist. If the order in which these changes occur can affect the final state of the circuit the race is said to be critical. Critical races should be avoided.

Essential hazards. Unger[8] has defined an essential hazard in terms of a flow table and has proved that, if a flow table contains an essential hazard, its circuit realization must contain at least one delay element for it to operate reliably. In terms of circuit operation the essential hazard is basically a critical race between an input signal change and a feedback signal change. The delay is needed to make the input signal always "win" the race.

Although the treatments of these three types of problems have been of outstanding significance in the theory of sequential circuit design, they do not offer a complete or easy solution to the problem of determining whether or not a given sequential circuit will respond reliably to a given input-state change. The object of this section is to describe an easy procedure for solving this problem.

• *Transition analysis using ternary techniques*

The basic problem of determining the transient response of a gate-type sequential circuit with no feedback delay elements can be divided into two parts. The first part is to determine all the feedback signals that may be changing as a result of the input change. The second part is to determine whether or not these feedback signals will eventually stabilize in some predetermined state. It will be shown that both of these problems can be solved by a ternary evaluation of the logic.

The first part of the problem can be solved in the following way. Consider the model shown in Fig. 5. The first step is to determine which of the Y signals can be changing as a result of the specified x variable changes. The second step is to determine whether any additional Y signals may be changing as a result of both the x variable changes and the y variable changes. This second step is repeated until no additional Y signal changes are determined. These steps may be accomplished as follows:

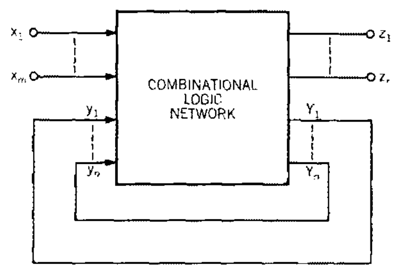
Procedure A—Determining all changing Y signals. With the changing x variables equal to $\frac{1}{2}$ and all other x and y variables as originally specified, evaluate the Y_i^* functions to determine if one or more have changed from 1 or 0 to $\frac{1}{2}$. If one or more Y_i^* functions have changed from 1 or 0 to $\frac{1}{2}$, change the corresponding y_i variables from 1 or 0 to $\frac{1}{2}$ and repeat the process until no additional changes in the Y_i^* functions are determined.

At the end of Procedure A all feedback signals, Y_i, that may experience some type of change during this transition will be indicated by Y_i^* functions equal to $\frac{1}{2}$.

The second half of the problem is to determine which of the changing feedback signals will eventually stabilize at a 1 or 0 as a result of the changing x variables stabilizing at their new values of 1 or 0. This is done as follows.

Procedure B—Determining which Y signals stabilize. With the changing x variables equal to their new values (1 or 0), and all other x and y variables equal to their values at the end of Procedure A, evaluate all Y_i^* functions. If one or more of these Y_i^* functions changes from $\frac{1}{2}$ to 1 or 0, change the corresponding y_i variable from $\frac{1}{2}$ to 1 or 0 and repeat the process until no additional changes in the Y_i^* functions are determined.

Figure 5 Model of sequential circuit.

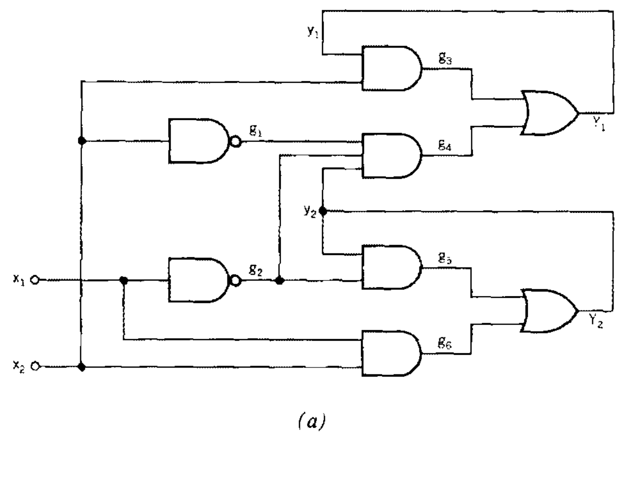


It is evident from Theorem 5 that if it is possible for one of the feedback signals, Y_k, to be changed by the changing of various x and y variables, then a ternary evaluation of the network will indicate $Y_k^* = \frac{1}{2}$. Thus the iterative process of Procedure A will detect all feedback signals that could possibly change as a result of the input change. It is also evident from Theorem 5 that if in the process of executing Procedure B it is determined that $Y_k^* = 1$ (0), then it must not be possible for the Y_k feedback signal to be changed from a 1 (0) by the changing y variables. Thus the following theorem is true.

Theorem 7. If $Y_k^* = 1$ (0) after applying Procedure A and Procedure B to a sequential circuit for a given input-state change starting in a given internal state, then the Y_k feedback signal must stabilize at 1 (0) for this transition regardless of the values of the (finite) delays associated with the logic gates.

Although Procedures A and B are difficult to explain they are very easy to perform since they are basically a ternary simulation of the logic. An example of this technique is shown in Fig. 6. In this case the problem is to determine whether or not changing x_1 and x_2 from 0 to 1 can result in an indeterminate final internal state. This is done by repeatedly determining the ternary functions Y_1^* and Y_2^* for each new set of values for the variables x_1, x_2, y_1, and y_2.

Figure 6 Transition analysis from logic network, using Procedures A and B. (a) Logic diagram; (b) Analysis table.



(a)

	x_1	x_2	y_1	y_2	g_1	g_2	g_3	g_4	g_5	g_6	Y_1^*	Y_2^*
1	0	0	0	0	1	1	0	0	0	0	0	0
2	½	½	0	0	½	½	0	0	0	½	0	½
3	½	½	0	½	½	½	0	½	½	½	½	½
4	1	1	½	½	0	0	½	0	½	1	½	1
5	1	1	½	1	0	0	½	0	1	1	½	1

(b)

Line 1 in Fig. 6b represents the initial state of the circuit. Lines 2 and 3 correspond to Procedure A and lines 4 and 5 correspond to Procedure B. The final value for Y_1^* Y_2^* is $\frac{1}{2}1$, so this transition results in an indeterminate value for the Y_1 feedback signal.

Procedures A and B as illustrated in Fig. 6 represent the solution to the key problem described in Fig. 1. To determine the response of a sequential circuit to a given input sequence, taking into account hazards, races, and oscillations, it is only necessary to insert between each input combination a new-input combination with $\frac{1}{2}$ values corresponding to changing variables, and then perform a ternary simulation of the sequential circuit. This solution to the problem is illustrated in Fig. 7.

• *Ternary simulation of sequential circuits*

Although the ternary simulation technique of Procedure A and Procedure B were developed to determine whether or not a sequential circuit would respond in a determinate way to a given input change, such a ternary simulation is a very powerful tool in evaluating transitions in very large logic networks.

In performing binary simulations of large logic networks the following problems exist:

(1) It is necessary to detect oscillations.

(2) Feedback lines must somehow be identified.

(3) For n feedback lines as many as 2^n evaluations of the network may be required.

(4) Hazards and races are not normally detected.

In using a ternary simulation as defined in Procedure A and Procedure B, the following is true.

(1) Oscillations are detected automatically.

(2) Feedback lines need not be identified.

(3) For n feedback lines, at most only $2n$ evaluations are required.

(4) Hazards and races are automatically detected.

(5) During Procedure A, any logic gate whose output value is $\frac{1}{2}$ need not be further considered, since its output value cannot possibly change (Theorem 3).

(6) During Procedure B, any logic gate whose output value is not $\frac{1}{2}$ need not be further considered since its output value cannot possibly change (Theorem 4).

Although Procedure A and procedure B are described in terms of feedback signals, Y_i, these signals are not handled any differently than the output signals of all other logic gates in the network. Thus feedback lines need not be identified. Conditions (5) and (6) can greatly reduce the number of logic gates that must be evaluated for any transition.

The following algorithm can be easily programmed to perform Procedures A and B in evaluating the effect of an input change on a sequential circuit.

• *Algorithm for transition analysis*

(1) With the changing input signals equal to $\frac{1}{2}$, place all logic gates that are fed by these signals and have a present output that is not $\frac{1}{2}$ on the "A-list."

(2) Remove one of the logic gates from the A-list and evaluate its output. (i) If its new output is $\frac{1}{2}$, place all logic gates that are fed by it and have a present output that is not $\frac{1}{2}$ on the A-list. (ii) If its new output is not $\frac{1}{2}$, do nothing.

(3) Repeat step (2) until the A-list is empty.

(4) With the changing input signals equal to their final value (1 or 0), place all logic gates that are fed by these signals and have a present output equal to $\frac{1}{2}$ on the B-list.

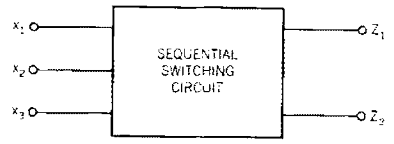
(5) Remove one of the logic gates from the B-list and evaluate its output. (i) If its new output is not $\frac{1}{2}$, place all logic blocks that are fed by it and have a present output of $\frac{1}{2}$ on the B-list. (ii) If its new output is $\frac{1}{2}$, do nothing.

(6) Repeat step (5) until the B-list is empty.

• *Manual technique for ternary evaluation*

In looking for a better manual technique for evaluating the Y^* functions, the first thought is to form the ternary truth table or maps of the Y^* functions. This is only practical for very small problems since there are 3^n entries for an n-variable ternary function. If the combinational logic generating the Y function is assumed to be free of all logic hazards (which is always possible) then only

Figure 7 Solution to key problem of Fig. 1.



x_1	0	1	0	0	GIVEN INPUT SEQUENCE
x_2	0	1	0	1	
x_3	0	0	1	1	

x_1	0	1/2	1	1/2	0	0	0	SIMULATED INPUT SEQUENCE
x_2	0	1/2	1	1/2	0	1/2	1	
x_3	0	0	0	1/2	1	1	1	

Table 5 Transition analysis from transition table using Procedures A and B; (a) Transition table; (b) Analysis table.

y_1 y_2	0	1
0 0	00	01
0 1	11	01
1 1	10	11
1 0	10	11
	Y_1	Y_2

(a)

	x	y_1	y_2	Y_1^*	Y_2^*
1	0	0	0	0	0
2	$\frac{1}{2}$	0	0	0	$\frac{1}{2}$
3	$\frac{1}{2}$	0	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
4	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1
5	1	$\frac{1}{2}$	1	$\frac{1}{2}$	1

(b)

function hazards can cause the Y^* function to change to a value of $\frac{1}{2}$ as a result of certain x and y variable changes. This allows the Y^* functions to be evaluated by examining the transition table entries for the Y functions.

The example in Table 5 shows how a transition can be analyzed by Procedures A and B by using a binary transition table. Line 1 of the analysis table corresponds to the initial stable state; lines 2 and 3 correspond to Procedure A and lines 3 and 4 to Procedure B. In line 2 the Y_1^* and Y_2^* values are obtained by examining the Y_1 and Y_2 entries in the transition table for $y_1y_2 = 00$. Since both Y_1 entries are 0, $Y_1^* = 0$, and since one Y_2 entry is 1 and the other is 0, $Y_2^* = \frac{1}{2}$. For line 3 all Y_1 and Y_2 entries in the transition table corresponding to $y_1 = 0$ are considered. Since Y_1 and Y_2 both have 1 and 0 entries, $Y_1^* = Y_2^* = \frac{1}{2}$. For line 4 all entries corresponding to $x = 1$ are considered and for line 5, all entries corresponding to $xy_2 = 11$. The final value of $Y_1^* Y_2^* = \frac{1}{2}1$ indicates that the Y_1 signal is indeterminate for this transition.

Transition analysis of sequential circuits containing delay elements

The ternary analysis technique will now be extended to include sequential circuits containing delay elements. This can be done by considering the delay element inputs as "special" outputs of the sequential circuit, and by considering the delay element outputs as "special" inputs to the sequential circuit.

To determine the response of the sequential circuit to an input change, the transition analysis is performed in the usual manner (Procedures A and B) with all delay element output signals held at their initial values. If one or more delay element inputs change during this analysis, the signal changes are then applied to the circuit at the corresponding delay element output terminals and the transition analysis is repeated.

It is possible that this process will not terminate if the transition involves an oscillation through the delay elements. Consequently some technique must be used to recognize an oscillatory condition and terminate the ternary simulation. To be completely rigorous all feedback lines would have to be identified and monitored to determine if the circuit is oscillating. Since this is difficult to accomplish, a much easier technique would be to monitor just the delay element output signals and arbitrarily decide that these signals are oscillating if they change more than a certain number of times. Once an oscillating signal is detected in this way, it is set to a value of $\frac{1}{2}$ and the process is continued. It is evident that this technique will terminate rather quickly and is fairly easy to perform.

Conclusions

It has been shown that there are two types of hazards associated with the changing of two or more input signals in a combinational network. The first type, called a logic hazard, is similar to a static hazard and can only be eliminated in a sum-of-products realization, by including all prime implicants. The second type, called a function hazard, cannot be eliminated by modifying the logic network. A technique using ternary algebra has been described for detecting both types of hazards. This technique has been extended to sequential circuits and a general procedure has been established for detecting whether or not the feedback signals will stabilize after a given transition.

Acknowledgments

The author is indebted to M. P. Marcus for his lively discussion and helpful suggestions during the preparation of this paper and to J. S. Jephson, C. B. Stieglitz and L. H. Tung who originally suggested the problem and whose use of a "third value" to represent "don't-know" conditions in logic simulation was very helpful in devel-

oping the techniques of transition analysis. The author is also grateful to R. M. Karp for his suggestion on the definition of logic hazards.

References and footnotes

1. D. A. Huffman, "The Design and Use of Hazard-Free Switching Networks," *J. ACM* 4, 47 (1957).
2. E. J. McCluskey, Jr., "Transients in Combinational Logic Circuits," from *Redundancy Techniques for Computing Systems*, Spartan Book Co. 1962, pp. 9–46.
3. D. E. Muller, "Treatment of Transition Signals in Electronic Switching Circuits by Algebraic Methods," *IRE Trans. on Electronic Computers* **EC-8**, 401 (1959).
4. M. Yoeli and S. Rinon, "Applications of Ternary Algebra to the Study of Static Hazards," *J. ACM* **11**, 84 (1964).
5. A prime implicant of F is a set of literals, P, such that (1) if all literals in P are 1, then $F = 1$, and (2) if any literal is removed from P, condition (1) no longer holds.
6. S. H. Caldwell, *Switching Circuits and Logical Design*, John Wiley & Sons, New York, 1958.
7. D. A. Huffman, "The Synthesis of Sequential Switching Circuits," *J. Franklin Inst.* **257**, 161, and **257**, 275 (1954).
8. S. H. Unger, "Hazards and Delays in Asynchronous Sequential Switching Circuits," *IRE Trans. on Circuit Theory* **CT-6**, 12 (1959).

Received October 5, 1964.

Revised manuscript received February 10, 1965.