SRC TR 85-39

A New Approach for Compiling Boolean Functions

by

J. Ja'Ja' & S. M. Wu

A New Approach for Compiling Boolean Functions*

(Preliminary Draft)

Joseph Ja'Ja'

Department of Electrical Engineering

and

Institute for Advanced Computer Studies

University of Maryland

College Park, MD 20740

Sau-Mou Wu

Chaitali Chakrabarti

Department of Electrical Engineering

University of Maryland, College Park

Abstract

We propose a new approach for laying out Boolean functions which is based on extracting the symmetries of a given set of functions and applying optimization procedures especially tailored to exploit these symmetries. This paper establishes a rigorous foundation for this approach and shows that it will outperform existing methods for many classes of functions. The different components of a newly developed system, SYMBL, will be briefly described.

* Supported partially by the U.S. Army Research Office, Contract No. DANG29-82-k-0110, NSA Contract No. MDA-904-85H-0015, by NSF Contract No. MCS-83-15890, and by the Systems Research Center Contract No. OIR-8500108.

1. Introduction

The recent advances in the integrated circuit technology [1] have substantially increased the amount of hardware that can be put on a single chip, and hence the overall design process has become much more complex. Good and efficient automated tools that can assist the user in all the different design phases have become a necessity. In particular, the automatic layout of Boolean functions is an essential ingredient of any set of design tools. The problem of the automatic generation of good layouts for a given set of functions has been considered by many researchers most of who have refined and elaborated on the PLA approach. The main goal of this research has been to develop algorithms that can efficiently generate layouts that are regular, small and whose overall delay is minimal.

Programmable Logic Arrays (PLA's) are certainly the most popular tools used for laying out Boolean functions. These structures implement two-level logic in a simple and regular fashion using two planes: AND and OR. The main strategy used revolves around the Quine-McCluskey procedure to determine a minimum cover of prime imiplicants. There are two main difficulties with this method. The first, and the more serious, is that the optimal PLA's tend to be quite large even for some simple and natural functions. This is due to the fact that the restriction to two-level logic causes circuits for many problems to become excessively large. The second difficulty is related to the complexity of the corresponding optimization problems most of which can be shown to be NP-complete. Many researchers have developed good heuristics to handle these optimization problems [2,3](see [4] for more references).

Another approach that seems to offer more flexibility is based on using Weinberger arrays[5]. In this case, Boolean functions are expressed in multi-level NAND and NOR logic, and are then laid out in a structure with pull-up transistors at the top and pulldown transistors along with the connecting wires under the pullups. In spite of the added flexibility, however, it is not clear that the resulting layout will in general be much smaller, and moreover no good general layout scheme has appeared in the literature. In addition, the corresponding optimization problems are much more difficult than those which have risen in the PLA case.

In this paper, we propose a new approach to layout a set of Boolean functions . It consists of (i) extracting all the possible symmetries in the given set of functions, (ii) applying optimization procedures designed for symmetric functions, and finally (iii) obtaining a regular

layout whose main components are variations of PLA's and Weinberger arrays. We establish a rigorous foundation for our approach and show that our method will always yield a layout that is at least as good as the corresponding PLA. We also show that any set of functions which have a reasonable degree of symmetry can be laid out in a much smaller area. A system SYMBL, SYMmetric Boolean Layout, has been partially written and our initial experiments are encouraging.

The rest of the paper is organized as follows. The next section addresses the class of functions that are completely symmetric and shows how to lay them out efficiently in a very small area. Section 3 introduces the class of functions that have partial symmetries and discusses an extension of the layout method developed for the symmetric functions. An overview of the software system is given in the last section together with an analysis of the time complexity of some of the main procedures involved.

2. Layout of Symmetric Function

In this section, we briefly introduce the class of symmetric functions and show that any such function can be laid out quite compactly with a relatively small overall delay. A software system, SYMMETRIC, checks whether a given set of Boolean functions are symmetric[6]and, in the affirmative, produces the layout. A brief outline of the overall algorithm will be given and its time complexity will be also analyzed.

2.1 Brief Review of Symmetric Function

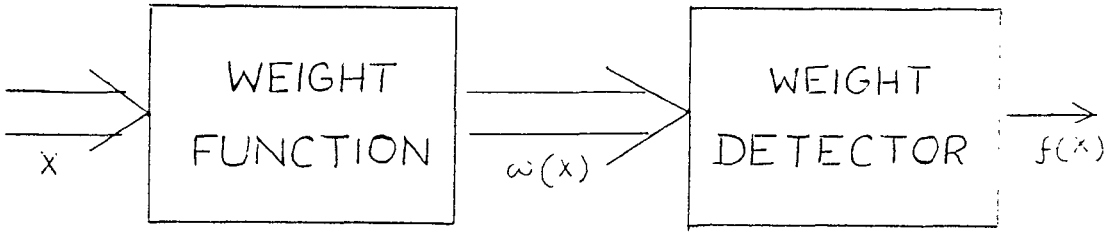
Definition. *A Boolean function* $f(x_1, \ldots, x_n)$ *is symmetric if for any permutation* $\pi \in S_n$, $f(x_1, \ldots, x_n) = f(x_{\pi(1)}, \ldots, x_{\pi(n)})$.

Let $X = (x_1, \ldots, x_n)$ be a binary n-tuple and let the weight $w(X)$ of X be the number of 1's in X. Then it is well-known that the value of a symmetric function $f(X)$ depends on $w(X)$; in other words, f has the same value on all binary n-tuples that have the same weight. Therefore, if f(X) has value a_i, $a_i \in \{0, 1\}$ on n-tuples X of weight i, for $0 \leq i \leq n$, then $f(X)$ can be written in the form

$$f(X) = \sum_{i=0}^{n} a_i \bullet \sigma_i(X) \tag{2.1}$$

where $\sum$ and $\bullet$ denote AND and OR respectively, and the i-th elementary symmetric function, $\sigma_i(x_1, \ldots, x_n)$ equals to 1 when $w(x_1, \ldots, x_n) = i$.

Let $F = \{f_i(x_1, \ldots, x_n)\}_{i=1}^{t}$ be a set of symmetric Boolean functions. As shown in *Fig(2.1)*, a layout for F will consist of two parts: (1) a layout of the weight(or counting) function $f_w(x_1, \ldots, x_n)$ that counts the number of 1's among the x's and outputs its binary expansion ; (2) a small PLA weight detector that takes the output of the weight function and produce the set F.



Fig(2.1)

2.2 Weight Function

2.2.1 Basic Logic Analysis

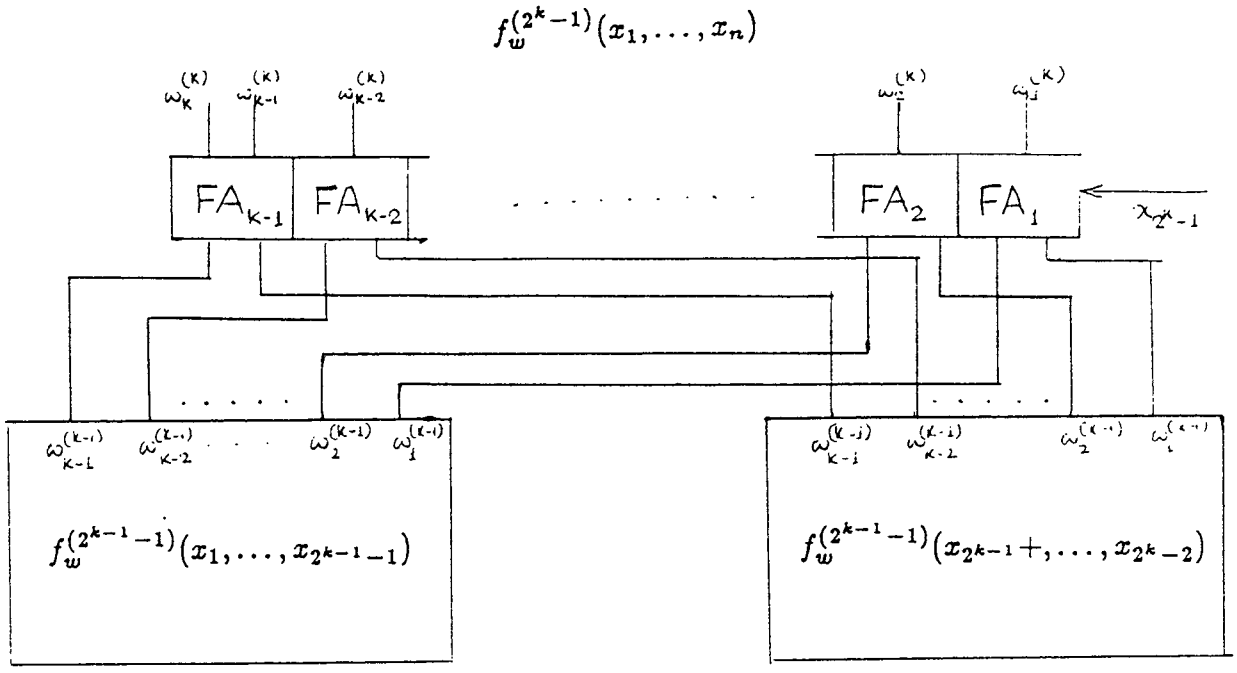
Recall that the weight function is a mapping $f_w : \{0,1\}^n \rightarrow \{0,1\}^{\lceil log\,(n+1)\rceil}$ that outputs the binary expansion of the weight. For simplicity, assume $n = 2^k - 1$ (the general case for any n will be discussed later). It is well-known that f_w can be realized by a divide-and-conquer strategy as follows.

$$f_w^{(2^k-1)}(x_1, \ldots, x_n) = f_w^{(2^{k-1}-1)}(x_1, \ldots, x_{2^{k-1}-1}) + f_w^{(2^{k-1}-1)}(x_{2^{k-1}+}, \ldots, x_{2^k-2}) + x_{2^k-1} \tag{2.2}$$

If we take full adder as base operator, then equation (2.2) can be implemented as shown in *Fig(2.2)*

Lemma. *Let the full adder be the base operator of the weight function $f_w(x_1, \ldots, x_n)$ with $n = 2^k - 1$. Then we need $2^k - k - 2 = O(n)$ full adders to implement f_w with delay $O(log\, n)$.*

Now suppose that $n \neq 2^k - 1$. Let l be the maximum integer such that $n > 2^l - 1$. Put l adders along with one input variable at the root of the tree and make the left subtree



Fig(2.2)

a complete tree that corresponds to $f_w^{2^l-1}(x_1,\ldots,x_{2^l-1})$. For the remaining $n-(2^l-2)$ variables, apply the same procedure rescursively at the right subtree. One can show that the number of adders used by the above construction is $O(n)$ and the delay is $O(logn)$.

2.2 Layout Scheme

A scheme that compactly lays out the tree of a weight function will be outlined in this section. We require that all the inputs and outputs be on the boundary and that the enclosing rectangle be as close to a square as possible.

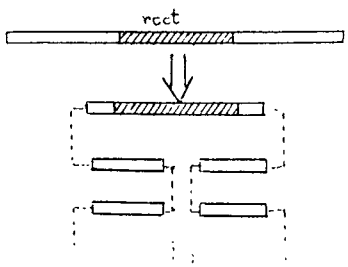
Theorem 1. *It is possible to lay out the tree of the weight function of n variables in a square of size $O(\sqrt{n}log\ n)$ such that all the inputs and outputs lie on the boundary.*

Proof: We first show how to lay out the tree linearly in an area of $O(nlog^2\ n)$ and then show how to turn it into square.

Lay out the tree of (say) $(k-1)$-adjacent adders in the middle. Recusively, lay out the left subtree on the left side of the root and the right subtree on the right side. The length of the layout is clearly $O(n)$. Its height $H(k)$, i.e. the number of horizontal tracks, is given by the recurrence

$$H(k) \leq H(k-1)+k-1 \quad \text{and} \quad H(2)=1$$

It then follows that $H(k) = O(k^2) = O(log^2\ n)$.



Next, as illustrated on the left, we fold the linear layout into a square with the root on the top row. This folding procedure does not increase the area by more than a factor of 2[7]. Thus we can make the layout into a square with the desired properties. $\diamond$

2.3 Weight Detector

A symmetric function f can be written in the form (2.1).

$$f(x_1, \ldots, x_n) = \sum_{i=0}^{n} a_i \bullet \sigma_i(x_1, \ldots, x_n) \tag{2.1}$$

The weight detctor is the function $f_p : \{0,1\}^{\lceil log_2(n+1) \rceil} \rightarrow \{0,1\}$ that maps the outputs of the weight function into $\{0,1\}$.

Let $m = \lceil log_2(n+1) \rceil$ and $I \subseteq \{0, 1, \ldots, n\}$ be such that $a_i = 1$ for all $i \in I$. Then f_p can be written in the following form.

$$f_p(y_1, \ldots, y_m) = \sum_{i \in I} \delta_i(y_1, \ldots, y_m)$$

where δ_i equals to 1 only when the binary expansion of i is equal to $(y_1, \ldots, y_m)$.

It is very easy to use a PLA to implement f_p once I is known. Since $I \subseteq \{0, 1, \ldots, n\}$, the area occupied by the PLA part is at most $O(nlog\ n)$, which is small compared with that of the weight function. On the other hand, if there are more than one symmetric function, only the OR plane will grow linearly with the number of functions.

Theorem 2. *It is possible to lay out m n-variable symmetric functions in an area $O(nlog^2 n + mn)$ with $O(logn)$ delay.*

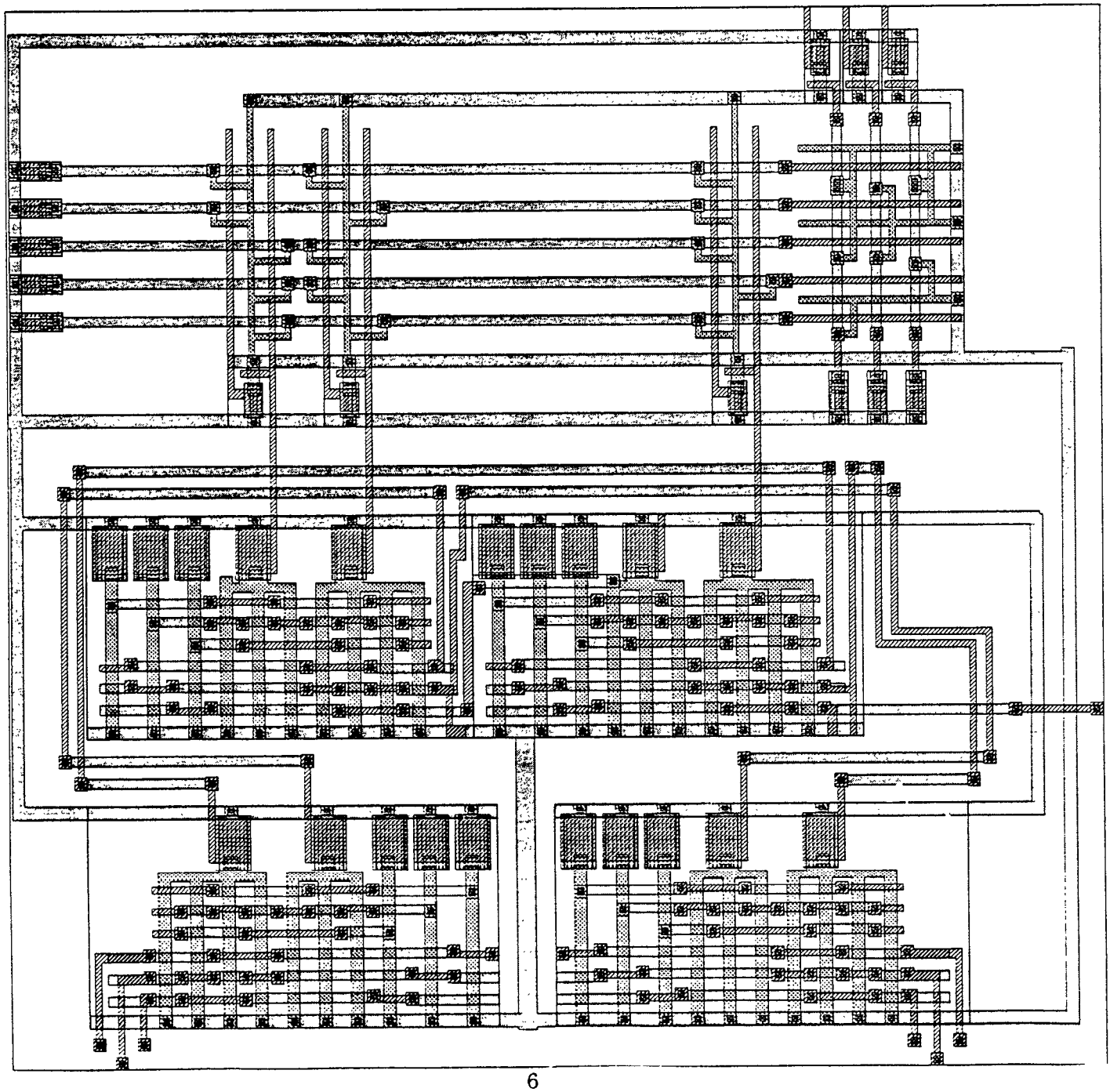
We end this section by presenting the layout produced by SYMMETRIC for the three symmetric functions on next page.

Example: layout of a set of totally symmetric functions

$f_1(x_1, x_2, \ldots x_7) = x_1 \oplus x_2 \oplus \ldots \oplus x_7$;

$f_2(x_1, \ldots, x_7) = \sum($ 112 104 100 98 97 88 84 82 81 76 74 73 70 69 67 56 52 50 49 44 42 41 38 37 35 28 26 25 22 21 19 14 13 11 7 124 122 121 118 117 115 110 109 107 103 94 93 91 87 79 62 61 59 55 47 31)

$f_3(x_1, \ldots, x_7) = \sum($ 127 124 122 121 118 117 115 110 109 107 103 94 93 91 87 79 62 61 59 55 47 31 96 80 72 68 66 65 48 40 36 34 33 24 20 18 17 12 10 9 6 5 3);



3. Layout of Partially Symmetric Functions

In this section, we will introduce the notion of partially symmetric functions and establish some of their basic properties. Later we show how to layout these functions efficiently. Given *any function*, our overall strategy is to extract all the partial symmetries in the function and then apply some optimization procedures particularly tailored for partially symmetric functions which will yield the final layout. We will show that we always end up with a layout at least as good as the corresponding PLA. As a matter of fact, we will generalize the concept of prime implicants and show that a slight modification of the Quine-McCluskey procedure will yield optimal solutions for our case.

3.1 Partially Symmetric Functions

Let $f(x_1, x_2, \ldots, x_n)$ be a Boolean function and let $\rho = \{X_1, X_2, \ldots, X_s\}$ be a partition of $\{x_1, x_2, \ldots, x_n\}$ such that $|X_i| = n_i$, $1 \leq i \leq s$ and $\sum_{i=1}^{s} n_i = n$. f is callel ρ-symmetric if

$$f(X_1, X_2, \ldots, X_s) = f(X_1', X_2', \ldots, X_s')$$

where $X_i' = \prod_i(X_i)$, $\prod_i$ is an arbitrary permutation on X_i.

It follows that f is determined by the weight functions on the X_i's, i.e., by $\omega(X_i) = k_i$, $1 \leq i \leq s$. Defince $C(f)$ to be

$$C(f) = \{\, (k_1, k_2, \ldots, k_s) \mid f = 1 \ for \ \omega(X_i) = k_i \,,\ 1 \leq i \leq s \,\}$$

Let $(k_1, k_2, \ldots, k_s) \in C(f)$ and let $\sigma_{(k_1,k_2,\ldots,k_s)}(X) = \sigma_{k_1}(X_1)\sigma_{k_2}(X_2)\ldots\sigma_{k_s}(X_s)$, where $\sigma_{k_i}(X_i)$ is the $k_i th$ elementary symmetric function on X_i. Clearly,

$$f(x_1, x_2, \ldots, x_n) = \sum_{\underline{k}=(k_1,k_2,\ldots,k_s)\in C(f)} \sigma_{\underline{k}}(X)$$

We want to rewrite f as follows:

$$f(x_1, x_2, \ldots, x_n) = \sum f_{i_1}(X_1) f_{i_2}(X_2) \ldots f_{i_s}(X_s)$$

where $f_{i_j}(X_j)$ is a symmetric function over X_j and the number of terms is as small as possible. Such a realization is called a *minimum symmetric realization*. It turns out that a variation of

the Quine-McCluskey procedure will allow us to find such a realization. Notice that if f has no symmetries, i.e. $n_i = 1$ for all i, the realization is a minimum cover of f. We now introduce two definitions.

A product of symmetric functions $P = f_{i_1}(X_1)f_{i_2}(X_2)...f_{i_s}(X_s)$ will be called a ***symmetric implicant*** of f if $f = 1$ whenever $P = 1$. P will be called a ***symmetric prime implicant*** if P is a symmetric implicant such that there is no symmetric implicant P' with fewer terms with the property that $P' = 1$ whenever $P = 1$.

Lemma. *Let f be a partially symmetric function with respect to $\{X_1, X_2, \ldots, X_s\}$. Then there exists a minimal symmetric realization of f such that each of the products is a symmetric prime implicant.*

3.2 Layout Scheme

Our scheme for laying out partially symmetric functions consists of two parts. The first part is a combination of layouts of the form presented in section 2, while the second is a regular structure that can be viewed as an extension of a Weinberger array.

Theorem 3. *Let f be partially symmetric with respect to $\{X_1, X_2, \ldots, X_s\}$ such that $|X_i| = n_i$, $1 \leq i \leq s$, and let $c_s(f)$ be the number of terms in a minimal symmetric ralization of f. Then we can layout f in an area of*

$$O\left(\sum_{i=1}^{s} n_i log^2\, n_i + nc_s(f)\right)$$

Proof: We have already seen how to layout $\omega(X_i)$ in an area of $O(n_i log^2\, n_i)$. It is easy to see that we can get all the elementary symmetric functions on X_i in essentially the same area. Hence let's assume that we have cells $ES(X_i)$ to compute the elementary symmetric functions of X_i for all i. Useing the configuration in $Fig(3.1)$, we can realize each of the products in a minimal symmetric form. $\diamond$

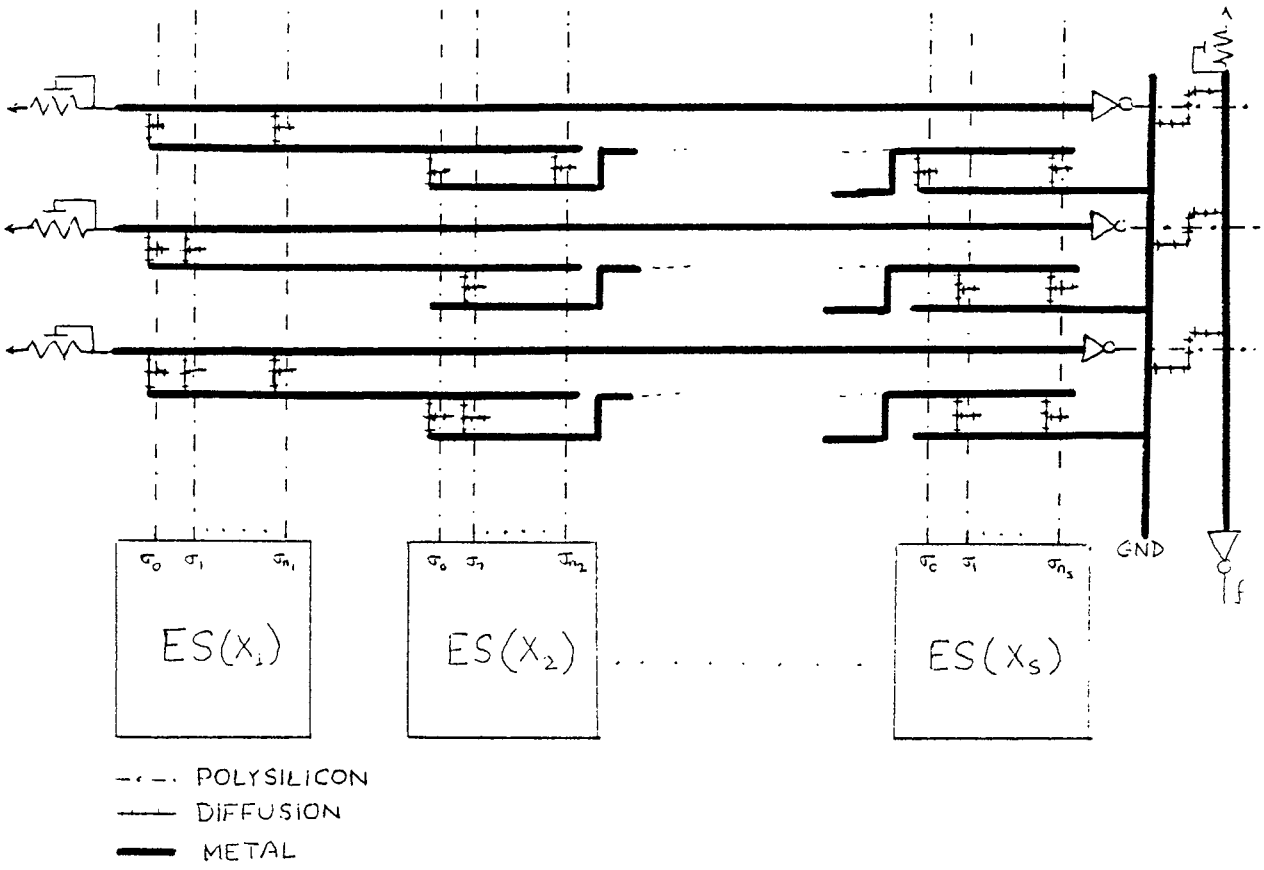


Fig (3.1)

3.3 Comparison with PLA's

Let f be a partially symmetric function with respect to $\{X_1, X_2, \ldots, X_s\}$ such that $|X_i| = n_i$, $1 \leq i \leq s$ and $n = \sum n_i$. Let $c(f)$ be the number of prime implicants in a minimum cover of f. Clearly the size of an optimal PLA realizing f is $O(nc(f))$. As we have seen in the previous subsection, our method produces a layout of size $O(nlog^2\, n + nc_s(f))$. It is obvious that $c_s(f) \leq c(f)$. The exclusive OR is a trivial example for which $c_s(f)$ $(= 1)$ is substantially smaller than $c(f)$ $(= 2^{n-1})$. We now show that if the function has a reasonable degree of symmetry, then $c(f)$ will be much larger than $c_s(f)$. First, let $p \leq q \leq n$ be positive integers and let $X = \{x_1, x_2, \ldots, x_n\}$. Then

$$\omega^{[p,q]}(X) = \begin{cases} 1, & \text{if } \omega(X) \in [p,q]; \\ 0, & \text{otherwise} \end{cases}$$

Note that

$$c\{\omega^{[p,q]}(X)\} = \binom{n}{p}\binom{n-p}{n-q}$$

One can check that the partially symmetric function f can be rewritten in the form

$$f(X_1, X_2, \ldots, X_s) = \sum_{i=1}^{c'_s(f)} W_i \tag{3.1}$$

where

$$W_i = \prod_{j=1}^{s} \omega^{[v_{ij}, t_{ij}]}(X_j)$$

Then we are ready for the following theorem.

Theorem 4. *Let f be a function partially symmetric over $\{X_1, \ldots, X_s\}$ such that $|X_i| = n_i$, $n_i \geq 2$ for $1 \leq i \leq s$. Suppose that there exist at least a pair of intervals $[v_{ik}, t_{ik}]$ and $[v_{jk}, t_{jk}]$ that are not adjacent and nonoverlapping for every i and j, $i \neq j$. If we replace each $\omega^{[v_{ij}, t_{ij}]}(X_j)$ with its minimum cover, then after applying the distributive law, the products obtained are all prime implicants of f. Moreover, they form a minimum cover.*

Proof: We start by observing the following fact. Let $\{p_{1i}\}$ and $\{p_{2j}\}$ be the minimum covers of $f_1(X_1)$ and $f_2(X_2)$ respectively such that $X_1 \cap X_2 = \emptyset$. Then $\{p_{1i}p_{2j}\}$ is a minimum cover of prime implicants of $f_1(X_1)f_2(X_2)$.

It follows that the products obtained from $\omega^{[v_{ij}, t_{ij}]}(X_j)$, $1 \leq j \leq s$ form a minimum cover of W_i, $1 \leq i \leq c'_s(f)$.

Claim. *$Minterm(W_i) \cap Minterm(W_j) = \emptyset$, $\forall i \neq j$, where $Minterm(g)$ is the set of minterms of g.*

Proof of claim: Let $m \in Minterm(W_i) \cap Minterm(W_j)$. Then for some k, $[v_{ik}, t_{ik}]$ and $[v_{jk}, t_{jk}]$ are not adjacent and nonoverlapping. Assume, without loss of generality, that $v_{jk} > t_{ik}$. Since $m \in Minterm(W_i)$, m has at least $n_i - t_{ik}$ complemented variables from X_k. On the other hand, $m \in Minterm(W_j)$ implies that at most $n_i - v_{jk}$ complemented variables from X_k. But $n_i - v_{jk} < n_i - t_{ik}$, a contradiction and hence the claim follows. $\diamond$

We now show that each element p of the minimum cover of W_i is a prime implicant of f. Suppose not. Then there exists an implicant q of f such that either $p = 1 \Rightarrow q = 1 \Rightarrow f = 1$ or $p + q$ is an implicant of f. In the first case, q covers some minterms of f. By the disjointness

property, q can only cover minterm belonging to one W_j, for some j. It is then easy to check that the first case can not happen. The second case implies that W_i and W_j are adjacent, which is impossible.

Using the above claim, it is easy to check that the resulting prime implicants form a minimum cover. $\diamond$

Corollary. *For the function f, the following holds:*

$$c(f) \geq \sum_{i=1}^{c'_s(f)} \left[\prod_{j=1}^{s} \binom{n_j}{v_{ij}} \binom{n_j - v_{ij}}{n_j - t_{ij}} \right]$$

4 Main Components of SYMBL

There are three main procedures in SYMBL: PARTITION, SYMCOVER, and LAYOUT. PARTITION takes the description of a set of functions and *partitions* the Boolean variables into sets such that all the the given functions are partially symmetric with respect to this partition. SYMCOVER applies a variation of the Quine-McCluskey procedure to obtain a minimal symmetric realization. The procedure LAYOUT takes input from SYMCOVER and generates the layout using the program SYMMETRIC described in section 2. In this version of the paper, we will briefly sketch the basic algorithms behind the first two programs.

4.1 Partitioning

Given a Boolean function $f(x_1, \ldots, x_n)$, the relation $x_n \sim x_j$ holds, if and only if f has the same value whenever x_i and x_j are interchanged. It is easy to see that $\sim$ is an equivalence relation. Our problem is to find the equivalence classes of the set $\{x_1, x_2, \ldots, x_n\}$ [8–10].

For a function $f(x_1 \ldots x_n)$, let us denote

$$f_{x_i x_j}(a_i, a_j) \equiv x_i^{(a_i)} x_j^{(a_j)} f(x_1 \ldots x_{i-1}, a_i, \ldots x_{j-1}, a_j, \ldots x_n)$$

where

$$x_k^{(a_i)} = \begin{cases} x_k, & \text{if } a_k = 1; \\ \bar{x}_k & \text{if } a_k = 0. \end{cases}$$

Then for any x_i and x_j, $i \neq j$

$$f(x_1, \ldots x_n) = f_{x_i x_j}(0,0) + f_{x_i x_j}(0,1) + f_{x_i x_j}(1,0) + f_{x_i x_j}(1,1)$$

and the following lemma is obvious.

Lemma 3. $x_i \sim x_j \iff f_{x_i x_j}(0,1) = f_{x_i x_j}(1,0)$

Hence, it is obvious that given a truth table of a function $f(x_1 \ldots x_n)$, $x_i \sim x_j$ iff for any row containing $x_i x_j = 01$, there must exist another row containing $x_i x_j = 10$ such that the values of the other variables in these two rows are the same.

Theorem 5. *Given a truth table of a n-variable function, it takes $O(n^2T)$ time to partition the input variables into the partial symmetric blocks, where T is the number of rows of the table.*

Proof: In order to check the compatibility of any two variables x_i and x_j, $O(nT)$ comparisions are required to ensure that $f_{x_i x_j}(01) = f_{x_i x_j}(10)$. Thus $O(n^2T)$ is needed to partition n-variable function.$\diamond$

The procedure Partition given in $Fig(4.1)$ can be extanded to generate an array representation of $C(f)$ i.e. a $t \times s$ array K such that the j-th tuple of $C(f)$ is represented by j-th row of K. Also, this procedure can be easily generalized to work for a set of functions.

Algorithm: Partition

Input: truth table T_0 of an n-variable function.

Output: the partial-symmetric partitioning block.

Method:

```
k=1;
P1 = {x1}   /* Pi: the i-th partition block */
p1 ≡ x1   /* pi: the representative ofthe variables in Pi. */
for j = 2 to n do
  for i = 1 to k do
    if(xj is equivalent to variable pi)
    {   Pi = Pi ∪ {xj}
      exitfor-i
    }
    else/* xj can not be in Pi */
      if(i = k)
      {   k = k + 1; /*creat new block*/
        Pk = {xj}; pk ≡ xj;
        exitfor-i;
      }
  endfor-i
endfor-j
```

Fig(4.1)

4.2 Minimization

Symmetric Prime Implicants:

Let t be the number of the symmetric implicants and s be the number of equivalence partitions. The output of the partitioning procedure is an array of vectors $K_j = (k_{1j}, \ldots, k_{sj})$, $j = 1, \ldots, t$ and $0 \leq k_{ij} \leq n_i$. Quine McCluskey's method for generating prime implicants has been modified to generate the symmetric prime implicants from this array. Unlike the grouping of implicants of two valued Boolean variables (which was on the basis of number of

0's), here the grouping of K_j's has to be done on the basis of a value p which occurs in all the s columns of the array. As a result, only K_j's in the same group and adjacent group can be combined. At the i-th iteration ($i > 1$), the vectors take the form $K'_j = (k'_{1j}, \ldots, k'_{sj})$, where each of the k'_{ij} is a combination of k_{il}'s, $0 \leq k_{il} \leq n_i$. The procedure for combining two vectors K'_j and K'_l is as follows.

$K'_j = (k'_{ij}, \ldots, k'_{sj})$ can be combined with $K'_l = (k'_{il}, \ldots, k'_{sl})$ to form $K'_p = (k'_{ip}, \ldots, k'_{sp})$ iff $s-1$ of the k_{ij}'s and k_{jl}'s are same. If the two vectors differ only in the t-th column, i.e. $k'_{tj} \neq k'_{tl}$ then k'_{tp} is a combination of k'_{tj} and k'_{tl}. The procedure Symmetric Prime Implicants is given in $Fig(4.2)$.

Algorithm: Symmetric Prime Implicants

1. Choose a suitable value p and group K_j's accordingly. Let q be the number of groups.
2. **for** $j = 1$ **to** t **do**
   {
   $K'_j = K_j$;
   mark K'_j ;
   }
3. $i = 1$;
   while $i <= q - 1$ **do**
   {
   compare every K'_j in group i with K'_l in group i and $i+1$.
   if combination is possible
   create K'_p and mark it
   else unmark K'_j ;
   $i = i + 1$
   }
4. **if** there have been no combinations in step3 , **halt** .
   else
   { $q = q - 1$;
   go to step3;
   }

The symmetric prime implicants are the marked K_j's.

$Fig(4.2)$

Symmetric Minimum Cover:

The symmetric prime implicants are used to form a chart which is essentially an array of u rows and v columns where u is the number of symmetric prime implicats and v is the number of K_i's in $C(f)$. The (i,j)-th entry in the array is 1 if the i-th symmetric prime implicant covers K_j. Existing procedures for finding the minimum cover of prime implicants by identifying essential rows, essential columns, dominating rows, dominated columns have been used to find the symmetric minimum cover.

A few steps of the minimization procedure are sketched in the example below.

Example 4.1: The input to the procedure is an array of the form

```
1 0 0
1 0 1
1 1 0
2 1 0
1 0 2
0 0 1
0 2 0
1 1 1
```

Choice of p=0 leads to three groups.

```
  i=0          i=1                 i=2
1 0 0        1   (0,1)   0      1   (0,1)   (0,1)**
0 0 1        1     0     (0,1)  1     0     (0,1,2)**
0 2 0**      1     0     (0,2)  ------------------
-------    (0,1)   0      1**
1 1 0      ------------------
2 1 0      (1,2)   1      0**
1 0 2        1     1     (0,1)
1 0 1        1     0     (1,2)
-------      1   (0,1)    1
1 1 1      ------------------
-------
```

The symmetric prime implicants are

$\sigma_0(X_1)\sigma_2(X_2)\sigma_3(X_0)$,

$(\sigma_0(X_1)+\sigma_1(X_1))\sigma_0(X_2)\sigma_1(X_3)$, $\quad(\sigma_1(X_1)+\sigma_2(X_1))\sigma_1(X_2)\sigma_0(X_3)$,

$\sigma_1(X_1)\sigma_0(X_2)(\sigma_0(X_3)+\sigma_1(X_3)+\sigma_2(X_3))$, $\quad\sigma_1(X_1)(\sigma_0(X_2)+\sigma_1(X_2))(\sigma_0(X_3)+\sigma_1(X_3))$

We illustrate the layout of partially symmetric functions with two examples. Example 4.2 realizes three functions f_1, f_2 and f_3 of 23 variables. The partitioning procedure yields a partition with 7, 7 and 9 variables in each equivalence block, X_i, $i = 1,2,3$. The T_i's are the symmetric prime implicants which combine to form the minimum cover of the functions. Similarly, example 4.3 realizes another set of partially symmetric functions of 14 variables, partitioned into three blocks with 3, 5 and 6 variables.

Example 4.2 :

$T_1 = (\sigma_0(X_1) + \sigma_4(X_1) + \sigma_6(X_1))(\sigma_1(X_2) + \sigma_3(X_2))(\sigma_0(X_3) + \sigma_3(X_3) + \sigma_4(X_3))$

$T_2 = (\sigma_2(X_1) + \sigma_5(X_1) + \sigma_7(X_1))(\sigma_0(X_2) + \sigma_5(X_2) + \sigma_6(X_2) + \sigma_7(X_2))(\sigma_1(X_3) + \sigma_2(X_3) + \sigma_5(X_3))$

$T_3 = (\sigma_5(X_1))(\sigma_2(X_2) + \sigma_4(X_2) + \sigma_6(X_2))(\sigma_6(X_3) + \sigma_7(X_3))$

$T_4 = (\sigma_6(X_1) + \sigma_7(X_1))(\sigma_3(X_2))(\sigma_5(X_3) + \sigma_8(X_3) + \sigma_9(X_3))$

$T_5 = (\sigma_1(X_1) + \sigma_2(X_1))(\sigma_3(X_2) + \sigma_5(X_2) + \sigma_6(X_2))(\sigma_7(X_3) + \sigma_9(X_3))$

$T_6 = (\sigma_0(X_1) + \sigma_3(X_1) + \sigma_6(X_1))(\sigma_4(X_2))(\sigma_0(X_3) + \sigma_5(X_3) + \sigma_9(X_3))$

$T_7 = (\sigma_1(X_1) + \sigma_2(X_1) + \sigma_3(X_1) + \sigma_4(X_1) + \sigma_5(X_1))(\sigma_2(X_2) + \sigma_3(X_2))(\sigma_7(X_3))$

$T_8 = (\sigma_1(X_1) + \sigma_6(X_1))(\sigma_2(X_2) + \sigma_3(X_2) + \sigma_4(X_2))(\sigma_6(X_3) + \sigma_8(X_3))$

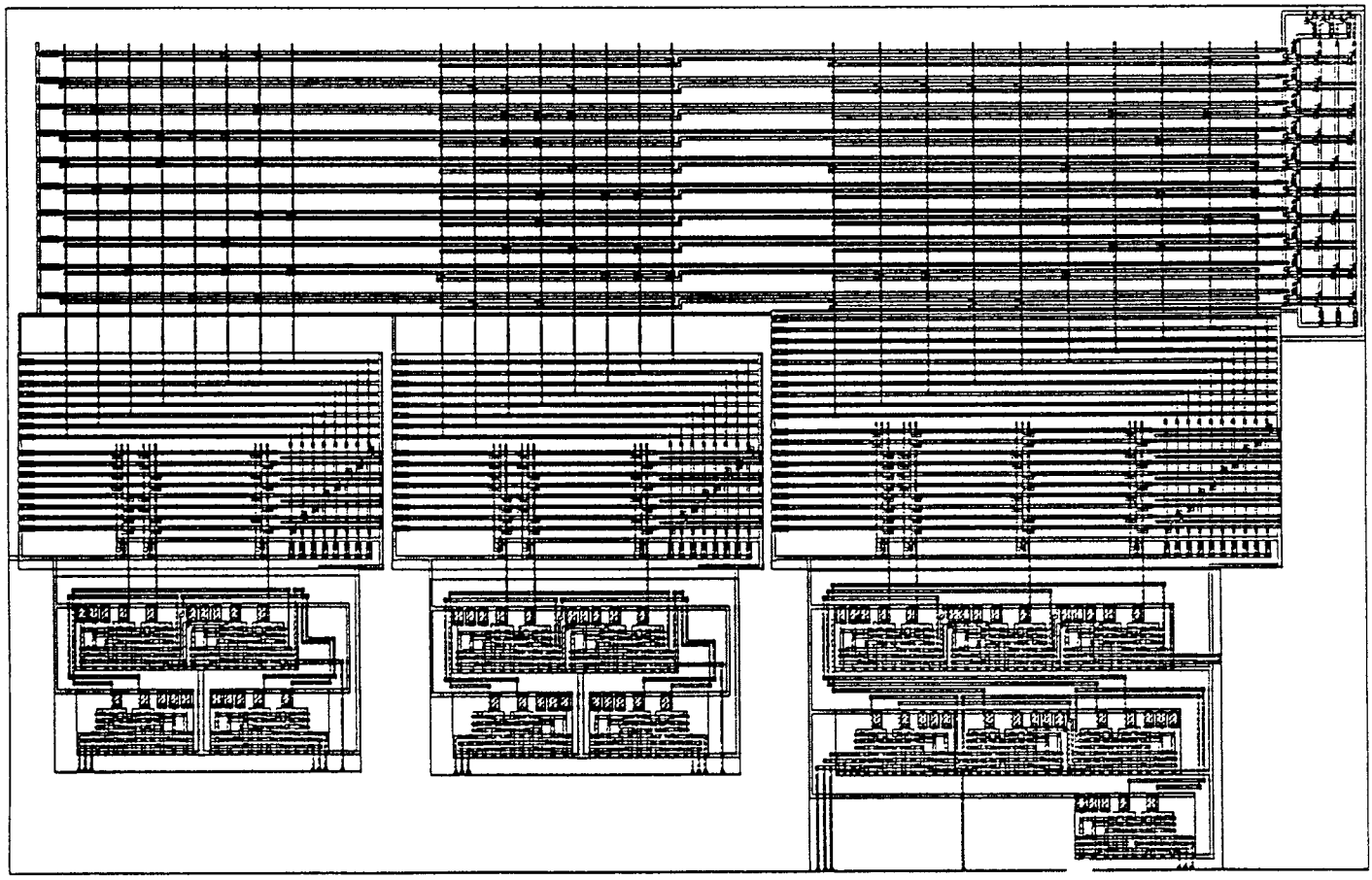
$T_9 = (\sigma_0(X_1))(\sigma_1(X_2) + \sigma_2(X_2) + \sigma_4(X_2))(\sigma_1(X_3) + \sigma_2(X_3) + \sigma_3(X_3) + \sigma_4(X_3))$

$T_{10} = (\sigma_5(X_1) + \sigma_6(X_1))(\sigma_7(X_2))(\sigma_0(X_3) + \sigma_8(X_3) + \sigma_9(X_3))$

$f_1 = T_1 + T_3 + T_5 + T_7 + T_8 + T_9 + T_{10}$

$f_2 = T_2 + T_4 + T_6$

$f_3 = T_1 + T_2 + T_7 + T_8 + T_{10}$



Example 4.3 :

$T_1 = (\sigma_0(X_1) + \sigma_1(X_1))(\sigma_1(X_2) + \sigma_3(X_2))(\sigma_2(X_3) + \sigma_4(X_3) + \sigma_6(X_3))$

$T_2 = (\sigma_1(X_1) + \sigma_2(X_1))(\sigma_2(X_2) + \sigma_4(X_2) + \sigma_5(X_2))(\sigma_0(X_3) + \sigma_1(X_3))$

$T_3 = (\sigma_3(X_1))(\sigma_0(X_2) + \sigma_1(X_2))(\sigma_2(X_3) + \sigma_3(X_3))$

$T_4 = (\sigma_0(X_1) + \sigma_3(X_1))(\sigma_3(X_2))(\sigma_4(X_3) + \sigma_5(X_3))$

$T_5 = (\sigma_2(X_1) + \sigma_3(X_1))(\sigma_3(X_2) + \sigma_5(X_2))(\sigma_0(X_3))$

$T_6 = (\sigma_1(X_1))(\sigma_1(X_2) + \sigma_4(X_2))(\sigma_2(X_3) + \sigma_4(X_3) + \sigma_5(X_3) + \sigma_6(X_3))$

$T_7 = (\sigma_0(X_1) + \sigma_1(X_1) + \sigma_2(X_1))(\sigma_2(X_2) + \sigma_3(X_2))(\sigma_1(X_3) + \sigma_2(X_3))$

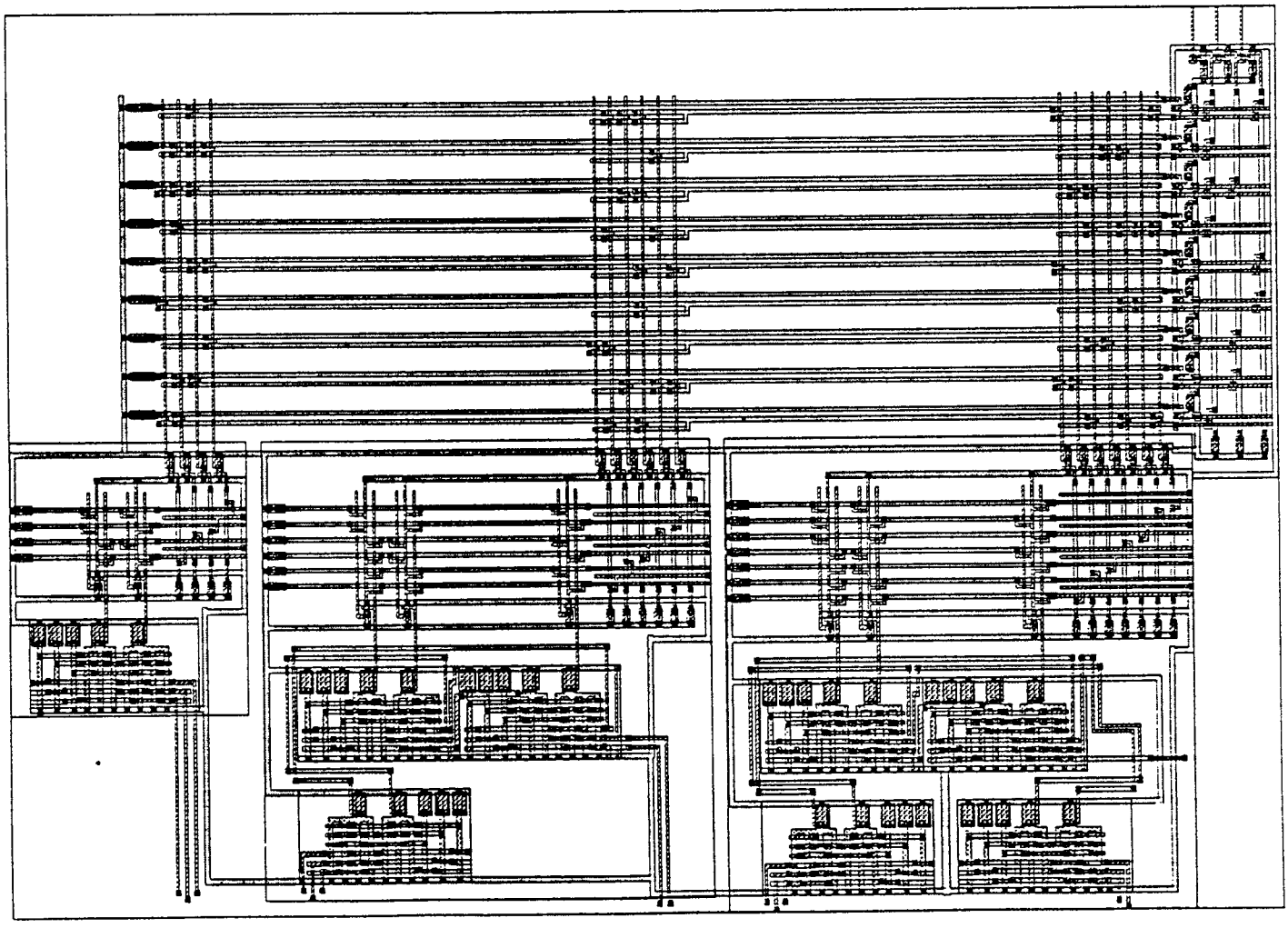
$T_8 = (\sigma_1(X_1) + \sigma_2(X_1) + \sigma_3(X_1))(\sigma_4(X_2))(\sigma_3(X_3) + \sigma_4(X_3))$

$T_9 = (\sigma_2(X_1))(\sigma_1(X_2) + \sigma_2(X_2) + \sigma_3(X_2))(\sigma_0(X_3) + \sigma_6(X_3))$

$f_1 = T_1 + T_6 + T_7 + T_8 + T_9$

$f_2 = T_2 + T_3 + T_7$

$f_3 = T_4 + T_5 + T_9$



References

[1] C. Mead & L. Conway, *Introduction to VLSI Systems*, Addison-Wiley, 1980.

[2] R. Brayton, G.D. Hachtel, L. Hemachandra, A.R. Newton & A.L. Sangiovanni-Vincentelli, "A Comparision of Logic Minimization Strategies Using ESPRESSO. An PLA Program Package for Partitioned Logic Minimization," *Proc. Int. Symp. on Cir. and Syst.* (May 1982).

[3] E. Morreale, "Recursive Operators for Prime Implicant and Irredundant Normal Form Determination," *IEEE T. Computer* C-19, No.6 (June 1970).

[4] R. Brayton, G.D. Hachtel, C.T. McMullen & A.L. Sangiovanni-Vincentelli, "Logic Minimization Algorithm for VLSI Synthesis," *Kluwer Academic Publishers* (1984).

[5] A. Weinberger, "Large Scale Integration of MOS Complex Logic: A Layout Method ," *IEEE J. Solid State Circuits* SC-2 (Dec. 1967), 182–190.

[6] Kohavi, *Switching and Finite Automata Theory* , McGraw-Hill, Reading, 1978.

[7] C. Leiserson, "Area Efficient Layouts," *21 Symp. on Fundation of Comp. Science IEEE* (1980).

[8] Das & Sheng, "On Detecting Total or Partial Symmetry of Switching Functions," *IEEE T. Computer* C-20 (1971), 352–355.

[9] Yao & Tang, "On Identification of Redundancy and Symmetry of Switching Functions," *IEEE T. Computer* C-20 (1971b), 1609–1613.

[10] M. Davio, J. Deschamps & M. Thayse, *Discrete and Switching Functions*, McGraw-Hill, Reading, 1978.