Tautology Checking Algorithms for Multiple-Valued Input Binary Functions and Their Application

Tsutomu SASAO

Department of Electronic Engineering
Osaka University, Osaka 565, Japan

Abstract: Multiple-valued input binary function is a mapping $\mathcal{F}: \bigtimes_{i=1}^{n} P \rightarrow B$, where $P_i=\{0,1,\ldots,p_i-1\}$ and $B=\{0,1\}$. In this paper, tautology checking methods for sum-of-products expressions of multiple-valued input binary functions are discussed. Firstly, methods for decomposing a tautology problem into smaller ones are shown. Secondly, a fast hardware tautology checker is proposed. The computation time of the hardware tautology checker is proportional to the number of the terms in the expression. The hardware tautology checker for n-variable p-valued input binary functions requires $4\cdot p^n$ copies of 2-input AND gates. Finally, applications of the tautology checker for generating prime implicants, for generating irredundant sum-of-products expressions, and for detection of the essential prime implicants are shown.

I. INTRODUCTION

In logic design, minimization or simplification of logical expression is important. A minimal sum-of-products expression for a two-valued input logic function corresponds to a minimal two-level AND-OR circuit or a minimal two-level PLA (Programmable Logic Array)[1]. Similarly, a minimal sum-of-products expression for a four-valued input two-valued output function (also called a four-valued input binary function) corresponds to a minimal PLA with two-bit decoders[2],[3],[4].

Suppose that we have to minimize n-variable two-valued logic function.

When $n \leq 10$, we can often obtain a minimum solution by using a Quine-McCluskey method. But this method is impractical for larger problems because it needs all the prime implicants and their covering table. The average number of prime implicants is greater than 2^{n-1} when $n \geq 10$, and some class of functions have prime implicants proportional to $3^n/n$ [4].

When $n \leq 16$, several algorithms which use the truth table of a given function are reported[11]-[13]. They obtain good solutions in a relatively short time. Although these algorithms need not generate all the prime implicants at a time, they need memory space which is proportional to 2^n. So these algorithms are also impractical for larger problems.

When $n \leq 40$, we have very good algorithms such as MINI and ESPRESSO which first obtains a complement of a given function[2],[5]. In these algorithms, the complement is effectively used to make the implicants prime or near prime. These algorithms often simplifies larger practical problems. However, recently, we found a class of functions whose size of the complement increases exponentially with the number of products in an original expression[6]. For this class of functions both MINI and ESPRESSO failed to obtain the complement of the function even if sophisticated algorithms[9],[14] were used.

There are minimization methods without using the list of all the prime implicants, the truth table, nor the complement of the given function[5],[15]-[18]. However, these methods require much computation time if we need a near minimum solution : we have to check whether an array $\mathcal{F}$ covers a cube c or not thousands of times. Most computation time is spend for the checking of this implication relation $c \lessdot \mathcal{F}$. As will be shown in Section 5, $c \lessdot \mathcal{F}$ holds if and only if $\mathcal{F}(|c) \equiv 1$, i.e., the restriction of $\mathcal{F}$ to c is tautology. Therefore, a high speed tautology checker is useful for testing implication relations and thus, useful for logic minimization of many-variable problems. It is known that the implication relation $c \lessdot \mathcal{F}$ can be examined by computing $c \# \mathcal{F}$[16,17] or $c \circledast \mathcal{F}$ [2]. But the tautology checking of $\mathcal{F}(|c)$ is much faster.

In Section 2, the tautology problem of the binary functions is formally defined. The problem is Co-NP complete and there is virtually no hope to find a polynomial time algorithm to decide whether a given sum-of-products expression is tautology or not.

In Section 3, methods for decomposing a tautology problem into smaller ones are shown. Necessary conditions to be tautology are also given, which are useful for quick non-tautology detection.

In Section 4, a fast hardware tautology checker is introduced. The time to decide whether a given sum-of-products expression is tautology or not is proportional to the number of terms in the expression. The tautology checker for an n-variable p-valued input binary function requires $4\cdot p^n$ copies of 2-input AND gates. We can realize an efficient tautology checking system by combining the methods shown in Sections 3 and 4.

In Section 5, application of the tautology checking system for generating prime implicants, for generating irredundant sum-of-products expressions, and for detection of the essential prime implicants are shown. Experiments in [4] shows that more than a half of the prime implicants in minimum sum-of-products expressions are essential for control circuits of microprocessors. Therefore, we often obtain good solutions quickly by first detecting all the essential prime implicants. Theorem 5.3 gives an efficient essential prime implicants detection method without generating all the prime implicants at a time. Our experiments show that this method is much faster than local extraction algorithm[10].

II. BINARY FUNCTIONS AND TAUTOLOGY PROBLEM

Definition 2.1: A mapping $\overset{n}{\underset{i=1}{\times}} P_i \rightarrow B$ is called a multiple-valued input binary function (also called a multi-valued input two-valued output function), where $P_i=\{0,1,\ldots,p_i-1\}$, and $B=\{0,1\}$.

Definition 2.2: Let X_i be a variable on P_i. $X_i^{S_i}$ is a literal of X_i, where $S_i \subseteq P_i$. $X_i^{S_i}$ represents a function such that $X_i^{S_i}=0$ if $X_i \notin S_i$, and $=1$ if $X_i \in S_i$.

Definition 2.3: A product of literals $X_1^{S_1}\cdot X_2^{S_2}\cdot\cdots\cdot X_n^{S_n}$ is called a product. A sum of products is called a sum-of-products expression.

Theorem 2.1: An arbitrary binary function can be represented by a sum-of-products expression:

$$\mathcal{F}(X_1,X_2,\ldots,X_n)=\bigvee_{(S_1,S_2,\ldots,S_n)} X_1^{S_1}\cdot X_2^{S_2}\cdots X_n^{S_n},$$

where $S_i \subseteq P_i$.

From here, F (an upper case letter) represents a multiple-valued input binary function and $\mathcal{F}$ (a script letter) represents its expression, and so on.

Example 2.1: A binary function $F:\{0,1\}\times\{0,1,2\}\rightarrow\{0,1\}$ shown in Table 2.1 can be represented by the sum-of-products expression:

$$\mathcal{F}=X_1^0\cdot X_2^{\{0,2\}} \vee X_1^1\cdot X_2^1 \vee X_2^{\{0,1\}}.$$

For simplicity, $X_1^{\{0\}}$ is represented by X_1^0 and so on.

Table 2.1 Truth table

X_1	X_2	F
0	0	1
0	1	1
0	2	1
1	0	1
1	1	1
1	2	0

Definition 2.4: (Positional cube notation). A product $X_1^{S_1}\cdot X_2^{S_2}\cdot\cdots\cdot X_n^{S_n}$ can be represented as follows:

$$\begin{array}{ccccc} X_1 & & X_2 & & X_n \\ c_0^1 c_1^1 \cdots c_{p_1-1}^1 & - & c_0^2 c_1^2 \cdots c_{p_2-1}^2 & - \cdots - & c_0^n c_1^n \cdots c_{p_n-1}^n \end{array}$$

where $c_i^j=0$ if $i \notin S_j$ and $=1$ if $i \in S_j$. The above notation is called a positional cube notation[7]. A sum-of-products expression represented by a set of positional cubes is called an array. Arrays are denoted by the same symbols as the corresponding expressions.

Example 2.2: An array for the expression in Example 2.1 is

$$\begin{array}{cc} & \begin{array}{ccc} X_1 & & X_2 \end{array} \\ \mathcal{F}= & \begin{bmatrix} 10 & - & 101 \\ 01 & - & 010 \\ 11 & - & 110 \end{bmatrix} \end{array}$$

Definition 2.5: Let $\mathcal{F}$ be a sum-of-products expression. If $\mathcal{F}\equiv 1$, i.e., $\mathcal{F}$ is equal to 1 for all the input combinations, then $\mathcal{F}$ is said to be tautology. The problem to decide a given sum-of-products expression is tautology or not is said to be a tautology problem.

Example 2.3: A function in Example 2.1 is not tautology, which is obvious from Table 2.1.

Example 2.4: Consider a function $F:\{0,1\}\times\{0,1,2\}\times\{0,1,2,3\}\rightarrow B$, and its expression:

$$\mathcal{F}=X_1^0\cdot X_2^2 \vee X_2^{\{0,2\}}X_3^{\{1,3\}} \vee X_2^1 \vee X_2^{\{0,2\}}\cdot X_3^{\{0,2\}} \vee X_1^1\cdot X_3^{\{1,2\}}.$$

By making a truth table for $\mathcal{F}$, it is easy to verify that $\mathcal{F}$ is tautology.

(End of Example).

When the number of the input variable is large, it is quite time consuming to check the tautology by making a large truth table. It is interesting to find an efficient tautology checking algorithm. However, the next theorem shows that the problem is quite hard.

Theorem 2.2: The tautology problem for binary function is Co-NP complete.

(Proof) Similar to the two-valued logic function [8]. (Q.E.D)

The above theorem implies that there is virtually no hope to find a polynomial time algorithm for the tautology problem.

III. DECOMPOSITION OF A TAUTOLOGY PROBLEM

Definition 3.1: Let c be a cube. The cube restriction of $\mathcal{F}$ to c is obtained as follows and denoted by $\mathcal{F}(|c)$.

1) Make a logical product of $\mathcal{F}$ and c.
2) Delete null products.
3) Change 0's of $\mathcal{F}$ into 1's in the columns where c is 0.

Example 3.1: Consider an array $\mathcal{F}$ and a a cube c, where

$$c = (01-111-1100)$$

$$\mathcal{F} = \begin{bmatrix} 10-111-0011 \\ 11-101-0101 \\ 11-010-1111 \\ 01-111-0110 \end{bmatrix}$$

We can make $\mathcal{F}(|c)$ as follows:

1). By making a logical product of $\mathcal{F}$ and c, we have

$$c \cdot \mathcal{F} = \begin{bmatrix} 00-111-0000 \\ 01-101-0100 \\ 01-010-1100 \\ 01-111-0110 \end{bmatrix} \quad *$$

2). By deleting the null cube(denoted by *), we have

$$c \cdot \mathcal{F} = \begin{array}{c} \;\;*\qquad\quad ** \\ \begin{bmatrix} 01-101-0100 \\ 01-010-1100 \\ 01-111-0100 \end{bmatrix} \end{array}$$

3). By changing 0's in $\mathcal{F}$ into 1's in the columns where c is 0 (denoted by *), we have,

$$\mathcal{F}(|c) = \begin{bmatrix} 11-101-0111 \\ 11-010-1111 \\ 11-111-0111 \end{bmatrix}$$

(End of example).

Lemma 3.1: $c \cdot \mathcal{F} = c \cdot \mathcal{F}(|c)$.

(Proof) Clear from Definition 3.1.(Q.E.D)

Lemma 3.2: Let $\mathcal{F}$ be an array, and c_i $(i=1,2,\ldots,m)$ be cubes, where

$$\bigvee_{i=1}^{m} c_i \equiv 1 \text{ and } c_i \cdot c_j = 0 \ (i \neq j).$$

Then, $\mathcal{F} = \bigvee_{i=1}^{m} c_i \cdot \mathcal{F}(|c)$.

(Proof)

$\mathcal{F} = (\bigvee_{i=1}^{m} c_i) \cdot \mathcal{F} = \bigvee_{i=1}^{m} (c_i \cdot \mathcal{F})$. By Lemma 3.1,

we have $\mathcal{F} = \bigvee_{i=1}^{m} c_i \cdot \mathcal{F}(|c_i)$. (Q.E.D.)

Lemma 3.3: Let $\mathcal{F}$ be an array, and c_i $(i=1,2,\ldots,m)$ be cubes, where

$$\bigvee_{i=1}^{m} c_i \equiv 1 \text{ and } c_i \cdot c_j = 0 (i \neq j).$$

Then, $\mathcal{F} \equiv 1 \leftrightarrow \mathcal{F}(|c_i) \equiv 1 (i=1,2,\ldots,m)$.

(Proof) Obvious from Definition 2.5 and Lemma 3.1. (Q.E.D.)

Lemma 3.4: In the columns of X_k in $\mathcal{F}$, suppose that the i-th column covers j-th column(i.e., the i-th column has 1's in all the rows in which the j-th column has 1's). Let $\mathcal{G}$ be an array after deleting the i-th column from $\mathcal{F}$, we have $\mathcal{F} \equiv 1 \leftrightarrow \mathcal{G} \equiv 1$.

(Proof) Let $c_r = X_k^r$ $(r=0,1,\ldots,p_k-1)$. It is clear that $\bigvee_{r=0}^{p_k-1} c_r \equiv 1$ and $c_r \cdot c_s = 0 (r \neq s)$.

By Lemma 3.3, $\mathcal{F} \equiv 1 \leftrightarrow \mathcal{F}_r \equiv 1 (r=0,1,\ldots,p_k-1)$, where $\mathcal{F}_r = \mathcal{F}(|X_k^r)$. Because i-th column covers j-th column, $\mathcal{F}_j < \mathcal{F}_i$, which implies that if $\mathcal{F}_j \equiv 1$ then $\mathcal{F}_i \equiv 1$. Therefore, we need not check the tautology of $\mathcal{F}_i$ if $\mathcal{F}_j$ is tautology.

Next, consider array $\mathcal{G}$. Let $c_r = X_k^r$ $(r=0,1,\ldots,p_k-1.\ r \neq j)$.

In this case note that $\bigvee_{r=0\ (r \neq j)}^{p_k-1} c_r \equiv 1$ and $c_r \cdot c_s = 0$ $(r \neq s)$. By Lemma 3.3, $\mathcal{G} \equiv 1 \leftrightarrow \mathcal{G}_r \equiv 1$ $(r=0,1,\ldots,p_k-1, r \neq j)$, where $\mathcal{G}_r = \mathcal{G}(|X_k^r)$. Obviously, $\mathcal{G}_r = \mathcal{F}_r$. Hence $\mathcal{F} \equiv 1 \leftrightarrow \mathcal{G} \equiv 1$. (Q.E.D.)

Lemma 3.5: If $\mathcal{F}$ has a column with all 0's, then $\mathcal{F}$ is non-tautology.

(Proof) Suppose that the j-th column of X_i has all 0's. $\mathcal{F}$ is 0 for the inputs such that $X_i = j$. In other words, $\mathcal{F}$ is non-tautology. (Q.E.D.)

Theorem 3.1: Let $\mathcal{G}$ be an array obtained from an array $\mathcal{F}$ by applying the following operations repeatedly. Then $\mathcal{F} \equiv 1 \leftrightarrow \mathcal{G} \equiv 1$.

1) Delete a cube which has a variable with all 0's.
2) Delete a column with all 1's.

(Proof)

1) A cube which has a variable with all 0's is a null cube. So the deletion of the cube will not change the function represented by the array.
2) Suppose that the i-th column of X_k are are all 1's.
   a) When $p_k \geq 2$: The i-th column covers other columns in X_k. By Lemma 3.4, the i-th column can be deleted.
   b) When $p_k = 1$: In this case, $\mathcal{F}$ does not depend on X_k and we can delete X_k .

Example 3.2: Consider an array $\mathscr{F}$:

$$\mathscr{F}=\begin{array}{c} *\qquad\quad *\qquad\quad *\qquad\quad \\ \begin{bmatrix} 1011\text{—}1010\text{—}1101 \\ 1000\text{—}1110\text{—}1011 \\ 1111\text{—}1000\text{—}1010 \\ 1001\text{—}1110\text{—}1011 \\ 1111\text{—}1011\text{—}1101 \\ 1001\text{—}1000\text{—}1101 \\ 1111\text{—}1010\text{—}1101 \end{bmatrix}\end{array}$$

By deleting columns with all 1's (denoted by *'s), we have

$$\mathscr{G}_1=\begin{bmatrix} 011\text{—}010\text{—}101 \\ \underline{000}\text{—}110\text{—}011 \\ 111\text{—}\underline{000}\text{—}010 \\ 001\text{—}110\text{—}011 \\ 111\text{—}011\text{—}101 \\ 001\text{—}\underline{000}\text{—}101 \\ 111\text{—}010\text{—}101 \end{bmatrix}$$

By deleting cubes with all-0 variables (denoted by under lines), we have

$$\mathscr{G}_2=\begin{array}{c} \qquad *\qquad\quad *\qquad\quad * \\ \begin{bmatrix} 011\text{—}010\text{—}101 \\ 001\text{—}110\text{—}011 \\ 111\text{—}011\text{—}101 \\ 111\text{—}010\text{—}101 \end{bmatrix}\end{array}$$

By deleting columns with all 1's (denoted by *'s), we have

$$\mathscr{G}_3=\begin{bmatrix} 01\text{—}\underline{00}\text{—}10 \\ \underline{00}\text{—}10\text{—}01 \\ 11\text{—}01\text{—}10 \\ 11\text{—}\underline{00}\text{—}10 \end{bmatrix}$$

By deleting cubes with all-0 variables (denoted by under lines), we have

$\mathscr{G}_4=\{11\text{—}01\text{—}10\}$

By lemma 3.5, $\mathscr{G}_4$ is non-tautology. Hence, $\mathscr{F}$ is non-tautology. (End of example)

Theorem 3.2:(Split-by-variable decomposition)

$$\mathscr{F}\equiv 1 \;\leftrightarrow\; \mathscr{F}_i\equiv 1 \quad (i=0,1,2,\ldots,p_k-1),$$

where $\mathscr{F}_i=\mathscr{F}(|X_k^i)$.

(Proof) Let $c_i=X_k^i$ $(i=0,1,2,\ldots,p_k-1)$.

By Lemma 3.3, we have the theorem.(Q.E.D.)

Example 3.3: Consider an array $\mathscr{F}$:

$$\mathscr{F}=\begin{bmatrix} 10\text{—}101\text{—}1011 \\ 10\text{—}010\text{—}1101 \\ 10\text{—}110\text{—}0100 \\ 01\text{—}010\text{—}1011 \\ 01\text{—}101\text{—}1101 \\ 01\text{—}011\text{—}1011 \end{bmatrix}$$

In Theorem 3.2, let k=1, then

$$\mathscr{F}_0=\begin{bmatrix} 11\text{—}101\text{—}1011 \\ 11\text{—}010\text{—}1101 \\ 11\text{—}110\text{—}0100 \end{bmatrix} \quad \text{and}$$

$$\mathscr{F}_1=\begin{bmatrix} 11\text{—}010\text{—}1011 \\ 11\text{—}101\text{—}1101 \\ 11\text{—}011\text{—}1011 \end{bmatrix}$$

Therefore,

$\mathscr{F}\equiv 1 \leftrightarrow (\mathscr{F}_0\equiv 1$ and $\mathscr{F}_1\equiv 1$).

(End of Example).

In Example 3.3, every cube of $\mathscr{F}$ has singleton 1 in X_1. It is easy to see that when every cube has singleton 1 in a same variable, Theorem 3.2 is especially effective. *However, when many elements are* 1's in all the variables, Theorem 3.2 is not so effective. This theorem has been discussed for the two-valued logic functions[9].

Theorem 3.3:(Split-by-term decomposition).

Suppose that the given expression can be written as $\mathscr{F}=c\vee\mathscr{G}$, where $c=X_1^{S_1}\cdot X_2^{S_2}\cdots X_m^{S_m}$.

Then, $\mathscr{F}\equiv 1\leftrightarrow\mathscr{G}(|c_i)\equiv 1 (i=1,2,..,m)$, where

$$c_1=X_1^{\bar{S}_1},\; c_2=X_1^{S_1}\cdot X_2^{\bar{S}_2},\; c_3=X_1^{S_1}\cdot X_2^{S_2}\cdot X_3^{\bar{S}_3},\ldots,$$

$$c_m=X_1^{S_1}\cdot X_2^{S_2}\cdot\ldots\cdot X_m^{\bar{S}_m}.$$

(Proof) Let $c=c_{m+1}$. Then $\bigvee_{i=1}^{m+1} c_i\equiv 1$ and $c_i\cdot c_j=0$ $(i\neq j)$. By Lemma 3.3,

$\mathscr{F}\equiv 1 \;\leftrightarrow\; \mathscr{F}(|c_i)\equiv 1(i=1,\ldots,m+1)$. Because $\mathscr{F}=c_{m+1}\vee\mathscr{G}$, and $c_{m+1}\cdot c_i=0$ $(i=1,2,\ldots,m)$, we have $\mathscr{F}(|c_i)=\mathscr{G}(|c_i)$ $(i=1,2,\ldots,m)$.

Note that $\mathscr{F}(|c_{m+1})\equiv 1$. Hence, we have

$\mathscr{F}\equiv 1 \;\leftrightarrow\; \mathscr{G}(|c_i)\equiv 1(i=1,2,\ldots,m)$.(Q.E.D.)

Example 3.4: Consider an array $\mathscr{F}$:

$$\mathscr{F}=\begin{bmatrix} 01\text{—}001\text{—}1111\text{—}1111 \\ 01\text{—}101\text{—}1010\text{—}1011 \\ 11\text{—}011\text{—}0111\text{—}1101 \\ 11\text{—}110\text{—}1011\text{—}0111 \\ 11\text{—}001\text{—}1011\text{—}1111 \\ 10\text{—}001\text{—}0101\text{—}1101 \end{bmatrix}$$

$\mathscr{F}$ is written as $F=c\vee\mathscr{G}$, where $c=X_1^1\cdot X_2^2$, and

$$\mathscr{G}=\begin{bmatrix} 01\text{—}101\text{—}1010\text{—}1011 \\ 11\text{—}011\text{—}0111\text{—}1101 \\ 11\text{—}110\text{—}1011\text{—}0111 \\ 11\text{—}001\text{—}1011\text{—}1111 \\ 10\text{—}001\text{—}0101\text{—}1101 \end{bmatrix}$$

By Theorem 3.3,

$\mathscr{F}\equiv 1 \;\leftrightarrow\; (\mathscr{G}(|c_1)\equiv 1$ and $\mathscr{G}(|c_2)\equiv 1)$,

where $c_1=X_1^0$, $c_2=X_1^1\cdot X_2^{\{0,1\}}$ and

$$\mathscr{G}(|c_1)=\begin{bmatrix} 11\text{—}011\text{—}0111\text{—}1101 \\ 11\text{—}110\text{—}1011\text{—}0111 \\ 11\text{—}001\text{—}1011\text{—}1111 \\ 11\text{—}001\text{—}0101\text{—}1101 \end{bmatrix} \quad \text{and}$$

$$\mathscr{G}(|c_2)=\begin{bmatrix} 11\text{—}101\text{—}1010\text{—}1011 \\ 11\text{—}011\text{—}0111\text{—}1101 \\ 11\text{—}111\text{—}1011\text{—}0111 \end{bmatrix}$$

(End of Example)

As easily seen from Example 3.4, Theorem 3.3 is effective when the given expression has a cube with a small number of literals.

Definition 3.2: The volume of a cube

$c=X_1^{S_1}\cdot X_2^{S_2}\cdot\ldots\cdot X_n^{S_n}$ *is defined as*

$vol(c) = \prod_{i=1}^{n} |S_i|$,where $|S_i|$ denotes the number of elements in S_i. In other words, vol(c) is the number of minterms contained in c.

<u>Theorem 3.4:</u> Let $\mathcal{F} = \bigvee_{i=1}^{m} c_i$.

If $F \equiv 1$ then $\sum_{i=1}^{m} vol(c_i) \geq \prod_{i=1}^{n} p_i$.

The above theorem says that, if the sum of the volume of each cube is smaller than the number of minterms in the universal cube, then $\mathcal{F}$ is non-tautology. This theorem is useful for quick non-tautology detection.

<u>Example 3.5:</u> Consider an array $\mathcal{F}$:

$$\mathcal{F} = \begin{array}{ll} \begin{bmatrix} 10-101-101 \\ 01-011-101 \\ 10-010-111 \\ 11-111-010 \end{bmatrix} & \begin{array}{l} c_1 \\ c_2 \\ c_3 \\ c_4 \end{array} \end{array}$$

It is easy to see that $vol(c_1)=4$, $vol(c_2)=4$, $vol(c_3)=3$, and $vol(c_4)=6$.

We have $\sum_{i=1}^{n} vol(c_i)=17$. On the other hand $\prod_{i=1}^{n} p_i = 2 \times 3 \times 3 = 18$. By Theorem 3.4, $\mathcal{F}$ is non-tautology. (End of example)

IV. HARDWARE TAUTOLOGY CHECKER

As shown in the previous section, the tautology problem can be solved by an iterative applications of Theorems 3.1, 3.2, and 3.3. Another method to solve the tautology problem is to make a truth table of the array. The tautology checking by using a truth table is, in general, time consuming because we have to check $\prod_{i=1}^{n} p_i$ combinations. However, when the problem is small(say $n \leq 7$ and $p_i=2$ $(i=1,2,\ldots,n)$), tautology checking by the truth table is faster than by decomposition[9]. Therefore, the tautology problem can be solved efficiently first by decomposing the problem into small ones, and then by making the truth tables of the small ones.

In this section, we will show a fast tautology checking method by using a special logic circuit. The computation time for this tautology checker is proportional to the number of the terms in the expression.

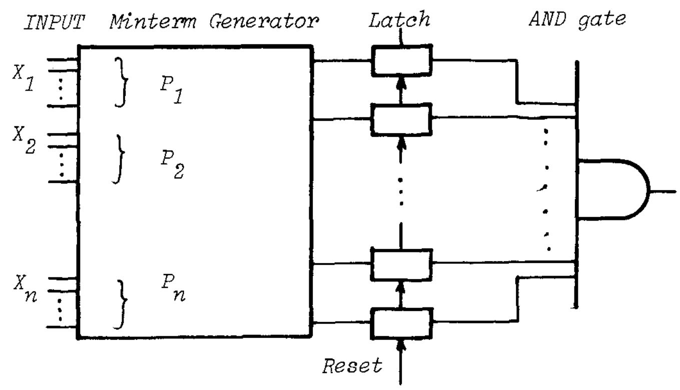
4.1 Tautology Checking Circuit

Schematic diagram of the hardware tautology checker is shown in Fig.4.1.

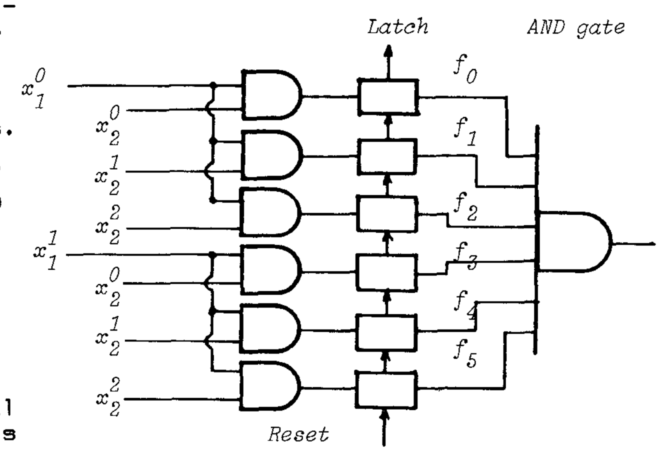
<u>Minterm Generator</u> has $H = \sum_{i=1}^{n} p_i$ inputs and $W = \prod_{i=1}^{n} p_i$ outputs. Each output corresponds to a minterm. When a cube c is applied to the input, all the outputs which corresponds to the minterms of c become one.

<u>Latch Part</u> consists of W latches. Every latch is reset to zero at the initial state. When a cube is applied to the minterm generator, all the latches which correspond to the minterms of c will be set to one. We have to apply all the cubes sequentially.

<u>AND Gate Part</u> has W inputs and one output. The output becomes one if all the W inputs are one, which shows that the given array is tautology.



<u>Fig. 4.1 Tautology Checking Circuit</u>



<u>Fig. 4.2 Tautology Checking Circuit for Exampple 4.1</u>

Example 4.1: Consider a binary function: $F: \{0,1\} \times \{0,1,2\} \rightarrow \{0,1\}$, and its array

$$\mathcal{F} = \begin{array}{cc|ccc|c} \multicolumn{2}{c|}{X_1} & \multicolumn{3}{c|}{X_2} & \\ x_1^0 & x_1^1 & x_2^0 & x_2^1 & x_2^2 & \\ \hline 1 & 0 & 1 & 0 & 1 & c_1 \\ 1 & 0 & 1 & 1 & 0 & c_2 \\ 0 & 1 & 1 & 1 & 0 & c_3 \\ 1 & 1 & 0 & 1 & 1 & c_4 \end{array}$$

The tautology checking circuit for this function is shown in Fig.4.2.
At the initial state, all the latches are reset to zero.
When cube c_1 is applied, f_0 and f_2 are set to one.
When cube c_2 is applied, f_0 and f_1 are set to one.
When cube c_3 is applied, f_3 and f_4 are
When cube c_4 is applied, f_1, f_2, f_4, and f_5 are set to one.
In the following table, X marks show the latches which are set by the application of each cube.

x_1	x_2		c_1	c_2	c_3	c_4
0	0	f_0	X	X		
0	1	f_1		X		X
0	2	f_2	X			X
1	0	f_3			X	
1	1	f_4			X	X
1	2	f_5				X

When all the cubes are applied to Fig.4.2, the output of the AND gate will be one, which shows the given array $\mathcal{F}$ is tautology. (End of example).

4.2 Estimation of the Gate Counts

In this part, we assume that $p=p_i$ $(i=1, 2,\ldots,n)$ and $n=2^m$, where m is an integer.
Minterm Generator: Let g(n) be the number of the 2-input AND gates to realize an n-variable minterm generator. A 2-variable minterm generator is realized by using p^2 copies of 2-input AND gate. So we have $g(2)=p^2$. An n-variable minterm generator is realized by two (n/2)-variable mintem generators and p^n copies of 2-input AND gates. From these, we have,

$$g(n)=2\cdot g(n/2)+p^n=p^n+2\cdot p^{(n/2)}+4\cdot p^{(n/4)}+\ldots\ldots +(n/2)\cdot p^2$$

Latch Part: Each latch is realized by a pair of 2-input AND gates. So the number of AND gates is $2\cdot p^n$.

AND Gate Part: p^n input AND gate is realized by (p^n-1) copies of 2-input AND gates.
Hence, the total number of 2-input AND gates is
$4\cdot p^n+2\cdot p^{(n/2)}+4\cdot p^{(n/4)}+\ldots\ldots+(n/2)\cdot p^2-1$.
Table 4.1 shows the number of 2-input AND gates necessary to realize a tautology checker when p=2.

Table 4.1 Number of 2-input AND gates to realize a Tautology checker

Number of Inputs n	6	8	10	12	14
Minterm Generator	88	304	1120	4272	16712
Latch part	128	512	2048	8192	32768
ANDgate part	63	255	1023	4095	16383
Total	279	1071	4191	16559	65863

4.3 Estimation of the Computation Time

Suppose that the bit pattern of each cube is sent to the output port of the computer by an output instruction, and that tautology is checked by the logic circuit shown in Fig.4.1. The circuit can be realized in at most $[n\cdot\log_2 p + 2 + \log_2 n]$ levels. When n=10 and p=2, it is 16 levels, and the delay time of the circuit is in the order of nano-seconds if we use TTL technology. On the other hand, the time t_1 to send a bit pattern of a cube to the output port of a computer is in the order of micro-seconds. So, the total computation time depends only on the output instruction time, and is $m\cdot t_1$, where m is the number of the cubes and t_1 is the time to send the bit pattern of a cube to the output port.

V. APPLICATION FOR LOGIC MINIMIZATION

Minimization methods in this section may be slower than MINI or ESPRESSO if we use only a software for tautology checking. But if we make a system which first decomposes a large tautology problem into many small problems, and then use a high-speed hardware tautology checker to solve the small problems, we can make a fast minimization system for many input problems.

Definition 5.1: A product $P=X_1^{S_1}\cdot X_2^{S_2}\cdot\ldots\cdot X_n^{S_n}$ is called an implicant of F if F is equal to one whenever P is equal to one, and denoted by $P \leq F$. P is called a prime implicant of F if $P \leq F$ and S_i $(i=1,2,\ldots,n)$ is maximal. When $P \leq F$, F is said to cover P.

From here, we will consider a problem to represent a given function f by using a minimum (or minimal) number of prime implicants.

Lemma 5.1: Let c be a cube and $\mathcal{F}$ be an an array. Then $c \leqslant \mathcal{F} \leftrightarrow \mathcal{F}(|c) \equiv 1$.

(Proof) By Lemma 3.1, $c \cdot \mathcal{F} = c \cdot \mathcal{F}(|c)$. Note that $c \leqslant \mathcal{F} \;\; \leftrightarrow \;\; c = c \cdot \mathcal{F}$. If $c \leqslant \mathcal{F}$, then $c = c \cdot \mathcal{F} = c \cdot \mathcal{F}(|c)$. Therefore $\mathcal{F}(|c) \equiv 1$. If $\mathcal{F}(|c) \equiv 1$, then $c = c \cdot \mathcal{F}$. Therefore $c \leqslant \mathcal{F}$. (Q.E.D.)

Example 5.1: Let c and $\mathcal{F}$ be a cube and an array as follows:

$$c = (11\text{—}111\text{—}0110)$$

$$\mathcal{F} = \begin{bmatrix} 01\text{—}011\text{—}0110 \\ 01\text{—}101\text{—}1110 \\ 10\text{—}101\text{—}0110 \\ 10\text{—}011\text{—}1110 \end{bmatrix}$$

The cube restriction of $\mathcal{F}$ is given by

$$\mathcal{F}(|c) = \begin{bmatrix} 01\text{—}011\text{—}1111 \\ 01\text{—}101\text{—}1111 \\ 10\text{—}101\text{—}1111 \\ 10\text{—}011\text{—}1111 \end{bmatrix}$$

By Theorem 3.1, $\mathcal{F}(|c) \equiv 1 \;\; \leftrightarrow \;\; \mathcal{G} \equiv 1$, where

$$\mathcal{G} = \begin{bmatrix} 01\text{—}01 \\ 01\text{—}10 \\ 10\text{—}10 \\ 10\text{—}01 \end{bmatrix}$$

By making the truth table for $\mathcal{G}$, we can see that $\mathcal{G}$ is tautology. Therefore, $\mathcal{F}(|c) \equiv 1$. By Lemma 5.1, we have $c \leqslant \mathcal{F}$. (End of Example).

Lemma 5.1 shows that whether an array $\mathcal{F}$ covers a cube c or not is examined by the tautology checking.

Theorems 5.1 through 5.3 shows methods for generating prime implicants, for deriving irredundant sum-of-products expressions, and for detecting the essential prime implicants. In these methods, we have to examine whether $c \leqslant \mathcal{F}$ or not thousands of times, which can be done efficiently by the tautology checking of $\mathcal{F}(|c)$.

Theorem 5.1: Let $\mathcal{F} = c \vee \mathcal{G}$, where

$c = X_1^{S_1} \cdot X_2^{S_2} \cdot \dots \cdot X_k^{S_k} \cdot \dots \cdot X_n^{S_n}$. Let

$b = X_1^{S_1} \cdot X_2^{S_2} \cdot \dots \cdot X_k^{a_k} \cdot \dots \cdot X_n^{S_n}$, and $a_k \notin S_k$.

1) If $b \leqslant \mathcal{G}$, then $\mathcal{F} = c' \vee \mathcal{G}$, where

$c_1' = X_1^{S_1} \cdot X_2^{S_2} \cdots X_k^{(S_k \cup a_k)} \cdots X_n^{S_n}$.

2) If $b \not\leqslant \mathcal{G}$ for all $a_k \notin S_k$, and for all k (k=1,2,...,n), then c is a prime implicant of $\mathcal{F}$.

Example 5.2: Let us obtain a prime implicant cover of the following array:

$$\mathcal{F} = \begin{bmatrix} 01\text{—}010\text{—}1010 \\ 01\text{—}100\text{—}1110 \\ 10\text{—}010\text{—}1110 \\ 10\text{—}100\text{—}0110 \end{bmatrix} \begin{matrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{matrix}$$

$\mathcal{F}$ can be written as $\mathcal{F} = c_1 \vee \mathcal{G}_1$, where

$c_1 = (01\text{—}010\text{—}1010)$, and

$$\mathcal{G}_1 = \begin{bmatrix} 01\text{—}100\text{—}1110 \\ 10\text{—}010\text{—}1110 \\ 10\text{—}100\text{—}0110 \end{bmatrix}$$

Consider a cube

$b_1 = (10\text{—}010\text{—}1010)$

Because $b_1 \leqslant \mathcal{G}_1$, c_1 is expandable to the direction X_1^0.

So $\mathcal{F}$ can be written as $\mathcal{F} = c_1' \vee \mathcal{G}_1$, where

$c_1' = c_1 \vee b_1 = (11\text{—}010\text{—}1010)$

c_1' is a prime implicant because it is unexpandable to any other direction. Similarly, both c_2 and c_3 are prime implicants.

Now, $\mathcal{F}$ can be written as $\mathcal{F} = c_4 \vee \mathcal{G}_2$, where

$c_4 = (10\text{—}100\text{—}0110)$, and

$$\mathcal{G}_2 = \begin{bmatrix} 11\text{—}010\text{—}1010 \\ 01\text{—}100\text{—}1110 \\ 10\text{—}010\text{—}1110 \end{bmatrix}$$

Consider a cube

$b_4 = (10\text{—}010\text{—}0110)$

Because $b_4 \leqslant \mathcal{G}_2$, b_4 is expandable to the direction X_2^1.

So $\mathcal{F}$ can be written as $\mathcal{F} = c_4' \vee \mathcal{G}_2$, where

$c_4' = c_4 \vee b_4 = (10\text{—}110\text{—}0110)$

c_4' is a prime implicant because it is unexpandable to any other direction.

Hence, $\mathcal{F} = c_1' \vee c_2 \vee c_3 \vee c_4'$ is a sum-of-products expression consisting of prime implicants. (End of example).

Theorem 5.2: Let $\mathcal{F} = c \vee \mathcal{G}$, and c be a cube.

1) If $c \leqslant \mathcal{G}$, then cube c can be deleted , i.e., $\mathcal{F} \equiv \mathcal{G}$.

2) Let $\mathcal{F}$ be represented as $\mathcal{F} = c_1 \vee c_2 \vee \dots \vee c_m$,where $c_i (i=1,2,\dots,m)$ is a prime implicant. Let $\mathcal{G}_i$ be a sum of prime implicants other than c_i. If $c_i \not\leqslant \mathcal{G}_i$ for all i (i=1,2,..,m), then $\mathcal{F}$ is irredundant (minimal).

Example 5.3: Consider an array consisting of prime implicants:

$$\mathcal{F} = \begin{bmatrix} 11\text{—}010\text{—}1010 \\ 01\text{—}100\text{—}1110 \\ 10\text{—}010\text{—}1110 \\ 10\text{—}110\text{—}0110 \end{bmatrix} \begin{matrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{matrix}$$

$\mathcal{F}$ can be written as $\mathcal{F} = c_1 \vee \mathcal{G}_1$, where

$c_1 = (11\text{—}010\text{—}1010)$, and

$$\mathcal{G}_1 = \begin{bmatrix} 01\text{—}100\text{—}1110 \\ 10\text{—}010\text{—}1110 \\ 10\text{—}110\text{—}0110 \end{bmatrix}$$

Because $c_1 \not\leqslant \mathcal{G}_1$, c_1 cannot be deleted.

Similarly, c_2 cannot be deleted.

$\mathcal{F}$ is also written as $\mathcal{F} = c_3 \vee \mathcal{G}_3$, where

$c_3 = (10\text{—}010\text{—}1110)$, and

$$\mathcal{G}_3 = \begin{bmatrix} 11\text{—}010\text{—}1010 \\ 01\text{—}100\text{—}1110 \\ 10\text{—}110\text{—}0110 \end{bmatrix}$$

Because $c_3 \lessdot \mathcal{G}_3$, c_3 can be deleted.

Similarly, $\mathcal{F}$ is written as $\mathcal{F}=c_4 \vee \mathcal{G}_4$, where

c_4 = (10—110—0110) , and

$$\mathcal{G}_4 = \begin{bmatrix} 11\text{—}010\text{—}1010 \\ 01\text{—}100\text{—}1110 \end{bmatrix}$$

Because $c_4 \not\lessdot \mathcal{G}_4$, c_4 cannot be deleted.

Hence, we have an irredundant sum-of-products expression:

$$\mathcal{G}_3 = \begin{bmatrix} 11\text{—}010\text{—}1010 \\ 01\text{—}100\text{—}1110 \\ 10\text{—}110\text{—}0110 \end{bmatrix}$$

(End of example).

<u>Definition 5.2</u>: Let c be a cube of f, and let v be a minterm of c. If the prime implicant which covers v is unique, then c is <u>an essential prime implicant</u> , and v is <u>a distinguished minterm.</u>

<u>Definition 5.3:</u> A sum-of-products expression is said to be <u>minimum</u> if it consists of the minimum number of prime implicants.

<u>Lemma 5.2</u>: A minimum sum-of-products expression for f contains all the essential prime implicants of f, if any.

<u>Definition 5.4</u>: Let c_1 and c_2 be cubes, where

$c_1=X_1^{S_1}\cdot X_2^{S_2}\cdot\ldots\cdot X_n^{S_n}$ and $c_2=X_1^{T_1}\cdot X_2^{T_2}\cdot\ldots\cdot X_n^{T_n}$.

A <u>consensus</u> of c_1 and c_2 is defined as

$$\bigcup_{i=1}^{n=m} X_1^{S_1 \cap T_1}\cdot X_2^{S_2 \cap T_2}\ldots X_i^{S_i \cup T_i}\ldots X_n^{S_n \cap T_n}.$$

and denoted by $\mathrm{cons}(c_1,c_2)$.

<u>Definition 5.5</u>: Let c be a cube and $\mathcal{G}$ be an array. A consensus of c and $\mathcal{G}$ is defined as $\mathrm{cons}(c,\mathcal{G}) = \bigcup_{c_i \in \mathcal{G}} \mathrm{cons}(c,c_i)$.

<u>Theorem 5.3</u>: Suppose that $\mathcal{F}$ can be written as $\mathcal{F}=c \vee \mathcal{G}$, where c is a prime implicant. Let $\mathcal{H}=\mathrm{cons}(c,\mathcal{G})$. If $c \not\lessdot \mathcal{H}$, then c is essential.

(Proof)

Suppose that $c=X_1^{S_1}\cdot X_2^{S_2}\cdot\ldots\cdot X_n^{S_n}$ and $c \not\lessdot \mathcal{H}$.

There exists a minterm, where

$v=X_1^{a_1}\cdot X_2^{a_2}\cdots X_k^{a_k}\cdots X_n^{a_n}$ such that $v \in c\cdot\overline{\mathcal{H}}$, where $a_i \in S_i (i=1,2,\ldots,n)$.

Suppose that a prime implicant c′ which is different from c covers v, where

$c'=X_1^{T_1}\cdot X_2^{T_2}\cdots X_k^{T_k}\cdots X_n^{T_n}$. Because $c \cap c' \neq \phi$ and $c \not\subseteq c'$, we can assume that $T_k - S_k \neq \phi$, and that there is a minterm in c′ such that

$v'=X_1^{a_1}\cdot X_2^{a_2}\cdot\ldots\cdot X_{k-1}^{a_{k-1}}\cdot X_k^{b_k}\cdot X_{k+1}^{a_{k+1}}\cdot\ldots\cdot X_n^{a_n}$, where $b_k \in T_k - S_k$.

Because $v' \notin c$ and $v' \in c'$, v′ is a *minterm* of $\mathcal{G}$. Therefore, there exists a cube d in $\mathcal{G}$ which covers v′.

Let $d=X_1^{D_1}\cdot X_2^{D_2}\cdot\ldots\cdot X_k^{D_k}\cdot\ldots\cdot X_n^{D_n}$.

Note that $a_i \in D_i$ $(i=1,2,\ldots,n,\ i \neq k)$ and $b_k \in D_k$. Consider a consensus of c and d: $h_k=\mathrm{cons}(c,d) \supseteq$

$X_1^{S_1 \cap D_1}\cdot X_2^{S_2 \cap D_2}\ldots X_k^{S_k \cup D_K}\ldots X_n^{S_n \cap D_n}$.

Because $a_i \in S_i \cap D_i (i \neq k)$ and $a_k \in S_k \cup D_k$, we have $v \in h_k$.

However, this contradicts the hypothesis that $v \in c\cdot\overline{\mathcal{H}}$ because $h_k \lessdot \mathcal{H}$.

Hence, *the prime implicants which covers c* is unique. In other words, v is distinguished minterm and c is an essential prime implicant. (Q.E.D.)

<u>Example 5.4</u>: Consider an array consisting of prime implicants:

$$\mathcal{F} = \begin{bmatrix} 01\text{—}01\text{—}1110 \\ 01\text{—}10\text{—}0111 \\ 10\text{—}01\text{—}0111 \\ 10\text{—}11\text{—}0001 \end{bmatrix} \begin{matrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{matrix}$$

Let′s find the *essenqtial prime implicants* of the array.

$\mathcal{F}$ is written as $\mathcal{F}=c_1 \vee \mathcal{G}_1$, where

c_1= (01—01—1110) , and

$$\mathcal{G}_1 = \begin{bmatrix} 01\text{—}10\text{—}0111 \\ 10\text{—}01\text{—}0111 \\ 10\text{—}11\text{—}0001 \end{bmatrix}$$

First, make a consensus of c_1 and $\mathcal{G}_1$.

$$\mathcal{H}_1=\mathrm{cons}(c_1,\mathcal{G}_1)= \begin{bmatrix} 01\text{—}11\text{—}0110 \\ 11\text{—}01\text{—}0110 \end{bmatrix}$$

Because $c_1 \not\lessdot \mathcal{H}_1$, c_1 *is an essential prime* implicant.

$\mathcal{F}$ is written as $\mathcal{F}=c_2 \vee \mathcal{G}_2$, where

c_2= (01—10—0111) , and

$$\mathcal{G}_2 = \begin{bmatrix} 01\text{—}01\text{—}1110 \\ 10\text{—}01\text{—}0111 \\ 10\text{—}11\text{—}0001 \end{bmatrix}$$

Similarly, make a consensus of c_2 and $\mathcal{G}_2$:

$$\mathcal{H}_2=\mathrm{cons}(c_2,\mathcal{G}_2)= \begin{bmatrix} 01\text{—}11\text{—}0110 \\ 11\text{—}10\text{—}0001 \end{bmatrix}$$

Because $c_2 \lessdot \mathcal{H}_2$, c_2 is not essential.

Similarly, we can see that neither c_3 nor c_4 are essential.(End of Example).

VI. CONCLUSION

1. Two methods for decomposing a tautology problem into smaller ones are shown.
2. A hardware tautology checker is proposed. The computation time of the checker is proportional to the number of products in a given sum-of-products expression.
3. Application of the tautology checker for simplifying logical expressions with many variables is shown.

REFERENCES

[1] S.Muroga, Logic design and Switching Theory, Wiley-Interscience Publication, 1979.

[2] S.J.Hong, R.G.Cain, and D.L.Ostapko, 'MINI: A heuristic approach for logic minimization,' IBM J. Res. and Develop., pp.443-458, Sept. 1974.

[3] T. Sasao, 'Multiple-valued decomposition of generalized Boolean functions and the complexity of programmable logic arrays,'IEEE Tans. on Computers, vol.C-30, No.9,pp.635-643, Sept. 1981.

[4] T. Sasao, 'Input variable assignment and output phase optimization of PLA's' IEEE Trans. on Comput.(to be published)

[5] R.K.Brayton, et.al, 'A comparison of logic minimization strategies using ESPRESSO', Proc. of 1982 ISCAS pp.42-48, May 1982.

[6] T.Sasao, S.J.Hong, and R.K.Brayton, 'Minimization of PLA's by decomposition' (in preparation).

[7] Y.H.Su and P.T.Cheung, 'Computer minimization of multi-valued switching functions,' IEEE Trans. Comput. C-21, pp.995-1003,1972.

[8] M.R.Garey and D.S. Johnson, Computers and Intractability, W.H.Freeman and Company, San Francisco , 1979.

[9] R.K.Brayton, J.D. Cohen, G.D.Hachtel, B.M. Tragger, and D.Y.Y.Yun, 'Fast recursive Boolean function manipulation', Proc. 1982 ISCAS,pp.58-62, May 1982.

[10] J. P.Roth, 'Algebraic topological methods in synthesis,',In Annals of Computational Laboratory of Harvard University, vol.29. pp.57-73, 1959.

[11] P.Bricaud and J.Campbell ,'Multiple output PLA minimization: EMIN', WESCON 78,33/3, 1978.

[12] Z. Arevalo and J.G. Bredeson, 'A method to simplify a Boolean function into a near minimal sum-of-products for programmable logic arrays,' IEEE Trans. Comput.,vol C-27,pp.1028-1039, Nov.1978.

[13] P.W. Besslich and P.Pichlbauer 'Fast transform procedure for the generation of near minimal covers of Boolean functions', IEE Proc. vol.128, Pt.E. No.6, pp.250-254, Nov. 1981.

[14] T.Sasao,'A fast complementaion algorithm for sum-of-products expressions of multiple-valued input binary functions', Proc. 13th ISMVL, pp.103-110, May 1983.

[15] D. W. Brown,'A state-machine synthesizer --SMS', Proc. of 18th Design Automation Conference ,June 1981.

[16] D.L. Dietmeyer, Logic Design of Digital Systems (Second Edition), Allyn and Bacon Inc., Boston, 1978.

[17] M. A. Breuer, Design Automation of Digital Systems, vol.1:Theory and Technique, Prentice-Hall, 1972.