A Design Method for AND-OR-EXOR Three-Level Networks

Tsutomu Sasao
Department of Computer Science and Electronics
Kyushu Institute of Technology
Iizuka 820, Japan

Abstract

This paper presents a design method of AND-OR-EXOR three-level networks, where single two-input EXOR gate is used for each output. The network realizes an EXOR of two sum-of-products expressions (EX-SOP): $F_1 \oplus F_2$, where F_1 and F_2 are sum-of-products expressions (SOPs). The problem is to minimize the total number of different products in F_1 and F_2. A heuristic optimization algorithm is presented, and the experimental results are shown. The algorithm uses output phase optimized SOPs and ESOPs (exclusive-or sum-of-products) as inputs.

Index terms: AND-EXOR, ESOP, Reed-Muller expression, logic minimization, output phase optimization, three-level network, FPGA, decomposition.

1 Introduction

Most logic synthesis systems use AND and OR gates as basic elements. However, arithmetic functions are efficiently realized by using EXOR gates as well as AND and OR gates. For 5-variable functions, on the average, SOPs (sum-of-products expressions) require 7.46 products while ESOPs (exclusive-or sum-of-products expressions) require 6.16 products. To realize an arbitrary function, an SOP requires 16 products, whereas an ESOP requires only 9 products. For example, to represent $f = x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5$, an SOP requires 16 products, while an ESOP requires only 5 products. However, there exist functions whose ESOPs require more products than SOPs. For example, $f = x_1x_2 \vee x_3x_4 \vee \cdots \vee x_{2n-1}x_{2n}$ require n products in an SOP, while $2^n - 1$ products in an ESOP. So, we have to choose the better expressions according to the functions. This paper presents a design method for AND-OR-EXOR three-level networks, where single two-input EXOR gate is used for each output (Fig. 1.1). Such a network can be realized by an AND-OR-EXOR PLA shown in Fig. 1.2.

An AND-OR-EXOR network realizes an EXOR of two SOPs (EX-SOP): $F = F_1 \oplus F_2$, where F_1 and F_2 are SOPs. We consider the problem to minimize total number of products in F_1 and F_2. Note that F_2 can be a constant 0 or a constant 1. Thus, an optimized EX-SOP requires not more products than SOPs. In the case of multiple-output functions, PLAs for EX-SOP never require more products than output phase optimized PLAs (Fig. 1.3). The AND-OR-EXOR architecture efficiently realizes adders [21, 13]. In the past, design methods for EX-SOP were considered [7, 1, 19], but no efficient algorithms were presented.

Table 1.1: Number of products in various expressions.

t	SOP	ESOP	EX-SOP
0	1	1	1
1	81	81	81
2	1804	2268	2316
3	13472	21744	22896
4	28904	37530	37634
5	17032	3888	2608
6	3704	24	
7	512		
8	26		
m	4.13	3.66	3.62

Table 1.1 compares the numbers of 4-variable functions that require t products. To realize an arbitrary function, an SOP requires 8 products while an EX-SOP requires at most 5 products. On the average, EX-SOPs require fewer products than ESOPs [4].

Because two-input EXORs are faster and less expensive than EXORs with more inputs, the networks based on EX-SOPs are quite desirable. AND-OR-EXOR three level networks, where the output EXOR gates may have unlimited fan-in were considered in [17]. AND-OR-AND three-level networks, where the outputs AND gates have unlimited fan-in were considered in [14]. AND-OR-AND three-level networks, where the outputs AND gates have only two inputs were considered in [10].

2 Definitions and Basic Properties

This section introduce a minimization method for ESOPs, as well as the output phase optimization.

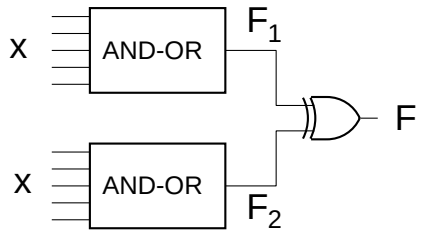

Figure 1.1: AND-OR-EXOR three-level network.

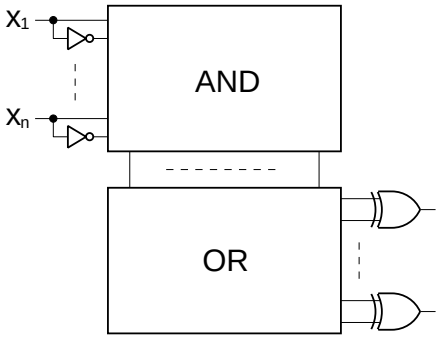

Figure 1.2: AND-OR-EXOR PLA.

2.1 ESOP minimization

For the minimization of logical expressions, we use the rule $1 \vee 1 = 1$ in SOPs, while $1 \oplus 1 = 0$ in ESOPs. This implies that in a Karnaugh map for an ESOP, 1-cells must be covered by an odd number of loops.

Example 2.1 *Consider the function:* $f = \bar{z}(x \vee y \vee \bar{w}) \vee z\bar{x}\bar{y}w$.
1) AND-OR realization.
Fig. 2.1 shows the SOP for f: $\bar{z}x \vee \bar{z}y \vee \bar{z}\bar{w} \vee z\bar{x}\bar{y}w$. *Fig. 2.2 shows the AND-OR two-level network.*
2) AND-EXOR realization.
Fig. 2.3 shows the ESOP for the same function: $\bar{z} \oplus \bar{x}\bar{y}w$. *Note that the 1-cell for* $\bar{x}\bar{y}\bar{z}w$ *is covered by two loops. Fig. 2.4 shows the AND-EXOR two-level network.* *(End of Example)*

When an ESOP is converted into an SOP, the number of products often increases. However, some ESOPs can be converted into SOPs without increasing the number of products.

Definition 2.1 *Let* $F = p_1 \vee p_2 \vee \cdots \vee p_k$ *be an SOP. F is said to be a disjoint SOP (DSOP) iff* $p_i \cdot p_j = 0$ $(i \neq j)$.

Lemma 2.1 *Let* $F_1 = p_1 \vee p_2 \vee \cdots \vee p_k$ *and* $F_2 = p_1 \oplus p_2 \oplus \cdots \oplus p_k$. *If* $p_i \cdot p_j = 0$ $(i \neq j)$, *then* F_1 *and* F_2 *represents the same function.*

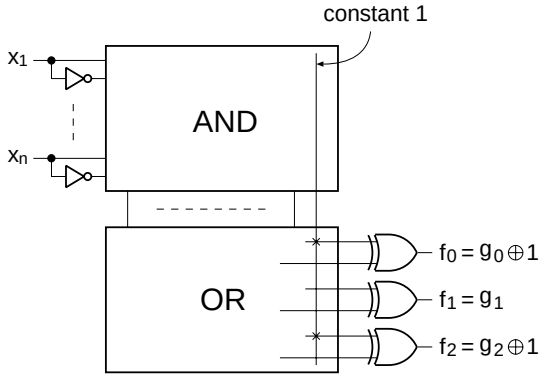

Figure 1.3: Output phase optimization technique for AND-OR-EXOR PLA.

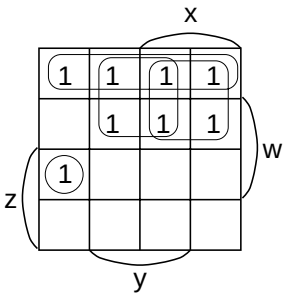

Figure 2.1: Karnaugh map for SOP.

This means that a disjoint SOP can be converted into an ESOP without changing the number of products. The next lemma shows the case when a given ESOP can be converted into an EX-SOP.

Lemma 2.2 *Let a function f be represented by an ESOP:* $F_1 = (p_1 \oplus p_2 \oplus \cdots \oplus p_k) \oplus (q_1 \oplus q_2 \oplus \cdots \oplus q_k)$. *If* $p_i \cdot p_j = 0$ $(i \neq j)$ *and* $q_k \cdot q_l = 0$ $(k \neq l)$, *then f can be represented by the EX-SOP:* $F_2 = (p_1 \vee p_2 \vee \cdots \vee p_k) \oplus (q_1 \vee q_2 \vee \cdots \vee q_k)$.

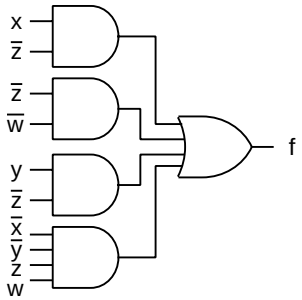

Figure 2.2: AND-OR two-level network.

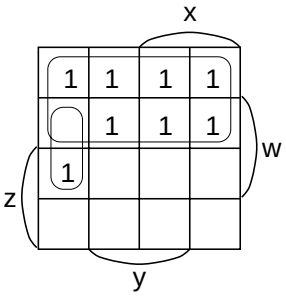

Figure 2.3: Karnaugh map for ESOP.

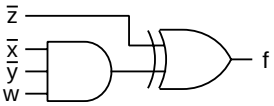

Figure 2.4: AND-EXOR two-level network.

Example 2.2 *Let an ESOP of the function f be $F_1 = (\bar{y}\bar{w} \oplus \bar{x}yw) \oplus (x\bar{z} \oplus \bar{x}y\bar{z})$. Since $(\bar{y}\bar{w})(\bar{x}yz) = 0$ and $(x\bar{z})(\bar{x}z\bar{y}) = 0$, f can be represented by the EX-SOP: $F_2 = (\bar{y}\bar{w} \vee \bar{x}yw) \oplus (x\bar{z} \vee \bar{x}\bar{y}\bar{z})$ (Fig. 2.5). The AND-OR-EXOR three-level realization is shown in Fig. 2.6. (End of Example)*

In the previous example, the given ESOP was converted into an EX-SOP without increasing the number of products. However, for many ESOPs, converting into EX-SOPs will increase the number of products. For example, to convert $F = x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_6$ into EX-SOP, we have to increases the number of products. In Section 3, we will consider a method to convert ESOPs to EX-SOP that adds as few products as possible.

2.2 Output Phase Optimization

To realize a function, in many cases, we have an option to realize f (the positive phase function) or $\bar{f}$ (the negative phase function) [6]. Consider the function $f = x_1x_2 \vee x_3x_4 \vee \cdots \vee x_{2n-1}x_{2n}$. The SOP for f requires n products, while the SOP for $\bar{f}$ requires 2^n products, since $\bar{f} = (\bar{x}_1 \vee \bar{x}_2)(\bar{x}_3 \vee \bar{x}_4) \cdots (\bar{x}_{2n-1} \vee \bar{x}_{2n})$. Thus, this function require fewer products in the positive phase SOP, than the negative phase SOP. In EX-SOPs, we can choose the better phase of the function since $F = G \oplus c$, where c is 0 or 1. If $c = 0$, then G is the positive phase SOP of f, and if $c = 1$ then G is the negative phase SOP of f. For m-output case, 2^m different combinations of phases exist. To find a set of phases that minimizes the total number of products is called an output phase optimization [12].

Definition 2.2 *Given a PLA for m-output function $f_0, f_1, \ldots, f_{m-1}$. The PLA that realizes either f_i or $\bar{f}_i$ for $i = 0, 1, \ldots, m-1$ is called an output phase optimized PLA if the PLA requires the minimum number of products.*

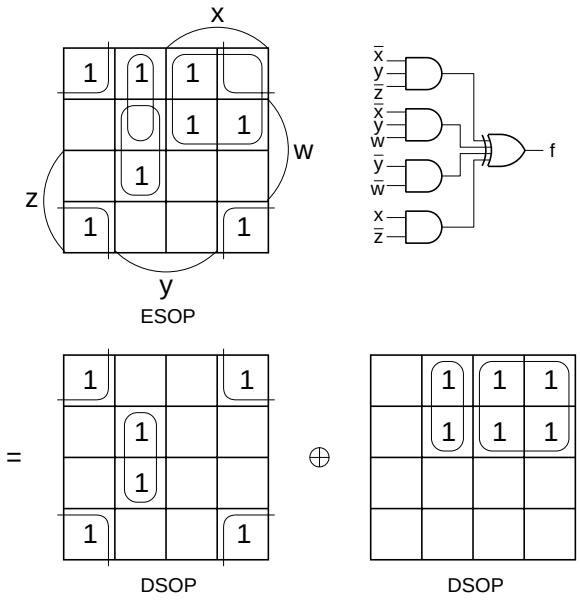

Figure 2.5: Decomposition of an ESOP.

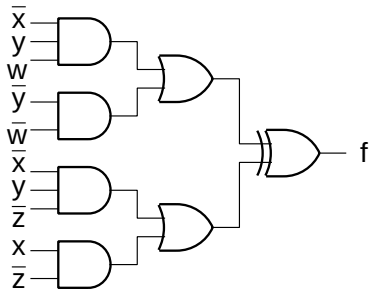

Figure 2.6: AND-OR-EXOR network.

Heuristic algorithms for the output phase optimization are available [12, 22].

Example 2.3 *Consider the two-bit adder (ADR2):*

$$\begin{array}{cccc} & & x_1 & x_0 \\ +) & & y_1 & y_0 \\ \hline & s_2 & s_1 & s_0 \end{array}$$

An ordinary PLA for ADR2 requires 11 products. However if $\bar{s}_2$, the complement of s_2, is realized instead of s_2, the PLA requires only 9 products. (End of Example)

3 Optimization of EX-SOP

3.1 Generation of EX-SOP

We assume that an output phase optimized SOP and an optimized ESOP are available for a given multiple-output function. As shown in the introduction, some functions require fewer products in ESOPs, and others require fewer products in SOPs. If the ESOP requires fewer products than the SOP, then we use the ESOP as an initial solution, otherwise we use the SOP as an initial solution.

All the algorithms use the following modifications:

$$F_1^* \leftarrow SOP(F_1 \oplus c),$$
$$F_2^* \leftarrow SOP(F_2 \oplus c),$$

where $SOP(F_i \oplus c)$ denotes an optimized SOP for $F_i \oplus c$ $(i = 1, 2)$. The function represented by EX-SOP is invariant under the above modification, since $F_1^* \oplus F_2^* = (F_1 \oplus c) \oplus (F_2 \oplus c) = F_1 \oplus F_2 = f$. If the number of products is reduced, we adopt this modification.

Algorithm 3.1 *(Generation of EX-SOP) Suppose that the ESOP requires fewer products than the SOP. Then, we use the ESOP as an initial solution, and decompose it into two ESOPs: $F_1 \oplus E_2$, where F_1 is a disjoint ESOP and E_2 is an ESOP consisting of other products than F_1. In this case, we choose F_1 so that the number of products in E_2 is minimal. Then, we convert E_2 into an SOP and obtain F_2, we denote this operation as follows: $F_2 \leftarrow SOP(E_2)$. In many cases, F_2 has more products than E_2. In this step, we have obtained an EX-SOP: $F_1 \oplus F_2$, where F_1 is a DSOP and F_2 is an SOP derived from E_2. From here, we will reduce the number of products in $F_1 \oplus F_2$. For each product c_i in E_2, we obtain*

$$F_1^* \leftarrow SOP(F_1 \oplus c_i), and$$
$$F_2^* \leftarrow SOP(F_2 \oplus c_i).$$

If the total number of products in F_1^* and F_2^* is reduced, then we modify F_1 and F_2 as follows:

$$F_1 \leftarrow F_1^*, and$$
$$F_2 \leftarrow F_2^*.$$

Example 3.1 *Suppose that $f = x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_6$ is the function to be represented as an EX-SOP. Note that the ESOP for f requires 6 products, while the SOP requires 32 products. The initial solution is $F_1 \oplus E_2$, where $F_1 = x_1$, and $E_2 = x_2 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_6$. We convert E_2 into SOP: $F_2 \leftarrow SOP(E_2)$. Note that F_2 requires 16 products. In this step, we obtained EX-SOP: $F_1 \oplus F_2$, where the total number of products is $1 + 16 = 17$. Let $c_1 = x_2$ be a product of E_2. We obtain the modified functions as follows:*

$$F_1^* \leftarrow SOP(F_1 \oplus x_2) = SOP(x_1 \oplus x_2), and$$
$$F_2^* \leftarrow SOP(F_2 \oplus x_2) = SOP(x_3 \oplus x_4 \oplus x_5 \oplus x_6).$$

Here, F_1^* requires 2 products and F_2^* requires 8 products. So, the total number of products is decreased to $2 + 8 = 10$. Because the modified functions require fewer products, we replace F_1 and F_2 with the modified ones:

$$F_1 \leftarrow SOP(x_1 \oplus x_2), and$$
$$F_2 \leftarrow SOP(x_3 \oplus x_4 \oplus x_5 \oplus x_6).$$

In the next, let $c_2 = x_3$ be a product of E_2. We modify the functions as follows:

$$F_1^* \leftarrow SOP(F_1 \oplus x_3) = SOP(x_1 \oplus x_2 \oplus x_3), and$$
$$F_2^* \leftarrow SOP(F_2 \oplus x_3) = SOP(x_4 \oplus x_5 \oplus x_6).$$

Here, both F_1^* and F_2^* require 4 products. So, the total number of products is decreased to $4 + 4 = 8$. Also, in this case, modified functions require fewer products, so we replace F_1 and F_2 with the modified ones:

$$F_1 \leftarrow SOP(x_1 \oplus x_2 \oplus x_3), and$$
$$F_2 \leftarrow SOP(x_4 \oplus x_5 \oplus x_6).$$

Next, we do the same procedure for $c_3 = x_4$. However, the total number of products increases. So, we do not modify the functions, and stop the procedure. Thus, we have decomposed the function $f = x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_6$ into $F_1 \oplus F_2$, where $F_1 = SOP(x_1 \oplus x_2 \oplus x_3)$ and $F_2 = SOP(x_4 \oplus x_5 \oplus x_6)$. (End of Example)

The above procedure often reduces the number of products.

3.2 Iterative Improvement

Suppose that we have an EX-SOP for a function $f : F_1 \oplus F_2$. Such EX-SOPs are obtained by Algorithm 3.1, or an output phase optimization algorithm. In the latter case, F_1 is the output phase optimized SOP and, F_2 is a constant (0 or 1 for a single output function, and a binary vector for a multiple-output function). In this part, we will present two algorithms to reduce the total number of products in EX-SOPs. Both algorithms are based on the modification of functions.

$$F_1^* \leftarrow SOP(F_1 \oplus c_i), and$$
$$F_2^* \leftarrow SOP(F_2 \oplus c_i).$$

However, the above modification is very time consuming, since it requires minimization of logical expressions. To reduce the computation time, Algorithm 3.2 finds a product c_j that does not increase the number of products. Before showing the algorithm, we will illustrate it by using a simple example.

Example 3.2 *Consider the EX-SOP (Fig. 3.1(a)) $F = F_1 \oplus F_2$, where*

$$F_1 = \bar{x}\bar{z}\bar{w} \vee \bar{x}\bar{y}\bar{z}w \vee x\bar{y}zw \vee xz\bar{w}, \quad and$$
$$F_2 = xy\bar{z}w \vee \bar{x}yzw.$$

In F_1, the loops for $\bar{x}\bar{z}\bar{w}$ and $\bar{x}\bar{y}\bar{z}w$ can be merged into a larger ones if $\bar{x}y\bar{z}w$ was the 1-cell in the map (the cell denoted by the dotted loop in Fig. 3.1(b)). So, let $c_1 = \bar{x}y\bar{z}w$, and we modify the functions as follows:

$$F_1^* \leftarrow SOP(F_1 \vee c_1), and$$
$$F_2^* \leftarrow SOP(F_2 \vee c_1).$$

In this step, SOPs for F_1^* and F_2^* are simplified as shown in the maps of Fig. 3.1(c). So, we adopt this

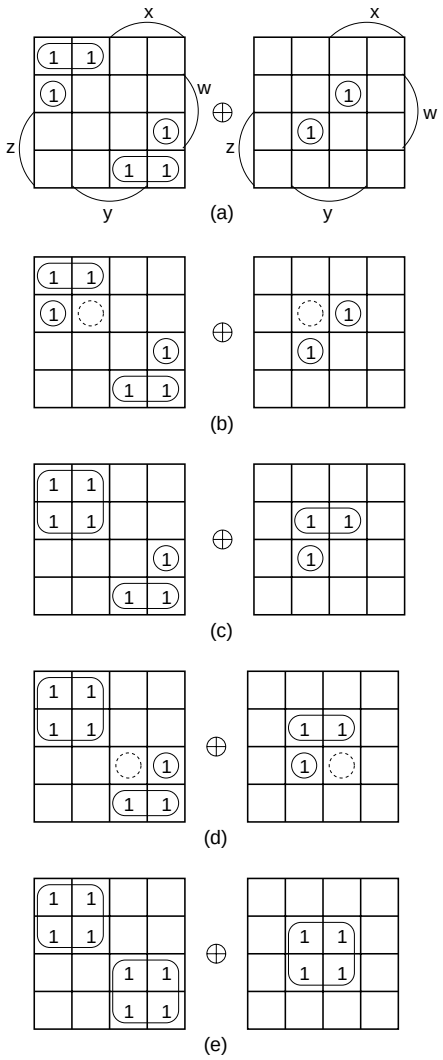

Figure 3.1: Iterative improvement I.

modification:

$$F_1 = \bar{x}\bar{z} \vee x\bar{y}zw \vee xz\bar{w},$$
$$F_2 = y\bar{z}w \vee \bar{x}yzw.$$

In the map in Fig. 3.1(d), the loops for $x\bar{y}zw$ and $xz\bar{w}$ can be merged into a larger one if $xyzw$ was the 1-cell (the cell denoted by the dotted loop in Fig. 3.1(d)). So, let $c_2 = xyzw$, and we modify the function as follows:

$$F_1^* \leftarrow SOP(F_1 \vee c_2), and$$
$$F_2^* \leftarrow SOP(F_2 \vee c_2).$$

In this step, SOPs for F_1^* and F_2^* are simplified as shown in Fig. 3.1(e). So, we adopt this modification:

$$F_1 = \bar{x}\bar{z} \vee xz,$$
$$F_2 = yw.$$

(End of Example)

To find a pair of products that can be merged into a larger product is easy. Such a pair of products satisfies the "reshaping condition" [9]. We will use the product c such that $F_1 \cdot c = 0$ and $F_2 \cdot c = 0$. This means the modification of the function will not increase the number of products.

Algorithm 3.2 *(Iterative Improvement I)*

1. *Let the EX-SOP be $F = F_1 \oplus F_2$.*
2. *Find a pair of cubes c_1 and c_2 in F_1 that satisfies the reshaping condition:*

$$c_1 = x_j^* e$$
$$c_2 = x_i^* \overline{x}_j^* e,$$

where $x_i^* = x_i$ or $\bar{x}_i$, $x_j^* = x_j$ or $\bar{x}_j$, and e does not contain variable x_i nor x_j.
Let $c_3 = \overline{x}_i^* \cdot \overline{x}_j^* \cdot e$. If $c_3 \cdot F_2 = 0$, then

$$F_1^* \leftarrow SOP(F_1 \oplus c_3), \quad and$$
$$F_2^* \leftarrow SOP(F_2 \oplus c_3).$$

Example 3.3 *Let $F_1 = \bar{x}\bar{z}\bar{w} \vee \bar{x}\bar{y}\bar{z}w \vee x\bar{y}zw \vee xz\bar{w}$. Let*

$$c_1 = \bar{w}(\bar{x}\bar{z}) \quad and$$
$$c_2 = \bar{y}w(\bar{x}\bar{z}).$$

Then, c_1 and c_2 satisfy the reshaping condition. So, we generate $c_3 = yw(\bar{x}\bar{z})$. Also, let

$$c_4 = \bar{w}(xz) \quad and$$
$$c_5 = \bar{y}w(xz).$$

Then, c_4 and c_5 satisfy the reshaping condition. So, we generate $c_6 = yw(xz)$. *(End of Example)*

The next algorithm is more powerful, but more time consuming. First, we will illustrate the idea by a simple example.

Example 3.4 *Consider the EX-SOP shown in Fig. 3.2(a) and (b): $F = F_1 \oplus F_2$, where*

$$F_1 = x\bar{z} \vee y\bar{z} \vee \bar{z}w,$$
$$F_2 = xyw.$$

In F_1 of Fig. 3.2(a), if the cell for $c = \bar{x}\bar{y}\bar{z}\bar{w}$ was the 1-cell, then the map would be simplified. So, we modify the functions as follows:

$$F_1^* \leftarrow SOP(F_1 \vee c), and$$
$$F_2^* \leftarrow SOP(F_2 \vee c).$$

In this step, SOPs for F_1^* and F_2^* are simplified as shown in the map of Fig. 3.2(c) and (d). So, we adopt this modification:

$$F_1 = \bar{z},$$
$$F_2 = xyw \vee \bar{x}\bar{y}\bar{z}\bar{w}.$$

(End of Example)

In Example 3.4, we assumed that the product $c = \bar{x}\bar{y}\bar{z}\bar{w}$ is the 1-cell. Such a product can be found from the cover for $\overline{F_1 \vee F_2}$. The next example will illustrate this.

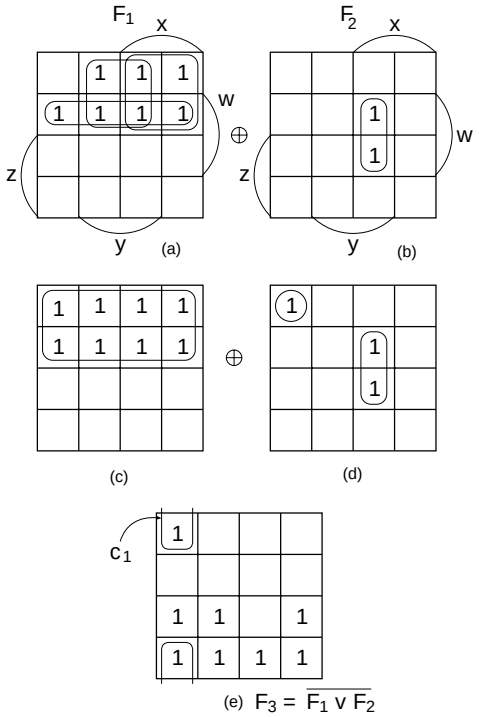

Figure 3.2: Iterative improvement II.

Example 3.5 *Consider the function in Example 3.4. $F_3 \leftarrow SOP(\overline{F_1 \vee F_2})$ is shown in Fig. 3.2(e). Let $c_1 = \bar{x}\bar{y}\bar{w}$. After the operation, $F_A \leftarrow expand(F_1, DC = \bar{x}\bar{y}\bar{w})$, we have $F_A = \bar{z}$. Because the number of the cubes in F_1 is reduced, we compute $c_1 = F_A \cdot \overline{F}_1 = \bar{z}(\overline{x\bar{z} \vee y\bar{z} \vee \bar{z}w}) = \bar{x}\bar{y}\bar{z}\bar{w}$. Because c_1 is a cube, then we modify the function as follows:*

$$F_1 \leftarrow SOP(F_1 \oplus \bar{x}\bar{y}\bar{z}\bar{w})$$
$$F_2 \leftarrow SOP(F_2 \oplus \bar{x}\bar{y}\bar{z}\bar{w}).$$

For other cubes in F_3, we cannot reduce the number of products in F_1, so we stop the algorithm. *(End of Example)*

Thus, we have the following algorithm.

Algorithm 3.3 *(Iterative Improvement II)*

1. *Let the EX-SOP be $F = F_1 \oplus F_2$.*
2. *$F_3 \leftarrow SOP(\overline{F_1 \vee F_2})$.*
3. *For each cube c_i in F_3, do steps 4 and 5. If all the cubes are checked, then return.*
4. *$F_A \leftarrow expand(F_1, DC = c_i)$. (Expand the cubes in F_1, using c_i as a don't care).*
5. *If (the number of cubes in F_1 is reduced) then*
   $c_1 \leftarrow F_A\overline{F}_1$

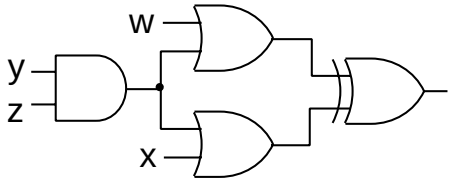

Figure 3.3: Sharing of a product.

else
   choose the next cube c_i in F_3 and go to 4.
endif.
If (c_1 is a cube) then
   $F_1 \leftarrow SOP(F_1 \oplus c_1)$,
   $F_2 \leftarrow SOP(F_2 \oplus c_1)$, and
   go to 2.
else
   choose the next cube c_i in F_3, and go to 4.
endif.

In Step 5, if $c_1 = F_A \cdot \overline{F}_1$ is not a cube, then the number of products in F_2 can increase by more than one. Thus, the reduction on the number of products in F_1 is usually offset by the increase of the number of products in F_2. Therefore, we discard c_1 if it is not a cube.

3.3 Final Optimization

In Algorithms 3.1, 3.2, and 3.3, we converted an ESOP into two SOPs: $F_1 \oplus F_2$, and simplified each SOP independently. However, we can often reduce the number of products by using multiple-output optimization. The following example illustrates this.

Example 3.6 *Let the ESOP of the given function be $G_1 = w \oplus yz\bar{w} \oplus x \oplus \bar{x}yz$. We can decompose this into an EXOR of two ESOPs: $G_2 = (w \oplus yz\bar{w}) \oplus (x \oplus \bar{x}yz)$. Because the products in each ESOP are mutually disjoint, G_2 can be converted into an EX-SOP: $G_3 = (w \vee yz\bar{w}) \oplus (x \vee \bar{x}yz)$. After simplification of two SOPs, we have $G_4 = (w \vee yz) \oplus (x \vee yz)$. Note that the product yz exists in both SOPs. Thus, we have the network in Fig. 3.3. This means that simultaneous minimization of two SOPs is useful.* *(End of Example)*

Note that simultaneous optimization of two SOPs F_1 and F_2 can be done by using standard SOP minimizers [2, 9, 11, 12].

4 Design of Adders

Let ADRn be the n-bit adder without carry input as follows:

$$\begin{array}{cccccc} & x_{n-1} & x_{n-2} & \cdots & x_0 \\ + & y_{n-1} & y_{n-2} & \cdots & y_0 \\ \hline s_n & s_{n-1} & s_{n-2} & \cdots & s_0 \\ & c_{n-1} & c_{n-2} & \cdots & c_0, \end{array}$$

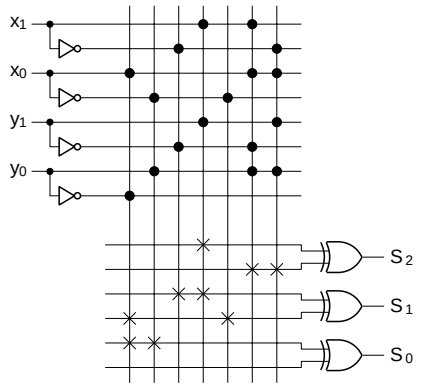

Figure 4.1: AND-OR-EXOR PLA for ADR2.

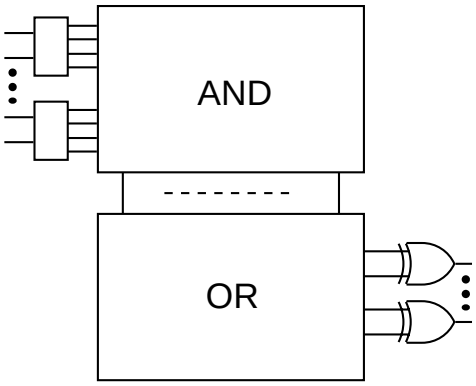

Figure 4.2: AND-OR-EXOR PLA with two-bit decoders.

where s_i's are sums and c_i's are carries. Note that $s_n = c_{n-1}$.

In this section, we consider the number of products to realize adders by EX-SOPs.

4.1 AND-OR-EXOR PLA

For ADRn, we have the following relations:

$$s_i = (x_i \oplus y_i) \oplus c_{i-1},$$
$$c_i = x_i y_i \oplus c_{i-1}(x_i \oplus y_i),$$
$$s_0 = x_0 \oplus y_0, \quad and$$
$$c_0 = x_0 y_0.$$

Example 4.1 *Let us realize ADR2 by AND-OR-EXOR PLA (Fig. 1.2). By definition, we have*

$$\begin{aligned} s_0 &= x_0 \oplus y_0 = x_0\bar{y}_0 \vee \bar{x}_0 y_0 \\ s_1 &= x_1 \oplus y_1 \oplus c_0 \\ &= (x_1 \oplus y_1) \oplus c_0 = (\overline{x_1 \oplus y_1}) \oplus \overline{x_0 y_0} \\ &= (x_1 y_1 \vee \bar{x}_1 \bar{y}_1) \oplus (\bar{x}_0 \vee x_0 \bar{y}_0) \\ s_2 &= c_1 = x_1 y_1 \oplus x_0 y_0 (x_1 \oplus y_1) \\ &= x_1 y_1 \oplus (x_0 y_0 x_1 \bar{y}_1 \vee x_0 y_0 \bar{x}_1 y_1) \end{aligned}$$

Note that $x_0\bar{y}_0$ in s_0 and s_1, and $x_1 y_1$ in s_1 and s_2 can be shared. Thus, the total number of different products is 7. Fig. 4.1 shows the PLA realizing ADR2. (End of Example)

4.2 AND-OR-EXOR PLA with 2-bit decoders

In an AND-OR-EXOR PLA (Fig. 1.2), replace the inverters with two-bit decoders [12, 17], and we have an AND-OR-EXOR PLA with two-bit decoders (Fig. 4.2).

Theorem 4.1 *An AND-OR-EXOR PLA with two-bit decoders realizes ADRn by using at most $(n^2 + n + 2)/2$ products.*

(Proof) Let $X_i = (x_i, y_i)$ $(i = 0, 1, \ldots, n-1)$ be the partition of the input variables. By definition, we have

$$\begin{aligned} s_i &= (x_i \oplus y_i) \oplus c_{i-1} = X_i^{\{01,10\}} \oplus c_{i-1} \\ c_i &= x_i y_i \oplus c_{i-1}(x_i \oplus y_i) = X_i^{\{11\}} \oplus c_{i-1} X_i^{\{01,10\}} \\ s_0 &= x_0 \oplus y_0 = X_0^{\{01,10\}} \\ c_0 &= x_0 y_0 = X_0^{\{11\}}. \end{aligned}$$

Note that $X_i^{\{01,10\}} = 1$ iff $X_i = (0,1)$ or $(1,0)$, and $X_i^{\{11\}} = 1$ iff $X_i = (1,1)$, and so on.

Let t_i be the number of products in EX-SOP for c_i. Note that $t_0 = 1$, and $t_{i+1} = t_i + 1$. From these, we have $t_i = i + 1$. Also, t_i denotes the number of products in EX-SOP for s_i. For the most significant two bits, we have

$$\begin{aligned} s_n &= c_{n-1} = X_{n-1}^{\{11\}} \oplus c_{n-2} X_{n-1}^{\{01,10\}} \\ &= X_{n-1}^{\{00,01,10\}} \oplus (\bar{c}_{n-2} \vee X_{n-1}^{\{00,11\}}), \quad and \\ s_{n-1} &= c_{n-2} \oplus X_{n-1}^{\{01,10\}} \\ &= \bar{c}_{n-2} \oplus X_{n-1}^{\{00,11\}} \end{aligned}$$

Thus, the products for $\bar{c}_{n-2}$ and $X_{n-1}^{\{00,11\}}$ can be shared. So, the total number of products is

$$\sum_{i=0}^{n-1} t_i + 1 = 1 + \sum_{i=0}^{n-1} (i+1) = (n^2 + n + 2)/2.$$

(Q.E.D.)

Example 4.2 *Let us realize ADR2 by an AND-OR-EXOR PLA with two-bit decoders (Fig. 4.3). By definition, we have*

$$\begin{aligned} s_0 &= x_0 \oplus y_0 = X_0^{\{01,10\}} \\ s_1 &= (x_1 \oplus y_1) \oplus c_0 = X_1^{\{01,10\}} \oplus X_0^{\{11\}} \\ &= X_1^{\{00,11\}} \oplus X_0^{\{00,01,10\}} \\ s_2 &= c_1 = x_1 y_1 \oplus x_0 y_0 (x_1 \oplus y_1) \\ &= X_1^{\{11\}} \oplus X_1^{\{01,10\}} X_0^{\{11\}} \\ &= X_1^{\{00,01,10\}} \oplus (X_1{}^{\{00,11\}} \vee X_0{}^{\{00,01,10\}}). \end{aligned}$$

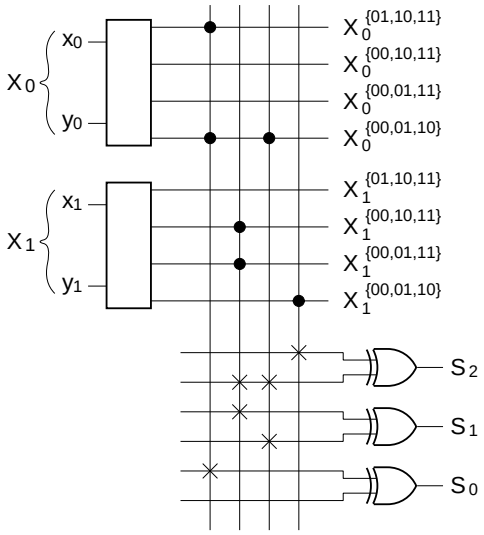

Figure 4.3: AND-OR-EXOR PLA with 2-bit decoders for ADR2.

Table 4.1: Number of products for n-bit adder.

		n=4
AND-OR	$6 \cdot 2^n - 4n - 5$	75
AND-EXOR	$2^{n+1} - 1$	31
AND-OR with 2-bit decoders	$n^2 + 1$	17
same above OPTOUT	$n^2 - n + 2$	14
AND-EXOR with 2-bit decoders	$(n^2 + n + 2)/2$	11
AND-OR-EXOR with 2-bit decoders	same as above	

Note that $X_1^{\{00,11\}}$ and $X_0^{\{00,01,10\}}$ in s_1 and s_2 can be shared. Thus, the total number of different products is four. Fig. 4.3 shows the PLA realizing ADR2. (End of Example)

In this way, we can efficiently realize the adders by using AND-OR-EXOR PLAs with two-bit decoders [21]. Table 4.1 compares upper bounds on the number of products for various realizations.

5 Experimental Results

5.1 Randomly generated functions

We generated logic functions with $n = 4$ to $n = 10$ by using a pseudo-random generator, where each function has 2^{n-1} true minterms. Table 5.1 compares the number of products of SOPs, ESOPs and EX-SOPs. For these functions, the ESOPs require the fewest products among the three.

Table 5.1: Number of products for randomly generated functions.

n	SOP	ESOP	EX-SOP	γ
4	4	3	3	0.833
5	6	5	5	0.909
6	13	10	10	0.911
7	24	19	20	0.889
8	46	38	45	1.036
9	86	64	80	0.979
10	167	142	163	1.023

Table 5.2: Number of products for arithmetic functions.

	n	m	SOP	SOP optu	ESOP	EX-SOP	γ
ard4	8	5	75	61	31	37	0.611
inc8	8	9	37	36	15	15	0.551
log8	8	8	123	121	99	116	1.257
mlp4	8	8	121	112	67	109	1.201
nrm4	8	5	120	103	71	93	0.960
rdm8	8	8	76	76	32	54	0.947
rot8	8	5	57	48	37	49	1.064
sqr8	8	16	182	175	112	176	1.467
wgt8	8	4	255	186	59	135	0.635

The last column of Table 5.1 shows γ, the relative size of PLAs.

$$\gamma = \frac{\text{size of AND-OR-EXOR PLA}}{\text{size of AND-OR PLA}},$$

where

$$\begin{aligned} \text{size of AND-OR PLA} &= W_1(2n+m), \\ \text{size of AND-OR-EXOR PLA} &= W_2(2n+2m), \\ n &= \text{number of input variables}, \\ m &= \text{number of output functions}, \\ W_1 &= \text{number of products in SOP, and} \\ W_2 &= \text{number of products in EX-SOP.} \end{aligned}$$

Table 5.1 shows that PLA based realization of EX-SOP is not so attractive for randomly generated functions. These functions should be realized by random networks.

5.2 Arithmetic functions with small number of inputs

We optimized the expressions for arithmetic functions with 8-input. Table 5.2 compare the number of products. Also in this case, ESOPs require the fewest products. The column headed with SOP optu shows the number of products in output phase (near) optimized SOPs. In most cases, EX-SOPs require fewer products than the output phase optimized SOPs. The last column of Table 5.2 shows that PLA based realizations are efficient for adr4, inc8, and wgt8.

Table 5.3: Number of products for other functions.

NAME	n	m	SOP	SOP optu	ESOP	EX-SOP	γ
5xp1	7	10	63	62	33	47	1.057
9sym	9	1	87	72	53	73	0.883
apex5	117	88	1088	1088	398	870	1.018
clip	9	5	118	116	67	92	0.949
rd73	7	3	127	97	35	83	0.769
sao2	10	4	58	37	29	33	0.664
seq	41	35	350	346	259	234	0.869
t481	16	1	481	363	13	364	0.780

Table 5.4: Number of products for n-bit adders.

n	SOP	ESOP	EX-SOP	γ
2	8	7	7	1.114
	5	4	4	1.018
3	31	15	26	1.048
	10	8	8	1.000
4	75	31	37	0.611
	17	11	11	0.801
5	167	63	79	0.553
	26	17	17	0.805

The upper numbers show products in PLAs with 1-bit decoders.
The lower numbers show products in PLAs with 2-bit decoders.

5.3 Other benchmark functions

Table 5.3 shows the number of products for MCNC benchmark functions [23]. In this table, only the functions whose ESOPs require fewer products than SOPs are shown. For some functions, only the output phase optimizations are effective, and EX-SOP optimizations were not so effective. The last column of Table 5.3 shows that PLA based realizations are efficient for rd73, sao2 and t481. For apex5, it requires a larger PLA even if the number of products is reduced by 218 products. Thus, this function should be realized by a random network.

5.4 Adders

We designed ADRn, an n bit adder without carry inputs, for $n = 2$ to $n = 5$. Table 5.4 compares the number of products for SOPs, ESOPs, and EX-SOPs. The upper figures show the number of products of PLAs with 1-bit decoders, while the lower numbers show that of PLAs with 2-bit decoders. In the cases of PLAs with 2-bit decoders, ESOP and EX-SOPs require the same number of products. For adders, the PLA based realizations are efficient for adr8 and adr10.

5.5 Symmetric functions

We also minimized the expressions for symmetric functions.

Table 5.5: Number of products for symmetric functions $SB(n,k)$.

	SOP	ESOP	EX-SOP	γ
SB(8,1)	128	8	16	0.132
SB(8,2)	84	21	61	0.769
SB(8,3)	64	33	52	0.860
SB(8,4)	70	35	58	0.877
SB(8,5)	64	33	42	0.695
SB(9,1)	256	9	24	0.099
SB(9,2)	168	26	110	0.689
SB(9,3)	120	48	109	0.956
SB(9,4)	126	57	91	0.760

Definition 5.1 *$SB(n,k)$ is a symmetric function of n variables [15]. They are represented by an EXOR sum of all possible products with k positive literals:*

$$SB(n,k) = \sum \oplus x_{i1}x_{i2}\cdots x_{ik}.$$

For example, $SB(4,2) = x_1x_2 \oplus x_1x_3 \oplus x_1x_4 \oplus x_2x_3 \oplus x_2x_4 \oplus x_3x_4$, and $SB(n,1) = x_1 \oplus x_2 \oplus \cdots \oplus x_n$.

Table 5.5 compares the number of products for SOPs, ESOPs, and EX-SOPs. For these functions, EX-SOPs require many fewer products than SOPs. For $SB(n,k)$ functions, the PLA based realizations are efficient in most cases.

5.6 Comparison with other method

Malik-Harrison-Brayton considered a design method of AND-OR-AND three-level networks, where a single two-input AND gate is used for each output [10]. Their method decomposes a function into the form $F = F_1 \cdot F_2$, where F_1 and F_2 are SOPs. They developed an algorithm to reduce the total number of products. Unfortunately, the comparison is difficult, since their benchmark functions are not explicitly shown.

6 Conclusion and Comments

In this paper, we presented a design method for AND-OR-EXOR three-level networks (EX-SOPs). The algorithm uses ESOPs and output phase optimized SOPs as input data. EX-SOPs never require more products than SOPs, and often require fewer products. Experimental results show that the EX-SOPs for arithmetic functions, such as adders require many fewer products than SOPs. EX-SOPs also efficiently realize symmetric functions such as $SB(n,k)$. Algorithm 3.1 is not so time consuming, but reduces the number of products considerably. On the other hand, Algorithm 3.3 is time consuming, so it should be used only if it is necessary.

Dr. Masahiro Fujita pointed output that the exact optimization of EX-SOPs is formulated as an optimization of Boolean relation R:

$$f(\mathbf{x}) = 1 \Leftrightarrow (\mathbf{x},(0,1)) \in R \text{ or } (\mathbf{x},(1,0)) \in R$$

However, the current performance of the optimization tools for Boolean relations seems to be impractical for the functions with many inputs.

Up to $n = 4$, we have obtained exact minimum EX-SOPs for all functions [4]. An EX-SOP with n variables can be composed of two EX-SOPs with $(n-1)$ variables without increasing the total number of products [5]. An optimization tool for EX-SOP based on this approach is under development. Currently, we have no exact optimization algorithm for EX-SOPs with many inputs, so we cannot evaluate the quality of solutions of the heuristic algorithm. Experimental results show that many functions are efficiently realized by EX-SOPs. In this paper, for illustration, we used PLA architecture to implement EX-SOPs. However, in many cases, EX-SOPs are more efficiently realized by random networks as shown in Fig. 2.6.

Acknowledgments

The author started this research in 1986. Discussion with the late Prof. Besslich [1] was quite useful. This work was supported in part by a Grant in Aid for Scientific Research of the Ministry of Education, Science and Culture of Japan. The author is also grateful to Dr. Fujita and reviewers for their useful comments.

References

[1] P. Besslich, Private communication, Dec. 1989.

[2] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis,* Boston, MA. Kluwer Academic Publishers, 1984.

[3] D. Brand, and T. Sasao, "Minimization of AND-EXOR expressions using rewrite rules," *IEEE Transactions on Computers*, Vol. 42, No. 5, May 1993, pp. 568-576.

[4] D. Debnath, and T. Sasao, "Optimization of AND-OR-EXOR using table look up approach,"(in preparation).

[5] E. V. Dubrova, D. M. Miller, and J. C. Muzio, "Upper bounds on the number of products in AND-OR-XOR expansion of logic functions," *Electronics Letters,* 30th March, 1995, vol. 31, No. 7, pp. 541-542.

[6] H. Fleisher and L. I. Maissel, "An introduction to array logic," *IBM J. Res. & Develop.*, vol. 19, pp. 98-109, March 1975.

[7] H. Fleisher J. Giraldi, D. B. Martin, R. L. Phoenix, and M. A. Tavel, "Simulated annealing as a tool for logic optimization in a CAD environment," *ICCAD-85*, Nov. 1985, pp. 203-205.

[8] M. Helliwell and M. Perkowski,"A fast algorithm to minimize multi-output mixed-polarity generalized Reed-Muller forms," *25th DAC*, pp. 427-432, 1988.

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

[10] A. A. Malik, D. Harrison, and R. K. Brayton, "Three-level decomposition with application to PLDs," *ICCD-1991*, pp.628-633, Oct. 1991.

[11] R. L. Rudell and A. L. Sangiovanni-Vincentelli, "Multiple-valued minimization for PLA optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 6, No. 5, Sept. 1987, pp. 727-750.

[12] T. Sasao, "Input variable assignment and output phase optimization of PLA's," *IEEE Trans. Comput.*, Vol. C-33, No. 10, pp. 879-894, Oct. 1984.

[13] T. Sasao, *Programmable Logic Array: How to use and how to make*, (in Japanese) Nikkan Kougyo Publishing Co., May 1985.

[14] T. Sasao, "On the complexity of three-level logic circuits," *International Workshop on Logic Synthesis,* Research Triangle Park, North Carolina, May 1989.

[15] T. Sasao and P. Besslich, "On the complexity of MOD-2 sum PLA's," *IEEE Trans. Comput.*, vol. 39, No. 2, pp. 262-266, Feb. 1990.

[16] T. Sasao, "EXMIN2: A simplification algorithm for exclusive-OR sum-of-products expressions for multiple-valued input two-valued output functions, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, No. 5, May 1993, pp. 621-632.

[17] T. Sasao (ed.), *Logic Synthesis and Optimization*, Kluwer Academic Publishers, 1993.

[18] M.S. Schmookler, "Design of large ALUs using multiple-PLA macros," *IBM Journal of Research and Development*, Vol. 24, No. 1, pp. 2-14, Jan. 1980.

[19] K. Shu, H. Yasuura, and S. Yajima, "Optimization of PLDs with output parity gates," (in Japanese) *National Convention, Information Processing Society of Japan*, March 1985.

[20] N. Song and M. A. Perkowski, "EXORCISM-MV-2: Minimization of exclusive sum of products expressions for multiple-valued input incompletely specified functions," *International Symposium on Multi-valued Logic*, May 1993, pp. 132-137.

[21] A. Weinberger, "High speed programmable logic array adders," *IBM Journal of Research and Development*, vol. 23, pp. 163-178, March 1979.

[22] C-L. Wey and T-Y Chan, "An efficient output phase assignment for PLA minimization," *IEEE Transactions on Computer-Aided Design*, vol. 9, No. 1, pp. 1-7, Jan. 1990.

[23] S. Yang,"Logic synthesis and optimization benchmark user guide, version 3.0", *MCNC,* Jan. 1991.