STRUCTURE THEORY FOR THE REALIZATION OF FINITE STATE AUTOMATA

NASA GRANT NGR-44-012-049

C. L. Coates

Final Report

November 1, 1966 - October 31, 1968

ELECTRONICS RESEARCH CENTER

THE UNIVERSITY OF TEXAS AT AUSTIN

AUSTIN, TEXAS 78712

FINAL REPORT

STRUCTURE THEORY FOR THE REALIZATION OF FINITE STATE AUTOMATA
NASA Grant NGR-44-012-049
November 1, 1966 - October 31, 1968

I. SUMMARY

The results of the investigations that received support from this grant are described in research papers that have been prepared for publication. Copies of these papers constitute Section II of this report and represent the technical description of the results of the research. Each paper is listed below and summarized in regard to technical content.

1. C. Harlow and C. L. Coates, "Feedback in Sequential Machine Realizations," Submitted for publication to the IEEE Transactions on Computers. A condensed version appears in the Proceedings of the Second Annual Princeton Conference, Princeton University, 1968, pp. 96-98, under the title "On Feedback and Memory Elements."

   Summary. This investigation considered the effect the memory element has on the feedback in a Sequential Machine. Three different memory elements are studied - the unit delay, trigger flip-flop and set-reset flip-flop. Methods are given for determining when a sequential machine can be realized, using trigger or set-reset flip-flop memory elements, such that the amount of feedback in the machine is given by a function f. In addition, it is shown that if a machine can be realized with unit delay memory elements with feedback function f then the machine can be realized with set-reset flip-flop memory elements where f is the feedback in the

machine. Also it is shown that if a sequential machine can be realized without feedback using trigger flip-flop memory elements then it cannot be realized without feedback using unit delay memory elements. The converse statement is also true. The above results imply that the feedback in a sequential machine depends on the memory element used to realize the machine.

2. C. Harlow and C. L. Coates, "Inessential Errors in Sequential Machines," Submitted to the IEEE Transactions on Computers. Condensed version appears in the Proceedings of the Hawaii International Conference on Systems Sciences, University of Hawaii, 1968, pp. 616-618.

Summary. State errors in sequential machines have been classified by Hartmanis and Stearns as temporary and permanent. A subclass of the permanent errors is called inessential and is defined as follows:

At some time the machine is supposed to enter state a but enters state b instead due to a temporary malfunction. After this the state behavior of the machine is error free. The error is inessential if any infinite input sequence yields only a finite number of outputs that are different from those that would have occurred had the state error not taken place.

State errors as described above are denoted by state partitions τ_{ab} where a and b are the only two states in the same block of τ_{ab}. It has been shown by Hartmanis and Stearns that for a Moore machine the state partition $\Pi_E = \sum\{\tau_{ab} | \tau_{ab}$ is an inessential error$\}$ has the substitution property and

that every τ_{ab} is an inessential error if and only if $\tau_{ab} \leq \Pi_e$. They have also shown that state partition pairs do not characterize inessential errors.

The partition Π_E is of interest since it contains all state errors that cause only a finite number of faulty outputs. The definition given above however, does not constitute an effective way to compute Π_E. This paper presents several results concerning Π_E which imply a method for its computation. An algorithm for determining Π_E is stated and illustrated by means of an example.

3. F. O. Hadlock and C. L. Coates, "Realization of Sequential Machines with Threshold Elements," accepted for publication IEEE Transactions on Electronic Computers, 1969.

Summary. This paper presents an algorithm which, given a synchronous sequential machine with completely specified state and output tables, yields all code assignments for which the state variable and output variable functions are 2-assumable. Since the condition of 2-assumability is necessary and sufficient for linearly separability for completely specified functions of fewer than eight variables, the algorithm yields all code assignments for which the corresponding threshold gate realizations are one-level. For other functions 2-assumability is a necessary condition. In these cases the set of code assignments, if any, that yield one-level threshold gate realizations are contained in the set provided by the algorithm.

4. A. B. Howe and C. L. Coates, "Hazards in Threshold Networks," IEEE Transactions on Computers, vol. C-17, March 1968, pp. 233-251.

Summary. This paper is concerned with the study of logic hazards in threshold gate networks. Eichelberger has proved that logic hazards are not present in a sum-of-product (product-of-sum) realization which realizes all of the 1(0) prime implicants of the given Boolean function. Logic gates of the AND or NOR (OR or NAND) variety realize single 1(0) prime implicants; therefore, a gate is required for each 1(0) prime implicant to be realized, and the problem of eliminating logic hazards is straightforward.

A single-threshold gate, however, realizes a number of prime implicants. Moreover, the number of prime implicants realized by a network that incorporates more than a single-threshold gate is not uniquely determined either by the Boolean function being realized or by the number of gates involved. As a result, it is often possible to control the prime implicants and hence the hazards without greatly increasing the number of gates required. In fact, in some cases no additional gates are required.

Three methods are presented for determining if a given threshold realization contains any logic hazards, the first of which is an extension of McCluskey's work. Two methods are then presented for synthesizing logic hazard-free threshold realizations. The first method is based on the tree method of synthesizing threshold gate networks, whereas the second method is based on expressing the given Boolean function as a sum of threshold functions.

II. TECHNICAL RESULTS

FEEDBACK IN SEQUENTIAL MACHINE REALIZATIONS

INTRODUCTION

In this paper we shall study feedback in sequential machines that are realized with either trigger or set-reset flip-flop memory elements. Previous studies (Reference 3) of feedback have tacitly assumed that the memory elements were unit delays. The results in this paper show that the different memory elements affect the feedback in the machine.

Before we proceed we need to state the following preliminary concepts.

Definition 1. A sequential machine is a five tuple $M = (\{s\}, \{x\}, \{0\}, \delta, \lambda)$. Where

1. $\{s\}$ is a finite set called the states of M.
2. $\{x\}$ is a finite set called the inputs to M.
3. $\{0\}$ is a finite set called the outputs of M.
4. δ is a function with the domain of δ a subset of $\{s\} \times \{x\}$ and range a subset of $\{s\}$. That is, $\delta: \{s\} \times \{x\} \rightarrow \{s\}$.
5. λ is a function with domain a subset of $\{s\} \times \{x\}$ and range $\{0\}$. Thus $\lambda: \{s\} \times \{x\} \rightarrow \{0\}$.

For our purposes λ and $\{0\}$ are not important and we suppose the inputs to be n-tuples of $\{0,1\}$. Frequently, we discuss partitions on the states of a machine M. These partitions will be denoted by Greek letters. A definition and an example of this concept follows.

Definition 2. A partition ρ on a set $\{s\}$ is a collection of subsets of $\{s\}$ such that

1. $\bigcup_{A \in \rho} A = \{s\}$
2. If A and B are in ρ, then $A \cap B = \varphi$.

Example. If $\{s\} = \{1,2,3,4,5\}$, then a partition ρ is given by $\rho = \{\{1,2\},\{3,4\},\{5\}\}$. It is more convenient, however, to use the notation $\rho = (\overline{1,2};\overline{3,4};\overline{5})$. The subsets of ρ are often called blocks of ρ. For example, $\overline{1,2}$ is a block of ρ. When we discuss partitions we frequently need to discuss their blocks. If ρ is a partition on a set $\{s\}$ and if $a \in \{s\}$, then $\rho[a]$ will denote the block of ρ which contains a. In the above example $\rho[3] = \overline{3,4}$.

A trigger flip-flop is the two state sequential machine specified in Figure 1. A particular input to a trigger flip-flop will be denoted by T.

s	T	$\delta(s,T)$	$\lambda(s,T)$
1	0	1	0
1	1	2	0
2	0	2	1
2	1	1	1

$\{s\} = \{1,2\}$ states
$\{x\} = \{0,1\}$ inputs
$\{0\} = \{0,1\}$ output

Figure 1

If we are discussing more than one trigger flip-flop, we will index them with integers and refer to the i^{th} flip-flop with input T_i where i is an integer.

A set-reset flip-flop is the two state machine specified in Figure 2. A set-reset flip-flop has 2 inputs. A particular input will be denoted by (S,R) where S is called the set input and R is called the reset input. Again if we are discussing more than one set-reset flip-flop, we shall index them with integers and refer to the i^{th} flip-flop with inputs R_i and S_i where i is an integer.

s	S	R	$\delta(s,S,R)$	$\lambda(s,S,R)$
1	0	0	1	0
1	0	1	1	0
1	1	0	2	0
2	0	0	2	1
2	0	1	1	1
2	1	0	2	1

$\{s\} = \{1,2\}$

$\{x\} = \{0,1\} \times \{0,1\}$

$\{0\} = \{0,1\}$

Figure 2

In order to realize a machine M, it is necessary to code the states of M into n tuples of $\{0,1\}$. The coding function will be called h and $h:\{s\} \rightarrow \{0,1\}^n$ is a 1-1 function. The i^{th} projection of h will be called h_i; that is, for every state $h_i(s) = y_i$ where $h(s) = (y_1, \ldots, y_i, \ldots, y_n)$ and y_i is in $\{0,1\}$. It should be noted that our concept of a realization and that of Reference 3 are not the same in that we do not expand the machine.

We are often interested in subspaces of $\{0,1\}^n$. To be specific, let $n = 5$. $(y_1, \ldots, y_n) = (0,1,0,1,1)$ is in $\{0,1\}^5$. We want a general way to refer to specific coordinates of $(y_1, \ldots, y_n)$, say coordinates 3 and 5 where $(y_3, y_5) = (0,1)$ an element of $\{0,1\}^2$. We will use the following formalism to do this. If $\Lambda \leq \{1, \ldots, n\}$, we let $\{0,1\}^\Lambda$ be $\{0,1\}^j$ where j is the number of elements in Λ. If $y = (y_1, \ldots, y_n) \in \{0,1\}^n$, we denote by y_Λ the j tuple $(y_{i_1}, \ldots, y_{i_j}) \in \{0,1\}^\Lambda$ where $i_1 < i_2 < \ldots < i_j$ and $i_1, i_2, \ldots, i_j$ are all in Λ. In the above example $\Lambda = \{3,5\}$, $\{0,1\}^\Lambda = \{0,1\}^2$ and when $y = (y_1, \ldots, y_n) = (0,1,0,1,1)$, then $y_\Lambda = (0,1) \in \{0,1\}^\Lambda$.

If we are given a coding function $h: \{s\} \to \{0,1\}^n$ for a machine M, we can associate the following partitions with it. For every $i \in \{1, \ldots, n\}$ we define the partition ρ_i by $\rho_i[a] = \rho_i[b]$ iff $h_i(a) = h_i(b)$. We call ρ_i the partition associated with h_i. Conversely, given a two block partition ρ_i on a machine M we can define a function h_i on $\{s\}$ such that $h_i(a) = 1$ if a is in block 1 of ρ_i and $h_i(a) = 0$ if a is in block 2 of ρ_i. This h_i will be called the function associated with ρ_i. If there are n such ρ_i, then $h(a) = (h_1(a), \ldots, h_n(a))$ is 1-1 if $\prod_1^n \rho_i = \emptyset$ the zero partition. Often we are given $\Lambda \leq \{1, \ldots, n\}$ and we want to discuss $h_\Lambda(a)$ for $a \in \{s\}$. It should be noted that if $\tau = \prod_\Lambda \rho_i$ then $\tau[a] = \tau[b]$ implies that $h_\Lambda(b) = h_\Lambda(a)$.

If we code a machine by h into $\{0,1\}^n$, then h need not be onto. We denote by Y_i a function such that $Y_i: \{0,1\}^n \times \{x\} \to \{0,1\}$ and $Y_i(h(a), x) = h_i(\delta(a,x))$ for every $a \in \{s\}$ and $x \in \{x\}$. Thus, Y_i is an extension of the i^{th} next state function to all of $\{0,1\}^n$.

A problem in many of our results is filling in the "don't care" terms properly. In the results pertaining to unit delay realizations this is fairly easy. But when one considers flip-flop realizations, the situation is more complicated. This is the reason for much of the complexity in some of our proofs relating to reduced dependence. With this in mind we state some results given in Reference 3.

Definition 3. Let $M = (\{s\}, \{x\}, \{0\}, \delta, \lambda)$ be a sequential machine. Let τ and ρ be state partitions and let $f: \{s\} \times \{x\} \rightarrow D$ be some function where D is a set. We say τ "pf" ρ iff for every two states a, b such that $\tau[a] = \tau[b]$ and for every input x such that $\delta(a,x)$ and $\delta(b,x)$ are specified and $f(a,x) = f(b,x)$ then $\rho[\delta(a,x)] = \rho[\delta(b,x)]$.

The next theorem relates the relation pf to unit delay realizations.

Result 1 (Theorem).

Let M be a machine coded by h into $\{0,1\}^n$. Also, let $\tau = \prod_{\Lambda} \rho_i$ where $\Lambda \leq \{1, \ldots, n\}$ and let $\rho = \rho_g$ where $g \in \{1, \ldots, n\}$. Then τ "pf" ρ iff $Y_g(h(a), x) = F_g(h_\Lambda(a), f(a,x), x)$ when $\delta(a,x)$ is specified.

Proof.

i) Suppose τ "pf" ρ. Define F_g as follows. For every (y_Λ, d, x), where $d \in D$ and $y_i \in \{0,1\}$ for every i in Λ, let $F_g(y_\Lambda, d, x) = h_g[\delta(a,x)]$ if there exists a $\in \{s\}$ such that $\delta(a,x)$ is specified and $(h_\Lambda(a), f(a,x)) = (y_\Lambda, d)$. Show F_g is well defined. If there exists $a, b \in \{s\}$ such that $\delta(a,x)$ and $\delta(b,x)$ are specified, $\tau[a] = \tau[b]$ and $f(a,x) = f(b,x)$ then $\rho_g[\delta(a,x)] = \rho_g[\delta(b,x)]$ which implies $h_g[\delta(a,x)] = h_g[\delta(b,x)]$. Hence,

F_g is well defined. For all (y_Λ, d, x) such that there is no $a \in \{s\}$ and input x such that $(h_\Lambda(a),\ f(a,x)) = (y_\Lambda, d)$ then $F_g(y_\Lambda, d, x)$ can be specified in any manner.

ii) Suppose $h_g[\delta(a,x)] = F_g(h_\Lambda(a),\ f(a,x), x)$. Let $a, b \in \{s\}$ such that $\delta(a,x)$ and $\delta(b,x)$ are specified, $\tau[a] = \tau[b]$ and $f(a,x) = f(b,x)$. Since $\tau[a] = \tau[b]$ implies $h_i(a) = h_i(b)$ for every $i \in \Lambda$, this implies from the hypothesis that $h_g[\delta(a,x)] = h_g[\delta(b,x)]$ which implies $\rho[\delta(a,x)] = \rho[\delta(b,x)2$. ‖

Definition 4. Let M be a machine and τ a state partition of M, then $m^1_{pf}(\tau) = \Pi(\rho \mid \tau \text{ "pf" } \rho)$ and $m^{i+1}_{pf}(\tau) = m^1_{pf}(m^i_{pf}(\tau))$.

Our m_{pf} is the same as the m operator of Hartmanis and Stearns. We now give the result of Hartmanis and Stearns on feedback which we shall not prove. We do not use this result except to relate flip-flop realizations to delay realizations.

Result 2 (Theorem).

Let M be a q state sequential machine and let $f: \{s\} \times \{x\} \to D$. Then M can be realized using f for feedback iff $m^{q-1}_{pf}(I) = \emptyset$. I is the unit partitions.

Feedback and Trigger Flip-Flop Realizations

In order to determine when a function f can be used as feedback in a machine M realized with trigger flip-flops, we define the following relation. The definition of what we mean by the expression "using f for feedback" will be given later.

Definition 5. Let τ and ρ be state partitions on machine M and $f:\{s\} \times \{x\} \rightarrow \{0,1\}^{\ell}$ where ℓ is a positive integer. Then τ "tf" ρ iff

1. ρ is a two block partition

2. $\tau \cdot \rho$ "pf" ρ

3. For every 2 states a, b and for every input x such that $\delta(a,x)$ and $\delta(b,x)$ are specified, $\tau[a] = \tau[b]$, $\rho[a] \neq \rho[b]$, and $f(a,x) = f(b,x)$ then $\rho[\delta(a,x)] \neq \rho[\delta(b,x)]$.

If we have a machine M and a function $f:\{s\} \times \{x\} \rightarrow \{0,1\}^{\ell}$ and if we code M with h into $\{0,1\}^{n}$, then we can define a function f' on $\{0,1\}^{n}$ by $f'(h(a),x) = f(a,x)$. If h is not onto $\{0,1\}^{n}$, then we extend f' in any manner to all of $\{0,1\}^{n}$. We make no distinction between f and f'. It might be noted that we use the set $\{0,1\}^{\ell}$ rather than D for the range of f. We do this because we are interested in a realization which can be realized in a practical since. In other words the output of f will be fed into logical gates, hence, it is convenient to consider the outputs as n tuples of $\{0,1\}$. It is necessary to prove the next two results before we can characterize feedback in trigger flip-flop realizations.

Result 3 (Lemma).

Let machine M be realized with the g^{th} memory element a trigger flip-flop where $h:\{s\} \rightarrow \{0,1\}^{n}$ is the coding function and $g \in \{1,\ldots,n\}$. If

i) $f:\{s\} \times \{x\} \rightarrow \{0,1\}^{\ell}$ and $\Lambda \leq \{1,\ldots,n\}$ with $g \notin \Lambda$.

ii) $F_g(y,x) = y_g M(y_\Lambda, f(y,x),x) + \bar{y}_g N(y_\Lambda, f(y,x),x)$ for every $x \in \{x\}$ and $y \in \{0,1\}^n$ with $M = \bar{N}$.

Then $T_g(y,x) = G_g(y_\Lambda, f(y,x),x)$.

Proof.

Since $T_g(y,x) = y_g \bar{Y}_g(y,x) + \bar{y}_g Y_g(y,x)$ for every $y \in \{0,1\}^n$ and for every $x \in \{x\}$, if we substitute for Y_g and simplify we get that $T_g(y,x) = y_g \bar{M}(y_\Lambda, f(y,x),x) + \bar{y}_g N(y_\Lambda, f(y,x),x)$. Since $M = \bar{N}$, this implies $T_g(y,x) = N(y_\Lambda, f(y,x),x)$. ‖

Result 4 (Lemma).

Let machine M be coded by h into $\{0,1\}^n$. Let $\tau = \prod_\Lambda \rho_i$ where $\Lambda \leq \{1,\ldots,n\}$, let $\rho = \rho_g$ where $g \in \{1,\ldots,n\}$ and let $f:\{s\}\times\{x\}\to\{0,1\}^\ell$. If τ "tf" ρ, then $T_g(y,x) = G_g(y_\Lambda, f(y,x),x)$.

Proof.

Since τ "tf" ρ, we know that $\tau\cdot\rho$ "pf" ρ and from Result 1 this implies that $Y_g(y,x) = F_g(y_\Lambda, y_g, f(y,x),x)$ for every y in $\{0,1\}^n$.

i) Suppose $g \in \Lambda$. Then since $T_g(y,x) = y_g \bar{Y}_g(y,x) + \bar{y}_g Y_g(y,x)$, we deduce that $T_g(y,x) = y_g \bar{F}_g(y_\Lambda, y_g, f(y,x),x) + \bar{y}_g F_g(y_\Lambda, y_g, f(y,x),x)$ $= G_g(y_\Lambda, f(y,x),x)$.

ii) Suppose $g \notin \Lambda$. Then $Y_g(y,x) = y_g M(y_\Lambda, f(y,x),x)$ $+ \bar{y}_g N(y_\Lambda, f(y,x),x)$ where $M(y_\Lambda, d, x) = F_g(y_\Lambda, y_g = 1, d, x)$ for every $d \in \{0,1\}^\ell$ and $N(y_\Lambda, d, x) = F_g(y_\Lambda, y_g = 0, d, x)$. Recall that there are certain freedoms on F_g. Namely, $F_g(h_\Lambda(a), h_g(a), f(h(a)\cdot x),x) = h_g[\delta(a,x)]$ if $\delta(a,x)$ is specified. Otherwise, $F_g(y_\Lambda, y_g, d, x)$ where $d \in \{0,1\}^\ell$ is

arbitrary. Specify F_g as follows. If $F_g(y_\Lambda, y_g, d, x)$ is not determined as above, let $F_g(y_\Lambda, y_g, d, x) = \bar{F}_g(y_\Lambda, \bar{y}_g, d, x)$ where $d \in \{0,1\}^\ell$, $y \in \{0,1\}^n$ and $x \in \{x\}$. Show $F_g(y_\Lambda, y_g, d, x) \neq F_g(y_\Lambda, \bar{y}_g, d, x)$ for all $y \in \{0,1\}^n$, $x \in \{x\}$ and $d \in \{0,1\}^\ell$. We must only consider the case where there exists $a, b, \in \{s\}$, $x \in \{x\}$ such that $h_\Lambda(a) = h_\Lambda(b)$, $h_g(a) \neq h_g(b)$, $f(a,x) = f(b,x)$ and $\delta(a,x)$ and $\delta(b,x)$ are specified. In this case $F_g(h_\Lambda(a), h_g(a), f(h(a),x),x) = h_g[\delta(a,x)]$. But $h_\Lambda(a) = h_\Lambda(b)$ implies that $\tau[a] = \tau[b]$ and $h_g(a) \neq h_g(b)$ implies that $\rho[a] \neq \rho[b]$. Since $f(a,x) = f(b,x)$, $\delta(a,x)$ and $\delta(b,x)$ are specified and τ "tf" ρ, this implies that $\rho[\delta(a,x)] \neq \rho[\delta(b,x)]$ and therefore that $h_g[\delta(a,x)] \neq h_g[\delta(b,x)]$. Thus, $F_g(h_\Lambda(a), h_g(a), f(h(a),x),x) \neq F_g(h_\Lambda(b), h_g(b), f(h(b),x),x)$.

Show $M = \bar{N}$ if F_g is so specified. This is clear since $M(y_\Lambda, d, x) = F_g(y_\Lambda, y_g = 1, d, x) \neq F_g(y_\Lambda, y_g = 0, d, x) = N(y_\Lambda, d, x)$ for every y_Λ with $y_i \in \{0,1\}^\ell$ for every $i \in \Lambda$, $x \in \{x\}$ and $d \in \{0,1\}^\ell$. From Result 3 this implies the theorem. ‖

Result 5 (Theorem).

If machine M is coded by h into $\{0,1\}^n$ and realized with trigger flip-flop memory elements such that $T_g(y,x) = G_g(y_\Lambda, f(y,x),x)$ where $\Lambda \leq \{1,\ldots,n\}$ and $g \in \{1,\ldots,n\}$ and $f:\{s\} \times \{x\} \to \{0,1\}^\ell$, then τ "tf" ρ where $\tau = \prod_\Lambda \rho_i$ and $\rho = \rho_g$.

Proof.

i) Suppose $g \in \Lambda$. Then τ "tf" ρ is equivalent to τ "pf" ρ since $\rho \geq \tau$. In general, $Y_g(y,x) = y_g \bar{T}_g(y,x) + \bar{y}_g T(y,x)$ thus $Y_g(y,x) = y_g \bar{G}_g(y_\Lambda, f(y,x),x) + \bar{y}_g G_g(y_\Lambda, f(y,x),x) = F_g(y_\Lambda, f(y,x),x)$. Therefore from Result 1 τ "pf" ρ.

ii) Suppose $g \notin \Lambda$. Then again $Y_g(y,x) = y_g \bar{G}_g(y_\Lambda, f(y,x), x)$ $+ \bar{y}_g G_g(y_\Lambda, f(y,x), x) = F_g(y_\Lambda, y_g, f(y,x), x)$ and therefore from Result 1 $\tau \cdot \rho$ "pf" ρ. Let $a,b \in \{s\}$ and $x \in \{x\}$ such that $\tau[a] = \tau[b]$, $\rho[a] \neq \rho[b]$, $f(a,x) = f(b,x)$, and $\delta(a,x)$ and $\delta(b,x)$ are specified. $\tau[a] = \tau[b]$ implies that $h_i(a) = h_i(b)$ for every $i \in \Lambda$ and $\rho[a] \neq \rho[b]$ implies that $h_g(a) \neq h_g(b)$. Assume $h_g(a) = 1$ which implies $h_g(b) = 0$. Then $Y_g(h(a),x) = \bar{G}_g(h_\Lambda(a), f(h(a),x),x)$ and $Y_g(h(b),x) = G_g(y_\Lambda, f(y,x),x)$. But since $(h_\Lambda(a), f(h(a),x),x)$ $= (h_\Lambda(b), f(h(b),x),x)$, this implies that $Y_g(h(a),x) = \bar{Y}_g(h(b),x)$. Since $\delta(a,x)$ and $\delta(b,x)$ are specified, this means that $h_g[\delta(a,x)] \neq h_g[\delta(b,x)]$ which implies that $\rho[\delta(a,x)] \neq \rho[\delta(b,x)]$. The same argument yields the same result assuming $h_g(a) = 0$ which implies $h_g(b) = 1$. Therefore τ "tf" ρ. ‖

With these results out of the way we can consider feedback in machines realized with trigger flip-flop memory elements. First we must define this concept. The basic idea is to lay the machine out from left to right in such a way that the input function to the ith flip-flop can be computed from f and the state of that portion of the machine which lies to the left of the ith flip-flop. This is shown in Figure 3. It should be noted in the figure that the set of $\Lambda_r - \Lambda_{r-1}$ flip-flops consists of those flip-flops i such that $i \in \Lambda_r - \Lambda_{r-1}$.

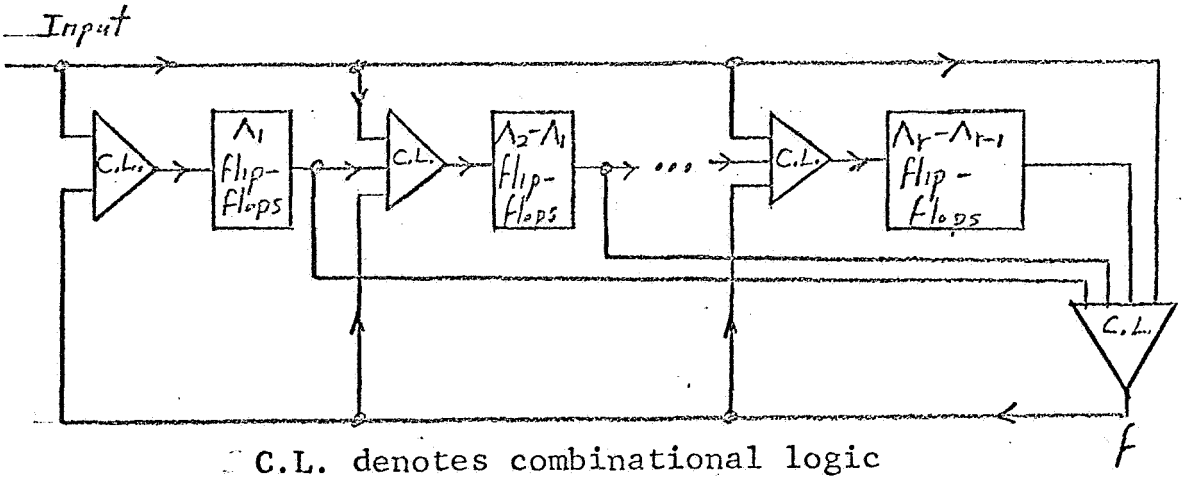
Definition 6. Let M be a machine and $f:\{s\} \times \{x\} \to \{0,1\}^\ell$. Then M can be realized with trigger flip-flop memory elements using f for feedback iff M can be coded by h into $\{0,1\}^n$ such that

1. There exists $\{\Lambda_1, \ldots, \Lambda_k\}$ a set of positive integers such that $u < v$ implies $\Lambda_u < \Lambda_v$.

2. If $i \in \Lambda_1$, then $T_i(y,x) = G_i(f(y,x),x)$ for every $y \in \{0,1\}^n$ and $x \in \{x\}$.

3. If $i \in \Lambda_r - \Lambda_{r-1}$ where $1 < r \leq k$, $T_i(y,x) = G_i(y_{\Lambda_{r-1}}, f(y,x), x)$.

4. $\prod_{\Lambda_k} \rho_i = \emptyset$ where ρ_i is the partition associated with h_i.



C.L. denotes combinational logic

Figure 3

It should be noticed that our definition of feedback depends on the memory element used. In order to prove the important theorem of this section we define the following quantity and prove a property about it.

Definition 7. Let M be a machine and τ be a state partition of M. Then $m_{tf}^1(\tau) = \Pi\{\rho \mid \tau \text{ "tf" } \rho\}$ and $m_{tf}^{i+1}(\tau) = m_{tf}^1(m_{tf}^i(\tau))$ for every integer i. If $\{\rho \mid \tau \text{ "tf" } \rho\} = \varphi$ the empty set, we define $m_{tf}^1(\tau) = I$ the unit partition.

We frequently designate $m_{tf}^1(\tau)$ by $m_{tf}(\tau)$. It should be observed that τ and $m_{tf}(\tau)$ are not in the relation "tf".

Result 6 (Lemma).

If $\tau.\tau_1$ and ρ are state partitions such that $\tau_1 \leq \tau$ and τ "tf" ρ where $f:\{s\} \times \{x\} \rightarrow \{0,1\}^{\ell}$ then τ_1 "tf" ρ.

Proof.

Let $\tau_1[a] = \tau_1[b]$. Then $\tau[a] = \tau[b]$ since $\tau \geq \tau_1$. If $\rho[a] = \rho[b]$, $f(a,x) = f(b,x)$ and $\delta(a,x)$ and $\delta(b,x)$ are specified then $\rho[\delta(a,x)$ $= \rho[\delta(b,x)]$ since $\tau.\rho$ "pf" ρ. If $\rho[a] \neq \rho[b]$, $f(a,x) = f(b,x)$ and $\delta(a,x)$ and $\delta(b,x)$ are specified then $\rho[\delta(a,x)] \neq \rho[\delta(b,x)]$ since τ "tf" ρ. ‖

Result 7.

Let M be a machine and $f:\{s\} \times \{x\} \rightarrow \{0,1\}^{\ell}$. If τ and τ_1 are state partitions such that $\tau_1 \leq \tau$ then $m_{tf}(\tau_1) \leq m_{tf}(\tau)$.

Proof.

Let ρ be such that τ "tf" ρ then τ_1 "tf" ρ from Result 6. This implies $m_{tf}(\tau_1) \leq m_{tf}(\tau)$. ‖

Result 8.

Let M be a machine and $f:\{s\} \times \{x\} \rightarrow \{0,1\}^{\ell}$. Then $m_{tf}^{i+1}(I) \leq m_{tf}^{i}(\tau)$ for every $i \in \{1,2,\ldots\}$.

Proof.

i) Show $m_{tf}^{2}(I) \leq m_{tf}^{1}(I)$. Clearly $m_{tf}^{1}(I) \leq I$. Thus $m_{tf}[m_{tf}^{1}(I)] \leq m_{tf}(I)$ from Result 7 which implies $m_{tf}^{2}(I) \leq m_{tf}(I)$.

ii) Suppose $m_{tf}^{i+1}(I) \leq m_{tf}^{i}(I)$. Then $m_{tf}[m_{tf}^{i+1}(I)] \leq m_{tf}[m_{tf}^{i}(I)]$ from Result 7 and therefore $m_{tf}^{i+2}(I) \leq m_{tf}^{i+2}(I)$.

It is clear that if $m_{tf}^{i}(I) = m_{tf}^{i+1}(I)$ then $m_{tf}^{i+2}(I) = m_{tf}^{i}(I)$. Therefore since I can be refined at most q-1 times if M is a q state machine we know that $m_{tf}^{q-1}(I) = m_{tf}^{q}(I)$.

It should be noticed that $m_{tf}(\tau)$ is a fairly difficult quantity to calculate. At this point we must consider every two block partition ρ and see if τ"tf"ρ and multiply these ρ together. Later we will give a better method. But first we prove a major result and then give an example of some of these concepts.

Result 9 (Theorem).

Let M be a machine and $f:\{s\} \times \{x\} \rightarrow \{0,1\}^{\ell}$. M can be realized with trigger flip-flops using f for feedback iff $m_{tf}^{q-1}(I) = \emptyset$ where q is the number of states of M.

Proof.

Suppose M can be realized with trigger flip-flops using f for feedback. Then M can be coded by h into $\{0,1\}^{n}$ such that Definition 6 is satisfied. Let $\tau_r = \prod_{\Lambda_r} \rho_j$.

1. From 2 of Definition 6 and Result 5 I"tf"ρ_i when $i \in \Lambda_r$ and ρ_i is the partition associated with h_i. This implies that $m_{tf}^{1}(I) < \prod_{\Lambda_i} \rho_j$ and $m_{tf}(I) \leq \tau_1$.

2. From 3 of Definition 6 and Result 5 $\prod_{\Lambda_{r-1}} \rho_j$"tf"$\rho_i$ for every $i \in \Lambda_r - \Lambda_{r-1}$. This together with Result 6 implies that $\prod_{\Lambda_{r-1}} \rho_j$"tf"$\rho_i$; that is, τ_{r-1}"tf"ρ_i for every $i \in \Lambda_r$.

3. From 1 we know $m_{tf}(I) \leq \tau_1$. Assume $m_{tf}^{r-1}(I) \leq \tau_{r-1}$ for every r such that $2 \leq r < k$ where k is given by Definition 6. Show $m_{tf}^{r}(I) \leq \tau_r$. From 2 $m_{tf}(\tau_{r-1}) \leq \prod_{\Lambda_r} \rho_i = \tau_r$. From the inductive hypothesis $\tau_{r-1} \geq m_{tf}^{r-1}(I)$ and therefore, from Result 7 we deduce that $m_{tf}[m_{tf}^{r-1}(I)] \leq \tau_r$ or equivalently $m_{tf}^{r}(I) \leq \tau_r$.

4. Show $m_{tf}^{q-1}(I) = \emptyset$. From 4 of Definition 6 $\prod_{\Lambda_k} \rho_i = \tau_k = \emptyset$. Hence $m_{tf}^{k}(I) \leq \tau_k = \emptyset$. Therefore $m_{tf}^{k}(I) = \emptyset$. From the comments after Result 8 this implies that $m_{tf}^{q-1}(I) = \emptyset$.

Suppose $m_{tf}^{q-1}(I) = \emptyset$. Let k be the first integer such that $m_{tf}^{k}(I) = \emptyset$. Then $k \leq q-1$.

1. Let $E_1 = \{\rho_i | i \in \Lambda_1\}$ be a set of partitions with the properties that I"tf"ρ_i for every $i \in \Lambda_1$ and $\prod_{\Lambda_1} \rho_i = m_{tf}(I)$. Such a set exists since $\{\rho | I"tf"\rho\}$ has these properties. However, it should be noted that one may not need to include all of these partitions in E_1.

2. Let $E_2 = \{\rho_i | i \in \Lambda_2\}$ be a set of partitions with the properties that $E_2 > E_1$, $m_{tf}(I)"tf"\rho_i$ for every $i \in \Lambda_2$ and $\prod_{\Lambda_2} \rho_i = m_{tf}^{2}(I)$. Again such a set exists since $E_2 = \{\rho | m_{tf}(I)"tf"\rho\}$ has the required properties. This follows from the fact that $m_{tf}^{2}(I) < m_{tf}(I)$.

3. Let $E_k = \{\rho_i | i \in \Lambda_k\}$ be a set of partitions with the properties that $E_k > E_{k-1}$, $m_{tf}^{k-1}(I)"tf"\rho_i$ for every $i \in \Lambda_k$ and $\prod_{\Lambda_k} \rho_i = m_{tf}^{k}(I)$. The set $E_k = \{\rho | m_{tf}^{k-1}(I)"tf"\rho\}$ satisfies these properties since $m_{tf}^{k}(I) < m_{tf}^{k-1}(I)$. Again it should be noted that one may not need to include all these partitions in E_k.

4. For every i such that $i \in \Lambda_k$, let $h_i(a) = h_i(b) \Leftrightarrow \rho_i[a] = \rho_i[b]$; i.e. let h_i be the function implies by ρ_i. Since $\prod_{\Lambda_k} \rho_i = m_{tf}^k(I) = \emptyset$, h is 1-1. Note that the range of h is $\{0,1\}^{j_k}$. Thus if we let n be the number of elements in Λ_k, to be consistent in notation, then $h:\{s\} \to \{0,1\}^n$.

5. From Result 4 since I"tf"ρ_i for every $i \in \Lambda_1$ we know that $T_i(y,x) = G_i(f(y,x),x)$ for every $y \in \{0,1\}^n$. Also from Result 4, since $\prod_{\Lambda_{r-1}} \rho_i = m_{tf}^{r-1}(I)$ and $m_{tf}^{r-1}(I)$"tf"ρ_i when $i \in \Lambda_r$, we know that $T_i(y,x)$ $= G_i(y_{\Lambda_{r-1}}, f(y,x),x)$ for every $i \in \Lambda_r$. Therefore, Definition 6 is satisfied and M can be realized with trigger flip-flop using f for feedback. ||

An example of this result is given in Figure 4 by machine A. $\{s\} = \{1,2,3,4,5\}$ and δ is given in the figure. Also a function $f:\{s\} \times \{x\} \to \{0,1\}$ is given in Figure 4. In machine A I"tf"$(\overline{1,2};\overline{3,4,5})$ and $m_{tf}(I) = (\overline{1,2};\overline{3,4,5})$ since this is the only partition with this property. $(\overline{1,2};\overline{3,4,5})$"tf"$\rho$ iff ρ is one of $(\overline{1,2};\overline{3,4,5})$, $(\overline{1,4};\overline{2,3,5})$ or $(\overline{2,4};\overline{1,3,5})$ and therefore $m_{tf}^2(I) = (\overline{1};\overline{2};\overline{3,5};\overline{4})$. $(\overline{1};\overline{2};\overline{3,5};\overline{4})$"tf"$\rho$ for every two block partition ρ and therefore $m_{tf}^3(I) = \emptyset$ which implies ρ can be realized with trigger flip-flops using f for feedback. Let $\rho_1 = (\overline{1,2};\overline{3,4,5})$, $\rho_2 = (\overline{1,4};\overline{2,3,5})$ and $\rho_3 = (\overline{1,2,3,4};\overline{5})$. Then $\Lambda_1 = \{1\}$, $\Lambda_2 = \{1,2\}$ and $\Lambda_3 = \{1,2,3\}$ satisfies the properties given in the proof of Result 9. In this case $E_1 = \{\rho_1\}$, $E_2 = \{\rho_1, \rho_2\}$ and $E_3 = \{\rho_1, \rho_2, \rho_3\}$. A coding function h corresponding to ρ_1, ρ_2 and ρ_3 is given in Figure 4.

If a machine can be realized using f for feedback and f is a constant, then we say f can be realized without feedback. Machine B in

Figure 5 gives a machine which can be realized without feedback using trigger flip-flops when f is any constant function. Note that I"tf"ρ iff $\rho = (\overline{1,2;3,4,5})$ and $(\overline{1,2;3,4,5})$"tf"ρ iff $\rho = (\overline{1,3,4;2,5})$, $(\overline{1,5;2,3,4})$ or $(\overline{1,2;3,4,5})$ and finally $(\overline{1;2;3,4;5})$"tf"ρ for every ρ which is a two block partition. Therefore $m^1_{tf}(I) = (\overline{1,2;3,4,5})$, $m^2_{tf}(I) = (\overline{1;2;3,4;5})$ and $m^3_{tf}(I) = \emptyset$.

States / Inputs

	0	1
1	1	3
2	3	4
3	4	1
4	5	2
5	1	2

δ

States / Inputs

	0	1
1	0	0
2	1	0
3	0	1
4	0	1
5	1	0

f

Machine A

$h(1) = (0,0,0)$

$h(2) = (0,1,0)$

$h(3) = (1,1,0)$

$h(4) = (1,0,0)$

$h(5) = (1,1,1)$

$$T_1(y_1,y_2,y_3,x) = \bar{x}\, f(y_1,y_2,y_3,x) + x$$

$$T_2(y_1,y_2,y_3,x) = y_1\, f(y_1,y_2,y_3,x) + x\bar{y}_1$$

$$T_3(y_1,y_2,y_3,x) = y_1 x\, f(y_1,y_2,y_3,x) + y_1\bar{\bar{x}}\bar{\bar{y}}_2 + y_1\bar{x}\, f(y_1,y_2,y_3,x)$$

	0	1	0	1	0	1
1	000	000	110	000	110	0	0
2	010	110	100	100	110	1	0
3	110	100	000	010	110	0	1
4	100	111	010	011	110	0	1
5	111	000	010	111	101	1	0

Y T f

Figure 4.

States	Inputs 0	Inputs 1	
1	5	2	h(1) = (0,0,0)
2	3	1	h(2) = (0,1,0)
3	d	4	h(3) = (1,1,1)
4	1	3	h(4) = (1,1,0)
5	2	5	h(5) = (1,0,0)

Machine B.

	Inputs 0	Inputs 1	Inputs 0	Inputs 1
000	100	010	100	010
010	111	000	101	010
111	d	110	d	001
110	000	111	110	001
100	010	100	110	000
	(Y_1, Y_2, Y_3)		(T_1, T_2, T_3)	

$$T_1(y_1, y_2, y_3, x) = \bar{x}$$

$$T_2(y_1, y_2, y_3, x) = x\bar{y}_1 + \bar{x}y_1$$

$$T_3(y_1, y_2, y_3, x) = y_2\bar{y}_1\bar{x} + xy_1y_2$$

Figure 5.

Now we relate feedback for the unit delay case to feedback with trigger flip-flop memory elements. One result we get is that the set of machines which can be realized without feedback using unit delays and the set of machines which can be realized without feedback using trigger flip-flops are disjoint when the machines are completely specified. Before we prove this result we consider a lemma.

Result 10 (Lemma).

Let M be a machine, $f:\{s\}\times\{x\} \to \{0,1\}^{\ell}$ and τ "tf" ρ. Let $a,b \in \{s\}$ such that $\tau[a] = \tau[b]$. If there exists $x \in \{x\}$ such that $f(a,x) = f(b,x)$ and $\delta(a,x) = \delta(b,x)$ then $m_{tf}(\tau)[a] = m_{tf}(\tau)[b]$.

Proof.

Consider any ρ such that τ "tf" ρ. If $\rho[a] \neq \rho[b]$ then $\rho[\delta(a,x)] \neq \rho[\delta(a,x)]$ which is impossible since $\delta(a,x) = \delta(b,x)$. Thus $\rho[a] = \rho[b]$ which implies $m_{tf}(\tau)[a] = m_{tf}(\tau)[b]$.

If $\delta:\{s\}\times\{x\} \to \{s\}$ and $\{x_i\}_1^n$ is a sequence in $\{x\}$, we define $\delta(a,\{x_i\}_1^n) = a$ if $n = 0$, $\delta(a,\{x_i\}_1^n) = \delta(a,x_1)$ if $n = 1$ and if $n > 1$ we inductively define $\delta(a,\{x_i\}_1^n) = \delta[\delta(a,\{x_i\}_1^{n-1}),x_n]$. With this notation we can prove the following result.

Result 11 (Lemma).

Let M be a completely specified machine and $f:\{s\}\times\{x\} \to \{0,1\}^{\ell}$. Let there exist $a,b \in \{s\}$ with $a \neq b$ and a sequence $\{x_i\}_1^{q-1}$ with $q \geq 2$ in $\{x\}$ such that $\delta(a,\{x_i\}_1^j)$ and $\delta(b,\{x_i\}_1^j)$ are specified when $1 \leq j \leq q-1$ and, in addition, $f[\delta(a,\{x_i\}_1^{j-1}),x_j] = f[\delta(b,\{x_i\}_1^{j-1}),x_j]$ for every j such that $1 \leq j \leq q-1$. Then $m_{pf}^{q-1}(I) = \emptyset$ implies that $m_{tf}^{q-1}(I) \neq \emptyset$.

Proof.

Since $I[a] = I[b]$, the hypothesis implies that $m_{pf}^{j}(I)[a] = m_{pf}^{j}(I)[b]$ for all $j \leq q-1$. Thus $m_{pf}^{q-1}(I) = \emptyset$ and the hypothesis implies that $\delta(a,\{x_i\}_1^{q-1}) = \delta(b,\{x_i\}_1^{q-1})$. Let r be the first integer such that $\delta(a,\{x_i\}_1^r) = \delta(b,\{x_i\}_1^r)$. Then $1 \leq r \leq q-1$ and if $a_1 = \delta(a,\{x_i\}_1^{r-1})$ and $b_1 = \delta(b,\{x_i\}_1^{r-1})$ then $a_1 \neq b_1$

and $\delta(a_1, x_r) = \delta(b_1, x_r)$. Show $m_{tf}^j(I)[a_1] = m_{tf}^j(I)[b_1]$ for every j such that j is a positive integer. Clearly $I[a_1] = I[b_1]$ and since $f(a_1, x_r) = f(b_1, x_r)$ and $\delta(a_1, x_r) = \delta(b_1, x_r)$ from Result 10 this implies that $m_{tf}(I)[a_1] = m_{tf}(I)[b_1]$. Suppose $m_{tf}^k(I)[a_1] = m_{tf}^k(I)[b_1]$. Again from Result 10 this implies that $m_{tf}(m_{tf}^k(I))[a_1] = m_{tf}(m_{tf}^k(I))[b_1]$ or $m_{tf}^{k+1}(I)[a_1] = m_{tf}^{k+1}(I)[b_1]$. If we let $j = q-1$ then $m_{tf}^{q-1}(I)[a_1] = m_{tf}^{q-1}(I)[b_1]$ where $a_1 \neq b_1$. Therefore $m_{tf}^{q-1}(I) \neq \emptyset$. ‖

From Result 11 we get the following result on feedback free machines.

Result 12 (Corollary).

Let M be a completely specified sequential machine with q states and $q > 1$.

i) If $m_{pf}^{q-1}(I) = \emptyset$, then $m_{tf}^{q-1}(I) \neq \emptyset$.

ii) If $m_{tf}^{q-1}(I) = \emptyset$, then $m_{pf}^{q-1}(I) \neq \emptyset$.

In particular, the set of machines which can be realized without feedback using unit delays is disjoint from the set of machines that can be realized without feedback using trigger flip-flop memory elements.

Proof.

i) Since $q > 1$ there exists $a, b \in \{s\}$ such that $a \neq b$. Let $\{x_i\}_1^{q-1}$ be any sequence in $\{x\}$. Since f is a constant, Result 11 holds and $m_{tf}^{q-1}(I) \neq \emptyset$.

ii) Suppose $m_{pf}^{q-1}(I) = \emptyset$, then from i) $m_{tf}^{q-1}(I) \neq \emptyset$ which is a contradiction. Therefore $m_{pf}^{q-1}(I) = \emptyset$.

The last statement follows from Results 9 and 2.‖

It is clear Result 12 does not necessarily hold if the machine is not completely specified. For example, one could consider a machine where $\delta(a,x)$ is not specified for any state a and input x. We now turn our consideration to the computation of $m_{tf}(\tau)$.

Definition 8. Let M be a machine and $f:\{s\}\times\{x\} \rightarrow \{0,1\}^{\ell}$. Let τ be a state partition.

1. Let $A(\tau) = \{(b,c) \mid \tau[b] = \tau[c]$ and there exists x such that $f(c,x) = f(b,x)$ and $\delta(b,x) = \delta(c,x)\} \cup \{(b,c) \mid$ there exists a $\epsilon\{s\}$ and input x such that $f(a,x) = f(b,x)$, $\tau[a] = \tau[b]$ and $c = \delta(a,x)$ while $a = \delta(b,x)$ or $\delta(a,x) = a$ and $\delta(b,x) = c\}$.

2. Let $A^{\#}(\tau)$ be the smallest equivalence class which contains $A(\tau)$ and let β_1 be the state partition implied by $A^{\#}(\tau)$.

Result 13.

If τ and β_1 are defined as in Definition 8 then $\beta_1 \leq m_{tf}(\tau)$.

Proof.

Let $b,c \epsilon\{s\}$ such that $\beta_1[b] = \beta_1[c]$. Let ρ be a partition such that τ "tf" ρ.

i) Suppose $(b,c) \epsilon A(\tau)$. If $\tau[b] = \tau[c]$ and there exists x such that $f(c,x) = f(b,x)$ and $\delta(c,x) = \delta(b,x)$ then $\rho[b] = \rho[c]$ from Result 10. If there exists a $\epsilon\{s\}$ and $x \epsilon\{x\}$ such that $f(a,x) = f(b,x)$, $\tau[a] = \tau[b]$ and $c = \delta(a,x)$ while $a = \delta(b,x)$, then if $\rho[a] = \rho[b]$ we have that $\rho[\delta(a,x)] = \rho[\delta(b,x)]$ since $\tau . \rho$ "pf" ρ or $\rho[c] = \rho[a] = \rho[b]$. If $\rho[a] \neq \rho[b]$, then $\rho[\delta(a,x)] \neq \rho[\delta(b,x)]$ or $\rho[c] \neq \rho[a]$ which implies $\rho[b] = \rho[c]$

since ρ has only two blocks. Therefore $\rho[b] = \rho[c]$. The proof for the case $a = \delta(b,x)$ and $\delta(b,x) = c$ is identical.

ii) Suppose $b = a_o$, $c = a_{k+1}$ $k \geq 0$ and $(a_o, a_1), (a_1, a_2, \ldots, (a_k, a_{k+1})$ are all in $A(\tau)$. Then from i) $\rho[a_i] = \rho[a_{i+1}]$ for every i $0 \leq i \leq k$. Since ρ is a partition, this implies $\rho[b] = \rho[a_o] = \rho[a_{k+1}] = \rho[c]$.

Cases i, ii cover all cases for b and c such that $\beta_1[b] = \beta_1[c]$ except when $b = c$ which is obvious. Thus $\beta_1[b] = \beta_1[c]$ implies $\rho[b] = \rho[c]$ for every ρ such that τ "pf" ρ. Therefore $m_{tf}(\tau)[b] = m_{tf}(\tau)[c]$. This implies that $\beta_1 \leq m_{tf}(\tau)$. ‖

It should be noted that $A(\tau)$ can be determined by inspection. One has only to observe that $(b,c) \in A(\tau)$ if $\delta(b,x) = \delta(c,x)$ for some x with $f(a,x) = f(b,x)$ or if any two of a, b, $\delta(a,x)$, $\delta(b,x)$ are equal when $\tau[a] = \tau[b]$ and $f(a,x) = f(b,x)$ then the other two are a pair in $A(\tau)$.

Definition 9. Let M be a machine and let τ, β_i be state partitions such that $m_{tf}(\tau) \geq \beta_i$ where $i \geq 1$.

1. Let $B(\beta_i) = \{(\beta_i[b], \beta_i[c]) \mid \tau[b] = \tau[c]$ and there exists input x such that $f(b,x) = f(c,x)$ and $\beta_i[\delta(b,x)] = \beta_i[\delta(c,x)]\} \cup \{\beta_i(b), \beta_i(c)) \mid \beta_i(c) = \beta_i[\delta(a,x)]$ and $\beta_i[a] = \beta_i[\delta(b,x)]$ or $\beta_i[a] = \beta_i[\delta(a,x)]$ and $\beta_i[c] = \beta_i[\delta(b,x)]$ for $a,b \in \{s\}$ and $x \in \{x\}$ such that $\tau[a] = \tau[b]$, $f(a,x) = f(b,x)\}$.

2. Let $B^{\#}(\beta_i)$ be the smallest equivalence relation which contains $B(\beta_i)$. Let β_{i+1} be a partition on $\{s\}$ defined by $\beta_{i+1}[a] = \beta_{i+1}[b]$ iff $(\beta_i[a], \beta_i[b]) \in B^{\#}(\beta_i)$.

Result 14.

$\beta_{i+1} \leq m_{tf}(\tau)$ and $\beta_{i+1} \geq \beta_i$.

Proof.

Let $b, c \in \{s\}$ such that $\beta_{i+1}[b] = \beta_{i+1}[c]$ which implies that $(\beta_i[c], \beta_i[b]) \in B^{\#}(\beta_i)$. Let ρ be a partition such that τ "pf" ρ.

i) Suppose $(\beta_1[c], \beta_1[b]) \in B(\beta_i)$. If $\tau[b] = \tau[c]$ and there exists x such that $f(b,x) = f(c,x)$ and $\beta_1[\delta(b,x)] = \beta_1[\delta(c,x)]$, then $\rho[\delta(b,x)] = \rho[\delta(c,x)]$ since $\rho \geq \beta_1$. Because τ "tf" ρ, $\rho[b] \neq \rho[c]$ implies $\rho[\delta(b,x)] \neq \rho[\delta(c,x)]$, which is a contradiction, thus $\rho[b] = \rho[c]$. If there exists $a \in \{s\}$, $x \in \{x\}$ such that $\tau[a] = \tau[b]$, $f(a,x) = f(b,x)$ and $\beta_i[c] = \beta_i[\delta(a,x)]$ while $\beta_i[a] = \beta_i[\delta(b,x)]$; then $\rho[a] = \rho[\delta(b,x)]$ and $\rho[c] = \rho[\delta(a,x)]$ since $\rho \geq \beta_i$. If $\rho[a] = \rho[b]$ then $\rho[\delta(a,x)] = \rho[\delta(b,x)]$ since $\tau . \rho$ "pf" ρ which implies $\rho[a] = \rho[\delta(a,x)] = \rho[c]$. Therefore $\rho[b] = \rho[c]$. If $\rho[a] \neq \rho[b]$ then $\rho[\delta(a,x)] = \rho[\delta(b,x)]$ since τ "tf" ρ. This implies $\rho[c] \neq \rho[a]$ and therefore $\rho[b] = \rho[c]$ since ρ has only two blocks. Therefore $(\beta_i[c], \beta_i[b]) \in B(\beta_i)$ implies $\rho[b] = \rho[c]$. The case when $\beta_i[a] = \beta_i[\delta(a,x)]$ and $\beta_i[c] = \beta_i[\delta(b,x)]$ is proved in a similar manner.

ii) Suppose $\beta_i[c] = \beta_i[a_o]$, $\beta_i[b] = \beta_i[a_{k+1}]$ and $(\beta_i[a_o], \beta_i[a_1])$, $(\beta_i[a_1], \beta_i[a_2]) \ldots (\beta_i[a_k], \beta_i[a_{k+1}])$ are such that $(\beta_i[a_j], \beta_i[a_{j+1}] \in B(\beta_i)$ when $0 \leq j \leq k$. Then from i) $\rho[a_j] = \rho[a_{j+1}]$ for every j such that $0 \leq j \leq k$. Therefore $\rho[c] = \rho[a_o] = \rho[a_{k+1}] = \rho[b]$.

Parts i, ii imply that if $(\beta_i[c], \beta_i[b]) \in B^{\#}(\beta_i)$, then $\rho[b] = \rho[c]$ which implies that $m_{tf}(\tau)[b] = m_{tf}(\tau)[c]$. Therefore $m_{tf}(\tau) \geq \beta_{i+1}$.

Results 13 and 14 imply a way to compute $m_{tf}(\tau)$. First compute β_1 as in Definition 8. Then using β_1 compute β_2 as in Definition 9. Continue until $\beta_{i+1} = \beta_i$ for some $i \geq 1$. This must happen since $\beta_{i+1} \geq \beta_i$ for every i and $\{s\}$ is finite. Let $\theta(\tau) = \beta_i$. One must then consider only those $\rho \geq \theta(\tau)$ to see if τ"tf"ρ when one computes $m_{tf}(\tau)$.

For an example of this consider machine A in Figure 4. Here it is seen by inspection that $A(I) = \{(3,4),(4,5),(3,5)\}$ which implies $A^{\#}(I) = \{(1,1),(2,2),(3,3),(4,4),(5,5),(3,4),(4,5),(3,5),(5,3),(5,4),(4,3)\}$ and $\beta_1 = (\overline{1;2;3,4,5})$. β_1 could have been easily determined from A(I). Now compute β_2. Again by inspection of Figure 4 $B(\beta_1) = \{(\overline{1;2})\}$ and $B^{\#}(\beta_1) = \{(\overline{1;2}),(\overline{3,4,5;3,4,5}),(\overline{2;1}),(\overline{1;1}),(\overline{2;2})$. Therefore $\beta_2 = (\overline{1,2;3,4,5})$. Compute β_3. By inspection $B(\beta_2) = \varphi$ and therefore $B^{\#}(\beta_2) = \{(\overline{1,2,1,2}),(\overline{3,4,5,3,4,5})\}$. Hence $\beta_3 = (\overline{1,2;3,4,5}) = \beta_2$ and therefore $\theta(I) = \beta_2$. From Results 13, 14 we know that $\theta(I) \leq m_{tf}(I)$. Hence to compute $m_{tf}(I)$, we need only consider all 2 block state partitions ρ such that $\rho \geq \theta(I)$ and an easy check shows I"tf"$\theta(I)$ and hence $m_{tf}(I) = (\overline{1,2;3,4,5})$. We have made this computation longer than needed. $B(\beta_i)$ can be written down by inspection of the state table and β_{i+1} can be written down directly from $B(\beta_i)$ without looking at $B^{\#}(\beta_i)$.

Let us again consider machine A in Figure 4. We have already determined that when we begin with I then $\theta(I) = (\overline{1,2;3,4,5})$. Repeat the calculations this time beginning with $\tau = \theta(I)$. Then by inspection $A(\tau) = \{(3,5)\}$ and $\beta_1 = (\overline{1;2;3,5;4})$. Continuing $B(\beta_1) = \varphi$ and therefore $\beta_2 = \beta_1$. Thus when we begin with $\tau = \theta(I)$ we get $\theta(\tau) = \theta(\theta(I)) = (\overline{1;2;3,5;4})$

which we label as $\theta^2(I)$. $\theta^2(I)$ must be less than $m_{tf}^2(I)$. Repeat the process letting $\tau = \theta^2(I)$. By inspection of Figure 14 $A(\tau) = \varphi$ and therefore $\beta_1 = \beta_2 = (\overline{1;2;3;4;5})$. Thus $\theta(\tau) = \theta(\theta^2(I)) = \theta^3(I) = (\overline{1;2;3;4;5})$. And $\theta^3(I) \leq m_{tf}^3(I)$. In this case $\theta^1(I) = m_{tf}(I)$, $\theta^2(I) = m_{tf}^2(I)$ and $\theta^3(I) = m_{tf}^3(I)$. To check a machine for feedback, one should do a computation as above. If one does not end up with the $\emptyset$ partition, the machine cannot be realized with trigger flip-flops using f for feedback. If one does end up with the $\emptyset$ partition, then he must continue to investigate by, for example, considering those two block partitions $\rho \geq \theta(I)$ to see if I"tf"ρ.

Feedback in Set-Reset Flip-Flop Realizations

In order to determine when a function f can be used as feedback in a machine M realized with set-reset memory elements we define the following relations.

Definition 10. Let τ and ρ be state partitions in machine M and $f:\{s\} \times \{x\} \rightarrow \{0,1\}^{\ell}$ where ℓ is a positive integer. Then τ"rf"ρ iff

1. ρ is a 2 block partition.

2. $\tau \cdot \rho$"pf"ρ.

3. For every two states a, b and every input x such that $\delta(a,x)$ and $\delta(b,x)$ are specified, $\tau[a] = \tau[b]$, $\rho[a] \neq \rho[b]$ and $f(a,x) = f(b,x)$; then $\rho[\delta(a,x)] = \rho[\delta(b,x)]$ or $\delta(a,x) \in \rho[a]$ and $\delta(b,x) \in \rho[b]$.

Before we consider the subject of feedback in set-reset realizations we must prove the next results which are similar to the ones proved in the trigger flip-flop development.

Result 15 (Lemma).

Let M be a machine and let $h:\{s\} \to \{0,1\}^n$ and $f:\{s\} \times \{x\} \to \{0,1\}^\ell$. Let $\Lambda \leq \{1,\ldots,n\}$ and $g \in \{1,\ldots,n\}$ but $g \notin \Lambda$. If for every $y \in \{0,1\}^n$ and for every input x $Y_g(y,x) = Y_g\ U(y_\Lambda, f(y,x),x) + \bar{y}_g\ W(y_\Lambda, f(y,x),x)$ where $U \geq W$, then $R_g(y,x) = I_g(y_\Lambda, f(y,x),x)$ and $S_g(y,x) = H_g(y_\Lambda, f(y,x),x)$.

Proof.

Let $R_g(y,x) = \bar{U}(y_\Lambda, f(y,x),x)$ and $S_g(y,x) = W(y_\Lambda, f(y,x),x)$. Show that this is allowable. That is, show that if $y_g = 1$ and $Y_g(y,x) = 0$ then $R_g(y,x) = 1$ and if $y_g = 0$ and $Y_g(y,x) = 1$ then $S_g(y,x) = 1$. In addition, one must show that R_g and S_g are not both one for any (y,x). Suppose $Y_g(y,x) = 0$. Then if $y_g = 1$, $U(y_\Lambda, f(y,x),x) = 0$. Thus $\bar{U}(y_\Lambda, f(y,x),x) = 1 = R_g(y,x)$. Suppose $Y_g(y,x) = 1$ and $y_g = 0$. Then $W(y_\Lambda, f(y,x)) = 1 = S_g(y,x)$. If $R_g(y,x) = 1$ then $U(y_\Lambda, f(y,x),x) = 0$ and since $U \geq W$ $W(y_\Lambda, f(y,x),x) = 0$. which implies $S_g(y,x) = 0$. If $S_g(y,x) = 1$, then $W(y_\Lambda, f(y,x),x) = 1$ and since $U \geq W$ $U(y_\Lambda, f(y,x),x) = 1$ which implies $R_g(y,x) = 0$. ‖

Result 16 (Theorem).

Let M be a machine and $f:\{s\} \times \{x\} \to \{0,1\}^\ell$. Let M be coded by h into $\{0,1\}^n$. Let $\tau = \prod_\Lambda \rho_i$ where $\Lambda \leq \{1,\ldots,n\}$ and $\rho = \rho_g$ where $g \in \{1,\ldots,n\}$. If τ "rf" ρ then $R_g(y,x) = I_g(y_\Lambda, f(y,x),x)$ and $S_g(y,x) = H_g(y_\Lambda, f(y,x),x)$.

Proof.

i) Suppose $g \in \Lambda$. Then $\rho \geq \tau$ and since τ "rf" ρ implies $\tau \cdot \rho$ "pf" ρ we deduce that τ "pf" ρ. Therefore $Y_g(y,x) = F_g(y_\Lambda, f(y,x),x)$ from Result 1. Define $R_g(y,x) = \bar{F}_g(y_\Lambda, f(y,x),x)$ and $S_g(y,x) = F_g(y_\Lambda, f(y,x),x)$.

ii) Suppose $g \notin \Lambda$. Then again τ . ρ "pf" ρ and from Result 1 $Y_g(y,x) = F_g(y_\Lambda, y_g, f(y,x), x)$. If we let $U(y_\Lambda, d, x) = F_g(y_\Lambda, y_g = 1, d, x)$ for every $d \in \{0,1\}^\ell$ and $W(y_\Lambda, d, x) = F_g(y_\Lambda, y_g = 0, d, x)$ for every $d \in \{0,1\}^\ell$ then $Y_g(y,x) = y_g\, U(y_\Lambda, f(y,x), x) + \bar{y}_g\, W(y_\Lambda, f(y,x), x)$. Recall that there are freedoms on F_g in the proof of Result 1. Namely, $F_g(h_\Lambda(a), h_g(a), f(h(a),x), x) = h_g[\delta(a,x)]$ if $\delta(a,x)$ is specified. For every other (y_Λ, y_g, d, x) with $d \in \{0,1\}^\ell$ and $y_\Lambda \in \{0,1\}^\Lambda$ F_g can be specified in any manner. We specify it as follows:

For every (y_Λ, y_g, d, x) define $F_g(y_\Lambda, y_g, d, x) = F_g(y_\Lambda, \bar{y}_g, d, x)$ when there is no $a \in \{s\}, x \in \{x\}$ such that $f(a,x) = d$, $h_\Lambda(a) = y_\Lambda$ and $\delta(a,x)$ is specified. Show when F_g is so specified that $U \geq W$. Suppose $W(y_\Lambda, d, x) = 1$. Then $F_g(y_\Lambda, y_g = 0, d, x) = 1$. Claim $F_g(y_\Lambda, y_g = 1, d, x) = 1$. This clearly is true from the above statements unless there exists $a, b \in \{s\}$ and $x \in \{x\}$ such that $(y_\Lambda, y_g = 0) = (h_\Lambda(a), h_g(a))$; $(y_\Lambda, y_g = 1) = (h_\Lambda(b), h_g(b))$; $f(a,x) = f(b,x) = d$ and $\delta(a,x)$ and $\delta(b,x)$ are specified. But $h_\Lambda(b)$ implies $\tau[a] = \tau[b]$ and $h_g(a) = h_g(b)$ implies $\rho[a] \neq \rho[b]$. Since $h_g(b) = 1$ from $F_g(h_\Lambda(a), h_g(a), f(h(a),x), x) = F_g(y_\Lambda, y_g = 0, d, x) = 1$ we infer that $h_g[\delta(a,x)] = 1$ or $\delta(a,x) \in \rho[b]$. But since τ "rf" ρ, this implies $\delta(b,x) \in \rho[b]$ and $h_g[\delta(b,x)] = F_g(h_\Lambda(b), h_g(b), f(h(a),x), x) = F_g(y_\Lambda, y_g = 1, d, x) = 1$. Therefore $W(y_\Lambda, d, x) = 1$ implies that $F_g(y_\Lambda, y_g = 1, d, x) = 1$ which implies $U(y_\Lambda, d, x) = 1$. Therefore $U \geq W$. From Result 15 this implies the theorem.‖

Result 17 (Theorem).

Let M be a sequential machine coded by h into $\{0,1\}^n$. Let $f:\{s\} \times \{x\} \rightarrow \{0,1\}^{\ell}$. If M is realized by set-reset flip-flops such that $R_g(y,x) = I_g(y_\Lambda, f(y,x),x)$ and $S_g(y,x) = H_g(y_\Lambda, f(y,x),x)$ where $g \in \{1,\ldots,n\}$ and $\Lambda \leq \{1,\ldots,n\}$ then τ"rf"ρ where $\tau = \prod_\Lambda \rho_i$ and $\rho = \rho_g$.

Proof.

i) Suppose $g\epsilon\Lambda$. In general $Y_g(y,x) = y_g \bar{R}_g(y,x) + \bar{y}_g S_g(y,x)$. Therefore $Y_g(y,x) = y_g \bar{I}_g(y_\Lambda, f(y,x),x) + \bar{y}_g H_g(y_\Lambda, f(y,x),x)$ $= F_g(y_\Lambda, f(y,x),x)$.

From Result 1 this implies $\tau.\rho$"pf"ρ and since $g\epsilon\Lambda$ implies $\rho \geq \tau$ this implies τ"pf"ρ and τ"tf"ρ.

ii) Suppose $g\notin\Lambda$. Then, as before, $Y_g(y,x) = y_g \bar{I}_g(y_\Lambda, f(y,x)$ $+ \bar{y}_g H_g(y_\Lambda, f(y,x),x) = F_g(y_\Lambda, y_g, f(y,x),x)$. From Result 1 this implies $\tau.\rho$"pf"ρ. Let $a,b \in \{s\}$ and $x \in \{x\}$ such that $\tau[a] = \tau[b]$. $\rho[a] \neq \rho[b]$, and $f(a,x) = f(b,x)$. Note that $\tau[a] = \tau[b]$ implies $h_\Lambda(a) = h_\Lambda(b)$ and $\rho[a] \neq \rho[b]$ implies $h_g[a] \neq h_g[b]$. Suppose $\delta(a,x) \in \rho[b]$ and also suppose $h_g[a] = 1$. Then $h_g[\delta(a,x)] = 0$ which implies $Y_g(h(a),x) = 0 = \bar{I}_g(h_\Lambda(a), f(h(a),x),x)$ or, equivalently, $I_g(h_\Lambda(a), f(h(a),x),x) = 1$. Since $I_g = R_g$ and $R_g = 1$ implies $S_g = 0$, we must have $S_g = H_g(h_\Lambda(a), f(h(a),x),x) = 0$ which in turn equals $H_g(h_\Lambda(b), f(h(b),x),x)$. Therefore $Y_g(h(b),x) = 0$ which implies that $\delta(b,x) \in \rho[b]$. Suppose $h_g[a] = 0$. Then $h_g[\delta(a,x)] = 1$ which implies $Y_g(h(a),x) = 1 = H_g(h_\Lambda(a), f(h(a),x),x)$. This implies that $S_g(h(a),x) = 1$ and therefore $R_g(h(a),x) = 0 = I_g(h_\Lambda(a), f(h(a),x),x)$. Since $I_g(h_\Lambda(a), f(h(a),x) = I_g(h_\Lambda(b), f(h(b),x),x)$ and $h_g(b) = 1$, this implies that $Y_g(h(b),x) = 1$. Therefore $h_g[\delta(b,x)] = 1$ and $\delta(b,x) \in \rho[b]$. Hence τ"rf"ρ.

With these results out of the way we are ready to define the concept of feedback in machines realized with set-reset memory elements. This definition is similar to Definition 6.

Definition 11. Let M be a machine and $f:\{s\} \times \{x\} \rightarrow \{0,1\}^{\ell}$. M can be realized with set-reset flip-flops using f for feedback iff M can be coded by h into $\{0,1\}^{n}$ such that

1. There exists $\{\Lambda_1, \ldots, \Lambda_k\}$ a set of positive integers such that $u < v$ implies $\Lambda_u < \Lambda_v$.

2. If $i \epsilon \Lambda_1$ then $R_i(y,x) = I_i(f(y,x),x)$ and $S_i(y,x) = H_i(f(y,x),x)$ for every $y \epsilon \{0,1\}^{n}$ and $x \epsilon \{x\}$.

3. If $i \epsilon \Lambda_r - \Lambda_{r-1}$ where $1 < r \leq k$ then $R_i(y,x) = I_i(y_{\Lambda_{r-1}}, f(y,x),x)$ and $S_i(y,x) = H_i(y_{\Lambda_{r-1}}, f(y,x),x)$.

4. $\prod_{\Lambda_k} \rho_i = \emptyset$ where ρ_i is the partition associated with h_i.

Again we define the m operator this time with respect to "rf", and then prove some results concerning it.

Definition 12. Let M be a machine and $f:\{s\} \times \{x\} \rightarrow \{0,1\}^{\ell}$. Let τ be a state partition.

i) $m_{tf}^{1}(\tau) = \prod\{\rho \mid \tau \text{ "rf" } \rho\}$. If $\{\rho \mid \tau \text{ "rf" } \rho\} = \varphi$, then define $m_{tf}^{1}(\tau) = I$. We frequently call $m_{rf}^{1}(\tau)$ by $m_{rf}(\tau)$ deleting the 1.

ii) $m_{rf}^{i+1}(\tau) = m_{rf}(m_{rf}^{i}(\tau))$ for i in $\{1,2,\ldots\}$.

Result 18.

Let τ, τ_1, and ρ be state partitions in a machine M and $f:\{s\} \times \{x\} \rightarrow \{0,1\}^{\ell}$. If $\tau_1 \leq \tau$ and τ "rf" ρ, then τ_1 "rf" ρ.

Proof.

Let $a,b \in \{s\}$ such that $\tau_1[a] = \tau_1[b]$, $f(a,x) = f(b,x)$, and $\delta(a,x)$ and $\delta(b,x)$ are specified. $\tau_1[a] = \tau_1[b]$ implies $\tau[a] = \tau[b]$ since $\tau \geq \tau_1$. Suppose $\rho[a] = \rho[b]$, then $\rho[\delta(a,x)] = \rho[\delta(b,x)]$ since $\tau_1 \cdot \rho$"pf"ρ. Suppose $\rho[a] \neq \rho[b]$ then $\delta(a,x) \in \rho[b]$ implies $\delta(b,x) \in \rho[b]$ since τ"rf"ρ.‖

Result 19.

Let τ and τ_1 be state partitions. If $\tau \geq \tau_1$ then $m_{rf}(\tau) \geq m_{rf}(\tau_1)$.

Proof.

Let ρ be such that τ"rf"ρ. Then τ_1"rf"ρ from Result 18 which implies $m_{rf}(\tau) \geq m_{rf}(\tau_1)$.‖

Result 20.

Let M be a machine and $f:\{s\} \times \{x\} \rightarrow \{0,1\}^{\ell}$. Then $m_{rf}^{i+1}(I) \leq m_{rf}^{i}(I)$ for every $i \in \{1,2,\ldots\}$.

Proof.

i) Show $m_{rf}^{2}(I) \leq m_{rf}(I)$. Clearly $m_{rf}(I) \leq I$. Therefore $m_{rf}(m_{rf}(I)) \leq m_{rf}(I)$ from Result 19. Thus $m_{rf}^{2}(I) \leq m_{rf}(I)$.

ii) Suppose $m_{rf}^{j}(I) \leq m_{rf}^{j-1}(I)$ with $2 \leq j$. Then $m_{rf}(m_{rf}^{j}(I)) \leq m_{rf}(m_{rf}^{j-1}(I))$ from Result 19. Thus $m_{rf}^{j+1}(I) \leq m_{rf}^{j}(I)$.‖

If M is a q state machine then since I can be refined at most q-1 times $m_{rf}^{q-1}(I) = m_{rf}^{q}(I)$. With this we are ready to state and prove the main result of this section.

Result 21 (Theorem).

Let M be a q state machine and $f:\{s\} \times \{x\} \rightarrow \{0,1\}^{\ell}$. M can be realized using set-reset flip-flop memory elements for feedback iff $m_{rf}^{q-1}(I) = \emptyset$.

Proof.

Suppose M can be realized using f for feedback. Then M can be coded by h into $\{0,1\}^{n}$ such that Definition 11 is satisfied. Let $\tau_r = \prod_{\Lambda_r} \rho_j$.

1. From 2 of Definition 11 and Result 17 $I"rf"\rho_i$ when $i \epsilon \Lambda_1$ and ρ_i is the partition associated with h_i. Therefore, $m_{rf}(I) \leq \prod_{\Lambda} \rho_i = \tau_1$.

2. From 3 of Definition 11 and Result 17 $(\prod_{\Lambda_{r-1}} \rho_j)"rf"\rho_i$ for every $i \epsilon \Lambda_r - \Lambda_{r-1}$. This together with Result 18 implies that $\prod_{\Lambda_{r-1}} \rho_j "tf" \rho_i$; that is, $\tau_{r-1}"tf"\rho_i$ for every $i \epsilon \Lambda_r$.

3. From 1 we know $m_{rf}(I) \leq \tau_1$. Assume $m_{rf}^{r-1}(I) \leq \tau_{r-1}$ for every r such that $2 \leq r < k$ where k is given by Definition 11. Show $m_{rf}^{r}(I) \leq \tau_r$. From 2 $m_{rf}(\tau_{r-1}) \leq \prod_{\Lambda_r} \rho_i = \tau_r$. From the inductive hypothesis and Result 19 $m_{rf}[m_{rf}^{r-1}(I)] \leq m_{rf}(\tau_{r-1}) \leq \tau_r$. Thus $m_{rf}^{r}(I) \leq \tau_r$.

4. Show $m_{rf}^{q-1}(I) = \emptyset$. From 4 of Definition 11 $\prod_{\Lambda_k} \rho_i = \emptyset$. From 3 of definition 11 $m_{rf}^{k}(I) \leq \tau_k = \prod_{\Lambda_k} \rho_i = \emptyset$. Therefore $m_{rf}^{k}(I) = \emptyset$. From Result 20 this implies $m_{rf}^{q-1}(I) = \emptyset$.

We now consider the converse. Suppose $m_{rf}^{q-1}(I) = \emptyset$. Let k be the first integer such that $m_{rf}^{k}(I) = \emptyset$. Then $1 \leq k \leq q-1$.

1. Let $E_1 = \{\rho_i | i \epsilon \Lambda_1\}$ be a set of partitions with the properties that $I"rf"\rho_i$ for every $i \epsilon \Lambda_1$ and $\prod_{\Lambda_1} \rho_i = m_{rf}(I)$. Such a set exists since the set $\{\rho | I"rf"\rho\}$ has these properties.

2. Let $E_2 = \{\rho_i | i\epsilon\Lambda_2\}$ be a set of partitions with the properties that $E_2 > E_1$, $m_{rf}(I)$"rf"ρ_i for every $i\epsilon\Lambda_2$ and $\prod_{\Lambda_2} \rho_i = m_{rf}^2(I)$. Again, such a set exists since $\{\rho | m_{rf}(I)$"rf"$\rho\}$ has the desired properties. This follows from Result 18 and the fact that $m_{rf}^2(I) < m_{rf}(I)$.

3. Let $E_k = \{\rho_i | i\epsilon\Lambda_k\}$ be a set of partitions with the properties that $E_k > E_{k-1}$, $m_{rf}^{k-1}(I)$"tf"ρ_i for every $i\epsilon\Lambda_k$ and $\prod_{\Lambda_k} \rho_i = m_{rf}^k(I)$. The set $\{\rho | m_{rf}^{k-1}(I)$"rf"$\rho\}$ has these properties. This follows from Result 18 and the fact that $m_{rf}^k(I) < m_{rf}^{k-1}(I)$.

4. For every $i\epsilon\Lambda_k$ let h_i be the function associated with ρ_i, i.e., $h_i:\{s\} \to \{0,1\}$ and $h_i(a) = h_i(b) \Leftrightarrow \rho_i[a] = \rho_i[b]$. Since $m_{rf}^k(I) = \emptyset = \prod_{\Lambda_k} \rho_i$, h is a 1-1 coding function. Note that the range of h is $\{0,1\}^{\Lambda_k}$. Thus if we let n be the number of elements in Λ_k, to be consistent in notation, then $h:\{s\} \to \{0,1\}^n$.

5. From Result 16 since I"rf"ρ_i for every i in Λ_1 $R_i(y,x) = I_i(f(y,x),x)$ and $S_i(y,x) = H_i(f(y,x),x)$ for every $y \epsilon \{0,1\}^n$. Also, from Result 16 since $\prod_{\Lambda_{r-1}} \rho_i = m_{rf}^{r-1}(I)$ and $m_{rf}^{r-1}(I)$"rf"ρ_i, when $i \epsilon \Lambda_r - \Lambda_{r-1}$ and $2 \leq r \leq k$, then $R_i(y,x) = I_i(y_{\Lambda_{r-1}}, f(y,x), x)$ and $S_i(y,x) = H_i(y_{\Lambda_{r-1}}, f(y,x), x)$. Thus Definition 11 is satisfied and M can be realized with set-reset flip-flops using f for feedback. ‖

For an example of the previous results consider machine C in Figure 6. This machine can be realized without feedback using set-reset flip-flops. This can be seen as follows. Let f be a constant. The only partitions ρ such that I"rf"ρ are $\overline{(1,2,3,5;4)}$ and $\overline{(1,2;3,4,5)}$.

Thus $m_{rf}(I) = \overline{(1,2;3,5;4)}$. If ρ is one of $\overline{(1,2,3,5;4)}, \overline{(1,2;3,4,5)}, \overline{(1,2,3,4;5)}, \overline{(1,4,5;2,3)}$ then $\overline{(1,2;3,5;4)}$"rf"ρ. Thus $m_{rf}^2(I) = \emptyset$ and machine C can be realized without feedback. This is done in Figure 6. Let $\rho_1 = \overline{(1,2;3,4,5)}$, $\rho_2 = \overline{(1,2,3,5;4)}$ and $\rho_3 = \overline{(1,4,5;2,3)}$. Then $\Lambda_1 = \{1,2\}$ and $\Lambda_2 = \{3\}$ satisfies the properties given in the proof of Result 21. In this case $E_1 = \{\rho_1, \rho_2\}$ and $E_2 = \{\rho_3\}$. A coding function h corresponding to ρ_1, ρ_2 and ρ_3 is given in Figure 6.

The following results relate set-reset feedback realizations to unit delay feedback realizations.

Result 22.

Let M be a machine and $f:\{s\} \times \{x\} \rightarrow \{0,1\}^{\ell}$. Let τ and γ be state partitions such that τ"pf"γ. If ρ is a two block state partition such that $\rho \geq \gamma$ then τ"rf"ρ.

Proof.

Let $a, b, \in \{s\}$ and let $x \in \{x\}$ such that $\tau[a] = \tau[b]$, $f(a,x) = f(b,x)$ and $\delta(a,x)$ and $\delta(b,x)$ are specified. Since τ"pf"γ, this implies $\gamma[\delta(a,x)] = \gamma[\delta(b,x)]$ and since $\rho \geq \gamma$ this means $\rho[\delta(a,x)] = \rho[\delta(b,x)]$. Therefore τ"rf"γ. ‖

Observe that any state partition $\gamma = \Pi\{\rho \mid \rho \geq \gamma$ and ρ is a two block partition$\}$. With this in mind we can easily prove the next result.

Inputs

	0	1
1	5	1
2	5	2
3	3	2
4	4	2
5	3	1

δ

f = constant

$m^1_{rf}(I) = (\overline{1,2;3,5;4})$

$m^2_{rf}(I) = (\overline{1;2;3;4;5})$.

Machine C

$h(1) = (0,0,0)$ $\quad R_1 = x \quad S_1 = \bar{x}$

$h(2) = (0,0,1)$ $\quad R_2 = x \quad S_2 = 0$

$h(3) = (1,0,1)$ $\quad R_3 = \bar{x}\bar{y}_1\bar{y}_2 \quad S_3 = y_1\bar{y}_2 + xy_1y_2$

$h(4) = (1,1,0)$

$h(5) = (1,0,0)$

		0	1	0	1	0	1
1	000	100	000	0dd	ddd	100	000
2	001	100	001	0d1	dd0	100	00d
3	101	101	001	0d0	1d0	d0d	00d
4	110	110	001	00d	110	dd0	001
5	100	101	000	0d0	1dd	dd1	00d
		(Y_1,Y_2,Y_3)		(R_1,R_2,R_3)		(S_1,S_2,S_3)	

Figure 6.

Result 23.

Let M be a machine and $f:\{s\} \times \{x\} \to \{0,1\}^{\ell}$. Let τ be a state partition. Then $m_{pf}(\tau) \geq m_{rf}(\tau)$.

Proof.

Let γ be a partition such that τ "pf" γ. Let ρ be a 2 block partition greater than γ. Then τ "rf" ρ from Result 22, which implies $m_{pf}(\tau) \geq m_{rf}(\tau)$. ||

This implies immediately a result relating set-reset flip-flop realizations to unit delay realizations. Results 19 and 23 imply immediately that for every i in $\{1,2,\ldots\}$ that $m^{i}_{pf}(\tau) \geq m^{i}_{rf}(\tau)$ because $m_{pf}(\tau) \geq m_{rf}(\tau)$ from Result 23 and, if $m^{i-1}_{pf}(\tau) \geq m^{i-1}_{rf}(\tau)$ then $m_{pf}[m^{i-1}_{pf}(\tau)] \geq m_{rf}[m^{i-1}_{pf}(\tau)] \geq m_{rf}(m^{i-1}_{rf}(\tau))$ from Results 19 and 23. This implies the following result.

Result 24 (Theorem).

If machine M can be realized with unit delays using f for feedback, then M can be realized with set-reset flip-flop using f for feedback.

Proof.

The hypothesis implies from Result 3 that $m^{q-1}_{pf}(I) = \emptyset$ where q is the number of states. But this implies $m^{q-1}_{rf}(I) = \emptyset$. From Result 21 this implies the theorem. ||

It should be noticed in Figure 6 that machine C cannot be realized with unit delays using f = constant for feedback. To determine if a machine M can be realized using f for feedback, it is necessary to compute $m_{rf}(\tau)$ for various τ which is not an easy problem. In general one must consider all two block partitions ρ to see if τ "rf" ρ. For a q state machine there are $2^{q-1}-1$ such partitions. It is wise to compute $m_{pf}(\tau)$ first. From Result 23 we know that $m_{rf}(\tau) \leq m_{pf}(\tau)$, therefore, it is not necessary to consider these ρ such that $\rho \geq m_{pf}(\tau)$.

In this paper we have developed a method for determining when a machine can be realized using a function f for feedback with either set-reset or trigger flip-flop memory elements. This method is more difficult to apply than the one given in Reference 3 for the unit delay case. We have also shown in Results 12 and 24 that for a given machine its feedback properties will be different for trigger, set-reset and unit delay type realizations. Thus in general one must first decide the type of memory element he wants to use before making a study of the feedback characteristics of a machine.

REFERENCES

[1] Ginzburg, A., and Yoeli, M., "Products of Automata and the Problem of Covering", Technion, Report No. 15 (July 1963).

[2] Hartmanis, J., "On the State Assignment Problem for Sequential Machines, I.", IRE Trans. on Electronic Computers, EC-10, pp. 157-165 (1961).

[3] Hartmanis, J., and Stearns, R. E., Algebraic Structure Theory of Sequential Machines, Prentice-Hall, 1966 .

[4] Hartmanis, J., and Stearns, R. E., "A Study of Feedback and Errors in Sequential Machines", IRE Trans. on Electronic Computers, EC-12 (June 1963).

[5] Harlow, C. A. and Coates, C. L., "On the Structure of Realizations Using Flip-Flop Memory Elements", Information & Control, vol. 10, pp. 159-174 (February 1967).

[6] Harlow, C. A. and Coates, C. L., "On the Structure of Sequential Machine Realizations", Technical Report No. 27, Laboratories for Electronics and Related Science Research, The University of Texas, Austin, Texas (July 19, 1967).

[7] Liu, C. L., "Some Memory Aspects of Finite Automata", MIT Research Laboratory of Electronics, Report 411 (May 1963).

[8] Stearns, R. E., and Hartmanis, J., "On the State Assignment Problem for Sequential Machines, II.", IRE Trans. on Electronic Computers, EC-10, pp. 593-603 (December 1961).

[9] Yoeli, M., "The Cascade Decomposition of Sequential Machines", IRE Trans. on Electronic Computers, EC-10, pp. 587-592 (December 1961).

INESSENTIAL ERRORS IN SEQUENTIAL MACHINES*

In papers by Hartmanis and Stearns (Reference 1 and 2) the concept of an inessential error is defined and some of the properties of inessential errors are derived. An error partition Π_E is defined and investigated. However, it is not shown in these papers how to calculate Π_E. The purpose of this paper is to give an algorithm for determining Π_E. First we shall review some concepts that are given in Reference 1.

Definition 1

A Moore type sequential machine is a quintuple $M = (\{s\}, \{x\}, \{0\}, \delta, \lambda)$, $\{s\}$ is a finite set called the set of states, $\{x\}$ is the set of inputs, $\{0\}$ is the set of outputs, $\delta:\{s\} \times \{x\} \rightarrow \{s\}$ and $\lambda:\{s\} \rightarrow \{0\}$.

In this paper the only machines we will consider are Moore machines which are completely specified; that is, the domain of δ is all of $\{s\} \times \{x\}$. The next definition extends the function δ to all sequences of inputs.

Definition 2

Let M be a sequential machine. Let $\{x_i\}_1^j$ be a sequence of inputs of length $j \geq 0$. We define a function $\bar{\delta}$ as follows. Let $a \in \{s\}$

$\bar{\delta}(a,\{x_i\}_1^j) = a$ if $j = 0$, $\bar{\delta}(a,\{x_i\}_1^j) = \delta(a,x_1)$ if $j = 1$. In general $\bar{\delta}(a,\{x_i\}_1^j) = \delta(\bar{\delta}(a,\{x_i\}_1^{j-1}),x_j)$ for every $j \geq 2$.

Definition 3

Let M be a sequential machine. Let $j \geq 0$, $\{x_i\}_1^j$ be a sequence of inputs and let $a \in \{s\}$. Then we define $\bar{\lambda}(a,\{x_i\}_1^j) = \lambda(\bar{\delta}(a,\{x_i\}_1^j))$.

Definition 4

Let M be a sequence machine.

i) Let τ be a partition on $\{s\}$. If $a,b \in \{s\}$, $\tau[a] = \tau[b]$ means a and b are in the same set, sometimes called a block, of τ. Moreover $A \in \tau$ means A is a block of τ.

Example 1. If $\{s\} = \{1,2,3,4,5\}$ and $\tau = (\overline{1,2};\overline{3,4,5})$ then $\tau[3] = \tau[4] = \tau[5]$ and $\tau[2] \neq \tau[3]$.

ii) An S.P. partition τ is a partition in $\{s\}$ such that for every $a,b \in \{s\}$ with $\tau[a] = \tau[b]$ then $\tau[\delta(a,x)] = \tau[\delta(b,x)]$ for every input x.

Definition 5

Let M be a machine. An error is a partition τ_{ab} where $a,b \in \{s\}$ and a and b are the only two states in the same block of τ_{ab}.

Example 2. If $\{s\} = \{1,2,3,4,5\}$ then $\tau_{13} = (\overline{1,2};\overline{2};\overline{4};\overline{5})$.

Definition 6

An error τ_{ab} is an inessential error iff there exists a finite set $\Delta \leq \{1,2,\ldots\}$ for every input sequence $\{x_i\}_1^\infty$ such that $\bar{\lambda}(a,\{x_i\}_1^k) \neq \bar{\lambda}(b,\{x_i\}_1^k)$ if and only if $k \in \Delta$.

Definition 7

Let $\Pi_E = \sum\{\tau_{ab} | \tau_{ab}$ is an inessential error$\}$.

It is proved in Reference 1 that if τ_{ab} and τ_{bc} are inessential error, then τ_{ac} is an inessential error. This implies the next result.

Result 1 (Theorem)

Let M be a machine. If $\tau_{ab} \leq \Pi_E$ then τ_{ab} is an inessential error and conversely if τ_{ab} is an inessential error then $\tau_{ab} \leq \Pi_E$.

We conclude the introductory concepts with a brief discussion of set systems. It turns out that the set system is the principal concept to be used in determining an algorithm for computing Π_E.

Definition 8

A set system on $\{s\}$ is a collection $\rho = \{A_i | i \in \Lambda\}$ where Λ is a finite index set and $A_i \leq \{s\}$ for every $i \in \Lambda$. Also

i) $\bigcup_{\Lambda} A_i = \{s\}$

ii) $A_i \geq A_j$ implies that $A_i = A_j$ for every $i, j \in \Lambda$.

Given a set system ρ, we say $\rho[a] = \rho[b]$ for two states a, b if a and b are both in some set of ρ. It should be observed that a partition is a set system.

Definition 9

Let M be a Moore machine

i) Let $E = \{[a,b] | a, b \in \{s\}$ and $\lambda(a) \neq \lambda(b)\}$

ii) Let $K = \{[a,b] | a, b \in \{s\}$ with $a \neq b\}$

Definition 10

i) If τ and ρ are set systems on $\{s\}$ such that $\delta(A,x) \leq B$ for every $A \in \tau$ and $x \in \{x\}$ where B is some number of ρ, then we say τ "p" ρ. Note that $\delta(A,x) = \{b \mid b = \delta(a,x)$ for some $b \in A\}$.

ii) We say τ is an S.P. set system if τ "p" τ.

iii) If τ is a set system on $\{s\}$, we define $m_{ss}(\tau) = \Pi\{\rho \mid \tau$ "p" $\rho\}$. Note that if $B \in m_{ss}(\tau)$ and B is not a singleton, $B = \delta(A,x)$ for some $A \in \tau$ and $x \in \{x\}$.

iv) $m_{ss}^{i+1}(\tau) = m_{ss}(m_{ss}^{i}(\tau))$ for every integer $i \geq 1$.

Result 2

If τ is an S.P. set system on $\{s\}$, then $m_{ss}(\tau) \leq \tau$ and $m_{ss}^{i+1}(\tau) \leq m_{ss}^{i}(\tau)$ for every $i \geq 1$.

Proof:

Since τ has S.P., τ "p" τ which implies $\tau \leq \Pi\{\rho \mid \tau$ "p" $\rho\}$. This implies $m_{ss}(\tau) \leq \tau$. The second part of the result is easily proved by induction.

Figure 1 contains an example of these concepts. In Figure 1 τ has S.P. and the m_{ss} operators on τ are computed. For this example $E = \{[1,2],[1,4],[2,3],[2,5],[3,4],[4,5]\}$.

At this point we turn to the problem of computing Π_E. Thus far all we know is that Π_E is a sum of inessential error partitions which is not a good characterization as far as its computation is concerned.

	0	1	λ	I
1	2	4	0	$\tau = \overline{(1,2,4,5;3)}$
2	2	5	1	$m^1_{ss}(\tau) = \overline{(1,2,4;2,4,5;3)}$
3	3	1	0	$m^2_{ss}(\tau) = \overline{(1,2,4;2,4,5;3)}$
4	4	2	1	φ
5	1	4	0	

Figure 1. Machine A

Result 3 which follows is one of the principal results of this paper in that it gives a necessary condition that Π_E must satisfy. In the following results when we write [a,b] = [c,d] this means a = c and b = d or a = d and b = c.

Result 3 (Theorem)

Let M be a q state machine where $q \geq 2$. Let $p = \binom{q}{2}$ the combination coefficient. If τ is a partition of {s} such that $\tau \leq \Pi_E$ and τ has S.P., then for every [a,b] $\in E$ such that $\tau[a] = \tau[b]$ we have that $m^i_{ss}(\tau_{ab})[a] \neq m^i_{ss}(\tau_{ab})[b]$ for every i such that $1 \leq i \leq p$.

Proof:

Suppose there exists [a,b] $\in E$ such that $\tau[a] = \tau[b]$ and an integer with $1 \leq i \leq p$ where $m^i_{ss}(\tau_{ab})[a] = m^i_{ss}(\tau_{ab})[a]$. This implies that there exists states a_{i-1}, b_{i-1}, and an x_i such that $m^{i-1}_{ss}(\tau_{ab})[a_{i-1}] = m^{i-1}_{ss}(\tau_{ab})[b_{i-1}]$ and $[\delta(a_{i-1},x_i), \delta(b_{i-1},x_i)] = [a,b]$. Continue in this fashion until we have a_1,b_1 such that $m^1_{ss}(\tau_{ab})[a_1] = m^1_{ss}(\tau_{ab})[b_1]$. This implies that there exists x_1 such that $[a_1,b_1] = [\delta(a,x_1), \delta(b,x_1)]$. Thus we have a sequence $\{x_j\}^i_1$ such that $[\bar{\delta}(a,\{x_j\}^i_1), \bar{\delta}(b,\{x_j\}^i_1)] = [a,b]$. Form the sequence $\{y_i\}^\infty_1$ as follows.

$y_n = x_n$ if $1 \leq n \leq i$

$y_n = x_{n-i}$ if $n > i$

Let $\Lambda = \{r | r = ni,\ n \text{ a positive integer}\}$. If $k \epsilon \Lambda$ then $k = ni$ which implies $[\bar{\delta}(a, \{x_j\}_1^{ni}),\ \bar{\delta}(b, \{x_j\}^{ni})] = [a,b]$. Which in turn implies that $\bar{\lambda}(a, \{x_i\}_1^{ni}) = \lambda(a)$ and $\bar{\lambda}(b, \{x_i\}_1^{ni}) = \lambda(b)$. Since $[a,b] \epsilon E$, this implies $\bar{\lambda}(a, \{x_i\}_1^{k}) \neq \bar{\lambda}(b, \{x_i\}_1^{k})$ for every $k \epsilon \Lambda$. Since Λ is an infinite set, this means τ_{ab} is not an inessential error. Thus from Result 1 τ_{ab} is not less than Π_E. But since $\tau[a] = \tau[b]$ and $\Pi_E \geq \tau$, this is a contradiction which proves the theorem. ‖

The remainder of this paper will be concerned with proving the converse of Result 3 which is stated in Result 6. The proof of the converse is fairly involved. For this reason we will isolate certain parts of the proof with the following lemmas.

Result 4 (Lemma)

Let M be a machine. If $a,b \epsilon \{s\}$ and $\Pi_E[a] \neq \Pi_E[b]$ then there exists a sequence of inputs $\{x_i\}_1^{\infty}$ and $[a_o, b_o] \epsilon E$ such that $[a_o, b_o] = [\delta(a, \{x_i\}_1^{k}),\ \delta(b, \{x_i\}_1^{k})]$ for every $k \epsilon J_o$ where J_o is an infinite subset of $\{1,2,\ldots\}$.

Proof:

Since $\Pi_E[a] \neq \Pi_E[b]$, we know that τ_{ab} is not an inessential error from Result 1. This implies there exists $\{x_i\}_1^{\infty}$ such that $\lambda(a, \{x_i\}_1^{k}) \neq \lambda(b, \{x_i\}_1^{k})$ for every $k \epsilon J'$ where J' is an infinite subset of $\{1,2,\ldots\}$. Let $a_k = \bar{\delta}(a, \{x_i\}_1^{k})$ and $b_k = \bar{\delta}(b, \{x_i\}_1^{k})$. Let $J_{[c,d]} = \{k \epsilon J' \,|\, [c,d] = [a_k, b_k]\}$. $J' \leq \bigcup_{[c,d] \epsilon} J_{[c,d]}$ since if $k \epsilon J'$ this implies $[a_k, b_k] \epsilon E$ or

$[a_k, b_k] = [c,d] \in E$. This implies $k \in J_{[c,d]}$ which implies that $k \in \bigcup_{[c,d] \in E} J_{[c,d]}$. Since J' is infinite, this means $J_{[a_o, b_o]} = J_o$ is infinite for some $[a_o, b_o] \in E$. Thus $[a_o, b_o] = [a_k, b_k] = [\delta(a, \{x_i\}_1^k), \delta(b, \{x_i\}_1^k)]$ for every $k \in J_o$ which is an infinite set. ‖

Result 5 (Lemma)

Let M be a q state machine with $q \geq 2$. Let $p = \binom{q}{2}$. Further suppose there exists $\{x_i\}_1^k$ a sequence of inputs with $k \geq 1$ such that $[a_o, b_o] = [\delta(a, \{x_i\}_1^k), \delta(b, \{x_i\}_1^k)]$ where a_o, b_o, a, and b are states of M. Then there exists a sequence of inputs $\{y_i\}_1^\ell$ where $1 \leq \ell \leq p$ and $1 \leq \ell \leq k$ and $[a_o, b_o] = [\delta(a, \{y_i\}_1^\ell), \delta(b, \{y_i\}_1^\ell)]$.

Proof:

i) If $k \leq p$ then the theorem is satisfied. Suppose $k > p$. For n such that $0 \leq n \leq k$ let $B_n = \{[c,d] \mid c \neq d, [c,d] = [\delta(a, \{x_i\}_1^j), \delta(b, \{x_i\}_1^j)]$ for some j such that $0 \leq j \leq n\}$. Clearly $B_{n+1} \geq B_n$. Also $B_{n+1} = B_n$ iff $0 \leq n \leq k-1$ and $[\delta(a, \{x_i\}_1^{n+1}), \delta(b, \{x_i\}_1^{n+1})] = [\delta(a, \{x_i\}_1^j), \delta(b, \{x_i\}_1^j)]$ for some j such that $0 \leq j \leq n$. Note that for every n $B_n \leq K$ and the cardinality of K is p. This implies that there exists integer r such that $B_{r+1} = B_r$ where $0 \leq r \leq p-1$. Let r_{min} be the minimum such r and let $\ell_1 = r_{min} + 1$. Then $1 \leq \ell_1 \leq p$ and $B_{\ell_1} = B_{\ell_1 - 1}$. This implies that there exists j_1 such that $0 \leq j_1 \leq \ell_1 - 1$ and $[\delta(a, \{x_i\}_1^{j_1})] = [\delta(b, \{x_i\}_1^{\ell_1}), \delta(b, \{x_i\}_1^{\ell_1}]$. Note that $p - 1 \geq \ell_1 - j_1 \geq 1$ and $k - (\ell_1 - j_1) > p - (\ell_1 - j_1) \geq j_1 \geq 0$. Let $k_1 = k - (\ell_1 - j_1)$. Then $k > k_1 \geq 1$. Form the sequence $\{x_{i,1}\}_1^{k_1}$ as follows. Let

$x_{i,1} = x_i$ if $1 \leq i \leq j_1$

$x_{i,1} = x_i + (\ell_1 - j_1)$ if $j_1 < i \leq k_1$.

Show $[\delta(a,\{x_{i,1}\}_1^{k_1}), \delta(b,\{x_{i,1}\}_1^{k_1})] = [a_o,b_o]$. From the definition of $\{x_{i,1}\}_1^{k_1}$ $[\delta(a,\{x_{i,1}\}_1^{j_1}), \delta(b,\{x_{i,1}\}_1^{j_1})] = [\delta(a,\{x_i\}_1^{j_1}), \delta(b,\{x_i\}_1^{j_1})] = [\delta(a,\{x_i\}_1^{\ell_1}), \delta(b,\{x_i\}_1^{\ell_1})]$. Thus if $j_1 < r < k_1$, since for i such that $j_1 < i \leq k_1$ $x_{i,1} = x_i + (\ell_1 - j_1)$, $[\delta(a,\{x_{i,1}\}_1^{r}), \delta(b,\{x_{i,1}\}_1^{r})] = [\delta(\delta(a,\{x_{i,1}\}_1^{j_1}), \{x_{i,1}\}_{j_1+1}^{r}), \delta(\delta(b,\{x_{i,1}\}_1^{j_1}), \{x_{i,1}\}_{j_1+1}^{r})] = [\delta(\delta(a,\{x_i\}_1^{\ell_1}), \{x_i\}_{\ell_1+1}^{r+\ell_1-j_1}), \delta(\delta(b,\{x_i\}_1^{\ell_1}), \{x_i\}_{\ell_1+1}^{r+\ell_1-j_1})] = [\delta(a,\{x_i\}_1^{r+\ell_1-j_1}), \delta(b,\{x_i\}_1^{r+\ell_1-j_1})]$. If we let $r = k_1$, we get that $[\delta(a,\{x_{i,1}\}_1^{k_1}), \delta(b,\{x_{i,1}\}_1^{k_1})] = [\delta(a,\{x_i\}_1^{k}), \delta(b,\{x_i\}_1^{k})] = [a_o,b_o]$. Thus $\{x_{i,1}\}_1^{k_1}$ is a sequence of inputs such that $1 \leq k_1 < k$ and $[\delta(a,\{x_{i,1}\}_1^{k_1}), \delta(b,\{x_{i,1}\}_1^{k_1})] = [a_o,b_o]$.

ii) If $k_1 \leq p$, then the lemma is satisfied. If not, repeat step i) and get a sequence $\{x_{i,2}\}_1^{k_2}$ where $1 \leq k_2 < k_1$ and $[\delta(a,\{x_{i,2}\}_1^{k_2}), \delta(b,\{x_{i,2}\}_1^{k_2})] = [a_o,b_o]$. Continue in this manner until for some k_j we have $k_j \leq p$. Let $\ell = k_j$. Then the sequence $\{x_{i,j}\}_{i=1}^{k_j} = \{y_i\}_{i=1}^{\ell}$ satisfies the lemma. ‖

The next result is the converse of Result 3. Its proof follows easily from the two preceeding lemmas.

Result 6 (Theorem)

Let M be a q state Moore Machine where $q \geq 2$. If τ is a S.P. partition on $\{s\}$ and if for every $[a,b] \in E$ such that $\tau[a] = \tau[b]$ we have

that $m^i_{ss}(\tau_{ab})[a] \neq m^i_{ss}(\tau_{ab})[b]$ for every i such that $1 \leq i \leq p$ then $\tau \leq \Pi_E$.

Proof:

Suppose τ is not less than Π_E. Then there exists a,b such that $\tau[a] = \tau[b]$ and $\Pi_E[a] \neq \Pi_E[b]$. From Result 4 this implies that there exists a sequence of inputs $\{x_i\}_1^\infty$ and $[a_o,b_o] \in E$ such that $[a_o,b_o] = [\delta(a,\{x_i\}_1^k), \delta(b,\{x_i\}_1^k)]$ for every $k \in J_o$ an infinite subset of $\{1,2,3,\ldots\}$. Let k_1 be the minimum element of J_o. This means that $m^{k_1}_{ss}(\tau)[a_o] = m^{k_1}_{ss}(\tau)[b_o]$ which implies $\tau[a_o] = \tau[b_o]$ since $\tau \geq m^{k_1}_{ss}(\tau)$ from Result 2. Let k_2 be the minimum element of $J_o - \{k_1\}$. Then $[a_o,b_o] = [\delta(a,\{x_i\}_1^{k_2}), \delta(b,\{x_i\}_1^{k_2})] = [\delta(\delta(a,\{x_i\}_1^{k_1}), \{x_i\}_{k_1+1}^{k_2}), \delta(\delta(b,\{x_i\}_1^{k_1}), \{x_i\}_{k_1+1}^{k_2})] = [\delta(a_o,\{x_i\}_{k_1+1}^{k_2}), \delta(b_o,\{x_i\}_{k_1+1}^{k_2})]$. Thus we have a sequence $\{x_i\}_{k_1+1}^{k_2}$ of length k_2-k_1 such that $[a_o,b_o] = [\delta(a_o,\{x_i\}_{k_1+1}^{k_2}), \delta(b_o,\{x_i\}_{k_1+1}^{k_2})]$. From Result 5 there exists $\{y_i\}_1^\ell$ such that $[a_o,b_o] = [\delta(a_o,\{y_i\}_1^\ell), \delta(b_o,\{y_i\}_1^\ell)]$ and $1 \leq \ell \leq p$. This implies $[a_o,b_o]$ $m^\ell_{ss}(\tau_{a_ob_o})$. Since $[a_o,b_o]$ E and $\tau[a_o] = \tau[b_o]$ this is a contradiction. ‖

Results 3 and 6 imply the following Theorem which is an algorithm for finding Π_E.

Result 7 (Theorem)

Let M be a q state machine with $q \geq 2$. Let $F = \{\tau \mid \tau$ satisfies i and ii below$\}$.

i) τ is an S.P. partition of $\{s\}$.

ii) For every $[a,b] \in E$ such that $\tau[a] = \tau[b]$ $m^i_{ss}(\tau_{ab})[a] \neq m^i_{ss}(\tau_{ab})[b]$ when $1 \leq i \leq p$. Then there is a largest partition in F and this partition is Π_E.

Proof:

From Result 3 $\Pi_E \in F$. If $\tau \in F$, then $\tau \leq \Pi_E$ from 6. ‖

Therefore to compute Π_E examine the S.P. partitions of $\{s\}$ beginning with the largest ones. Next check these partitions for property ii) above. The largest partition which satisfies ii) is Π_E.

We conclude this paper with some examples of Result 7. Consider machine A in Figure 1. The only S.P. partitions are those in the figure. We compute F. Clearly $\emptyset$ the zero partition is in F. Consider $\tau = \overline{(1,2,4,5;3)}$. Note that $[1,2] \in E$ and $\tau[1] = \tau[2]$. $m^1_{ss}(\tau_{12}) = (\overline{1;2;3;4,5})$, $m^2_{ss}(\tau_{12}) = \overline{(1,4;2,4;3;5)}$, $m^3_{ss}(\tau_{12}) = \overline{(2,4;2,5;3;1)}$ and $m^4_{ss}(\tau_{12}) = \overline{(3;2,4;2,5;1,2;4,5)}$. Since $p = \binom{5}{2} = 10$ and $[1,2] \in m^4_{ss}(\tau_{12})$, this implies $\tau \in F$. Thus $F = \{\emptyset\}$ and $\Pi_E = \emptyset$.

Consider machine B in Figure 2. All the S.P. partitions are given in the figure. We again want to compute F. Since $[4,5] \in E$ and $m^1_{ss}(\tau_{45})[4] = m^1_{ss}(\tau_{45})[5]$, clearly $I \in F$. Consider τ_1. The only states a,b such that $\tau_1[a] = \tau[b]$ and $[a,b] \in E$ are $\{[1,3],[2,3],[3,4]\}$. Consider these pairs. $m^1_{ss}(\tau_{13}) = \emptyset$ which implies that $m^i_{ss}(\tau_{13}) = \emptyset$ for every i. $m_{ss}(\tau_{23}) = \overline{(1,2;1,3;4;5)}$, $m^2_{ss}(\tau_{23}) = \overline{(1,2;1,3;4;5)}$. Since $m^2_{ss}(\tau_{23}) = m_{ss}(\tau_{23})$, we can stop. $m_{ss}(\tau_{34}) = \overline{(2,4;1,3,5)}$, $m^2_{ss}(\tau_{34}) = \overline{(1,4,2;3;5)}$, $m^3_{ss}(\tau_{34}) = \overline{(2,4;1,3;5)}$ and again we can stop. By noting that state pairs $\{1,3\},\{2,3\}$

	Inputs		Output	
	0	1		I
1	2	1	0	$\tau_1 = \overline{(1,2,3,4;5)}$
2	1	3	0	$\tau_2 = \overline{(1,2,3,5;4)}$
3	2	1	1	$\tau_3 = \overline{(1,2,3;4,5)}$
4	4	3	0	$\tau_4 = \overline{(1,2,3;4;5)}$
5	5	2	1	$\emptyset$

Figure 2. Machine B

and $\{3,4\}$ are not in the same block of $m^i_{ss}(\tau_{13})$, $m^i_{ss}(\tau_{23})$ and $m^i_{ss}(\tau_{34})$ respectively for any i we conclude that $\tau_1 \epsilon F$. Since there can be no other partitions larger than τ_1 in F, this implies that $\tau_1 = \Pi_E$.

References

[1] J. Hartmanis and R. E. Stearns (1966), Algebraic Structure Theory of Sequential Machines, Prentice-Hall.

[2] J. Hartmanis and R. E. Stearns (1963), "A Study of Feedback and Errors in Sequential Machines, IRE Trans. on Electronic Computers, EC-12.

[3] J. Hartmanis (1961), "On the State Assignment Problem for Sequential Machines, I", IRE Trans. on Electronic Computers, EC-10.

REALIZATION OF SEQUENTIAL MACHINES WITH THRESHOLD ELEMENTS

I

INTRODUCTION

Sequential Machines

A finite-state, synchronous, sequential machine consists of three finite sets and two functions. The three sets are the input set I, the state set S, and the output set O. The two functions are the next-state function f^S and the output function f^O. Both functions are defined on the set of state-input pairs, SxI. Their ranges are respectively S and O.

In order to realize a machine, binary codes are first assigned to each of the inputs, states and outputs. The functions which assign the binary codes are called assignment functions. They are one-one. The assignment function for the input set is denoted by A^I. Thus we have A^I: $I \rightarrow \{0,1\}^{n_1}$. Similarly, for the state and output sets, we have the assignment functions A^S: $S \rightarrow \{0,1\}^{n_2}$ and A^O: $O \rightarrow \{0,1\}^{n_3}$. The set $\mathcal{A} = \{A^I, A^S, A^O\}$ is called an assignment.

Two functions are induced by the assignment of binary codes. The coded next-state function, $\hat{f}^S$, maps the ordered pair consisting of the codes for S_i and I_j onto the code for $f^S(S_i, I_j)$. This same pair is mapped onto the code for $f^O(S_i, I_j)$ by $\hat{f}^O$, the coded output function. The induced functions are defined formally by the following two equations:

I. $\hat{f}^S(A^S(S_i), A^I(I_j)) = A^S(f^S(S_i, I_j))$

II. $\hat{f}^O(A^S(S_i), A^I(I_j)) = A^O(f^O(S_i, I_j))$

The tables for $\hat{f}^S$ and $\hat{f}^O$ are obtained from the tables for f^S and f^O by replacing the inputs, states and outputs by their codes. The component function $\hat{f}_k^S$ of the coded next -state function is called the kth state-variable function. The component function $\hat{f}_k^O$ is called the kth output-variable function.

Of primary concern is the behavior of a sequential machine. The behavior is the response or output sequence due to some initial state and an input sequence. Let x, y, z be respectively variables over the code sets $A^I(I)$, $A^S(S)$ and $A^O(O)$. If y^O is the code for the initial state and x^t is the code for (t)th term of the input sequence, then the (t)th terms of the corresponding state and output sequences are given by:

$$y^t = \hat{f}^S(y^{t-1}, x^{t-1}),\ t \geq 1 \quad \text{and}$$

$$z^t = \hat{f}^O(y^t, x^t),\ t \geq 0.$$

Objective

In this paper, we are concerned with finding the one-level assignments $\mathcal{A}$ for a sequential machine. Here $\mathcal{A}$ is one-level if every component function of $\hat{f}^S$ and $\hat{f}^O$ is threshold. To illustrate this, two threshold gate realizations for a sequential machine M, corresponding to two different assignments, are presented in Example 1.

EXAMPLE 1 - A finite-state, synchronous sequential machine M.

$I = \{I_1, I_2, I_3, I_4\}$, $S = \{S_1, S_2, S_3, S_4\}$,

$O = \{O_1, O_2, O_3, O_4\}$

f^S	I_1	I_2	I_3	I_4
S_1	S_2	S_3	S_1	S_4
S_2	S_2	S_2	S_1	S_4
S_3	S_2	S_3	S_1	S_4
S_4	S_4	S_2	S_4	S_4

f^O	I_1	I_2	I_3	I_4
S_1	O_1	O_1	O_3	O_2
S_2	O_1	O_1	O_3	O_2
S_3	O_4	O_4	O_3	O_2
S_4	O_2	O_3	O_2	O_2

Assignment $\mathcal{A}_1$

$\hat{f}^S$ / $A^S(S_j)$ \ $A^I(I_j)$	00	01	10	11
00	01	10	00	11
01	01	01	00	11
10	01	10	00	11
11	11	01	11	11

$\hat{f}^O$ / $A^S(S_j)$ \ $A^I(I_j)$	00	01	10	11
00	00	00	10	01
01	00	00	10	01
10	11	11	10	01
11	01	10	01	01

$A^O(O_1) = 00$

$A^O(O_2) = 01$

$A^O(O_3) = 10$

$A^O(O_4) = 11$

State-variable functions:

$$\hat{f}_1^S(y^t, x^t) \triangleq y_1^{t+1} = x_1^t x_2^t + x_2^t \bar{y}_2^t + y_1^t y_2^t \bar{x}_2^t$$

$$\hat{f}_2^S(y^t, x^t) \triangleq y_2^{t+1} = \bar{x}_1^t \bar{x}_2^t + \bar{x}_1^t y_2^t + x_1^t x_2^t + x_1^t y_1^t y_2^t$$

Output-variable functions:

$$\hat{f}_1^O(y^t, x^t) \triangleq z_1^t = y_1^t \bar{y}_2^t \bar{x}_1^t + y_1^t \bar{x}_1^t x_2^t + \bar{y}_1^t x_1^t \bar{x}_2^t + \bar{y}_2^t x_1^t \bar{x}_2^t$$

$$\hat{f}_2^O(y^t, x^t) \triangleq z_2^t = y_1^t \bar{x}_1^t \bar{x}_2^t + y_1^t \bar{y}_2^t \bar{x}_1^t + x_1^t x_2^t + x_1^t y_1^t y_2^t$$

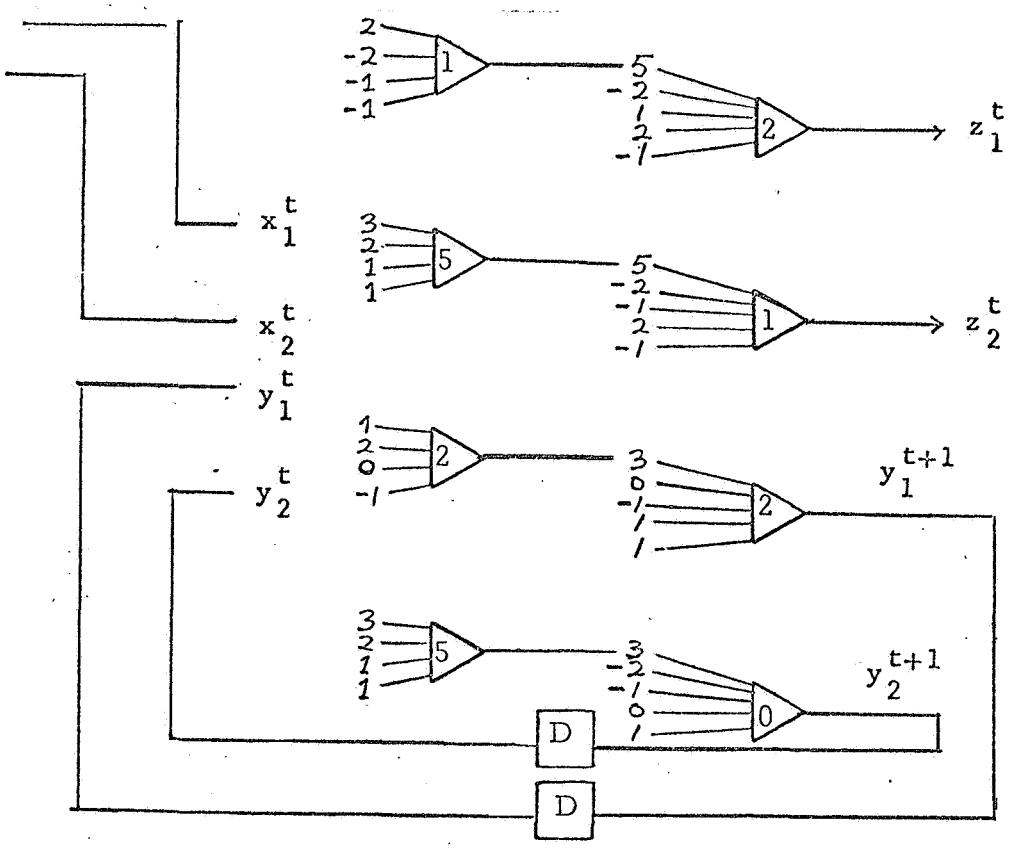


Figure 1. Realization of M for Assignment A_1.

Assignment A_2

$A^S(S_j)$

$\hat{f}^S$	$A^I(I_j)$ 00	01	10	11
00	11	01	00	10
11	11	11	00	10
01	11	01	00	10
10	10	11	10	10

$A^S(S_j)$

$\hat{f}^O$	$A^I(I_j)$ 00	01	11	10
00	000	000	001	011
11	000	000	001	011
01	100	100	001	011
10	011	001	011	011

$A^O(O_1) = 000$

$A^O(O_2) = 011$

$A^O(O_3) = 001$

$A^O(O_4) = 100$

State-variable functions

$$\begin{cases} \hat{f}^S_1(y^t, x^t) \triangleq y_1^{t+1} = y_1^t(\bar{x}_1^t + \bar{y}_2^t) + \bar{x}_2^t \\ \hat{f}^S_2(y^t, x^t) \triangleq y_2^{t+1} = \bar{x}_1^t(\bar{y}_1^t + y_2^t) \end{cases}$$

Output-variable functions

$$\begin{cases} \hat{f}^O_1(y^t, x^t) = z_1^t = \bar{y}_1^t y_2^t \bar{x}_1^t \\ \hat{f}^O_2(y^t, x^t) = z_2^t = y_1^t \bar{y}_2^t (x_1^t + \bar{x}_2^t) + x_1^t \bar{x}_2^t \\ \hat{f}^O_3(y^t, x^t) = z_3^t = y_1^t \bar{y}_2^t + x_1^t \end{cases}$$

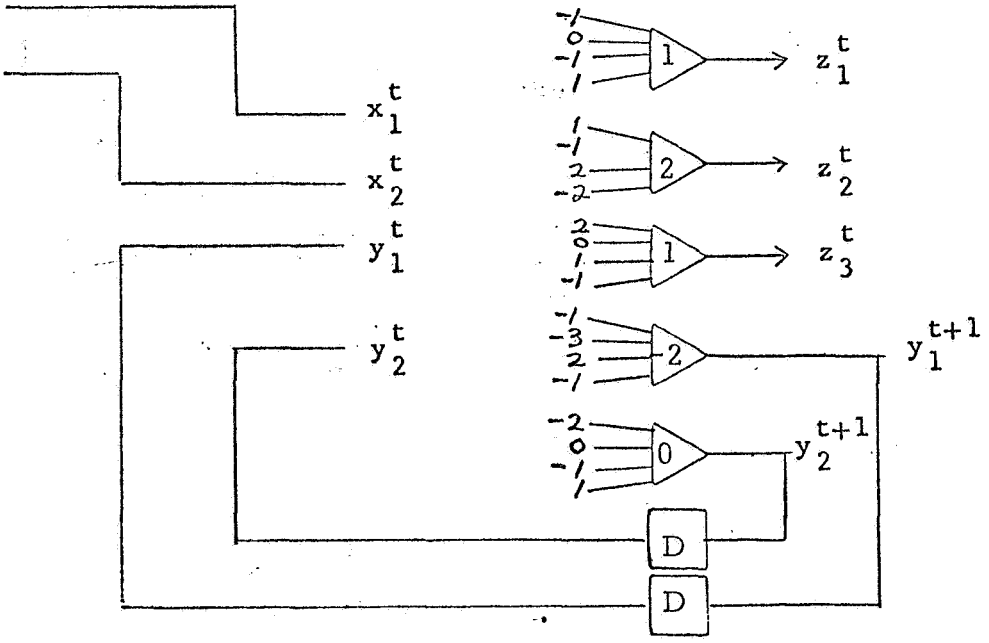


Figure 2. Realization for M for Assignment A_2.

The obstacle confronted in attempting to force the $\hat{f}^S_i$'s and $\hat{f}^O_j$'s to be threshold functions is that of characterization. By definition, a Boolean function F is threshold iff there exist an n-tuple $\vec{a}$ with real components and a real number T_r such that for every point P of the n-cube, $F(P) = 1$ iff $\vec{a} \cdot P \geq T_r$. The gate having the parameters $\vec{a}$, T_r then realizes F. Determination of $\vec{a}$ and T_r requires use of a lengthy algorithm [1], [3]-[14].

Thus the definition implies no characteristics of a Boolean function which make it readily recognizable as being a threshold function. Because of this, the following approach seems a reasonable one.

An algorithm is developed which yields all code assignments for which the state-variable and output-variable functions satisfy a necessary condition that they be threshold functions. This set of code assignments will contain the set of one-level assignments.

A necessary condition that a Boolean function F be a threshold function is that of domain 2-asummability. Let F be defined on $\{0,1\}^n$. Then F is 2-asummable iff for every pair of points P_1, P_2 which map onto zero and every pair of points P_3, P_4 which map onto one, the pairs are unequal in sum (i.e., $P_1 + P_2 \neq P_3 + P_4$). It is well known that if F is defined on the n-cube with $n \leq 7$, F is threshold iff F is 2-asummable. For $n > 7$, it is a necessary condition.

An incompletely specified function defined on the n-cube is domain 2-asummable if, over its domain, no pair of points which map onto zero is equal in sum to a pair which map onto one. Clearly this is necessary in order that it have a 2-asummable extension to the n-cube. If it does have such an extension, we will call the incompletely specified function 2-asummable.

If $\mathcal{A} = \{A^I, A^S, A^O\}$ is an assignment for some machine, then $\mathcal{A}$ will be called an acceptable assignment if each state-variable and output-variable is domain 2-asummable.

II

CHARACTERIZATION OF THE ACCEPTABLE CODE ASSIGNMENTS

Assignment of Codes According to Partition Sets

In this paper a partition of a set T will mean a two block partition of the set. If an element t_1 is contained in one block of a partition π and some other element t_2 is contained in the other block of π, then π separates the pair $\{t_1, t_2\}$. A partition set $\{\pi_k\}_1^n$ on a set T has zero-product if every pair of elements of T is separated by some partition π_k in $\{\pi_k\}_1^n$. If $2^n \geq \#(T)$, where $\#(T)$ is the number of elements in T, then a unique binary code of n digits may be assigned to each element of T according to a zero-product partition set $\{\pi_k\}_1^n$. For each π_k, define a function A_k from T into $\{0, 1\}$ which maps one block of π_k onto 0 and the other block onto 1. Consider the function $A: T \rightarrow \{0, 1\}^n$ defined by $A(t) = (A_1(t), \ldots, A_n(t))$.

That A is one-one may be verified by letting $t_1 \neq t_2$. Then some π_k separates $\{t_1, t_2\}$, since $\{\pi_k\}_1^n$ has zero-product, and so $A_k(t_1) \neq A_k(t_2)$. It follows that $A(t_1) \neq A(t_2)$. Hence A is an assignment function for T <u>defined</u> according to a zero-product partition set $\{\pi_k\}_1^n$ of T.

Since for each k, A_k may be defined in 2 ways according to π_k, A may be defined in 2^n ways according to $\{\pi_k\}_1^n$. Let $\mathcal{P} = \{\mathcal{A}, \mathcal{B}, \mathcal{C}\}$ where $\mathcal{A} = \{\alpha_i\}^{n_1}$, $\mathcal{B} = \{\beta_j\}^{n_2}$ and $\mathcal{C} = \{\gamma_k\}^{n_3}$ are zero-product partition sets of I, S, and O respectively. An assignment $\mathcal{A} = \{A^I, A^S, A^O\}$ is defined

according to $\mathcal{P} = \{\mathcal{A}, \mathcal{B}, \mathcal{C}\}$ if A^I, A^S, and A^O are defined according to $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ respectively.

There are $2^{n_1+n_2+n_3}$ assignments $\mathcal{A}$ which may be defined according to $\mathcal{P}$. The question arises as to whether the threshold property of the state-variable and output-variable functions is inherent in the set $\mathcal{P}$ of zero-product partition sets or whether it depends also on the assignment $\mathcal{A}$ defined according to $\mathcal{P}$. The answer is given by the following theorem.

FUNDAMENTAL THEOREM. Let M be a machine and $\mathcal{P} = \{\mathcal{A}, \mathcal{B}, \mathcal{C}\}$ where $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ are zero-product partition sets of I, S and O respectively. If $\mathcal{A}_1$, and $\mathcal{A}_2$ are two assignments for M, both defined according to $\mathcal{P}$ and if ${}_1\hat{f}^S_k$ and ${}_2\hat{f}^S_k$ are respectively the kth state-variable functions for $\mathcal{A}_1$ and $\mathcal{A}_2$, then ${}_1\hat{f}^S_k$ is threshold iff ${}_2\hat{f}^S_k$ is threshold.

PROOF: It is easily shown that the algebraic expression for ${}_2\hat{f}^S_k$ is obtained from that for ${}_1\hat{f}^S_k$ by simply complementing certain of the variables or else by complementing certain of the variables and then complementing the result. It follows that ${}_1\hat{f}^S_k$ is threshold iff ${}_2\hat{f}^S_k$ is threshold [2].

A similar result holds for the output-variable functions. For this reason, we may narrow the problem of looking for the acceptable assignments, for some given machine, down to the problem of looking for the acceptable sets $\mathcal{P}$ of zero-product partition sets on I, S and O. Here $\mathcal{P}$ is acceptable if some, and thus any, assignment $\mathcal{A}$, defined according to $\mathcal{P}$, is acceptable.

Accordingly we adopt the convention of calling $\mathcal{P} = \{\mathcal{A}, \mathcal{B}, \mathcal{C}\}$, where $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ are zero-product partition sets on I, S and O respectively, an assignment. Literally, then, we are still concerned with finding the set of acceptable assignments.

Decomposition of 2-asummability

In order to characterize the acceptable assignments $\mathcal{P}$, it is expedient to first decompose the condition of 2-asummability into its three constituent parts. Again, a function F is 2-asummable if no pair of points which map onto zero is equal in sum to a pair of points which map onto one. The first constituent of the 2-asummability condition is a pair of sets, each set being a pair of points. This structure will be encountered in situations when the parent set is other than the n-cube. Variations of the structure, such as that in which the points are not all distinct, are also encountered. The following definition, concerning the first constituent of 2-asummability, is thus necessarily vague as to the parent set and as to the relationship between the components of the structure.

DEFINITION I. Let T be any set with t_k in T, k = 1, . . , 4. The pair of sets, $\lambda = \{\{t_1, t_2\}, \{t_3, t_4\}\}$, will be called a T set-pair and denoted by $\langle t_1, t_2; t_3, t_4 \rangle$.

The notation is not unique in that there are eight such symbols, all representing λ. The modifier T is used to indicate the parent set. The sets $\{t_1, t_2\}$ and $\{t_3, t_4\}$ are called the blocks of λ while the points t_1, t_2, t_3 and t_4 are called the components of λ. Each block of a set-pair always has two distinct components even though they may not be distinct elements of T. For example, $\{\{0\}, \{0,2\}\}$ may be written $\langle 0,0; 0,2 \rangle$ (or in any of the other three permutations: $\langle 0,2; 0,0 \rangle$, etc.) or may be abbreviated by $\langle 0; 0,2 \rangle$. Regardless of how it is written, we will speak of it as having four components t_1, t_2, t_3, t_4 where $\{\{t_1, t_2\}, \{t_3, t_4\}\} = \{\{0\}, \{0,2\}\}$. A set-pair whose blocks are not disjoint will be called trivial.

We proceed with the decomposition of 2-asummability by making a definition concerning its second constituent, the equal sum property.

DEFINITION II. If $T \subseteq \{0,1\}^n$, i.e. if T is a subset of the n-cube, and if $\lambda = \langle t_1, t_2; t_3, t_4 \rangle$ is a T set-pair for which $t_1 + t_2 = t_3 + t_4$, then λ is said to be ES, (equal sum).

An example is obtained from M, the machine of our example. For Assignment A_1 of Example 1, if $T = A^S(S) \times A^I(I)$, the T set-pair $\theta = \langle (00,00), (00,11); (00,01)\ (00,10) \rangle$ is ES since in summing over either block, we obtain (00,11).

Finally, there is involved in the condition of 2-asummability the

idea of a function mapping one block of a set-pair onto zero while mapping the other block onto one. This third constituent of 2-asummability is formalized in the following definition.

DEFINITION III. Let F be a function, $F: T \rightarrow \{0,1\}$. F induces a set-pair $\lambda = \langle t_1, t_2; t_3, t_4 \rangle$ iff the sets $\{\{F(t_1), F(t_2)\}, \{F(t_3), F(t_4)\}\}$ and $\{\{0\},\{1\}\}$ are equal. That is, iff $\langle F(t_1), F(t_2); F(t_3), F(t_4)\rangle = \langle 0; 1\rangle$.

As an illustration, consider the example, $\theta = \langle (00, 00), (00, 11); (00, 01), (00, 10)\rangle$, given of an ES set-pair. From Example 1, for Assignment $\overset{A_1}{\wedge}$ it is seen that the state-variable function $\hat{f}_2^S$ induces θ. As a remark in passing, it will be noted that a necessary condition that a set-pair be induced by a function is that the set-pair be non-trivial. For example, $\hat{f}_2^S$ could not map one block of θ onto zero and the other onto one if the two blocks intersected. From the following definition, it will be seen that because $\hat{f}_2^S$ induces the ES set-pair θ, it fails to be 2-asummable. Again, the three constituent parts of the 2-asummability condition are: the set-pair structure, the ES (Equal Sum) property, and the concept of a function inducing a set-pair. Reuniting these three parts, we have an obviously equivalent definition of domain 2-asummability:

DEFINITION IV. A Boolean function F is domain 2-asummable iff F induces no ES set-pairs.

The advantage of this definition over the first one is that this definition isolates each of the three constituents of the 2-asummability

condition. To each constituent corresponds a question; the answers will enable us to effectively characterize the acceptable assignments.

Three Questions

We adopt the convention of reserving the symbol μ for SxI set-pairs. Once an assignment $\mathcal{P}$ is chosen and assignment functions A^I, A^S are defined according to zero-product partition sets $\mathcal{A}, \mathcal{B}$ on I and S respectively, we will talk about the $A^S(S) \times A^I(I)$ set-pair $A(\mu)$ as being the one obtained from μ by replacing states and inputs by their codes. Thus, if

$$\mu = \langle (S_i, I_m), (S_j, I_n); (S_k, I_o), (S_\ell, I_p) \rangle$$

then

$$A(\mu) = \langle (A^S(S_i), A^I(I_m)), (A^S(S_j), A^I(I_n)); (A^S(S_k), A^I(I_o)), (A^S(S_\ell), A^I(I_p)) \rangle .$$

Given an $A^S(S) \times A^I(I)$ set-pair θ, since the assignment functions are one-one, there will be a unique SxI set-pair μ such that $A(\mu) = \theta$. μ is called the pre-image of θ in this instance.

Question I roughly corresponds to the first constituent of 2-asummability, the set-pair structure over the domain of the function. The domain of the state-variable and output-variable functions is $A^S(S) \times A^I(I)$. Accordingly, we must concern ourselves with set-pairs over this set. Our concern extends to their pre-images as set forth in the following question.

QUESTION I. What are the possible pre-images of non-trivial

ES set-pairs over $A^S(S)xA^I(I)$? That is, for what SxI set-pairs μ do there exist assignment functions A^S and A^I such that $A(\mu)$ is non-trivial ES?

Question II corresponds to the second constituent of 2-asummability, the ES property. A function fails to be domain 2-asummable only if one of the induced set-pairs is ES. Accordingly, we are concerned only with those set-pairs over $A^S(S)xA^I(I)$ which are ES. The first question dealt with their pre-images; the next question deals with the assignments $\mathcal{P}$ involved.

QUESTION II. Given that μ is a possible pre-image of a non-trivial ES set-pair, for which assignments $\mathcal{P}$ is $A(\mu)$ actually ES?

Question III corresponds to the third constituent of 2-asummability, the induction of a set-pair by a function. While there may be set-pairs over the domain of the function which are ES, the function fails to be domain 2-asummable only if it induces one of them. Accordingly, we must concern ourselves with the induction of set-pairs by state-variable and output-variable functions.

QUESTION III. Given an SxI set-pair μ, for which assignments $\mathcal{P}$ does one of the state-variable functions induce $A(\mu)$?

To deal with these questions requires some preliminary definitions and several lemmas. The answers to the questions will then take the form of three theorems.

Three Theorems

DEFINITION V. Let T, T* be sets with F a function from T into

T^* and $\lambda = \langle t_1, t_2; t_3, t_4 \rangle$ a T set-pair. Then λ <u>implies</u> $F(\lambda)$ with respect to F where $F(\lambda) = \langle F(t_1), F(t_2); F(t_3), F(t_4) \rangle$. Thus $F(\lambda)$ is obtained from λ by replacing each component of λ by its image under F.

There are several instances in which we will have occasion to employ this definition. In one instance, T is SxI, T^* is S (or O) and F is f^S (or f^O). An example from Example 1 would be $\mu = \langle (S_1, I_1), (S_1, I_4); (S_1, I_2), (S_1, I_3) \rangle$ for which $f^S(\mu) = \langle S_1, S_3; S_2, S_4 \rangle$. Incidentally, for Assignment $\mathcal{A}_1$, $A(\mu) = \langle (00, 00), (00, 11); (00, 01), (00, 10) \rangle$ is the $A^S(S) \times A^I(I)$ set-pair θ which was given earlier as an example of an ES set-pair. In the other instance in which this definition is employed, T is S (or I), T^* is $\{0, 1\}$, and F is A_k^S (or A_k^I). An example would be $\lambda = \langle S_1, S_3; S_2, S_4 \rangle$. From Example 1, we see that for Assignment $\mathcal{A}_1$, $A_1^S(\lambda) = \langle 0, 1; 0, 1 \rangle$ which is, incidentally, ES.

DEFINITION VI. Given an SxI set-pair $\mu = \langle (S_i, I_m), (S_j, I_n); (S_k, I_o), (S_\ell, I_p) \rangle$, the S set-pair, $p_S(\mu) = \langle S_i, S_j; S_k, S_\ell \rangle$ is the <u>state projection</u> of μ while the I set-pair $p_I(\mu) = \langle I_m, I_n; I_o, I_p \rangle$ is the <u>input projection</u> of μ.

Before stating the first lemma, it is necessary to categorize the different ways in which the components t_k of a T set-pair $\lambda = \langle t_1, t_2; t_3, t_4 \rangle$ can be distributed with respect to the blocks of a partition π of T. By the definition of a set-pair, t_i and t_j will be regarded as different components of λ even though they may be the same element of T (i.e. $t_i = t_j$).

One and only one of the following must characterize the distribution of the components t_k with respect to the blocks of π.

Case I. One block of π contains all four components t_k of λ.

Case II. Each block of π contains a component from each block of λ.

Case III. One block of π contains one block of λ while the other block of π contains the other block of λ. This distribution is particularly significant and will be indicated by writing $\lambda \leq \pi$.

Case IV. One block of π contains three components of λ while the other block of π contains the fourth component of λ.

To give examples of each case, let $T = \{1,2,3,4,5,6\}$ and $\pi = \{\{1,2,3,4,\}, \{5,6\}\}$. Then $\langle 1,2,; 3,4\rangle$, $\langle 3,5; 4,6\rangle$, $\langle 3,4; 5,6\rangle$, and $\langle 2,3; 4,5\rangle$ are respectively examples of cases I, II, III and IV.

DEFINITION VII. If either Case III or Case IV characterize the distribution of the components of λ with respect to the blocks of π, then π <u>cancels</u> λ. Here again, λ is a T set-pair while π is a partition of the set T.

In the last example, $\pi = \{\{1,2,3,4\}, \{5,6\}\}$ cancels both the set-pairs $\langle 3,4; 5,6\rangle$ and $\langle 2,3; 4,5\rangle$. The significance of this definition lies with the following lemma.

LEMMA I. <u>Let T be a set, F a function from T into</u> $\{0,1\}$ <u>and</u> $\{\overline{0},\overline{1}\}$ <u>the partition of T induced by F</u>. That is, $\overline{0}$ is the subset of T containing those elements which F maps onto 0; $\overline{1}$ is the complement of $\overline{0}$).

Then if $\lambda = \langle t_1, t_2; t_3, t_4 \rangle$ is any T set-pair, $F(\lambda)$ is ES iff $\{\overline{0}, \overline{1}\}$ does not cancel λ.

PROOF: $F(\lambda)$ is ES iff $F(t_1) + F(t_2) = F(t_3) + F(t_4)$ iff either 1) or 2) are true as follows:

1) $0 = F(t_1) + F(t_2) = F(t_3) + F(t_4)$ or

$2 = F(T_1) + F(t_2) = F(t_3) + F(t_4)$ so

that $\{t_k\}_1^4$ is contained in $\overline{0}$ or $\overline{1}$ and Case I characterizes the distribution of the t_k with respect to the blocks of $\{\overline{0}, \overline{1}\}$.

2) $1 = F(t_1) + F(t_2) = F(t_3) + F(t_4)$ so

that $\{F(t_1), F(t_2)\} = \{F(t_3), F(t_4)\} = \{0, 1\}$ which is the case iff each block of $\{\overline{0}, \overline{1}\}$ contains an element from each block of $\lambda = \langle t_1, t_2; t_3, t_4 \rangle$. That is, $1 = F(t_1) + F(t_2) = F(t_3) + F(t_4)$ iff Case II characterizes the distribution of the t_k with respect to the blocks of $\{\overline{0}, \overline{1}\}$.

Thus $F(\lambda)$ is ES iff either Case I or Case II characterizes the distribution of the components of λ with respect to the blocks of $\{\overline{0}, \overline{1}\}$ iff $\{\overline{0}, \overline{1}\}$ does not cancel λ.

LEMMA II. Let T be a set and A an assignment function, $A: T \to \{0, 1\}^n$, where the kth component function of A is denoted by A_k. Let π_k be the partition of T induced by A_k. Then for any T set-pair λ, $A(\lambda)$ is ES iff no π_k cancels λ.

PROOF: $A(\lambda)$ is ES iff $A_k(\lambda)$ is ES for every k. Apply Lemma I.

Consider now the different relationships which may exist between the components of a set-pair $\lambda = \langle t_1, t_2; t_3, t_4 \rangle$. Writing $t_i = t_j$ if t_i and t_j

are the same element of T, we may categorize some of them as follows.

Case-c. One of the cross-over relationships $t_1 = t_3 \neq t_2 = t_4$ or $t_1 = t_4 \neq t_2 = t_3$ hold. E.G., $T = \{1,2,3,4\}$, $\lambda = \langle 1,3; 1,3\rangle$.

Case-d. The four components of λ are all distinct elements of T. E.G., $\lambda = \langle 1,2; 3,4\rangle$.

Case-s. The four components of λ are all the same element of T. E.G., $\lambda = \langle 1,1; 1,1\rangle$ or simply $\langle 1; 1\rangle$.

A set-pair λ will be called Type-c, Type-d or Type-s depending on whether Case-c, Case-d, or Case-s, respectively, is the case for the relationship between the components of λ.

LEMMA III. If $\lambda = \langle t_1, t_2; t_3, t_4\rangle$ is some T set-pair, then a necessary and sufficient condition that there exist an assignment function A for the set T such that $A(\lambda)$ is ES is that λ be Type-c, Type-d or Type-s. Moreover, $A(\lambda)$ is ES for every assignment function A if λ is either Type-c or Type-s.

PROOF: The necessity may be shown by supposing that λ is not one of these three types. Then either $t_1 = t_2 \neq t_3 = t_4$ or exactly three of the components are the same element of T. In either case, the four components of λ represent two distinct elements, say t and t*, of T.

Let A be any assignment function. If π_k is induced by A_k, it is easily seen that A's being one-one implies that $\{\pi_k\}$ has zero-product. Then necessarily π_k separates $\{t, t^*\}$ for some π_k. This same π_k then

cancels λ and consequently $A(\lambda)$ is not ES. This shows the necessity.

That $A(\lambda)$ is ES for every A if λ is either Type-s or Type-c is immediate. So is the existence of an A for which $A(\lambda)$ is ES if λ is Type-d. This establishes the sufficiency.

We are now in a position to deal with the three questions. To answer the first, we have:

THEOREM I. A necessary and sufficient condition that, given an SxI set-pair μ, there exist assignment functions A^I and A^S such that $A(\mu)$ is non-trivial ES is that μ be one of the following types: (d,d), (d,c), (d,s), (c,d), (s,d), (c,c). (Here the letter in the first position is respectively c, d, or s depending on whether $p_S(\mu)$ is Type-c, Type-d or Type-s. In like fashion, the letter in the second position tells what type of set-pair $p_I(\mu)$ is.)

PROOF: Obviously, $A(\mu)$ is ES iff both $A^I(p_I(\mu))$ and $A^S(p_S(\mu))$ are ES. By Lemma III, there exist assignment functions A^S and A^I such that $A^S(p_S(\mu))$ and $A^I(p_I(\mu))$ are ES iff $p_S(\mu)$ and $p_I(\mu)$ are each one of Type-c, Type-d or Type-s. Thus there exist A^S and A^I such that $A(\mu)$ is ES iff μ is one of the following types: (d,d), (d,s), (s,d), (c,d), (d,c), (c,c), (s,c), (c,s), or (s,s). It is easily verified that $A(\mu)$ is trivial if μ is one of the last three types, thus leaving only the first six.

Theorem I characterizes the possible pre-images of ES set-pairs, thereby answering the first question. To answer the second question we have:

THEOREM II. Given an SxI set-pair μ, $A(\mu)$ is ES for some choice of assignment functions A^S and A^I iff no α_i cancels $p_I(\mu)$ and no β_j cancels $p_S(\mu)$.

PROOF: The input assignment function A^I is defined according to the zero-product partition set $\{\alpha_k\}$ on I. Then its component functions A^I_k, induce the set $\{\alpha_k\}$. By Lemma II, $A^I(p_I(\mu))$ is ES iff no α_k cancels $p_I(\mu)$. Likewise, $A^S(p_S(\mu))$ is ES iff no β_j cancels $p_S(\mu)$. Thus $A(\mu)$ is ES iff no α_k cancels $p_I(\mu)$ and no β_j cancels $p_S(\mu)$.

Given an SxI set-pair μ, the question of which assignments $\mathcal{P} = \{\mathcal{A}, \mathcal{B}, \mathcal{C}\}$ is $A(\mu)$ an ES set-pair for is answered by Theorem II. By Theorem II it can be asserted that $A(\mu)$ is ES as long as $\mathcal{A}$ contains no α which cancels $p_I(\mu)$ and $\mathcal{B}$ no β which cancels $p_S(\mu)$. To answer the third question we have:

THEOREM III. Suppose $f^S(\mu) \leq \beta_k$ for some SxI set-pair μ and some partition β_k of S. Then for every pair of assignment functions A^S and A^I such that A^S_k is defined according to β_k, the kth state-variable function $\hat{f}^S_k$ induces $A(\mu)$.

PROOF: From the definition of the state-variable functions, it follows that $\hat{f}^S_k(A(\mu)) = A^S_k(f^S(\mu))$. Now $\hat{f}^S_k$ induces $A(\mu)$ iff $\hat{f}^S_k(A(\mu)) = \langle 0;1\rangle$ iff $A^S_k(f^S(\mu)) = \langle 0;1\rangle$.

Now write $\beta_k = \{\bar{0}, \bar{1}\}$ where $\bar{0}$ is the block of β_k which A^S_k maps onto zero and $\bar{1}$ is the block mapped onto one. $A^S_k(f^S(\mu)) = \langle 0;1\rangle$ iff

one block of $f^S(\mu)$ is contained in $\overline{0}$ and the other block in $\overline{1}$. But this is indicated by writing $f^S(\mu) \leq \{\overline{0}, \overline{1}\} = \beta_k$. Thus f^S_k induces $A(\mu)$ iff $f^S(\mu) \leq \beta_k$.

An identical result holds for the output-code functions, i.e., f^O_k induces $A(\mu)$ iff $f^O(\mu) \leq \gamma_k$. Given an SxI set-pair μ, for any assignment $\mathcal{P} = \{\alpha, \mathcal{B}, \mathcal{C}\}$ for which $\mathcal{B}$ contains $\beta \geq f^S(\mu)$ one of the state-variable functions induces $A(\mu)$ where A^S_k is defined according to β. Thus the third question is answered.

A Theorem Characterizing the Acceptable Assignments

Let us extend the notion of set-pairs implying other set-pairs to set-pairs implying partitions. Let λ, σ be respectively I and S set-pairs; let β, γ be respectively partitions of S and O. Then $\lambda \Rightarrow \beta$ ($\lambda \Rightarrow \gamma$) iff there exists some SxI set-pair μ which is (s,d) or (c,d) and for which $\lambda = p_I(\mu)$ and $f^S(\mu) \leq \beta$ ($f^O(\mu) \leq \gamma$). Similarly, $\sigma \Rightarrow \beta$ ($\sigma \Rightarrow \gamma$) iff there exists some SxI set-pair μ which is (d,s) or (d,c) and for which $\sigma = p_S(\mu)$ and $f^S(\mu) \leq \beta$ ($f^O(\mu) \leq \gamma$). Finally, $(\sigma, \lambda) \Rightarrow \beta$ ($(\sigma, \lambda) \Rightarrow \gamma$) iff there exists some SxI set-pair μ which is (d,d) and for which $\sigma = p_S(\mu)$ and $\lambda = p_I(\mu)$ while $f^S(\mu) \leq \beta$ ($f^O(\mu) \leq \gamma$).

The following theorem characterizes the acceptable assignments.

THEOREM IV. Let $M = (I, S, O, f^S, f^O)$ be a machine. Then an assignment $\mathcal{P} = \{\alpha, \mathcal{B}, \mathcal{C}\}$ is an acceptable assignment for M iff 1) for every β_j in $\mathcal{B}$ and γ_k in $\mathcal{C}$, neither $f^S(\mu) \leq \beta_j$ nor $f^O(\mu) \leq \gamma_k$ for any SxI set-pair μ which is (c,c). 2) For every β_j in $\mathcal{B}$ and γ_k in $\mathcal{C}$, $\lambda \Rightarrow \beta_j$ or

$\lambda \Rightarrow \gamma_k$ for some I set-pair λ only if $\mathcal{A}$ contains some α_i which cancels λ. 3) for every β_j in $\mathcal{B}$ and γ_k in $\mathcal{C}$, $\sigma \not\Rightarrow \beta_j$ or $\sigma \Rightarrow \gamma_k$ for some S set-pair σ only if $\mathcal{B}$ contains some β_i which cancels σ. 4) for every β_j in $\mathcal{B}$ and γ_k in $\mathcal{C}$, $(\sigma,\lambda) \Rightarrow \beta_j$ or $(\sigma,\lambda) \Rightarrow \gamma_k$ for some S set-pair σ and some I set-pair λ only if either $\mathcal{A}$ contains an α_h which cancels λ or if $\mathcal{B}$ contains a β_i which cancels σ.

PROOF: To show the necessity, let $\mathcal{P}$ be acceptable and show that 1), 2), 3) and 4) must hold. Since $\mathcal{P}$ is acceptable, each state-variable function $\hat{f}^S_j$ and output-variable function $\hat{f}^O_k$ is domain 2-asummable and thus induces no ES set-pairs.

To show that 1) holds, let μ be (c,c). Then $p_S(\mu)$ and $p_I(\mu)$ are both Type-c and so $A^S(p_S(\mu)$ and $A^I(p_I(\mu)$ are both ES. Consequently $A(\mu)$ is ES and thus induced by no state-variable or output-variable function. By Theorem III, this happens iff $f^S(\mu) \not\leq \beta_j$ and $f^O(\mu) \not\leq \gamma_k$ for every β_j in $\mathcal{B}$ and γ_k in $\mathcal{C}$. Thus, if μ is (c,c), for every β_j in $\mathcal{B}$ and γ_k in $\mathcal{C}$, neither $f^S(\mu) \leq \beta_j$ nor $f^O(\mu) \leq \gamma_k$.

To show that 2) holds, let λ be any I set-pair. Suppose that $\lambda \Rightarrow \beta_j$ or $\lambda \Rightarrow \gamma_k$ for some β_j in $\mathcal{B}$ or for some γ_k in $\mathcal{C}$. Then there exists some SxI set-pair μ which is (s,d) or (c,d) and for which $\lambda = p_I(\mu)$ and either $f^S(\mu) \leq \beta_j$ or $f^O(\mu) \leq \gamma_k$. Since $p_S(\mu)$ is Type-s or Type-c, $A^S(p_S(\mu)$ is ES. By Theorem III, either $\hat{f}^S_j$ or $\hat{f}^O_k$ induces $A(\mu)$. Then $A(\mu)$ must not be ES. Since $A^S(p_S(\mu))$ is ES, $A^I(p_I(\mu))$ must not be ES. By Theorem II, then, α_i cancels $p_I(\mu) = \lambda$ for some α_i in $\mathcal{A}$. Thus, if λ is some I set-pair,

for every β_j in $\mathcal{B}$ and γ_k in $\mathcal{C}$, $\lambda \Rightarrow \beta_j$ or $\lambda \Rightarrow \gamma_k$ only if $\mathcal{A}$ contains some α_i which cancels λ.

Part 3) may be shown to hold by an argument which is the dual of that for 2) while the argument for 4) is a composite of those for 2) and 3).

To show the sufficiency, let 1), 2), 3) and 4) hold and show that $\mathcal{P}$ must be acceptable. $\mathcal{P}$ is acceptable iff each state-variable and output-variable function is domain 2-asummable iff none of these functions induce an ES set-pair. Let θ be a non-trivial ES set-pair over the domain of these functions. Then, by Theorem I, there exists an SxI set-pair μ which is one of (d,d), (d,s), (d,c), (c,d), (s,d) or (c,c) and for which $A(\mu) = \theta$. If μ is (c,c), then by 1), for every β_j in $\mathcal{B}$ and γ_k in $\mathcal{C}$, neither $f^S(\mu) \leq \beta_j$ nor $f^O(\mu) \leq \gamma_k$.

If μ is (c,d) or (s,d), let $\lambda = p_I(\mu)$. Since $A(\mu)$ is ES, $A^1(\lambda)$ is ES. By Theorem II, no α_i cancels λ. Then by 2), neither $\lambda \Rightarrow \beta_j$ nor $\lambda \Rightarrow \gamma_k$ for any β_j or γ_k. Consequently, neither $f^S(\mu) \leq \beta_j$ nor $f^O(\mu) \leq \gamma_k$ for any β_j or γ_k.

Similarly, if μ is (d,c), (d,s) or (d,d) it may be shown that neither $f^S(\mu) \leq \beta_j$ or $f^O(\mu) \leq \gamma_k$ for any β_j or γ_k.

Thus, no matter what type of SxI set-pair μ, neither $f^S(\mu) \leq \beta_j$ nor $f^O(\mu) \leq \gamma_k$ for any β_j or γ_k. By Theorem III, neither $\hat{f}^S_j$ nor $\hat{f}^O_k$ induces $A(\mu) = \theta$ for any $\hat{f}^S_j$ or $\hat{f}^O_k$. Thus no state-variable or output-variable function induces an ES set-pair; hence each is domain 2-asummable and $\mathcal{P}$ is acceptable.

Having characterized the acceptable assignments $\mathcal{P}$, we now outline a procedure for finding them.

III

AN ALGORITHM YIELDING THE ACCEPTABLE ASSIGNMENTS

The algorithm consists of the following steps.

Step 1. Compute the implicants $f^S(\mu)$ and $f^O(\mu)$ for each (c,c),(d,d), (d,c), (c,d), (d,s) or (s,d) SxI set-pair μ.

Step 2. Compute the set of acceptable partitions,

$$D = \{\pi \mid \pi \text{ is a 2-block partition of S or of O such that } \pi \not\geq f^S(\mu) \text{ or } f^O(\mu) \text{ for any (c,c) SxI set-pair } \mu\}$$

Step 3. Compute L_π for each π in D where L_π is the list of all λ's, σ's, and (σ,λ)'s implying π (here λ and σ are respectively type-d I and S set-pairs.)

Step 4. Compute K_λ for each type-d I set-pair λ where K_λ is the list of all input partitions α which cancel λ.

Compute K_σ for each type-d S set-pair σ where K_σ is the list of all state partitions β in D which cancel σ.

Step 5. Compute M_p for each pair p of inputs where M_p is the list of all input partitions α which separate p.

Compute M_p for each pair p of states where M_p is the list of all state partitions β in D which separate p.

Compute M_p for each pair p of outputs where M_p is the list of all output partitions γ in D which separate p.

Step 6. Compute the points $\wp$ for which the discriminant function Δ is 1. The discriminant function is defined as follows. Let $\wp$ =

$\{\mathcal{A}, \mathcal{B}, \mathcal{C}\}$ where $\mathcal{A}$ is a set of input partitions, $\mathcal{B}$ is a set of state partitions in D, and $\mathcal{C}$ is a set of output partitions in D. Let Ψ be the set of all such $\mathcal{P}$. Define $X_\alpha : \Psi \to \{0,1\}$ by $X_\alpha(\mathcal{P}) = 1$ iff α in $\mathcal{A}$. Do this for each input partition α. Similarly, for each state partition β in D and for each output partition γ in D, define $Y_\beta : \Psi \to \{0,1\}$ and $Z_\gamma : \Psi \to \{0,1\}$ by $Y_\beta(\mathcal{P}) = 1$ iff β in $\mathcal{B}$ and $Z_\gamma(\mathcal{P}) = 1$ iff γ in $\mathcal{C}$.

Let P_1, P_2, P_3 be as follows:

$$P_1 = \prod_{\{I_j, I_k\} = p} \left(\sum_{\alpha \text{ in } M_p} X_\alpha \right)$$ where the product is taken over all pairs p of inputs I_j, I_k.

$$P_2 = \prod_{\{S_j, S_k\} = p} \left(\sum_{\beta \text{ in } M_p} Y_\beta \right)$$ where the product is taken over all pairs p of states S_j, S_k.

$$P_3 = \prod_{\{O_j, O_k\} = p} \left(\sum_{\gamma \text{ in } M_p} Z_\gamma \right)$$ where the product is taken over all pairs p of outputs O_j, O_k.

Let P_4^π, P_5^π, P_6^π be as follows for each π in D.

$$P_4^\pi = \prod_{\lambda \text{ in } L_\pi} \left(\sum_{\alpha \text{ in } K(\lambda)} X_\alpha \right)$$ where the product is taken over all I set-pairs λ in L_π.

$$P_5^\pi = \prod_{\sigma \text{ in } L_\pi} \left(\sum_{\beta \text{ in } K(\sigma)} Y_\beta \right)$$ where the product is taken over all S set-pairs σ in L_π.

$$P_6^\pi = \prod_{(\sigma,\lambda) \text{ in } L_\pi} \left[\sum_{\alpha \text{ in } K(\lambda)} X_\alpha + \sum_{\beta \text{ in } K(\sigma)} Y_\beta \right]$$ where the product is taken over all (σ,λ)'s in L_π.

Define Δ_1, Δ_2, and Δ all from Ψ into $\{0,1\}$ by:

$$\Delta_1(\mathcal{P}) = P_1(\mathcal{P}) \cdot P_2(\mathcal{P}) \cdot P_3(\mathcal{P})$$

$$\Delta_2(\mathcal{P}) = \left[\prod_{\beta \text{ in } D} (\overline{Y}_\beta + P_4^\beta \cdot P_5^\beta \cdot P_6^\beta) \right] \left[\prod_{\gamma \text{ in } D} (\overline{Z}_\gamma + P_4^\gamma \cdot P_5^\gamma \cdot P_6^\gamma) \right]$$

$$\Delta(\mathcal{P}) = \Delta_1(\mathcal{P}) \cdot \Delta_2(\mathcal{P}).$$

To show that the algorithm actually yields the acceptable assignments, it is sufficient to show that $\Delta(\mathcal{P}) = 1$ iff $\mathcal{P}$ is an acceptable assignment. It is easily shown that $\Delta_1(\mathcal{P}) = 1$ iff $\mathcal{P}$ is an assignment (i.e., iff $\mathcal{A}, \mathcal{B}, \mathcal{C}$ have zero-product).

To see that $\Delta_2(\mathcal{P}) = 1$ iff $\mathcal{P}$ satisfies the terms of Theorem IV, suppose that $\Delta_2(\mathcal{P}) = 1$. Since $\mathcal{B} \cup \mathcal{C} \subseteq D$, part 1) of Theorem IV is automatically satisfied. If $\lambda \Rightarrow \beta$ in $\mathcal{B}$, λ is in L_β. $\Delta_2(\mathcal{P}) = 1$ means $\overline{Y}_\beta + P_4^\beta \cdot P_5^\beta \cdot P_6^\beta = 1$. $\overline{Y}_\beta = 0$ since β in $\mathcal{B}$. Thus $P_4^\beta = 1$ and so $\sum_{\alpha \text{ in } K(\lambda)} X_\alpha = 1$. Thus $\mathcal{A}$ contains an α which cancels λ. Together with a dual result for the output partitions, this means part 2) of Theorem IV is satisfied. The argument that part 3) holds is analagous while the argument that part 4) holds is a composite of those for 2) and 3).

Conversely, if $\mathcal{P}$ satisfies the terms of Theorem IV, $\Delta_2(\mathcal{P})$ may be shown to be equal to 1. This may be done by showing $\overline{Y}_\pi + P_4^\pi \cdot P_5^\pi \cdot P_6^\pi$ to be equal to 1 for every π in D. If π is not in $\mathcal{B} \cup \mathcal{C}$, $\overline{Y}_\pi = 1$. Otherwise $\mathcal{P}$'s satisfying parts 2), 3), and 4) of Theorem IV imply respectively that P_4^π, P_5^π, and P_6^π are each equal to 1.

Since $\Delta_1(\mathcal{P}) = 1$ iff $\mathcal{P}$ is an assignment and $\Delta_2(\mathcal{P}) = 1$ iff $\mathcal{P}$ satisfies the terms of Theorem IV, $\Delta(\mathcal{P}) = 1$ iff $\mathcal{P}$ is an acceptable assignment. The algorithm then does indeed yield the acceptable assignments.

Execution of the algorithm is partially illustrated in Example 2,

using the machine of Example 1. Being rather lengthy, the first three steps are omitted here. However an example appears in [2] which illustrates execution of the algorithm for this same machine. Moreover, the three steps are now discussed here in greater detail and from this discussion, it is hoped that the reader will be able to construct these steps for himself.

Step 1) may be most efficiently accomplished by first computing the images of the blocks b where a block is a pair of elements of SxI. The images $f^S(b)$ of the blocks are then combined to obtain the implicants $f^S(\mu)$. If $\mu_1 = \{b_1, b_2\}$ and $\mu_2 = \{b_1, b_3\}$ are two SxI set-pairs, $f^S(\mu_1)$ can be obtained by combining $f^S(b_1)$ with $f^S(b_2)$ while $f^S(\mu_2)$ can be obtained by combining $f^S(b_1)$ with $f^S(b_3)$. In this way computation of $f^S(b_1)$ twice is avoided.

It is convenient to display the implicants $f^S(\mu)$ and $f^O(\mu)$ in tables, a table for each type of set-pair μ. Let t_s and t_i belong to $\{c, d, s\}$. Then the (t_s, t_i) implicant table has a column for each Type-t_i I set-pair λ and a row for each Type-t_s S set-pair σ. The entry in column λ and row σ is a list of all non-trivial $f^S(\mu)$'s and $f^O(\mu)$'s such that $p_S(\mu) = \sigma$ and $p_I(\mu) = \lambda$. Thus the (t_s, t_i) table displays all non-trivial implicants of (t_s, t_i) set-pairs μ.

Step 2) is accomplished simply by listing all state and output partitions. Each partition π is then compared with each implicant δ in the (c,c) implicant table. π is rejected if $\pi \geq \delta$ for some δ. The remaining partitions make up the set D of acceptable partitions.

Step 3) is accomplished most easily by first making a list for each Type-d I set-pair λ of all partitions in D implied by λ. The list for λ is formed by comparing each partition in D with implicants in column λ of the (c,d) and (s,d) implicant tables formed in step 1. A list for each type-d S set-pair σ of all partitions in D implied by σ is formed in like fashion. That is, each partition in D is compared with the implicants in row σ of the (d,c) and (d,s) implicant tables. Finally, a similar list is formed for each pair (σ,λ) by comparing the partitions in D with the implicants in row σ and column λ of the (d,d) implicant table.

The lists for the λ's, σ's, and (σ,λ)'s are then inverted to form the L_π's, that is, λ is in L_π iff π appeared in the list of partitions implied by λ.

Example 2 illustrates partially the execution of the algorithm. Again, the machine is that of Example 1. The input partitions, the set D obtained in Step 2, and the lists L_π obtained in Step 3 are first listed.

EXAMPLE 2

Input Partitions

$\alpha_1 = \{\{I_1, I_2\}, \{I_3, I_4\}\}$

$\alpha_2 = \{\{I_1, I_3\}, \{I_2, I_4\}\}$

$\alpha_3 = \{\{I_1, I_4\}, \{I_2, I_3\}\}$

$\alpha_4 = \{\{I_1\}, \{I_2, I_3, I_4\}\}$

$\alpha_5 = \{\{I_2\}, \{I_1, I_3, I_4\}\}$

$\alpha_6 = \{\{I_3\}, \{I_1, I_2, I_4\}\}$

$\alpha_7 = \{\{I_4\}, \{I_1, I_2, I_3\}\}$

The Set D of Acceptable Partitions

Acceptable State Partitions	Acceptable Output Partitions
$\beta_1 = \{\{S_1, S_3\}, \{S_2, S_4\}\}$	$\gamma_1 = \{\{O_1, O_4\}, \{O_2, O_3\}\}$
$\beta_2 = \{\{S_1, S_4\}, \{S_2, S_3\}\}$	$\gamma_2 = \{\{O_1\}, \{O_2, O_3, O_4\}\}$
$\beta_3 = \{\{S_1\}, \{S_2, S_3, S_4\}\}$	$\gamma_3 = \{\{O_2\}, \{O_1, O_3, O_4\}\}$
$\beta_4 = \{\{S_3\}, \{S_1, S_2, S_4\}\}$	$\gamma_4 = \{\{O_4\}, \{O_1, O_2, O_3\}\}$
$\beta_5 = \{\{S_4\}, \{S_1, S_2, S_3\}\}$	

Type-d S set-pairs	Type-d I set-pairs
$\lambda_1 = \langle I_1, I_2; I_3, I_4\rangle$	$\sigma_1 = \langle S_1, S_2; S_3, S_4\rangle$
$\lambda_2 = \langle I_1, I_3; I_2, I_4\rangle$	$\sigma_2 = \langle S_1, S_3; S_2, S_4\rangle$
$\lambda_3 = \langle I_1, I_4; I_2, I_3\rangle$	$\sigma_3 = \langle S_1, S_4; S_2, S_3\rangle$

The Lists L_π Constructed For Each Acceptable Partition π in D And Containing All λ's, σ's and (σ,λ)'s Implying π.

$L\beta_1$: $(\sigma_1,\lambda_3), (\sigma_2,\lambda_3), \sigma_2, \lambda_3$

$L\beta_2$: $(\sigma_1,\lambda_1), (\sigma_2,\lambda_1), (\sigma_3,\lambda_1), \lambda_1$

$L\beta_3$: empty

$L\beta_4$: σ_2

$L\beta_5$: $(\sigma_1,\lambda_1), (\sigma_1,\lambda_3), (\sigma_2,\lambda_1), (\sigma_2,\lambda_3), (\sigma_3,\lambda_1), (\sigma_3,\lambda_3), \lambda_1, \lambda_3$

$L\gamma_1$: $(\sigma_1,\lambda_1), (\sigma_2,\lambda_1), (\sigma_3,\lambda_1), \lambda_1$

$L\gamma_2$: $(\sigma_1,\lambda_1), \sigma_1, \lambda_1$

$L\gamma_3$: $(\sigma_1,\lambda_1), (\sigma_1,\lambda_3), (\sigma_2,\lambda_1), (\sigma_2,\lambda_3)$

$L\gamma_4$: λ_1

The Lists $K(\lambda)$ Constructed for Each Type-d Set-Pair λ And Containing all Input Partitions α Which Cancel λ And The Lists $K(\beta)$ Constructed For Each Type-d Set-pair σ And Containing All State Partitions β Which Cancel β.

$K(\lambda_1)$: $\alpha_1, \alpha_4, \alpha_5, \alpha_6, \alpha_7$

$K(\lambda_2)$: $\alpha_2, \alpha_4, \alpha_5, \alpha_6, \alpha_7$

$K(\lambda_3)$: $\alpha_3, \alpha_4, \alpha_5, \alpha_6, \alpha_7$

$K(\sigma_1)$: $\beta_3, \beta_4, \beta_5$

$K(\sigma_2)$: $\beta_1, \beta_3, \beta_4, \beta_5$

$K(\sigma_3)$: $\beta_2, \beta_3, \beta_4, \beta_5$

The Lists M_p

I_1I_2	I_1I_3	I_1I_4	I_2I_3	I_2I_4	I_3I_4
α_2	α_1	α_1	α_1	α_1	α_2
α_3	α_3	α_2	α_2	α_3	α_3
α_4	α_4	α_4	α_5	α_5	α_6
α_5	α_6	α_7	α_6	α_7	α_7
S_1S_2	S_1S_3	S_1S_4	S_2S_3	S_2S_4	S_3S_4
β_1	β_2	β_1	β_1	β_2	β_1
β_2	β_3	β_3	β_4	β_5	β_2
β_3	β_4	β_5			β_5
O_1O_2	O_1O_3	O_1O_4	O_2O_3	O_2O_4	O_3O_4
$\gamma_1, \gamma_2, \gamma_3$	γ_1, γ_2	γ_2, γ_4	γ_3	$\gamma_1, \gamma_3, \gamma_4$	γ_1, γ_4

$$\Delta_1 = (X_2+X_3+X_4+X_5)(X_1+X_3+X_4+X_6)(X_1+X_2+X_4+X_7)(X_1+X_2+X_5+X_6)$$
$$(X_1+X_3+X_5+X_7)(X_2+X_3+X_6+X_7)(Y_1+Y_2+Y_3)(Y_2+Y_3+Y_4)(Y_1+Y_3+Y_5)$$
$$(Y_1+Y_4)(Y_2+Y_5)(Y_1+Y_2+Y_5)(Z_1+Z_2+Z_3)(Z_1+Z_2)(Z_2+Z_4)\cdot Z_3 \cdot$$
$$(Z_1+Z_3+Z_4)(Z_1+Z_4)$$

Using the Boolean Law, $X(X+Y) = X$, we can simplify the product of sums expression for Δ_1 to read:

$$\Delta_1 = (X_2+X_3+X_4+X_5)(X_1+X_3+X_4+X_6)(X_1+X_2+X_4+X_7)(X_1+X_2+X_5+X_6)$$
$$(X_1+X_3+X_5+X_7)(X_2+X_3+X_6+X_7)(Y_1+Y_2+Y_3)(Y_2+Y_3+Y_4)(Y_1+Y_3+Y_5)$$
$$(Y_1+Y_4)(Y_2+Y_5)(Z_1+Z_2)(Z_2+Z_4)\, Z_3\, (Z_1+Z_4)$$

This type of simplification can be achieved for Δ_2 by omitting the factor corresponding to (σ_i, λ_j) if the list for β_j (or γ_k) contains, also, either σ_i or λ_j. Thus we have:

$$\Delta_2 = [\bar{Y}_1+(Y_1+Y_3+Y_4+Y_5)(X_3+X_4+X_5+X_6+X_7)][\bar{Y}_2+(X_1+X_4+X_5+X_6+X_7)]$$
$$[\bar{Y}_4+(Y_1+Y_3+Y_4+Y_5)][\bar{Y}_5+(X_1+X_4+X_5+X_6+X_7)(X_3+X_4+X_5+X_6+X_7)]$$
$$[\bar{Z}_1+(X_1+X_4+X_5+X_6+X_7)][\bar{Z}_2+(X_1+X_4+X_5+X_6+X_7)(Y_3+Y_4+Y_5)]$$
$$[\bar{Z}_3+(X_1+X_4+X_5+X_6+X_7)(X_3+X_4+X_5+X_6+X_7)][\bar{Z}_4+(X_1+X_4+X_5+X_6+X_7)]$$

Forming the product $\Delta = \Delta_1 \cdot \Delta_2$, and simplifying we obtain:

$$\Delta = (X_1+X_3+X_4+X_5 \cdot X_6)(X_1+X_2+X_4X_5+X_4X_6+X_5X_7+X_6X_7)(X_3+X_7+X_1X_2 +X_1X_6+X_2X_5+X_5X_6)(X_1X_3+X_4+X_5+X_6+X_7)(Y_1+Y_3+Y_2Y_5)(Y_1Y_2+Y_1Y_3+Y_4) (Z_1Z_2+Z_1Z_4+Z_2Z_4)\, Z_3\, (Y_2+Y_5)(\overline{Z}_2+Y_3+Y_4+Y_5)$$

Let $\overline{1}$ be the set of all combinations of values Q for the $X_i(\mathcal{P})$'s, $Y_j(\mathcal{P})$'s and $Z_k(\mathcal{P})$'s such that $\Delta(\mathcal{P}) = 1$. If Q is in $\overline{1}$, write W(Q) for the total number of Y_j's and Z_k's which equal 1 for Q. If $Y_j(\mathcal{P}) = 1$ for Q, then $\mathcal{B}$ contains β_j and if $\mathcal{A}$, defined according to $\mathcal{P}$, is one level, there will be a threshold gate in the corresponding realization for M which computes the jth digit of the next state for M. Similarly, if $Z_k(\mathcal{P}) = 1$ for Q, then $\mathcal{C}$ contains γ_k and if $\mathcal{A}$, defined according to $\mathcal{P}$, is one level, there will be a threshold gate in the corresponding realization for M which computes the kth digit of the present output. Thus W(Q) equals the number of gates in the realization for M if $\mathcal{A}$ is one-level where $\mathcal{A}$ is any assignment defined according to $\mathcal{P}$ and $Q = (X_1(\mathcal{P}), \ldots, Y_1(\mathcal{P}), \ldots Z(\mathcal{P}), \ldots)$.

To find the minimum one-level realization, it is clear that the optimal procedure from this point is to exhaust $\overline{1}$, the search process ordered by the rule: Q_1 is considered before Q_2 iff $W(Q_1) \leq W(Q_2)$. Whenever a point Q of $\overline{1}$ is encountered for which $\mathcal{A}$, defined according to $\mathcal{P}$ where $Q = (X_1(\mathcal{P}), \ldots, Y_1(\mathcal{P}), \ldots, Z_1(\mathcal{P}), \ldots)$, is one-level, a minimal one-level realization will have been obtained for M.

The most expedient form for Δ, then, is that form from which the

points of $\overline{1}$ may be most readily obtained in serial fashion, Q_1 before Q_2 iff $W(Q_1) \leq W(Q_2)$. This, in general, may be the minimum-sum-of-products form, obtained by expanding the product-of-sums form and applying the Quine-McCluskey Procedure. For the discriminant function of Example 16, it is unnecessary to proceed this far. The form given in Example 3 is adequate.

EXAMPLE 3 - A Terminal Form for Δ

$$\Delta(\mathcal{P}) = [X_1X_3+X_1X_6+X_2X_3X_4+X_3X_4X_5 + X_3X_4X_6+X_2X_3X_5+X_3X_5X_7+X_2X_3X_6 +X_2X_3X_7+X_3X_6X_7+X_2X_4X_7+X_4X_6X_7 + X_4X_5X_7+X_5X_6X_7+X_1X_2X_4 +X_1X_2X_5+X_2X_4X_5+X_2X_5X_6+X_4X_5X_6] \cdot Z_3 \cdot [Z_1\overline{Z}_2Z_3Y_1Y_2+(Z_1Z_2 +Z_1Z_4+Z_2Z_4)(Y_1Y_2Y_3+Y_2Y_3Y_4+Y_1Y_2Y_4+Y_1Y_2Y_5+Y_2Y_4Y_5+Y_1Y_3Y_5 + Y_1Y_4Y_5+Y_3Y_4Y_5)]$$

From Example 3 , it can be seen that if Q is the combination of values: $X_1 = X_3 = Y_1 = Y_2 = Z_1 = Z_3 = Z_4 = 1$ with the remaining variables 0, then $Q \in \overline{1}$ and $W(Q) \leq W(Q')$ for every $Q' \in \overline{1}$. If $\mathcal{P}$ is the assignment such that $Q = (X_1(\mathcal{P}), \ldots, Y_1(\mathcal{P}), \ldots Z_1(\mathcal{P}) \ldots)$, then $\Delta(\mathcal{P}) = 1$ and employs as few gates as possible. Hence $\mathcal{P} = \{\{\alpha_1, \alpha_3\}, \{\beta_1, \beta_2\}, \{\gamma_1, \gamma_3, \gamma_4\}\}$, is acceptable and may as well be considered first. It has a one-level assignment defined according to it as evidenced by Assignment A_2 of Example 1 . Should some other assignment, defined according to $\mathcal{P}$, have been chosen, nothing would be lost. By the Fundamental Theorem, it is also one-level; moreover, it results in the same number of gates.

IV

ADDITIONAL CONSIDERATIONS

This paper concerns itself with finding the minimum one-level code assignment for a sequential machine. To illustrate the size of the problem, there are 140^3 different irredundant assignments A for the machine M of Example 1. (An assignment function A: $T \to \{0,1\}^n$ is redundant if there exists k, $1 \leq k \leq n$, such that, while masking the kth component, $A(t_1) \neq A(t_2)$ for every $t_1 \neq t_2$. Thus the component function A_k is redundant. An assignment A is redundant if any of A^I, A^S, or A^O are redundant .)

The Fundamental Theorem greatly reduces the size of the problem as follows. Define a relation $\approx$ by $A_1 \approx A_2$ iff both A_1, A_2 induce the same set $\mathcal{P} = \{\alpha, \beta, \gamma\}$. The Fundamental Theorem states that $A_1 \approx A_2$ implies that A_1 is one-level iff A_2 is one-level. Accordingly we may look instead for $\mathcal{P}$ for which an arbitrary A defined according to $\mathcal{P}$, is one-level. The size of the problem remains formidable. There are 19^3 different assignments $\mathcal{P}$ for the machine M of Example 1.

The algorithm further reduces the problem. The discriminant function Δ yields the set of acceptable assignments $\mathcal{P}$. To illustrate how this reduces the number of assignments which must be considered, consider the discriminant function for the machine M in

its form given by Example 3 . The first factor has 19 terms; the second can be expanded to 25 terms. Thus the number of points for which Δ is 1 and the number of acceptable assignments is $19 \cdot 25 = 475$. Moreover, having the minimum-sum-of-products form allows us to order our search process so that, in general, all the acceptable assignments will not have to be tried.

Domain 2-asummability, 2-asummability, and linear separability are synonymous in a very special case. Let n_1, n_2 be positive integers with $n_1 + n_2 \leq 7$. Consider a machine $M = (I, S, O, f^S, f^O)$. If the cardinalities of I and S are n_1 and n_2 respectively, then the state-variable and output-variable functions are fully specified for any $\mathcal{P} = \{\mathcal{A}, \mathcal{B}, \mathcal{C}\}$ for which the cardinalities of $\mathcal{A}$ and $\mathcal{B}$ are n_1 and n_2 respectively. Since they are fully specified, domain 2-asummability and 2-asummability coincide. Since they are functions of $n_1 + n_2 \leq 7$ variables, 2-asummability and linear separability coincide. Thus $\mathcal{P}$ is acceptable iff $\mathcal{P}$ is one-level. This situation characterizes the machine M of Example 1 and the assignment $\mathcal{P}$ by which Assignment A_2 was defined.

State Splitting

In closing, it seems reasonable to comment on the size of the class of machines having one-level realizations. It appears impossible to give a positive characterization. However, it is easy to characterize a certain subclass of those machines which do not have a one-level

realization. Let $M = (I, S, O, f^S, f^O)$. If for some i, j, m, n we have $S_k = f^S(S_i, I_m) = f^S(S_j, I_n) \neq f^S(S_i, I_n) = f^S(S_j, I_m) = S_\ell$ then M has no one-level realization. To see this, let $\mu = \langle (S_i, I_m)(S_j, I_n); (S_i, I_n), (S_j, I_m) \rangle$. Then μ is (c,c) and $f^S(\mu) = \langle S_k; S_\ell \rangle$. Since $S_k \neq S_\ell$, some state partition β in $\mathcal{P}$ will have to separate S_k and S_ℓ for any assignment.

Then $f^S(\mu) \leq \beta$ and so $\mathcal{P}$ is not acceptable by Theorem IV, part 1). Thus M has no acceptable assignments and thus no one-level assignments.

There is a way around this difficulty if we are willing to accept, in lieu of the original machine, a machine internally different but with identical external behavior. To illustrate, consider the machines of Example 4.

EXAMPLE 4 - Two Equivalent Machines

M_1

$_1f^S$	I_1	I_2
S_1	S_1	S_2
S_2	S_2	S_1

$_1f^O$	I_1	I_2
S_1	O_1	O_1
S_2	O_2	O_2

M_2

$_2f^S$	I_1	I_2
S_1	S_1	S_2
S_2	S_2	S_3
S_3	S_1	S_2

$_2f^O$	I_1	I_2
S_1	O_1	O_1
S_2	O_2	O_2
S_3	O_1	O_1

It may be shown that M_1 and M_2 are outwardly equivalent. Inwardly they differ as evidenced by the assignment in Example 5.

EXAMPLE 5 :- An Assignment for M_2 and the Resulting Realization

$\hat{f}^S$	0	1
00	00	10
10	10	11
11	00	10

$\hat{f}^O$	0	1
00	0	0
10	1	1
11	0	0

$$\hat{f}_1^S(y^t, x^t) = x^t + y_1^t \bar{y}_2^t$$

$$\hat{f}_2^S(y^t, x^t) = x^t y_1^t \bar{y}_2^t$$

$$\hat{f}_1^O(y^t, x^t) = y_1^t \bar{y}_2^t$$

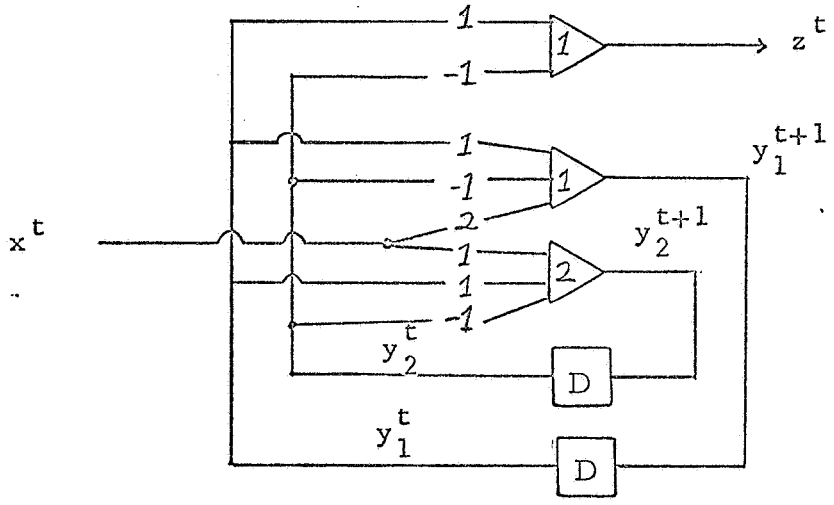


Figure 3 A one-level realization of M_2.

From this example it is seen that M_2 has a one-level threshold realization. For any assignment $\mathcal{P}$ for M_1 there must be some β in $\mathcal{B}$ which separates $\{S_1, S_2\}$. Then $\beta \geq \langle S_1; S_2 \rangle = f^S(\mu)$ where μ is the (c,c) set-pair; $\langle (S_1, I_1)(S_2, I_2); (S_1, I_2), (S_2, I_1) \rangle$. By the terms of Theorem IV, $\mathcal{P}$ fails to be acceptable and so M_1 has no one-level realization; thus M_1 and M_2 differ internally in this very significant respect.

M_2 is said to be equivalent to M_1 and is said to be derived from M_1 by splitting state ${}_1f^S(S_2, I_2)$. The origin of the term state-splitting becomes more obvious when viewing machine behavior in terms of state graphs. It is easily seen that by successively splitting states, starting with any machine M_1, a machine M_n equivalent to M_1 may be obtained which enjoys the following property. No state occurs in more than one column of M_n's state table. The significance of this observation lies in the following Theorem.

THEOREM V. Let $M = \langle I, S, O, f^S, f^O \rangle$ be a machine for which $f^S(S_i, I_j) = f^S(S_k, I_\ell)$ only if $I_j = I_\ell$. (i.e. no state appears in more than one column of M's state table.) Let β_k be the partition $\{ \{S_k\}, S - \{S_k\} \}$ of S and let $\mathfrak{B} = \{\beta_k | S_k \text{ in } S\}$. If α and $\mathfrak{C}$ are any zero-product partition-sets of I and O, and $\mathcal{A}$ is defined according to $\mathfrak{P} = \{\alpha, \mathfrak{B}, \mathfrak{C}\}$, then the state-variable functions for $\mathcal{A}$ are each domain 2-asummable.

PROOF: Show that $\mathfrak{B}$ satisfies the terms of Theorem IV.

The importance of this theorem is not in that it offers a practical alternative when the algorithm fails ($\Delta \equiv O$). Instead the theorem merely makes it possible to assert that if the algorithm fails for a machine M_1,

an equivalent machine M_2 may always be found for which there exists an assignment which is acceptable with respect to the feedback logic. Unfortunately nothing can be done about the output logic in the event that $f^O(\mu) = \langle O_j;O_k \rangle$, $O_j \neq O_k$, for some $(c,c)\,\mu$. There is nothing to be gained by splitting outputs since then logic must be included to identify outputs with same origin.

It is hoped that some state-splitting procedure might be formalized which creates a minimal number of new states and which yields an equivalent machine for which there exists a one-level threshold realization.

V

ACKNOWLEDGEMENT

The authors gratefully acknowledge the support for this work from the National Science Foundation under Grant GP-2724 and from the Joint Services Electronics Program under Grant AF-AFOSR-766-66.

REFERENCES

[1] C. L. Coates and P. M. Lewis, II, "Linearly Separable Switching Functions," J. Franklin Inst. Vol. 272, pp. 366-410, Nov. 1961.

[2] F. O. Hadlock, "Realization of Sequential Machines with Threshold Elements," Ph.D. Dissertation, The University of Texas, August 1966. Also available as Technical Report No. 17, Laboratories for Electronics and Related Science Research, Electrical Engineering Department, The University of Texas.

[3] R. O. Winder, "Threshold Logic," Ph.D. Dissertation, Princeton University, May 1962.

[4] S. B. Akers, "Threshold Logic and Two-Person, Zero-Sum Games," Switching Circuit Theory and Logical Design - AIEE (1961) pp. 27-33, September 1961.

[5] S. H. Cameron, "The Generation of Minimal Threshold Nets by an Integer Program," IEEE Trans. on Electronic Computers, Vol. EC-13, pp. 299-302, June 1964.

[6] C. K. Chow, "Boolean Functions Realizable with Single Threshold Devices," IRE Proceedings, Vol. 39, pp. 370-371, January 1961.

[7] M. Dertouzos, "An Approach to Single Threshold Element Synthesis," IRE Trans. on Electronic Computers, Vol. EC-

[8] C. C. Elgot, "Truth Functions Realizable by Single Threshold Organs," Switching Circuit Theory and Logical Design - AIEE S-134, pp. 225-245, September 1961.

[9] I. J. Gabelman, "The Synthesis of Boolean Functions Using A Single Threshold Element," IRE Trans. on Electronic Computers, Vol. EC-11, pp. 639-643, October 1962.

[10] C. A. Gaston, "A Simple Test for Linear Separability," IEEE Trans. on Electronic Computers, Vol. EC-12, pp. 134, April 1963.

[11] R. C. Minnick, "Linear Input Logic," IRE Trans. on Electronic Computers, Vol. EC-10, pp. 6-17, March 1961.

[12] S. Muroga, I. Toda, and S. Takasu, "Theory of Majority Decision Elements," J. Franklin Inst., Vol. 271, pp. 376-418, May 1961.

[13] C. L. Sheng, "A Method for Testing and Realization of Threshold Functions," IEEE Trans. on Electronic Computers, Vol. EC-13, pp. 232-240, June 1964.

[14] H. C. Torng, "An Approach for the Realization of Linearly-Separable Switching Functions," IEEE Trans. on Electronic Computers, Vol. EC-15, pp. 14-21, February 1966.

Logic Hazards in Threshold Networks

A. BART HOWE, MEMBER, IEEE, AND CLARENCE L. COATES, SENIOR MEMBER, IEEE

Abstract—This paper is concerned with the study of logic hazards in threshold gate networks. Eichelberger has proved that logic hazards are not present in a sum-of-product (product-of-sum) realization which realizes all of the 1(0) prime implicants of the given Boolean function.[2] Logic gates of the AND or NOR (OR or NAND) variety realize single 1(0) prime implicants; therefore, a gate is required for each 1(0) prime implicant to be realized, and the problem of eliminating logic hazards is straightforward.

A single-threshold gate, however, realizes a number of prime implicants. Moreover, the number of prime implicants realized by a network that incorporates more than a single-threshold gate is not uniquely determined either by the Boolean function being realized or by the number of gates involved. As a result, it is often possible to control the prime implicants and hence the hazards without greatly increasing the number of gates required. In fact, in some cases no additional gates are required.

Three methods are presented for determining if a given threshold realization contains any logic hazards, the first of which is an extension of McCluskey's work.[5] Two methods are then presented for synthesizing logic hazard-free threshold realizations. The first method is based on the tree method of synthesizing threshold gate networks, whereas the second method is based on expressing the given Boolean function as a sum of threshold functions.

Index Terms—Combinational logic, logic hazards, static hazards, threshold networks.

Threshold Definitions and Theorems

THE NOTATION of Lewis and Coates[1] will be used and will be briefly reviewed here.

A threshold gate has binary inputs $x_1, \cdots, x_n$ and a binary output y. The threshold gate has an internal threshold T, and each binary input has an internal weight a_i. Let $\{0, 1\}^n$ denote the collection of 2^n n-tuples of $x_1, \cdots, x_n$. Associated with each gate is a function f which is defined on $\{0, 1\}^n$ as

$$f(p) = \sum_{i=1}^{n} a_i x_i(p)$$

where $p \in \{0, 1\}^n$, $x_i(p)$ is the value of x_i at p, and where normal arithmetic operations are used. The function f is called the separating function.

If F is a Boolean function defined on $\{0, 1\}^n$, then F is linearly separable if and only if there exist numbers $a_1, \cdots, a_n$ and T such that $f(p) \geq T \Leftrightarrow F(p) = 1$ and $f(p) < T \Leftrightarrow F(p) = 0$.

Manuscript received May 11, 1967; revised November 13, 1967. This work was supported by NSF under Grants GP-2724 and GK-1146X, by NASA under Grant NGR-44-012-049, and by the Joint Services Program under Grants AF-AFOSR-766-66 and AF-AFOSR-766-67.

A. B. Howe was with the University of Texas, Austin, Tex. He is now with the Systems Development Div., IBM Corporation, Poughkeepsie, N. Y.

C. L. Coates is with the Laboratories for Electronics and Related Science Research, University of Texas, Austin, Tex.

The Boolean function F, which is realized by a threshold gate with threshold T and separating function f, can be represented as

$$F(x_1, \cdots, x_n) = \langle f(x_1, \cdots, x_n) \rangle_T = \langle a_1 x_1 + \cdots + a_n x_n \rangle_T.$$

The collection $\{F(p), f(p)\}$ is called the map of F. Let u denote the smallest $f(p)$ such that $F(p) = 1$ and l the largest $f(p)$ such that $F(p) = 0$. A map of F is separated if $l < u$. The gap for a separated map is the set of real numbers z such that $l < z \leq u$ and is denoted by $u:l$. If F is linearly separable, then for some f it follows that $l < T \leq u$. In terms of this f, the previous expression can be written as

$$F(x_1, \cdots, x_n) = \langle a_1 x_1 + \cdots + a_n x_n \rangle_{u:l} = \langle f \rangle_{u:l}.$$

Obviously, all Boolean functions are not linearly separable; hence, they cannot be realized with one threshold gate. When such a case occurs, the multigate realization can be represented as

$$\langle f(p) \rangle_{u:l} = \left\langle \sum_{i=1}^{m} \beta_i y_i(p) \right\rangle_{u:l}$$

where $u:l$ is the gap of the output gate, y_i the Boolean function realized by the ith input, and β_i the associated weight.

When using the reconstruction technique of Lewis and Coates,[1] the addition of a gate is accomplished by using the following theorem.

Theorem A:

$$[\langle f \rangle_{u:l}] = [\langle f + a \cdot 0 \rangle_{u:l}] = [\langle f + a \cdot 1 \rangle_{u+a:l+a}]$$

where 0 and 1 represent constant Boolean functions.

The constant functions, in Theorem A, represent gates which have been added. It has been shown[1] that the separating functions for these can be $0 = \langle 0 \rangle_{\infty:0}$ and $1 = \langle 0 \rangle_{0:-\infty}$.

Lewis and Coates[1] give a step-by-step procedure for the tree realization technique and numerous examples. Examples 3 and 4 also will illustrate this technique.

Definitions Concerning the Boolean Function F and Its Realization

Let p represent a variable on $\{0, 1\}^n$, where the ith component of p is the variable x_i^*. In this space x_i^* can be represented by either the literal x_i or $\bar{x}_i$ since $x_i = 1$ if and only if $\bar{x}_i = 0$.

A subcube K of $\{0, 1\}^n$ is denoted by $\{p \mid x_1^* = b_1, x_2^* = b_2, \cdots, x_m^* = b_m\}$, where $b_i \in \{0, 1\}$. For a Boolean

function F: $\{0, 1\}^n \rightarrow \{0, 1\}$, 1 and 0 subcubes are denoted by K_1 and K_0, respectively, and when these are prime implicants they are denoted by P_1 and P_0, respectively.

McCluskey[5] has shown that for detecting logic hazards it is necessary to draw a distinction between the literals of the Boolean function and the literals of the realization, the distinction being that in the realization the literal x_i and its complement must be treated independently of each other, whereas in the Boolean function F this is not true. The necessity of this distinction is based on the fact that during an input state change it is possible for the input lines x_i and $\bar{x}_i$ to be temporarily the same and, as shown in Huffman[4] and McCluskey,[5] it is exactly this property that causes a hazard. Henceforth let F^t denote the *transient* or *output function* that is realized by a given realization when x_i and its complement are treated as independent variables.

When considering the Boolean function F, the "barred" literal $\bar{x}_i$ will be used to denote the complement of x_i. The literals x_i and $\bar{x}_i$ are not independent of each other, whereas when considering the realization of F, the "primed" literal x_i' will be used to denote the input literal which is independent of x_i but which would be the complement of x_i if an input state change is not occurring (i.e., the input literals corresponding to $\bar{x}_i$ are represented as x_i').

For a given realization, F^t can be obtained by replacing all complement $(-)$ variables and operations by the corresponding prime $(')$ variables and operations where the only allowable identities are $(x')' = x$, $(x+y+\cdots+z)' = x' \cdot y' \cdots z'$, and $(x \cdot y \cdots z)' = x' + y' + \cdots + z'$.

Since F^t may contain both x_i and x_i', the domain of the transient function is $\{0, 1\}^{2n}$. Hence, when studying hazards the problem becomes that of distinguishing between the properties of F and the properties of F^t. Before continuing, note that since $x_i' = \bar{x}_i$ for the steady state condition, it follows that if x_i' is replaced by $\bar{x}_i$ in the function F^t, and if the usual Boolean operations are used, then F can be obtained from F^t.

Consider now the relations between F^t and F. Let q represent a variable on $\{0, 1\}^{2n}$, where the $(2i-1)$th component is x_i and the $2i$th component is x_i', and where x_i and x_i' are considered as independent of each other. Thus q is a function of $x_1, x_1', x_2, x_2', \cdots, x_n, x_n'$.

Definition 1: For each point p of $\{0, 1\}^n$ such that $p = (x_1^* = b_1, \cdots, x_n^* = b_n)$, there is a point q_p of $\{0, 1\}^{2n}$, called the *image point of* p, such that $q_p = (x_1^* = b_1, x_1^{*\prime} = \bar{b}_1, \cdots, x_n^* = b_n, x_n^{*\prime} = \bar{b}_n)$.

For example, consider the point $p = (x_1 = 1, x_2 = 0, x_3 = 1)$ of the space $\{0, 1\}^3$. The image point is $q_p = (x_1 = 1, x_1' = 0, x_2 = 0, x_2' = 1, x_3 = 1, x_3' = 0)$.

Definition 2: For each subcube K of $\{0, 1\}^n$ such that $K = \{p \mid x_1^* = b_1, \cdots, x_m^* = b_m\}$, there is an *image subcube* S of $\{0, 1\}^{2n}$ such that $S = \{q \mid x_1^* = b_1, x_1^{*\prime} = \bar{b}_1, \cdots, x_m^* = b_m, x_m^{*\prime} = \bar{b}_m\}$.

For example, consider the subcube $K = \{p \mid x_1 = 1, x_3 = 0\}$ of $\{0, 1\}^3$. The corresponding image subcube is $S = \{q \mid x_1 = 1, x_1' = 0, x_3 = 0, x_3' = 1\}$.

Obviously, if a subcube K contains 2^{n-m} points, the image subcube S will contain $2^{2(n-m)}$ points and 2^{n-m} of these points will be the image points of the points of K.

Note that since the inputs x_i and x_i' in a realization may not change simultaneously, it is possible for the two to be temporarily the same. Hence the realization $\langle f \rangle_{u:l}$ is actually defined on the space $\{0, 1\}^{2n}$. In fact, now that an image point and output function have been defined, a realization can be redefined as follows.

Definition 3: Let F be an arbitrary Boolean function and F^t the output function of an arbitrary realization $\langle f \rangle_{u:l}$ of F, where F and F^t are defined on $\{0, 1\}^n$ and $\{0, 1\}^{2n}$, respectively. Let p be an arbitrary point of $\{0, 1\}^n$ and q_p the corresponding image point. Then $\langle f \rangle_{u:l}$ is said to realize F if and only if for every $p \in \{0, 1\}^n$

$$F(p) = 1 \Leftrightarrow f(q_p) \geq T \qquad \text{(i.e., } F^t(q_p) = 1\text{)}$$

$$F(p) = 0 \Leftrightarrow f(q_p) < T \qquad \text{(i.e., } F^t(q_p) = 0\text{)}.$$

Note that Definition 3 does not place any requirement upon the nonimage points of $\{0, 1\}^{2n}$. Hence $F^t(q)$, for the nonimage points, can be either 1 or 0. These points, however, do determine the hazard conditions of the realization.

Definition 4: Let F^t be the transient function of an arbitrary threshold realization $\langle f \rangle_{u:l}$. A 1(0) *subcube of* F^t is a subcube S of $\{0, 1\}^{2n}$ such that $F^t(q) = 1$ (0) for all $q \in S$.

A logic hazard, first defined by Eichelberger,[2] can be defined in the following equivalent manner in terms of F^t.

Definition 5: A realization $\langle f \rangle_{u:l}$ of F contains a logic 1(0) *hazard* within the 1(0) subcube $K_1(K_0)$ of F if and only if the corresponding image subcube S_1 (S_0) of $\{0, 1\}^{2n}$ is not a 1(0) subcube of F^t, where F^t is the transient function of $\langle f \rangle_{u:l}$.

Consider the function $F^t(x_1, x_1', \cdots, x_n, x_n')$ and the subcube $S = \{q \mid x_1 = b_1, x_1' = \bar{b}_1, \cdots, x_m = b_m, x_m' = \bar{b}_m\}$. The function which results when x_i and x_i' are set equal to b_i and $\bar{b}_i$, respectively, in the function F^t, is referred to as a *reduced function* of F^t and is denoted by $F^t(S)$. $F^t(S) = 1(0)$ implies that the reduced function $F^t(S)$ is the constant function 1(0). By using the idea of reduced functions, Definition 5 can also be expressed in the following equivalent manner.

Definition 6: A realization $\langle f \rangle_{u:l}$ of F contains a *logic* 1(0) *hazard* within the 1(0) subcube $K_1(K_0)$ of F if and only if $F^t(S_1) \neq 1$ $(F^t(S_0) \neq 0)$, where $S_1(S_0)$ is the image subcube of $K_1(K_0)$, and F^t the transient function of $\langle f \rangle_{u:l}$.

Detection of Logic Hazards—Method 1

The following theorem will now give a method for determining if a realization $\langle f \rangle_{u:l}$ of F contains any logic

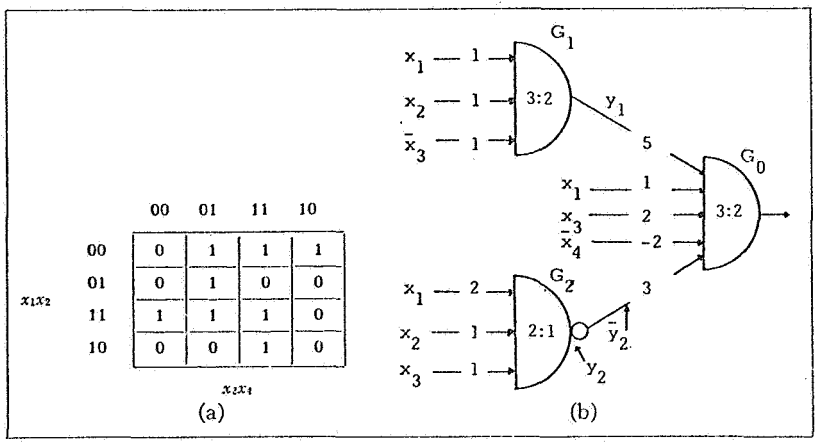


Fig. 1. Karnaugh map and realization for Example 1.

hazards. The proof follows directly from Definitions 2 and 6.

Theorem 1: Let $\{P_1^i\}$ ($\{P_0^i\}$) denote the set of 1(0) prime implicants of F and $\{S_1^i\}$ ($\{S_0^i\}$) the corresponding set of image subcubes in the space $\{0,1\}^{2n}$. A realization $\langle f\rangle_{u:l}$ of F will not contain any logic 1(0) hazards if and only if $F^t(S_1^i)=1$, ($F^t(S_0^i)=0$) for all $S_1^i \in \{S_1^i\}$, ($S_0^i \in \{S_0\}$), where F^t is the transient function of $\langle f\rangle_{u:l}$.

Summarizing, Theorem 1 can be used to determine if a given realization contains a logic hazard. If it does, then Definition 5 or 6 can be used to determine the subcube within which the logic hazard occurred. The following procedure can be used to determine if a realization contains any logic 1(0) hazards.

Procedure 1:

1) Determine the transient function F^t of $\langle f\rangle_{u:l}$.
2) Determine the set of 1(0) prime implicants $\{P_1^i\}$ ($\{P_0^i\}$) of F.
3) Determine $F^t(S_1^i)$, ($F^t(S_0^i)$) for all $S_1^i(S_0^i)$, where $S_1^i(S_0^i)$ is the image subcube of $P_1^i(P_0^i)$.
4) The realization $\langle f\rangle_{u:l}$ does not contain a logic 1(0) hazard within $P_1^i(P_0^i)$ if and only if $F^t(S_1^i) = 1(F^t(S_0^i)=0)$.

The following example will illustrate Procedure 1.

Example 1: Consider the Boolean function

$$F(x_1, x_2, x_3, x_4) = x_1x_2\bar{x}_3 + x_1x_3x_4 + \bar{x}_1\bar{x}_2x_3 + \bar{x}_1\bar{x}_3x_4. \quad (1)$$

The Karnaugh map and a realization are given in Fig. 1.

The problem is to determine if the given realization contains any logic hazards. From Procedure 1, the first step is to obtain F^t. In this case, F^t is given by

$$F^t = x_1x_2x_3' + x_1'x_2'x_4 + x_1'x_2'x_3 + x_1'x_3'x_4 + x_1'x_3'x_3 + x_1x_3x_4. \quad (2)$$

Table I contains the set of 1(0) prime implicants $\{P_1^i\}$ ($\{P_0^i\}$) of F and the corresponding set of reduced functions $\{F^t(S_1^i)\}$, ($\{F^t(S_0^i)\}$).

For example, from Table I, $P_1^4 = \{p \mid x_2=0,\ x_3=1,\ x_4=1\}$; then $S_1^4 = \{q \mid x_2=0,\ x_2'=1,\ x_3=1,\ x_3'=0,\ x_4=1,\ x_4'=0\}$. Hence the reduced function $F^t(S_1^4)$ is $F^t(x_2=0, x_2'=1, x_3=1, x_3'=0, x_4=1, x_4'=0) = x_1'+x_1$.

TABLE I
TABLE FOR EXAMPLE 1

	P_1						P_0^1					
i	x_1	x_2	x_3	x_4	$F^t(S_1^i)$	$\tilde{f}(q_0^i)$	x_1	x_2	x_3	x_4	$F^t(S_0^i)$	$\tilde{f}(q_1^i)$
1	1	1	0	—	1	6	—	1	1	0	0	3
2	0	0	1	—	1	5	0	1	—	0	$x_1'x_3x_2$	5
3	0	—	0	1	1	5	1	0	0	—	0	3
4	—	0	1	1	x_1+x_1'	4	—	0	0	0	0	4
5	1	—	1	1	1	5	0	1	1	—	0	4
6	0	0	—	1	1	5	1	—	1	0	0	3
7	1	1	—	1	x_3+x_3'	3	1	0	—	0	0	3
8	—	1	0	1	x_1+x_1'	2	0	—	0	0	0	3

Referring to Table I, $F^t(S_0^4)\neq 1$, $F^t(S_1^7)\neq 1$, $F^t(S_1^8)\neq 1$, and $F^t(S_0^2)\neq 0$. [For the present, disregard the columns labeled $\tilde{f}(q_0^i)$ and $\tilde{f}(q_1^i)$]. From Definition 6, the realization will contain a logic 1 hazard in P_1^4, P_1^7, and P_1^8 and a logic 0 hazard in P_0^2.

Procedure 1 requires the calculation of the transient function F^t and the calculation of all of the prime implicants of F. McCluskey[5] has presented several alternate methods for determining if a given realization contains any static hazards, all of which require the calculation of F^t. These methods can be readily extended to include logic hazards and for further detail see McCluskey.[5]

Before continuing, note that if the realization contains negative weights and/or inverting gates, one cannot determine F^t by successively applying the 2^{2n} possible combinations of $\{0,1\}^{2n}$ as inputs to the realization and determining the value of the output for each. In terms of the separating function this gives the surprising result that

$$F^t(q) = 1(0) \not\Rightarrow f(q) \geq u(f(q) \leq l)$$

or

$$f(q) \geq u(f(q) \leq l) \not\Rightarrow F^t(q) = 1(0)$$

where $q \in \{0,1\}^{2n}$.

For example, consider the point $q=(x_1=0,\ x_1'=0,\ x_2=0,\ x_2'=1,\ x_3=1,\ x_3'=0,\ x_4=1,\ x_4'=0)$. Referring to Fig. 1(b), $f_1(q)=0$; hence, the output of G_1 is 0. Also, $f_2(q)=1$; hence, the output of G_2 is 1. Likewise, $f_0(q)=5$; thus, the output of the realization is 1. Now referring to (2), $F^t(q)=0$. Hence $F^t(q)=0$, whereas $f(q)>u$.

The next section will be concerned with modifying the given realization $\langle f\rangle_{u:l}$ in such a manner that F^t can always be obtained by considering only the 2^{2n} possible input states. This modification will, in many cases, give an easier method for determining F^t. It will also yield a method for determining if a realization contains any logic hazards which does not require the calculation of F^t. But even more important, it will develop the fundamentals which will be needed to synthesize hazard-free threshold gate networks.

Detection of Logic Hazards—Method 2

Lemma 1 concerns the modification of a given realization $\langle f\rangle_{u:l}$ and Theorem 2 provides the desired results. Proof of each are given in Howe and Coates.[7]

Definition 7: A threshold realization which does not contain any negative weights or inverting gates will be called a *positive threshold realization* and will be denoted by $\langle \bar{f}\rangle_{\bar{u}:\bar{l}}$.

Lemma 1: For each realization $\langle f\rangle_{u:l}$ of F there exists a unique corresponding positive realization $\langle \bar{f}\rangle_{\bar{u}:\bar{l}}$ of F such that the transient functions of the two are the same.

Because of Lemma 1 no loss of generality results by considering only positive realizations. Moreover, Howe and Coates[7] give a procedure for converting a given realization $\langle f\rangle_{u:l}$ to the corresponding positive realization $\langle \bar{f}\rangle_{\bar{u}:\bar{l}}$.

Definition 8: Consider the subcube $S=\{q \mid x_1=b_1,\ x_1'=\bar{b}_1,\ \cdots,\ x_m=b_m,\ x_m'=\bar{b}_m\}$ of the space $\{0,\ 1\}^{2n}$. Define q_1 as the element $(x_1=b_1,\ x_1'=\bar{b}_1,\ \cdots,\ x_m=b_m,\ x_m'=\bar{b}_m,\ x_{m+1}=1,\ \cdots,\ x_n=1,\ x_n'=1)$ and q_0 as the element $(x_1=b_1',\ x_1'=\bar{b}_1,\ \cdots,\ x_m=b_m,\ x_m'=\bar{b}_m,\ x_{m+1}=0,\ \cdots,\ x_n=0,\ x_n'=0)$. The elements q_0 and q_1 will be called the *minimum* and *maximum elements* of S, respectively.

A theorem can now be given by which it can be determined, without the calculation of F^t, if a given realization contains a logic hazard within a specific subcube. The proof is given in Howe and Coates.[7]

Theorem 2: Let $\langle f\rangle_{u:l}$ be an arbitrary realization of F, and $\langle \bar{f}\rangle_{\bar{u}:\bar{l}}$ the corresponding positive realization. Let $\{P_1^i\}(\{P_0^i\})$ be the set of 1(0) prime implicants of F, $\{S_1^i\}(\{S_0^i\})$ the corresponding set of image subcubes, and $\{q_0^i\}(\{q_1^i\})$ the corresponding set of minimum (maximum) elements. The realization $\langle f\rangle_{u:l}$ will not contain any logic 1(0) hazards if and only if $\bar{f}(q_0^i)\geq\bar{u}(\bar{f}(q_1^i)\leq\bar{l})$ for all $q_0^i\in\{q_0^i\}(q_1^i\in\{q_1^i\})$.

The following procedure will outline a method for determining if a given realization contains any logic 1(0) hazards.

Procedure 2:

1) Determine the set of 1(0) prime implicants $\{P_1^i\}(\{P_0^i\})$ of F.
2) Obtain the corresponding positive realization $\langle \bar{f}\rangle_{\bar{u}:\bar{l}}$ of $\langle f\rangle_{u:l}$.
3) Determine $\bar{f}(q_0^i)(\bar{f}(q_1^i))$ for all $q_0^i(q_1^i)$, where $q_0^i(q_1^i)$ is the corresponding minimum (maximum) image point of $P_1^i(P_0^i)$.
4) The realization $\langle f\rangle_{u:l}$ will not contain a logic 1(0) hazard within $P_1^i(P_0^i)$ if and only if $\bar{f}(q_0^i)\geq\bar{u}$, $(\bar{f}(q_1^i)\leq\bar{l})$.

Suppose that we apply Procedure 2 to (1). The set of prime implicants are given in Table I and the corresponding positive realization is given by

$$\langle x_1+2x_3+2x_4+5\langle x_1+x_2+\bar{x}_3\rangle_{3:2}+3\langle 2\bar{x}_1+\bar{x}_2+\bar{x}_3\rangle_{3:2}\rangle_{5:4}. \quad (3)$$

Now consider step 3). For example, consider the 1 prime implicant P_1^4. The corresponding image subcube is $S_1^4=\{q \mid x_2=0,\ x_2'=1,\ x_3=1,\ x_3'=0,\ x_4=1,\ x_4'=0\}$. Hence the corresponding minimum image point is $q_0^4=(x_1=0,\ x_1'=0,\ x_2=0,\ x_2'=1,\ x_3=1,\ x_3'=0,\ x_4=1,\ x_4'=0)$. From (3), $\bar{f}(q_0^4)=4$. Since $\bar{u}=5$, it follows from Theorem 2 that the realization will contain a logic 1 hazard in P_1^4.

Table I contains $\bar{f}(q_0^i)$ for each $P_1^i\in F$ and $\bar{f}(q_1^i)$ for each $P_0^i\in F$. As can be seen, the results of Procedure 2 agree with those of Procedure 1.

Detection of Logic Hazards—Method 3

Methods 1 and 2 for detecting logic hazards were based on either F^t or the input–output relations of the realization. A method will now be given which will be based on the structure of the realization. As will be seen, this latter method is inferior to the previous two for detecting logic hazards; however, the results obtained from this method are needed to obtain a theorem for synthesizing a hazard-free threshold network.

Definition 9: Let $\langle f\rangle_{u:l}$ denote an arbitrary realization of the Boolean function F, and B a set of gates contained in $\langle f\rangle_{u:l}$. The set of gates B will be defined as an *output connected subset* of gates if B contains the output gate of $\langle f\rangle_{u:l}$ and, excluding the output gate of $\langle f\rangle_{u:l}$, the output of each gate in B is an input to another gate in B.

For example, in Fig. 1(b) the sets $\{G_1,\ G_0\}$ and $\{G_0\}$ are output-connected subsets of $\langle f\rangle_{u:l}$, whereas the set $\{G_1,\ G_2\}$ is not.

Definition 10: Let G_i denote an arbitrary gate in the positive realization $\langle \bar{f}\rangle_{\bar{u}:\bar{l}}$, and B^* an arbitrary *output-connected subset* of $\langle \bar{f}\rangle_{\bar{u}:\bar{l}}$. Also, let S be an arbitrary subcube in the space $\{0,\ 1\}^{2n}$ with minimum element q_0 and maximum element q_1. The set B^* is defined as a 1(0) *branch of* $\langle \bar{f}\rangle_{\bar{u}:\bar{l}}$ *which realizes* S if, when $\langle \bar{f}\rangle_{\bar{u}:\bar{l}}$ is modified such that all gates not in B^* have 0(1) output, then the value of $\bar{f}_i(q_0)$, $(\bar{f}_i(q_1))$ for the modified realization satisfies the condition $\bar{f}_i(q_0)\geq\bar{u}_i$, $(f_i(q_1)\leq\bar{l}_i)$ for all $G_i\in B^*$.

For example, consider the realization of (3), which is

$$\langle \bar{f} \rangle_{\bar{u}:\bar{l}} = \langle x_1 + 2x_3 + 2x_4 + 5\langle x_1 + x_2 + \bar{x}_3 \rangle_{3:2} \quad (G_0, G_1)$$

$$+3\langle 2\bar{x}_1 + \bar{x}_2 + \bar{x}_3 \rangle_{3:2} \rangle_{5:4}. \quad (G_2)$$

Suppose one wishes to determine if the set of gates $\{G_0, G_1\}$ is a 1 branch which realizes the subcube $S = \{q \mid x_1 = 1, x_1' = 0, x_2 = 1, x_2' = 0, x_3 = 0, x_3' = 1\}$. According to Definition 10, in order for $\{G_0, G_1\}$ to be a 1 branch which realizes S the condition $\bar{f}_1(q_0) \geq \bar{u}_1$ and $\bar{f}_0(q_0) \geq \bar{u}_0$ must exist when the output of G_2 is 0. Under the condition $G_2 = 0$, the modified realization becomes

$$\langle x_1 + 2x_3 + 2x_4 + 5\langle x_1 + x_2 + \bar{x}_3 \rangle_{3:2} + 0 \rangle_{5:4}. \quad (G_0, G_1)$$

The point q_0 is $(x_1 = 1, x_1' = 0, x_2 = 1, x_2' = 0, x_3 = 0, x_3' = 1, x_4 = 0, x_4' = 0)$. From the preceding equation, $\bar{f}_1(q_0) = 3$ and $\bar{f}_0(q_0) = 6$. Thus from Definition 10, the set $\{G_0, G_1\}$ is a 1 branch which realizes S.

The following lemma will give a relationship between the term $\bar{f}(q_0)$, $(\bar{f}(q_1))$ and a 1(0) branch.

Lemma 2: Let $\langle \bar{f} \rangle_{u:l}$ be an arbitrary positive realization of F. Also let $K_1(K_0)$ be an arbitrary 1(0) subcube of F, the set $\{q_p\}_1(\{q_p\}_0)$ the corresponding set of image points of $K_1(K_0)$, and $q_0(q_1)$ the corresponding minimum (maximum) element of $K_1(K_0)$. Then $\bar{f}(q_0) \geq \bar{u}$, $(\bar{f}(q_1) \leq \bar{l})$ if and only if there exists a 1(0) branch $B_1(B_0)$ of $\langle \bar{f} \rangle_{\bar{u}:\bar{l}}$ which realizes all $q_p \in \{q_p\}_1(\{q_p\}_0)$.

Proof: First we will prove that all $q_p \in \{q_p\}_1$ are realized by a particular 1 branch B_1 implies that $\bar{f}(q_0) \geq \bar{u}$. This result will be proved by induction.

Let G_j be an arbitrary gate of B_1 which is on the r logic level of $\langle \bar{f} \rangle_{\bar{u}:\bar{l}}$. Let G_i be an arbitrary gate of B_1 such that its output is an input to G_j (i.e., G_i must be on the $r+1$ or greater logic level). Let $K_1 = \{p \mid x_1^* = b_1, \cdots, x_m^* = b_m\}$ be an arbitrary 1 subcube of F.

Assume that Lemma 2 is true for all G_i (i.e., assume $\bar{f}_i(q_0) \geq u_i$). Thus the output of G_i is not a function of $x_{m+1}, x_{m+1}', \cdots, x_n, x_n'$.

Now consider the independent inputs to G_j. Let $\hat{x}_i$ denote an independent input to G_j. Assume that G_j does not have both $\hat{x}_i$ and its complement $\hat{x}_i'$ as inputs, which is true for all gates. From the hypothesis of Lemma 2, it is known that the output of G_j is a 1 for all $q_p \in \{q_p\}_1$. Hence there exists a q_p for which $\hat{x}_{m+1} = 1, \cdots, \hat{x}_n = 1$ and some other q_p for which $\hat{x}_{m+1} = 0, \cdots, \hat{x}_n = 0$ such that the output of G_j is 1 for both. Thus the output of G_j is not a function of $\hat{x}_{m+1}, \cdots, \hat{x}_n$. Now since the corresponding complements $\hat{x}_{m+1}', \cdots, \hat{x}_n'$ are not inputs to G_j and since the output of G_i is not a function of $x_{m+1}, x_{m+1}', \cdots, x_n, x_n'$, it follows that the output of G_j is not a function of $x_{m+1}, x_{m+1}', \cdots, x_n, x_n'$. Thus $\bar{f}_j(q_0) \geq \bar{u}_j$. Hence we have proved that if Lemma 2 is true for all G_i, it is true for G_j.

Next consider an arbitrary input gate of the branch. Since for the modified realization (i.e., all gates not contained in B_1 have 0 outputs) the gate contains only independent inputs, and since it does not contain both x_i and x_i', it follows by the preceding reasoning that Lemma 2 is true for all input gates of B_1. Thus it follows from induction that $\bar{f}(q_0) \geq \bar{u}$.

Next it will be proved that $\bar{f}(q_0) \geq u_0$ implies that all $q_p \in \{q_p\}_1$ are realized by the same 1 branch.

First, assume $\bar{f}(q_0) \geq \bar{u}$; hence, the output of $\langle \bar{f} \rangle_{\bar{u}:\bar{l}}$ is a 1 at q_0. Let B^* be the largest output-connected subset of gates that have unit output at q_0. Clearly this is nonempty. Since all coefficients are positive and no inverters exist in the realization, then the gates of B^* will have unit output for all $q \in S_1$, where S_1 is the image subcube of K_1, independent of the outputs of the gates not in B^*. Hence B^* is a 1 branch which realizes all $q \in S_1$, and hence all $q_p \in \{q_p\}_1$.

The proof concerning the inequality $\bar{f}(q_1) \leq \bar{l}$ is similar and will be omitted.

Theorem 3 follows directly from Lemma 2 and the proof of Theorem 2.

Theorem 3: Let $\langle f \rangle_{u:l}$ be an arbitrary realization of F, and $\langle \bar{f} \rangle_{\bar{u}:\bar{l}}$ the corresponding positive realization. Also, let $K_1(K_0)$ be an arbitrary 1(0) subcube of F, and the set $\{q_p\}_1(\{q_p\}_0)$ the corresponding set of image points for all $p \in K_1(K_0)$. The realization $\langle f \rangle_{u:l}$ will not contain a logic 1(0) hazard in $K_1(K_0)$ if and only if there exists a 1(0) branch $B_1(B_0)$ of $\langle \bar{f} \rangle_{\bar{u}:\bar{l}}$ which realizes all $q_p \in \{q_p\}_1 \cdot (\{q_p\}_0)$.

Summarizing, we have proved the following facts. Let K^α be an arbitrary 1(0) subcube of F, and S^α and $q_0^\alpha(q_1^\alpha)$ the corresponding image subcube and minimum (maximum) element, respectively. An arbitrary realization $\langle f \rangle_{u:l}$ of F will not contain a logic 1(0) hazard within $K^\alpha \Leftrightarrow F^t(S^\alpha) = 1(0) \Leftrightarrow \bar{f}(q_0^\alpha) \geq \bar{u}$, $(\bar{f}(q_1^\alpha) \leq \bar{l})$, if and only if there exists a 1(0) branch $B_1(B_0)$ of $\langle \bar{f} \rangle_{\bar{u}:\bar{l}}$ which realizes all of the image points of K^α, where $\langle \bar{f} \rangle_{\bar{u}:\bar{l}}$ is the corresponding positive realization.

As in the preceding cases, a theorem can now be given for determining if a given realization contains any logic hazards. The proof follows from Theorems 2 and 3.

Theorem 4: Let $\langle f \rangle_{u:l}$ be an arbitrary realization of F, and $\langle \bar{f} \rangle_{u:l}$ the corresponding positive realization. Also let $\{P_1^i\}(\{P_0^i\})$ be the set of all 1(0) prime implicants of F, and $\{\{q_p\}_1^i\}(\{\{q_p\}_0^i\})$ the corresponding collection of sets of image points. The realization $\langle f \rangle_{u:l}$ will not contain any logic 1(0) hazards if and only if there exists a 1(0) branch $B_1^i(B_0^i)$ of $\langle \bar{f} \rangle_{\bar{u}:\bar{l}}$ which realizes all $q_p \in \{q_p\}_1^i \cdot (\{q_p\}_0^i)$ for all $\{q_p\}_1^i \in \{\{q_p\}_1^i\}$, $(\{q_p\}_0^i \in \{\{q_p\}_0^i\})$.

Example 2 will illustrate the application of Theorem 4.

Example 2: Again consider the realization of Fig. 1(b). From (3) the positive realization is

$$\langle \bar{f} \rangle_{\bar{u}:\bar{l}} = \langle x_1 + 2x_3 + 2x_4 + 5\langle x_1 + x_2 + \bar{x}_3 \rangle_{3:2} + 3\langle 2\bar{x}_1 + \bar{x}_2 + \bar{x}_3 \rangle_{3:2} \rangle_{5:4}. \quad (4)$$

TABLE II
TABLE FOR EXAMPLE 1

j	p_j	q_p^j	B_1^n
	$x_1 x_2 x_3 x_4$	$x_1 x_1' x_2 x_2' x_3 x_3' x_4 x_4'$	
1	1 1 0 0	1 0 1 0 0 1 0 1	$\{G_1, G_0\}$
2	1 1 0 1	1 0 1 0 0 1 1 0	$\{G_1, G_0\}$
3	1 1 1 1	1 0 1 0 1 0 1 0	$\{G_0\}$
4	1 0 1 1	1 0 0 1 1 0 1 0	$\{G_0\}$
5	0 1 0 1	0 1 1 0 0 1 1 0	$\{G_0, G_2\}$
6	0 0 0 1	0 1 0 1 0 1 1 0	$\{G_0, G_2\}$
7	0 0 1 1	0 1 0 1 1 0 1 0	$\{G_0, G_2\}$
8	0 0 1 0	0 1 0 1 1 0 0 1	$\{G_0, G_2\}$

Only the logic 1 hazards will be considered. Therefore, Table II will contain only the points p_j of $\{0, 1\}^n$ such that $F(p_j)=1$, the corresponding image points q_p^j, and the set of 1 branches which realize each q_p^j. Actually, in this example each point q_p^j is realized by only one 1 branch.

In Table II, the points $p_j \in \{0, 1\}^n$ such that $F(p_j)=1$ can be obtained from Fig. 1(a), whereas the corresponding 1 branch can be obtained from (4).

Consider the 1 prime implicant $\{p | x_1=1,\ x_2=1,\ x_3=0\}$. The corresponding image points are $q_p^1=(x_1=1,\ x_1'=0,\ x_2=1,\ x_2'=0,\ x_3=0,\ x_3'=1,\ x_4=0,\ x_4'=1)$ and $q_p^2=(x_1=1,\ x_1'=0,\ x_2=1,\ x_2'=0,\ x_3=0,\ x_3'=1,\ x_4=1,\ x_4'=0)$. Referring to Table II, both image points are realized by the 1 branch $\{G_1, G_0\}$. Therefore, by Theorem 3, the realization will not contain a logic 1 hazard within $\{p | x_1=1, x_2=1, x_3=0\}$.

Next consider the 1 prime implicant $\{p | x_1=1, x_2=1, x_4=1\}$. The corresponding image points are q_p^2 and q_p^3. Referring to Table II, q_p^2 and q_p^3 are realized by the 1 branches $\{G_1, G_0\}$ and $\{G_0\}$, respectively. Hence from Theorem 3, the realization will contain a logic 1 hazard within $\{p | x_1=1, x_2=1, x_4=1\}$.

Likewise, it can be determined that the realization will contain a logic 1 hazard within the 1 prime implicants $\{p | x_2=1,\ x_3=0,\ x_4=1\}$ and $\{p | x_2=0,\ x_3=1,\ x_4=1\}$.

As previously mentioned, the application of Theorem 3 or 4 to determine if a given realization contains any logic hazards is considerably more involved than Procedure 2. Hence it is doubtful if Theorem 3 or 4 would be used to detect logic hazards. However, Theorem 4 will be used, in conjunction with a later lemma, to derive a theorem for synthesizing a hazard-free threshold network directly from the Boolean function F. How this is accomplished is the subject of the next section.

SYNTHESIS OF HAZARD-FREE THRESHOLD NETWORKS

This section will be concerned with synthesizing positive threshold gate realizations which are hazard free. As previously mentioned, however, positive weights can be changed to negative weights, noninverting gates can be changed to inverting gates and the resulting realizations will also be hazard free. Hence one can obtain any type of desired hazard-free threshold realization. The following synthesis technique will be primarily based on the multigate realization technique of Lewis and Coates.[1] However, at the end of this section, an alternate method is given for obtaining a two-level hazard-free threshold realization.

Since the material will be largely concerned with the n level of the function tree, the following terms are needed. Let $\{q_p\}$ be the subset of $\{0, 1\}^{2n}$ such that q_p is the image point of p for each $p \in \{0, 1\}^n$. Consider a function tree for a Boolean function F. A given position on the n level of the tree corresponds to a specific p of $\{0, 1\}^n$, in that each position corresponds to a unique reduced function $F(p)$, where p is the subcube of $\{0, 1\}^n$ consisting of the single point p. Moreover, any realization of $F(p)$ must have an output function F' such that $F'(q_p)=F(p)$. Thus the position corresponding to p also corresponds to q_p.

Now consider the reconstruction procedure of Lewis and Coates.[1] A separating function f^n is selected which, with appropriate gaps $u^n : l^n$, will realize the n-level reduced function $F'(q_p)=F(p)$. Unless specifically indicated, such n-level realizations $\langle f^n \rangle_{u^n:l^n}$ will not contain negative weights or inverters. Appendix I gives several possible n-level realizations.

Consider some n-level realization $\langle f^n \rangle_{u^n:l^n}$. If G_i denotes an arbitrary gate of $\langle f^n \rangle_{u^n:l^n}$, then let y_i^n denote the Boolean function realized by G_i. Note that y_i^n is a constant function of either 1 or 0. Referring to Appendix I, an example of an n-level realization and the corresponding Boolean functions is

$$\langle f^n \rangle_{u^n:l^n} = \langle 0 + \beta_1 \langle 0 + \beta_2 \langle 0 \rangle_{\infty:0} \rangle_{0:-\infty} \rangle_{\beta_1:-\infty} \qquad \begin{matrix} G_0 & G_1 & G_2 \end{matrix} \tag{5}$$

where $y_2^n=0$, $y_1^n=1$, and $y_0^n=1$.

The following definition will define a set of constants which can be associated with an n-level realization.

Definition 11: Consider an arbitrary n-level realization $\langle f^n \rangle_{u^n:l^n}$. Let G_i and G_j denote arbitrary gates of $\langle f^n \rangle_{u^n:l^n}$ such that the output of G_i is an input to G_j. Define C_j as

$$C_j = \sum_{i=1}^{m} \beta_i y_i^n + k_j$$

where β_i is the weight of input y_i^n to G_j, and k_j the weight of a constant input to G_j.

For instance, consider (5). The set of constants is $C_2=0$, $C_1=0$, $C_0=\beta_1$.

The following definition will define a branch which corresponds to the n level of the tree. The definition is similar to the previous definition of a branch except that it is defined for an n-level realization $\langle f^n \rangle_{u^n:l^n}$.

Definition 12: Let G_i denote an arbitrary gate in $\langle f^n \rangle_{u^n:l^n}$ and B^* an arbitrary *output connected subset* of $\langle f^n \rangle_{u^n:l^n}$. The set B^* is defined as an *n-level* 1(0) *branch* $B_1^n(B_0^n)$ and is said to realize the constant function 1(0)

if, when $\langle f^n\rangle_{u^n;\,l^n}$ is modified such that all the gates not contained in B^* have 0(1) output, then C_i for the modified n-level realization satisfies the condition $C_i \geq u_i^n \cdot (C_i \leq l_i^n) \forall G_i \in B^*$.

As an example of an n-level 0 branch, consider the following n-level realization, which can be obtained from Appendix I.

$$\begin{matrix} & G_0 & G_1 & G_2 \\ \langle f^n\rangle_{u^n;\,l^n} = & \langle 0 + \beta_1\langle 0 + \beta_2\langle 0\rangle_{o:\,-\infty}\rangle_{\infty:\,\beta_2}\rangle_{\infty:\,o}. \end{matrix}$$

Consider the set $\{G_1, G_0\}$ as a possible n-level 0 branch. According to Definition 12, the condition $C_0 \leq l_0^n$ and $C_1 \leq l_1^n$ must exist when the output of G_2 is a 1. For this condition, the modified n-level realization becomes

$$\langle 0 + \beta_1\langle 0 + \beta_2\rangle_{\infty:\,\beta_2}\rangle_{\infty:\,o}.$$

Hence $C_1 = \beta_2$ and $C_0 = 0$. Since $l_1 = \beta_2$ and $l_0 = 0$, it follows that the set $\{G_1, G_0\}$ is an n-level 0 branch.

The following lemma will now give a relationship between n-level branches and branches of a positive realization. The lemma assumes a configuration of gates for the n-level realization such that no void ranges occur during reconstruction (i.e., no additional gates are necessary during reconstruction).

Lemma 3: Consider an arbitrary point $q_p \in \{0, 1\}^{2n}$ such that $F^t(q_p) = 1(0)$. Assume that no void ranges occur during reconstruction. Given that a set of gates B^* is an n-level 1(0) branch $B_1^n(B_0^n)$ which realizes $F^t(q_p)$ on the n level of the tree, then B^* is a 1(0) branch $B_1(B_0)$ which realizes q_p in the final realization.

Proof: We will prove this result by induction. Let G_j be an arbitrary gate of B^* which is on the r logic level of $\langle f^n\rangle_{u^n;\,l^n}$, and hence of $\langle \bar{f}\rangle_{\bar{u};\,\bar{l}}$. Let G_i be one of the m arbitrary gates whose output is an input to G_j (i.e., G_i must be on the $r+1$ or greater logic level). Then the n-level realization for G_j can be expressed as

$$\langle f_j^n\rangle_{\substack{n\ n \\ uj:\,lj}} = \left\langle 0 + \sum_{i=1}^{m'} \beta_i y_i^n + \sum_{i=m'+1}^{m} \beta_i y_i^n \right\rangle_{\substack{n\ n \\ uj:\,lj}}$$

where $1 \leq i \leq m' \Leftrightarrow G_i \in B_1^n$ and $m'+1 \leq i \leq m \Leftrightarrow G_i \notin B_1^n$. Referring to the previous equation, when all $G_i \notin B_1^n$ have 0 output, then

$$C_j = \sum_{i=1}^{m'} \beta_1.$$

Since B^* is an n-level 1 branch, it follows from Definition 12 that

$$C_j = \sum_{i=1}^{m'} \beta_i \geq u_j^n. \tag{6}$$

From properties of reconstruction it is known that if $F^t(q_p) = 1$, then

$$u_j^n + \sum_{k=1}^{n} a_k x_k^*(q_p) \geq \bar{u}_j \tag{7}$$

where $a_k \geq 0$, x_k^* is a literal of x_k, and $\bar{u}_j$ is the upper gap for G_j in the final realization.

Assume that Lemma 3 is true for all G_i, where $1 \leq i \leq m'$. Thus all G_i for $1 \leq i \leq m'$ are elements of B_1 for the point q_p in the final realization. Therefore,

$$f_j(q_p) = \sum_{k=1}^{n} a_k x_k^*(q_p) + \sum_{i=1}^{m'} \beta_i \tag{8}$$

when all $G_i \notin B_1$ have 0 output.

From (6), (7), and (8) it follows that

$$f_j(q_p) \geq \bar{u}_j$$

when all $G_i \notin B_1$ have 0 output. Hence G_j is an element of B_1 for the point q_p in the final realization.

Now consider an arbitrary input gate G_i of B_1^n. For this case, $u_i^n = 0$. It follows from the preceding reasoning that the gate will be an element of B_1 for the point q_p in the final realization. Thus, it follows from induction that B_1 will be a 1 branch for the point q_p in the final realization.

The proof concerning 0 branches is similar and will be omitted.

Definition 13: Let $\langle f\rangle_{u:l}$ denote an arbitrary realization of the Boolean function F. Consider the real numbers u' and l' such that $u > u' > l' > l$. The gaps $u:l'$, $u':l$, or $u':l'$ are defined as *reduced gaps* of $u:l$.

Consider a realization $\langle f\rangle_{u:l}$ of the Boolean function F. Obviously, $u:l$ can be replaced by a reduced gap and the Boolean function F is still realized, provided T is properly selected. For instance, consider the gate G_0 of (5) which has an n-level gap of β_1: $-\infty$. A possible reduced gap is 0: $-\infty$. The n-level realization then becomes

$$\begin{matrix} & G_0 & G_1 & G_2 \\ & \langle 0 + \beta_1\langle 0 + \beta_2\langle 0\rangle_{\infty:\,o}\rangle_{o:\,-\infty}\rangle_{o:\,-\infty}. \end{matrix} \tag{9}$$

This still realizes the Boolean function 1 and $\{G_1, G_0\}$ is still an n-level 1 branch. However, now by Definition 12 the gate G_0 is also an n-level 1 branch, where before it was not. Hence by Lemma 3, if (9) is used to realize a specific $F^t(q_p)$ on the n level of the tree, then the 1 branches $\{G_0\}$ and $\{G_0, G_1\}$ will both realize q_p in the final realization. Thus, reduced n-level gaps provide a means for obtaining a final realization such that the point q_p will be realized by more than one branch.

The value of a point q_p being realized by more than one branch follows from Theorem 3. For example, let K^α and K^β be two 1 subcubes of F which have the set of points $\{p_j\}$ in common, and $\{q_p^j\}$ the corresponding set of image points. Let S^α and S^β be the corresponding image subcubes. Clearly $\{q_p^j\}$ belongs to the set of points common to S^α and S^β. Assume that for some positive realization S^α and S^β are realized by the 1 branches B^α and B^β, respectively, where $B^\alpha \neq B^\beta$. It follows from Theorem 3 that the realization will not contain a logic 1

hazard within K^{α} and K^{β}. However, note that the points of $\{q_p^j\}$ are realized by *both* B^{α} and B^{β}.

Once the set of n-level gaps is chosen, the reconstruction process of Lewis and Coates[1] is a technique for obtaining the coefficients of the independent input variables for each gate of the network. In some realizations the restricted n-level range, caused by using a reduced n-level gap for some arbitrary $F^i(q_p)$, will not have any effect upon the coefficients. When such a case occurs, the reduced n-level gap is referred to as an *unnecessary* reduced n-level gap. However, if the reduced n-level gap does effect the final coefficient, it is referred to as a *necessary* reduced n-level gap. Example 3 illustrates both types of reduced n-level gaps.

The following theorem can now be used for synthesizing a logic hazard-free threshold network directly from the Boolean function F.

Theorem 5: Let $\{P_1^i\}$ ($\{P_0^i\}$) be the set of 1(0) prime implicants of F, $\{\{q_p\}_1^i\}$ ($\{\{q_p\}_0^i\}$) the corresponding collection of sets of image points, and $\{\{F^i(q_p)\}_1^i\}$, ($\{\{F^i(q_p)\}_0^i\}$) the corresponding collection of sets of n-level reduced functions. A final realization $\langle \hat{f} \rangle_{\bar{u}:l}$ of F will not contain any logic 1(0) hazards if the n-level gaps are assigned such that there exists at least one n-level 1(0) branch which realizes all $F^i(q_p) \in \{F^i(q_p)\}_1^i$, ($\{F^i(q_p)\}_0^i$) for all $\{F^i(q_p)\}_1^i \in \{\{F^i(q_p)\}_1^i\}$, ($\{F^i(q_p)\}_0^i \in \{\{F^i(q_p)\}_0^i\}$) and reconstruction is possible without adding any additional gates.

Proof: Let P_1' be an arbitrary 1 prime implicant of F, $\{q_p\}_1'$ the corresponding set of image points, and $\{F^i(q_p)\}_1'$ the corresponding set of n-level reduced functions. Assume that the n-level gaps are assigned such that all $F^i(q_p) \in \{F^i(q_p)\}_1'$ are realized by the same n-level 1 branch, and that reconstruction is possible without void ranges. By Lemma 3, all $q_p \in \{q_p\}_1'$ will be realized by the same 1 branch in the final realization. Hence by Theorem 3, P_1' will not contain any logic 1 hazards.

The proof concerning 0 prime implicants is similar and will be omitted.

Examples 3 and 4 will illustrate the application of Theorem 5. However, several additional facts must be considered first.

In general, a set of n-level gaps, which will yield a hazard-free solution, will not be known. Therefore, suppose that while trying to obtain a hazard-free solution, the first condition of Theorem 5 is satisfied but the second condition is not. Two alternatives exist: either a procedure analogous to Reconstruction III of Lewis and Coates[1] can be used, or an additional threshold gate or gates can be added in such a way as to remedy the situation.

Only the second alternative will be considered here. Assume that an inconsistency occurs for a gate G_j on the k level of the tree. Normally one determines the inconsistency, uses Theorem A to add the necessary gate or gates so that the inconsistency is removed (see Lewis and Coates[1]), and then continues on up the tree. However, if the final realization is to be logic hazard free, Theorem 4 must also be satisfied; hence, the application of Theorem A is restricted in the following manner.

The set of n-level gaps consisting of the n-level gaps for the additional gate(s) plus the changed n-level gaps for the previously chosen gates must satisfy the first condition of Theorem 5. In practice, the most straightforward procedure for satisfying this restriction of Theorem A is the following.

Determine the inconsistency on the k level of the tree and then, instead of adding gates on the k level of the tree, add the gates on the n level of the tree so that the inconsistency is removed from the k level of the tree and the first condition of Theorem 5 is satisfied. The gates that need to be added on the n level of the tree to remove the inconsistency of the k level can easily be determined by tracing the gaps that caused the inconsistency to the bottom of the tree. Example 4 will illustrate this.

Assume that all additional gates are added in this manner. It is proved in Howe and Coates[7] that a final realization can always be obtained with this restriction on Theorem A. Clearly the final realization will be hazard free.

The next fact concerns the application of Theorem 5. To apply Theorem 5 it is necessary to associate each n-level reduced function $F(p) = F^i(q_p)$ with a specific set of prime implicants of F, viz., the set for which the corresponding point p is an element. For example, assume p is an element of the 1 prime implicants P_1^3, P_1^6, and P_1^7. Hence $F^i(q_p)$ must be associated with P_1^3, P_1^6, and P_1^7, in which case, $F^i(q_p) = 1$ can be labeled 1_3, 1_6, and 1_7 on the n level of the tree. A similar statement exists for 0 prime implicants. Now from Theorem 5 the final realization will be hazard free if 1) all n-level points bearing the same label are realized by the same n-level branch and 2) reconstruction is possible without additional gates. Examples 3 and 4 will illustrate this procedure. Also, Chapter 6 of Howe and Coates[7] gives an algorithm for identifying the points at the bottom of the tree.

The last fact to be considered is concerned with incomplete functions. A function is said to be incomplete if for some $p \in \{0, 1\}^n$, $F(p)$ is not specified as either 1 or 0. Such p's are called *don't cares*. This type of Boolean function should also be considered when studying hazards in threshold gate networks. A method has been presented for synthesizing incomplete logical functions by threshold gate networks.[1] The method, in effect, assigns the n-level gaps of the *don't care* points to be $\infty : -\infty$. Therefore, when assigning the n-level gaps in accordance with Theorem 5 (i.e., such that a logic hazard will not occur) the n-level gaps of the *don't care*

points are assigned as $\infty : -\infty$. Unfortunately, however, by this method one has no control of the logic hazards associated with the *don't care* points.

In summary, the following procedure is outlined for obtaining a hazard-free threshold network.

Procedure 3:

1) Using the function tree, decompose F in the usual manner.

2) Using the method previously described, identify the reduced functions $F^t(q_p)$ at the bottom of the tree with their associated prime implicants.

3) Referring to Theorem 5, assign the n-level gaps such that there exists at least one n-level branch which realizes all of the reduced functions $F^t(q_p)$ which bear the same label.

4) Reconstruct in the normal way. If no void common ranges occur, the final realization is hazard free; if a void common range occurs, go to step 5).

5) Using Theorem A, add a sufficient number of gates to eliminate the void common range. However, the set of n-level gaps consisting of the n-level gaps for the additional gates plus the changed n-level gaps for the previously chosen gates must satisfy step 3).

6) Repeat steps 3), 4), and 5) until a final realization is obtained.

The following two examples will illustrate Procedure 3.

Example 3: Consider the Boolean function

$$F = x_1x_2 + \bar{x}_2x_3 + x_3x_4. \tag{9}$$

The Karnaugh map is shown in Fig. 2. The problem is to obtain a hazard-free threshold realization directly from the Boolean function F by application of Theorem 5. The first step is to obtain the prime implicants of F. These are obtained from Fig. 2(a) and are given in Fig. 2(b).

Decompose F by removing x_1, x_2, x_3, and x_4, respectively. The resulting function tree is shown in Fig. 3. The prime implicants that each n-level reduced function corresponds to are identified at the bottom of the tree. For example the reduced function $F^t(x_1=1,\ x_1'=0,\ x_2=1,\ x_2'=0,\ x_3=1,\ x_3'=0,\ x_4=1,\ x_4'=0)$, henceforth to be denoted by $F^t(10,\ 10,\ 10,\ 10)$, corresponds to P_1^1, P_1^3, and P_1^4.

The next step is to assign the n-level gaps according to step 3) of Procedure 3. Referring to Fig. 3, all of the points labeled $1_1(0_1)$ should be realized by the same n-level 1(0) branch, etc. Disregarding the terms in parentheses, one such assignment is given in Fig. 3.

For example, consider the four reduced functions labeled 1_4. The n-level reduced function $F^t(10,\ 10,\ 10,\ 01)$ is realized by the n-level realization $\langle 0+2\langle 0\rangle_{\infty:0}\rangle_{0:-\infty}$, whereas the other three reduced functions labeled 1_4 are realized by the n-level realization $\langle 0+2\langle 0\rangle_{0:-\infty}\rangle_{0:-\infty}$. Hence $F^t(10,\ 10,\ 10,\ 01)$ is realized by the n-level 1 branch consisting of the gate $\{G_0\}$, whereas the other

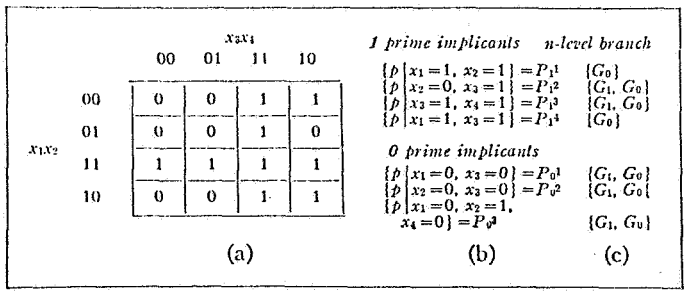


Fig. 2. Karnaugh map, prime implicant list, and n-level branch assignment for Example 3.

three are realized by the n-level 1 branches consisting of both $\{G_0\}$ and $\{G_1, G_0\}$. Thus all four are realized by the n-level branch $\{G_0\}$. Similarly the n-level branch which corresponds to each prime implicant of F is shown in Fig. 2(c).

Fig. 3 shows that reconstruction is possible, and that the final realization is

$$\langle f\rangle_{u:1} = \langle x_3 + x_2 + 2x_1 + 2\langle x_4 + 2x_3 + \bar{x}_2\rangle_{3:2}\rangle_{3:2}.$$

Thus the second condition of Theorem 5 is satisfied. Therefore, the final realization will not contain any logic hazards.

Consider the realization which results when $F^t(10,\ 10,\ 10,\ 10)$, $F^t(10,\ 01,\ 10,\ 10)$, and $F^t(10,\ 01,\ 10,\ 01)$ are assigned normal n-level gaps (see Fig. 3). This assignment and the reconstruction changes caused by this assignment are enclosed in parentheses in Fig. 3, the final realization being

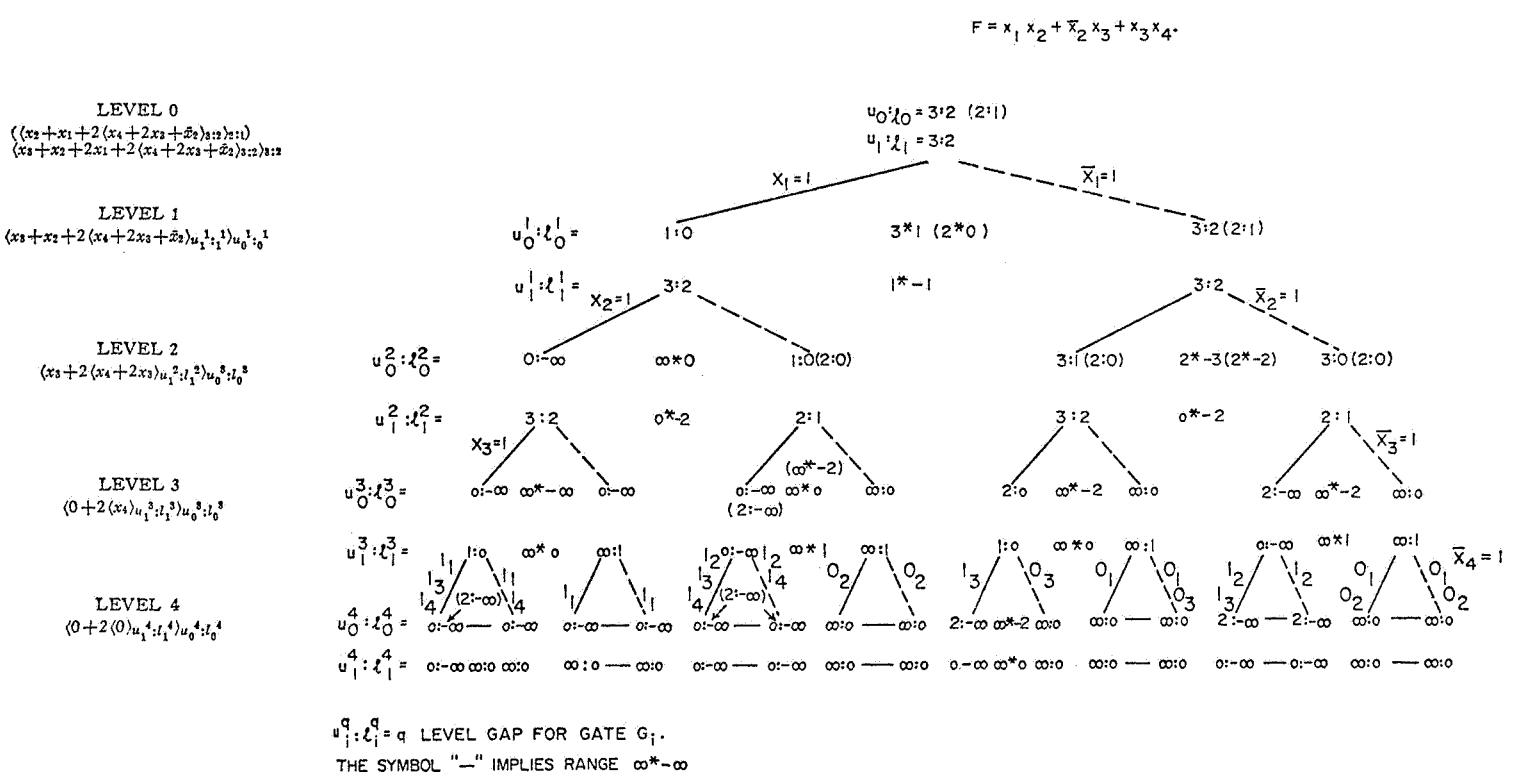
$$\langle f\rangle_{u:1} = \langle x_2 + x_1 + 2\langle x_4 + 2x_3 + \bar{x}_2\rangle_{3:2}\rangle_{2:1}.$$

Note that the 1 prime implicant $\{p|x_1=1,\ x_3=1\}$ is not contained in the latter realization. It will therefore contain a logic 1 hazard. Also note that on comparing the two realizations, the prime implicant can be realized without requiring any additional gates. This is not possible with conventional elements such as AND, OR, NAND, NOR, or relays (i.e., a conventional element can only realize one prime implicant).

By comparing the two previous reconstructions, it can be determined that the reduced n-level gap for $F^t(10,\ 10,\ 10,\ 10)$ is an unnecessary reduced n-level gap, whereas the reduced n-level gaps for $F^t(10,\ 01,\ 10,\ 10)$ and $F^t(10,\ 01,\ 10,\ 01)$ are necessary reduced n-level gaps. By definition, if a gap is an unnecessary reduced n-level gap, the normal gap could be used and the final realization would be the same. However, when the n-level gaps are being assigned it is not known whether or not a reduced n-level gap is an unnecessary reduced n-level gap. Therefore, to obtain a logic hazard-free realization, the best approach is to apply Theorem 5 and assume all reduced n-level gaps are necessary reduced n-level gaps.

Example 4: Consider the Boolean function

$$F = x_1\bar{x}_3 + \bar{x}_1x_3 + \bar{x}_2x_4 + x_2\bar{x}_4.$$



a_i^1 COMMON RANGE	SELECT
(2*0)	$(a_0^1 = 1)$
3*1	$a_0^1 = 2$
1*−1	$a_1^1 = 0$

a_i^2 COMMON RANGE	SELECT
2*0	$a_0^2 = 1$
0*−2	$a_1^2 = -1$

a_i^3 COMMON RANGE	SELECT
(∞*−2)	$(a_0^3 = 0)$
∞*0	$a_0^3 = 1$
∞*1	$a_1^3 = 2$

a_i^4 COMMON RANGE	SELECT
∞*−2	$a_0^4 = 0$
∞*0	$a_1^4 = 1$

Fig. 3. Function tree and realization for Example 3.

The Karnaugh map is shown in Fig. 4. Again the problem is to obtain a hazard-free threshold realization directly from the Boolean function F by application of Theorem 5. The first step is to obtain the prime implicants of F. Referring to Fig. 4(a), it is obvious that a logic 0 hazard can not occur; hence only the 1 prime implicants of F need be considered.

Decompose F by removing x_1, x_2, x_3, and x_4, in that order. The resulting function tree is shown in Fig. 5. The prime implicants that each $F^t(q_p)$ correspond to are identified at the bottom of the tree.

The next step is to obtain n-level realizations and to assign n-level gaps according to step 3) of Procedure 3. Since F is not unate, the initial n-level realization must contain at least two gates. Consider the n-level assignment labeled A in Fig. 5, where the n-level realization is

$$\langle f^n\rangle_{u^n:\,l^n} = \langle 0 + 2\langle 0\rangle_{u_1^n:\,l_1^n}\rangle_{u_0^n:\,l_0^n}.$$

$$G_0 \quad G_1$$

Referring to Definition 12 and assignment A, the n-level reduced functions labeled 1_1 are realized by the n-level 1 branch $\{G_0\}$, whereas the other n-level reduced functions which are 1 are realized by the n-level 1 branch $\{G_1, G_0\}$. Hence a hazard-free threshold realization will be obtained if reconstruction is possible. However, a void common range occurs for G_1 on the third level of reconstruction. Therefore, an additional threshold gate, or gates, must be added to complete reconstruction. However, before considering the addition of a gate or gates to correct the void range for G_1, further reconstruction for G_0 will be considered. Referring to Fig. 5, it is seen that reconstruction of G_0 can be completed without any void common range.

Now consider the inconsistency which caused the void common range for G_1 on the third level of the tree. Referring to Fig. 5, if the gaps identified by Δ are increased by more than 2 the void common range will not occur. Hence the n-level gaps for $F^t(01, 10, 10, 10)$, $F^t(01, 10, 10, 01)$, $F^t(01, 01, 10, 10)$, and $F^t(01, 01, 10, 01)$ must be increased by more than 2. Therefore, change these n-level gaps from $o\colon -\infty$ to $4\colon -\infty$ by adding the gate G_2 as input to G_1, where $\beta_2 = 4$. The n-level assignment for G_1 and G_2 labeled B is shown in Fig. 5. The n-level realization is now

$$\langle f^n\rangle_{u^n:\,l^n} = \langle 0 + 2\langle 0 + 4\langle 0\rangle_{u_2^n:\,l_2^n}\rangle_{u_1^n:\,l_1^n}\rangle_{u_0^n:\,l_0^n}.$$

Referring to assignment B and the previous assignment for G_0, the n-level reduced functions labeled 1_1 and 1_2 are realized by the n-level 1 branch $\{G_0\}$ and $\{G_1, G_0\}$, respectively, whereas the other n-level reduced functions which are 1 are realized by the n-level 1 branch $\{G_2, G_1, G_0\}$. By Theorem 5, if reconstruction is possible, the final realization will not contain any logic hazards.

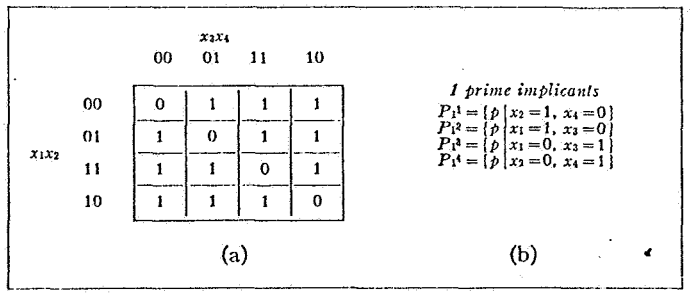


Fig. 4. Karnaugh map for Example 4.

Before continuing, note the following.

1) Since reconstruction was complete for G_0, the image points of P_1^1 will be realized by the 1 branch consisting of G_0 regardless of which gates are added.

2) In this example when the inconsistency is removed from the third level of G_1, it so happens that the first condition of Theorem 5 is also satisfied. This is not true in general, and in such cases some additional n-level gaps for G_1 must be changed.

The process of reconstruction will now be continued. Referring to Fig. 5, reconstruction of G_1 can now be completed without any void common ranges. Next consider the reconstruction for G_2. A void common range occurs for G_2 on the third level of the tree. If the gap, identified by δ, is increased by any positive amount, the inconsistency will not occur. Hence the n-level gap of $F^t(10, 01, 01, 10)$ is increased from $o\colon -\infty$ to $2\colon -\infty$ by adding G_3 as an input to G_2, where $\beta_3 = 2$. The n-level assignment for G_2 and G_3 labeled C is shown in Fig. 5. The corresponding n-level realization is also shown in Fig. 5. Reconstruction is now possible, the final realization being

$$\langle f\rangle_{u:\,l} = \langle \tfrac{1}{2}\bar{x}_1 + x_2 + \tfrac{1}{2}x_3 + \bar{x}_4 + 2\langle 2x_1 + \bar{x}_2 + 2\bar{x}_3 + x_4 + 4\langle 2\bar{x}_1 + \bar{x}_2 + 2x_3 + x_4 + 2\langle \bar{x}_2 + x_4\rangle_{2:1}\rangle_{4:3}\rangle_{4:3}\rangle_{2:1\frac{1}{2}}.$$

Hence from Theorem 5, the realization will not contain any logic hazards.

Theorem 5 gives a means for obtaining a general threshold realization which is hazard free. However, the following special realizations are also of interest.

1) Obviously, if the desired realization can contain logic 0 hazards, but not logic 1 hazards, then only the 1 prime implicants of F must be contained in the realization. Hence in Theorem 5, the n-level reduced functions $F^t(q_p)$ which are 0 can be chosen arbitrarily. A similar statement exists if the realization can contain logic 1 hazards, but not 0 hazards.

2) Eichelberger[2] has proved that a sum-of-product (product-of-sum) realization will not contain any logic hazards if each 1(0) prime implicant of F is realized by a unique AND (OR) gate. Consider the case where the

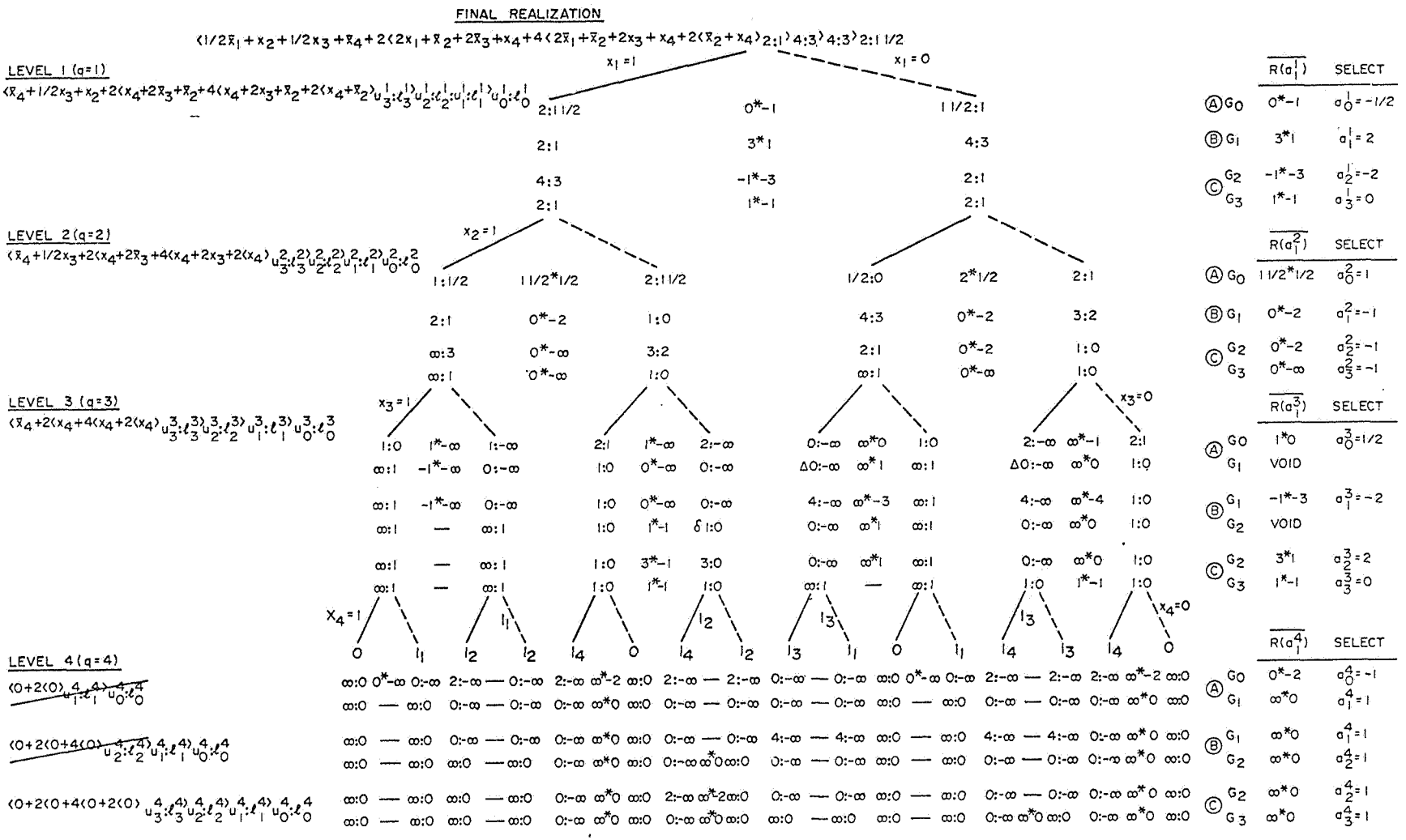


Fig. 5. Function tree and realization for Example 4.

number of 0 prime implicants of F is less than the number of 1 prime implicants of F (i.e., the number of 1 prime implicants of $\overline{F}$ is less than the number of 1 prime implicants of F). From the previous statement, the product-of-sum hazard-free realization requires fewer gates than the sum-of-product hazard-free realization. If the number of 1 prime implicants is less, the sum-of-product realization would require fewer gates.

Theorem 6 will show that a similar situation exists for the following type of two-level threshold realization.

Theorem 6: Let $\{P_1^i\}$ be the set of 1 prime implicants of the Boolean function F, and s_1^i the product Boolean function which realizes the 1 prime implicant P_1^i. Then the Boolean function F can be expressed as

$$F = s_1^1 + \cdots + s_1^i + \cdots + s_1^n \tag{10}$$

where $1 \leq i \leq n$. If (10) can be expressed as a sum of m Boolean threshold functions

$$F_{T_1} + \cdots + F_{T_{m-1}} + F_{T_m} \tag{11}$$

then it can be realized by the two-level positive threshold realization

$$\langle f_\alpha \rangle_{u_\alpha : l_\alpha} = \langle \beta_1 F_{T_1} + \cdots + \beta_{m-1} F_{T_{m-1}} + a_1 x_1^* + \cdots + a_n x_n^* \rangle_{u_\alpha : l_\alpha}$$

where $\beta_j = u_\alpha$ for $1 \leq j \leq m-1$ and $\langle a_1 x_1^* + \cdots + a_n x_n^* \rangle_{u_\alpha : l_\alpha}$ realizes F_{T_m}. Moreover, $\langle f_\alpha \rangle_{u_\alpha : l_\alpha}$ will not contain any logic hazards.

Proof: Since $\beta_j = u_\alpha$, it follows that (11) and hence F can be realized with m gates. Now consider the logic hazards.

All of the reduced functions such that $F^t(q_p) = 0$ are realized by the same 0 branch; namely, the 0 branch which consists of all of the gates contained in $\langle t_\alpha \rangle_{u_\alpha : l_\alpha}$. Thus from Theorem 4, $\langle f_\alpha \rangle_{u_\alpha : l_\alpha}$ will not contain any logic 0 hazards. Obviously, $\langle f_\alpha \rangle_{u_\alpha : l_\alpha}$ will not contain any logic 1 hazards.

The realization $\langle f_\alpha \rangle_{u_\alpha : l_\alpha}$ is equivalent to a sum-of-product realization.

Now consider the complement Boolean function $\overline{F}$ and express it in the form of (10) and (11), respectively. Let m^* denote the number of Boolean threshold functions obtained in the latter expression. From Sheng,[6] m^* will not necessarily equal m; it may be greater or less. From Theorem 6, there exists a realization $\langle f_\alpha^* \rangle_{u_\alpha^* : l_\alpha^*}$ which contains m^* gates, realizes $\overline{F}$, and does not contain any logic hazards. There exists a corresponding complement realization $\langle \bar{f}_\alpha^* \rangle_{\bar{u}_\alpha^* : \bar{l}_\alpha^*}$ which realizes F with m^* gates. Obviously, $\langle \bar{f}_\alpha^* \rangle_{\bar{u}_\alpha^* : \bar{l}_\alpha^*}$ will not contain any logic hazards.

Therefore, if $m^* < m$, the realization $\langle \bar{f}_\alpha^* \rangle_{\bar{u}_\alpha^* : \bar{l}_\alpha^*}$ will require fewer gates than $\langle f_\alpha \rangle_{u_\alpha : l_\alpha}$, whereas if $m^* > m$, the realization $\langle f_\alpha \rangle_{u_\alpha : l_\alpha}$ requires fewer gates.

3) Theorems 5 and 6 give means for synthesizing threshold networks which do not contain any logic hazards. However, from another point of view, they also enable one to design threshold networks which contain certain specified logic hazards and only those specified. The utilization of such networks and their design with conventional-type gates has been considered by Eichelberger.[3]

4) One of the most common applications of hazard-free combinational circuits is in the design of asynchronous systems. Moreover, in most asynchronous systems the assumption is made that only one input variable can change at a time. When such an assumption is made, it has been shown that it is not necessary to realize all of the 1 and 0 prime implicants of F.[4],[5] In such a case, Theorem 5 can be changed accordingly in that the set of necessary 1 and 0 prime implicants will be a subset of $\{P_1^i\}$ and $\{P_0^i\}$, respectively.

Appendix I

This appendix contains the possible n-level separating functions with n-level gaps for two gate and three gate three-level realizations. The parameters β can be selected equal to any number greater than unity. They should be selected equal to 2 or greater if it is desired not to decrease the gap length.[1] For Table III, the n-level realizations are

$\langle 0 + \beta_1 \langle 0 \rangle_{u_1 : l_1} \rangle_{u_0 : l_0}$	2 gate
$\langle 0 + \beta_1 \langle 0 + \beta_2 \langle 0 \rangle_{u_2 : l_2} \rangle_{u_1 : l_1} \rangle_{u_0 : l_0}$	3 gate, 3 level

where β_i, u_i, and l_i correspond to the gate G_i.

TABLE III
Normal Gaps

n-level separating function	n-level branch
$1 = \langle 0 + \beta_1 \langle 0 \rangle_{0 : -\infty} \rangle_{\beta_1 : -\infty}$	$\{G_0, G_1\}$
$1 = \langle 0 + \beta_1 \langle 0 \rangle_{\infty : 0} \rangle_{0 : -\infty}$	$\{G_0\}$
$1 = \langle 0 + \beta_1 \langle 0 + \beta_2 \langle 0 \rangle_{0 : -\infty} \rangle_{\beta_2 : -\infty} \rangle_{\beta_1 : -\infty}$	$\{G_0, G_1, G_2\}$
$1 = \langle 0 + \beta_1 \langle 0 + \beta_2 \langle 0 \rangle_{0 : -\infty} \rangle_{\infty : \beta_2} \rangle_{0 : -\infty}$	$\{G_0\}$
$1 = \langle 0 + \beta_1 \langle 0 + \beta_2 \langle 0 \rangle_{\infty : 0} \rangle_{0 : -\infty} \rangle_{\beta_1 : -\infty}$	$\{G_0, G_1\}$
$1 = \langle 0 + \beta_1 \langle 0 + \beta_2 \langle 0 \rangle_{\infty : 0} \rangle_{\infty : 0} \rangle_{0 : -\infty}$	$\{G_0\}$
$0 = \langle 0 + \beta_1 \langle 0 \rangle_{\infty : 0} \rangle_{\infty : 0}$	$\{G_0, G_1\}$
$0 = \langle 0 + \beta_1 \langle 0 \rangle_{0 : -\infty} \rangle_{\infty : \beta_1}$	$\{G_0\}$
$0 = \langle 0 + \beta_1 \langle 0 + \beta_2 \langle 0 \rangle_{\infty : 0} \rangle_{\infty : 0} \rangle_{\infty : 0}$	$\{G_0, G_1, G_2\}$
$0 = \langle 0 + \beta_1 \langle 0 + \beta_2 \langle 0 \rangle_{\infty : 0} \rangle_{0 : \infty} \rangle_{\infty : \beta_1}$	$\{G_0\}$
$0 = \langle 0 + \beta_1 \langle 0 + \beta_2 \langle 0 \rangle_{0 : -\infty} \rangle_{\infty : \beta_2} \rangle_{\infty : 0}$	$\{G_0, G_1\}$
$0 = \langle 0 + \beta_1 \langle 0 + \beta_2 \langle 0 \rangle_{0 : \infty} \rangle_{\beta_2 : -\infty} \rangle_{\infty : \beta_1}$	$\{G_0\}$

Reduced Gaps

n-level separating function	n-level branches
$1 = \langle 0 + \beta_1 \langle 0 \rangle_{0 : -\infty} \rangle_{0 : -\infty}$	$\{G_0, G_1\}$, $\{G_0\}$
$1 = \langle 0 + \beta_2 \langle 0 + \beta_1 \langle 0 \rangle_{0 : -\infty} \rangle_{0 : -\infty} \rangle_{\beta_1 : -\infty}$	$\{G_0, G_1, G_2\}$, $\{G_0, G_1\}$
$1 = \langle 0 + \beta_2 \langle 0 + \beta_1 \langle 0 \rangle_{0 : -\infty} \rangle_{0 : -\infty} \rangle_{0 : -\infty}$	$\{G_0, G_1, G_2\}$, $\{G_0, G_1\}$, $\{G_0\}$
$1 = \langle 0 + \beta_2 \langle 0 + \beta_1 \langle 0 \rangle_{\infty : 0} \rangle_{0 : -\infty} \rangle_{0 : -\infty}$	$\{G_0, G_1\}$, $\{G_0\}$
$0 = \langle 0 + \beta_1 \langle 0 \rangle_{\infty : 0} \rangle_{\infty : \beta_1}$	$\{G_0, G_1\}$, $\{G_0\}$
$0 = \langle 0 + \beta_2 \langle 0 + \beta_1 \langle 0 \rangle_{\infty : 0} \rangle_{\infty : \beta_2} \rangle_{\infty : 0}$	$\{G_0, G_1, G_2\}$, $\{G_0, G_1\}$
$0 = \langle 0 + \beta_2 \langle 0 + \beta_1 \langle 0 \rangle_{\infty : 0} \rangle_{\infty : \beta_2} \rangle_{\infty : \beta_1}$	$\{G_0, G_1, G_2\}$, $\{G_0, G_1\}$, $\{G_0\}$
$0 = \langle 0 + \beta_2 \langle 0 + \beta_1 \langle 0 \rangle_{0 : -\infty} \rangle_{\infty : \beta_2} \rangle_{\infty : \beta_1}$	$\{G_0, G_1\}$, $\{G_0\}$

REFERENCES

[1] P. M. Lewis, II and C. L. Coates, *Threshold Logic*. New York: Wiley, 1967.

[2] E. B. Eichelberger, "Hazard detection in combinational and sequential switching circuits," *IEEE Internat'l Conf. Rec. on Switching Circuits Theory and Logical Design*, vol. 12, pp. 111–121, 1964.

[3] ——, "The synthesis of combinational circuits containing hazards," Digital Systems Laboratory, Princeton University, Princeton, N. J., Tech. Rept. 17, March 1962.

[4] D. A. Huffman, "The design and use of hazard free switching networks," *J. ACM*, vol. 4, pp. 47–62, January 1957.

[5] E. J. McCluskey, "Transients in combinational logic circuits," in *Redundancy Techniques for Computing Systems*. Washington, D. C.: Spartan Books, 1962, pp. 9–46. Most of this material has been recently published in *Introduction to the Theory of Switching Circuits*. New York: McGraw-Hill, 1965, pp. 284–306.

[6] C. L. Sheng, "Compound synthesis of threshold-logic network for the realization of general Boolean functions," *IEEE Trans. Electronic Computers*, vol. EC-14, pp. 798–814, December 1965.

[7] A. B. Howe and C. L. Coates, "A study of hazards in threshold networks," Laboratories for Electronics and Related Science Research, University of Texas, Austin, Tex., Tech. Rept. 21, August 1966.