PAPER *Special Section on Recent Advances in Circuits and Systems*

Output Phase Optimization for AND-OR-EXOR PLAs with Decoders and Its Application to Design of Adders

Debatosh DEBNATH[†a)], ***Nonmember and*** **Tsutomu SASAO**[††b)], ***Member***

SUMMARY This paper presents a design method for three-level programmable logic arrays (PLAs), which have input decoders and two-input EXOR gates at the outputs. The PLA realizes an EXOR of two sum-of-products expressions (EX-SOP) for multiple-valued input two-valued output functions. We developed an output phase optimization method for EX-SOPs where some outputs of the function are minimized in the complemented form and presented techniques to minimize EX-SOPs for adders by using an extension of Dubrova-Miller-Muzio's AOXMIN algorithm. The proposed algorithm produces solutions with a half products of AOXMIN-like algorithm in 250 times shorter time for large adders with two-valued inputs. We also proved that an n-bit adder with two-valued inputs requires at most $3 \cdot 2^{n-2} + 7n - 5$ products in an EX-SOP while it is known that a sum-of-products expression (SOP) requires $6 \cdot 2^n - 4n - 5$ products.
key words: *three-level network, logic minimization, adder, programmable logic*

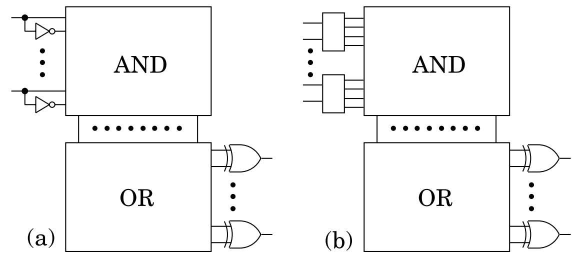

Fig. 1 AND-OR-EXOR three-level PLA with (a) one-bit and (b) two-bit decoders.

1. Introduction

Programmable logic arrays (PLAs) with two-input EXOR gates at the outputs, also known as AND-OR-EXOR PLAs (Fig. 1) [28], are a powerful architecture to realize many logic functions. The AND-OR-EXOR PLA realizes an EXOR of two sum-of-products expressions (EX-SOP). Minimization of the number of products in EX-SOPs is an important step in the optimization of AND-OR-EXOR PLAs, because the number of products is directly related to the cost of PLAs. EX-SOPs are promising because, for many practical logic functions, they often require many fewer products than sum-of-products expressions (SOPs) [7], [10], [11], [15], [16], [28].

AND-OR-EXOR three-level networks are suitable for implementing adders, which serve as building blocks for synthesizing many other arithmetic circuits [21]. For example, Texas Instruments' SN181 arithmetic circuit and SN283 four-bit adder have two-input EXOR gates in the outputs [31]; Monolithic Memories' ZHAL20X8A eight-bit counter realizes EX-SOPs [19]. An AND-OR-EXOR is one of the simplest three-level architecture, since it contains only a single two-input EXOR gate. However, its logic capability is quite high. Because of this, various programmable logic devices (PLDs) with two-input EXOR gates in the outputs were developed. Especially, RICOH, Lattice and AMD (MMI) produced series of such PLDs [19], [22], [23] and millions of complex PLDs (CPLDs) with output EXOR gates have been shipped [1], [2]. An AND-OR-EXOR three-level network is also suitable for efficient implementation of many random functions. For example, simplified EX-SOPs for six-variable pseudo-random functions require 25 percent fewer products and 40 percent fewer literals than simplified SOPs [5]. For an arbitrary function of six variables, minimum SOPs require up to 32 products [29], while minimum EX-SOPs require at most 15 products [5].

Minimization of EX-SOPs were considered in the past [13], [30], and a cut-and-try method was reported [22]. Design methods for adders by using AND-OR-EXOR PLAs with more than one-bit input decoders were developed at IBM [32]. Exact minimization algorithms for EX-SOPs and upper bounds on the number of products in EX-SOPs are also reported [4]–[6], [9]. AND-OR-EXOR networks where output EXOR gates have unlimited fan-in is considered [27]. During the last several years significant progress in the heuristic minimization of EX-SOPs have been made and many interesting results are reported [7], [10], [11], [15], [16], [28]. However, no efficient algorithm to design AND-OR-EXOR PLAs for adders is developed.

Important contributions of the paper are as follows:

- We present a method to reduce the number of products in EX-SOPs by considering output phase optimization [26], where some components of the function are implemented in the complemented form.
- We develop a heuristic method to minimize EX-SOPs for adders with two- and four-valued inputs by using an extension of the AOXMIN algorithm [10].
- We proved that an n-bit adder with two-valued inputs

Manuscript received October 9, 2004.
Manuscript revised February 3, 2005.
[†]The author is with the Department of Computer Science and Engineering, Oakland University, Rochester, Michigan 48309, U.S.A.
[††]The author is with the Department of Computer Science and Electronics, Kyushu Institute of Technology, Iizuka-shi, 820–8502 Japan.
a) E-mail: debnath@oakland.edu
b) E-mail: sasao@cse.kyutech.ac.jp
DOI: 10.1093/ietisy/e88–d.7.1492

requires at most $3 \cdot 2^{n-2} + 7n - 5$ products in an EX-SOP.

A crucial step in AOXMIN is to partition the products of an SOP of the given function into two sets, which is done by a random method. We propose a partitioning method for adders. Our experimental result demonstrates that, for an n-bit adder with sufficiently large n, the proposed algorithm produces solutions with a half products of the random partitioning method in 250 times shorter time.

The remainder of the paper is organized as follows: Section 2 reviews terminology. Section 3 considers output phase optimization techniques. Section 4 summarizes AOXMIN and describes its extensions. Section 5 presents design methods for adders. Section 6 derives an upper bound on the number of products in EX-SOPs for adders. Section 7 shows experimental results. Section 8 presents conclusion.

2. Definitions and Terminology

In this section, we review basic terminology related to multiple-valued functions [25], [26].

Definition 1: A *multiple-valued input two-valued output function*, or *function* in short, is a mapping

$$f(X_1, X_2, \dots, X_n) : \bigtimes_{i=1}^{i=n} P_i \rightarrow B,$$

where $P_i = \{0, 1, \dots, p_i - 1\}$, $p_i \geq 2$, $B = \{0, 1\}$, and X_i is a *multiple-valued variable* taking a value from P_i.

Definition 2: Let $S_i \subseteq P_i$. A *literal* $X_i^{S_i}$ represents 0 if $X_i \notin S_i$ and 1 if $X_i \in S_i$. A *product* $X_1^{S_1} X_2^{S_2} \cdots X_n^{S_n}$ is AND of literals. A *cube* is a convenient representation of a product for computer manipulation.

Definition 3: A *sum-of-products expression (SOP)*

$$\bigvee_{(S_1, S_2, \dots, S_n)} X_1^{S_1} X_2^{S_2} \cdots X_n^{S_n}$$

is OR of products. An SOP is represented by a *cover*, which is a set of cubes. An *EX-SOP*

$$\bigvee_{(S_1, S_2, \dots, S_n)} X_1^{S_1} X_2^{S_2} \cdots X_n^{S_n} \oplus \bigvee_{(S_1, S_2, \dots, S_n)} X_1^{S_1} X_2^{S_2} \cdots X_n^{S_n}$$

is the EXOR of two SOPs.

Definition 4: Let $f_i(X_1, X_2, \dots, X_n)$ $(i = 0, 1, \dots, m-1)$ be an n-input m-output function. The two-valued output function $F(X_1, X_2, \dots, X_n, X_{n+1})$, where X_{n+1} is an m-valued variable representing the outputs such that $F(X_1, X_2, \dots, X_n, i)$ $= f_i(X_1, X_2, \dots, X_n)$, is the *characteristic function* for the multiple-output function [26].

Definition 5: An *SOP for a multiple-output function* indicates an SOP for its characteristic function, and an *EX-SOP for a multiple-output function* indicates an EX-SOP for its characteristic function.

Definition 6: The *intersection* of the products $c_1 = X_1^{S_1} X_2^{S_2} \cdots X_n^{S_n}$ and $c_2 = X_1^{T_1} X_2^{T_2} \cdots X_n^{T_n}$, denoted by $c_1 \cap c_2$, is the product $X_1^{S_1 \cap T_1} X_2^{S_2 \cap T_2} \cdots X_n^{S_n \cap T_n}$. If $S_i \cap T_i = \emptyset$ for some i, then the intersection denotes a null cube.

Definition 7: *Disjoint sharp* of two covers F and G, denoted by $F \circledsharp G$, represents only those minterms of F which are not contained by G.

Definition 8: *ON-set*, *OFF-set*, and *DC-set* is the set of cubes for which the function value is 1, 0, and unspecified, respectively.

In this paper, we often use the same symbol for a function and its cover; and unless otherwise specified, *adder* refers to *adder without carry input*, and *adrn* represents an n-bit adder.

3. Output Phase Optimization

In many cases, we can realize a function f in either *positive phase* (f) or *negative phase* $(\bar{f})$. For m-output function, we can choose the output phases in 2^m ways. The choice of the output phases in the realization of a function influences on the number of products in its minimized expressions. To reduce the number of products by choosing the output phases is *output phase optimization* [26].

Definition 9: Let $(f_0, f_1, \dots, f_{m-1})$ be an m-output function. The minimized SOP G for the characteristic function of $(g_0, g_1, \dots, g_{m-1})$, where $g_i \in \{\bar{f_i}, f_i\}$ $(i = 0, 1, \dots, m-1)$ such that the number of products in G is minimal, is the *output phase optimized SOP* for $(f_0, f_1, \dots, f_{m-1})$.

Similarly, we can define an *output phase optimized EX-SOP*. We handle the output phase optimization of EX-SOPs by using the output phase optimization techniques for SOPs. We use an output phase optimized SOP as the input of the EX-SOP minimizer. For a function with m outputs, an EX-SOP minimizer produces two SOPs each having m outputs. We optimize the output phases of the $2m$-output SOP to obtain an output phase optimized EX-SOP.

Let the output phase for the function f_i be $a_i \in \{0, 1\}$, where $a_i = 0$ indicates f_i is in the positive phase and $a_i = 1$ indicates f_i is in the negative phase. Let the output phases of the two SOPs of the EX-SOP for f_i be b_{i0} and b_{i1}. Therefore, the output phase of the EX-SOP for f_i is $a_i \oplus b_{i0} \oplus b_{i1}$. When output phase optimization of the two m-output SOPs is impractical, we consider a_i as the output phase of the EX-SOP for f_i. The output phase optimization technique for AND-OR-EXOR three-level PLAs is shown in Fig. 2. An output phase optimized EX-SOP can be realized in an AND-OR-EXOR PLA, where the polarity of the outputs are programmable.

4. Minimization Techniques

In this section we review AOXMIN [10], which is a heuristic algorithm to simplify EX-SOPs. We then present an extension of AOXMIN.

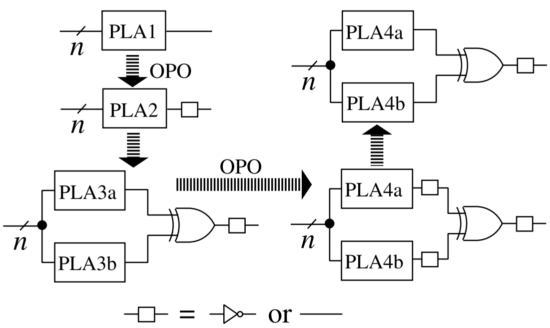

Fig. 2 Output phase optimization (OPO).

```
procedure AOXMIN_Specify_Both(F_A, F_B, R) {
    F_a ← Espresso(F_A, R, F_B);
    R_assigned ← F_a ⊕ F_A;
    F_temp ← F_B ∪ R_assigned;
    R_temp ← F_A ∪ (R ⊕ R_assigned);
    F_b ← Espresso(F_temp, ∅, R_temp);
    return (F_a, F_b);
}
```

Fig. 3 Pseudocode for *AOXMIN_Specify_Both*.

4.1 Overview of AOXMIN

Basic steps of AOXMIN are as follows:

1. Obtain a minimized cover F for the given function f and compute a cover R for $\bar{f}$.
2. Group the cubes of F into clusters of cubes. Two cubes are in the same *cluster* if they intersect or they are connected through a chain of intersecting cubes. (In [10], a cluster of cubes are called an equivalence class.)
3. Randomly partition the clusters of cubes into two covers, F_A and F_B.
4. Obtain two EX-SOPs by using *AOXMIN_Specify_Both*(F_A, F_B, R) and *AOXMIN_Specify_Both*(F_B, F_A, R) (Fig. 3). *AOXMIN_Specify_Both*returns two SOPs which form an EX-SOP. *Espresso*(F_k, D_k, R_k) in Fig. 3 obtains a minimized cover for a function, where F_k, D_k, and R_k represents the ON-set, DC-set, and OFF-set, respectively.
5. Iterate steps 3 and 4 for some specified number of times, and take the best EX-SOP among all the EX-SOPs generated so far.

In addition, AOXMIN simplifies complement of the given function and uses some output phase optimization technique to obtain better solution.

4.2 Extension of AOXMIN

The proposed heuristic method to simplify EX-SOPs, which is an extension of AOXMIN [10], have the following features:

- It can simplify EX-SOPs for functions with two- and four-valued variables, and can treat functions where different variables have different domains (two-valued or four-valued). On the other hand, AOXMIN simplifies only two-valued functions.
- It uses heuristic algorithms to partition the clusters of cubes for adders. In this regard, AOXMIN uses only a random partitioning method.
- During iterative improvement, it concurrently minimizes both SOPs of the EX-SOP to reduce the total number of products by increasing shared products between two SOPs. On the other hand, AOXMIN uses simultaneous minimization of both SOPs only once as part of its simplification technique for multiple-output functions.
- For multiple-output functions, it performs concurrent simplification of all the outputs. However, AOXMIN simplifies each output separately throughout the algorithm. A modified AOXMIN considers simplification of all the outputs simultaneously [11].
- For the output phase optimization of EX-SOPs, it uses techniques for the output phase optimization of SOPs [26]. AOXMIN handles the output phase optimization problem in a different way.
- To find good solutions quickly, especially for adders, it selects from two different minimizers for SOPs. On the other hand, AOXMIN uses only Espresso [3].
- The method makes efficient use of the given don't care conditions during grouping the cover into clusters of cubes and also during every minimization of the SOPs of the EX-SOP. AOXMIN does not use don't care conditions during these two operations.

The minimization of an SOP for a multiple-output function corresponds to the minimization of an SOP for its characteristic function [26]. Similarly, we can prove the following:

Theorem 1: The minimization of an EX-SOP for a multiple-output function corresponds to the minimization of an EX-SOP for its characteristic function.

Now, the definition of the clusters of cubes can be extended as follows:

Definition 10: Let F and D be the covers for the ON-set and DC-set, respectively, of the characteristic function for a multiple-output function. Then, two cubes $c_i, c_j \in F$ are in the same *cluster* if

(a) $G(i, j) \neq \emptyset$, or
(b) $G(i, i+1) \neq \emptyset, G(i+1, i+2) \neq \emptyset, \ldots, G(j-1, j) \neq \emptyset$,

where $G(p, q)$ denotes $(c_p \cap c_q) \; \#\!\!\!\!\bigcirc \; D$.

Section 4.1 shows that during every iteration AOXMIN calls *AOXMIN_Specify_Both* twice. We replaced these calls by *Modified_Specify_Both*(F_A, F_B, D, R) and *Modified_Specify_Both*(F_B, F_A, D, R) (Fig. 4). *Make_Double_Out_Cover*(F_k, G_k) in Fig. 4 receives n-input m-output covers F_k and G_k, and returns an n-input $2m$-output cover such that covers corresponding to outputs $0, 1, \ldots, m-1$ and $m, m+1, \ldots, 2m-1$ represent F_k and G_k, respectively.

In Fig. 4, both *Simplify_Single*(F_k, D_k, R_k) and *Simplify_Double*(F_k, D_k, R_k) obtain a minimized cover for a function, where F_k, D_k, and R_k represents the ON-set, DC-set, and OFF-set, respectively. *Simplify_Single* and *Simplify_Double* can be either *Simplify_Local* (Fig. 5) or Espresso-MV [25]. *Simplify_Local* uses a single pass of *Reduce*, *Expand*, and *Irredundant* operations to obtain a simplified SOP [25]. It reduces the number of cubes by locally changing the shape of the cubes. Espresso-MV iterates these operations as long as the solution improves. Sections 5 and 7 explain how the choice of the two-level minimizers influence the quality of the solution and execution time.

```
procedure Modified_Specify_Both(F_A, F_B, D, R) {
    F_AsharpD ← F_A ⊕ D;
    F_BsharpD ← F_B ⊕ D;

    F_a ← Simplify_Single(F_AsharpD, D ∪ R, F_BsharpD);
    R_assigned ← F_a ∩ R;
    R_remained ← R ⊕ R_assigned;

    F_b ← F_BsharpD ∪ R_assigned;
    R_a ← F_BsharpD ∪ R_remained;
    R_b ← F_AsharpD ∪ R_remained;

    F_dbl ← Make_Double_Out_Cover(F_a, F_b);
    R_dbl ← Make_Double_Out_Cover(R_a, R_b);
    D_dbl ← Make_Double_Out_Cover(D, D);

    F_EX-SOP ← Simplify_Double(F_dbl, D_dbl, R_dbl);
    return F_EX-SOP;
}
```

Fig. 4 Pseudocode for *Modified_Specify_Both*.

```
/* F = ON-set, D = DC-set, R = OFF-set */
procedure Simplify_Local(F, D, R) {
    F ← Reduce(F, D);
    F ← Expand(F, R);
    F ← Irredundant(F, D);
    return F;
}
```

Fig. 5 Pseudocode for *Simplify_Local*.

5. Design of Adders

In this section, we propose partitioning methods of the cluster of cubes for adders with one- and two-bit decoders, and discuss about the choice of the two-level minimizers. Note that EX-SOPs for functions with two- and four-valued inputs correspond to AND-OR-EXOR PLAs with one- and two-bit decoders, respectively (Fig. 1).

During minimization of adders, we use *Simplify_Local* for *Simplify_Single* and Espresso-MV for *Simplify_Double* in Fig. 4. We observe that if Espresso-MV is used for *Simplify_Single* then the resulting awkward shape of $R_{assigned}$ in Fig. 4 prevent us from obtaining a good solution in the next minimization by using *Simplify_Double*.

5.1 Adders with One-Bit Decoders

We found that an output phase optimized SOP for n-bit ($3 \leq n \leq 11$) adder with two-valued inputs has $4n-1$ clusters of cubes. Figure 6 shows the distribution of these clusters, where an entry c_k represents k clusters each having c cubes. It is interesting that the number of cubes in the clusters have a regular structure. To partition the clusters of cubes into two covers F_A and F_B, we use the following method:

1. Sort the clusters in descending order of the number of cubes in them.
2. Starting from the beginning of the sorted list of the clusters, alternately add a pair of clusters to F_A and F_B.
3. Add the remaining cluster to F_B.

Example 1: For three-bit adder with two-valued inputs, the number of cubes in the clusters which form F_A and F_B are 5, 5, 2, 2, 1, 1, and 3, 3, 1, 1, 1, respectively.

The above partitioning method is devised by considering outputs. Adders have pairs of clusters, where each pair belongs to a particular set of outputs. Roughly, the strategy is to put the clusters from such a pair into two different partitions. A similar method is also devised for adders with four-valued inputs.

Figure 7 shows Karnaugh map for a six variable function [20]. Its SOP requires 16 products and EX-SOP, $(p_1 \vee p_2) \oplus (p_3 \vee p_4 \vee p_5)$, requires five products as shown in Fig. 7. The EX-SOP is designed by using the method presented in this section.

adr3: $1_5, 2_2, 3_2, 5_2$
adr4: $1_5, 2_2, 3_2, 5_2, 7_2, 11_2$
adr5: $1_5, 2_2, 3_2, 5_2, 7_2, 11_2, 15_2, 23_2$
adr6: $1_5, 2_2, 3_2, 5_2, 7_2, 11_2, 15_2, 23_2, 31_2, 47_2$
adr7: $1_5, 2_2, 3_2, 5_2, 7_2, 11_2, 15_2, 23_2, 31_2, 47_2, 63_2, 95_2$
adr8: $1_5, 2_2, 3_2, 5_2, 7_2, 11_2, 15_2, 23_2, 31_2, 47_2, 63_2, 95_2, 127_2, 191_2$
adr9: $1_5, 2_2, 3_2, 5_2, 7_2, 11_2, 15_2, 23_2, 31_2, 47_2, 63_2, 95_2, 127_2, 191_2, 255_2, 383_2$
adr10: $1_5, 2_2, 3_2, 5_2, 7_2, 11_2, 15_2, 23_2, 31_2, 47_2, 63_2, 95_2, 127_2, 191_2, 255_2, 383_2, 511_2, 767_2$
adr11: $1_5, 2_2, 3_2, 5_2, 7_2, 11_2, 15_2, 23_2, 31_2, 47_2, 63_2, 95_2, 127_2, 191_2, 255_2, 383_2, 511_2, 767_2, 1023_2, 1535_2$

Fig. 6 Distribution of the clusters of output phase optimized SOPs for adders with two-valued inputs.

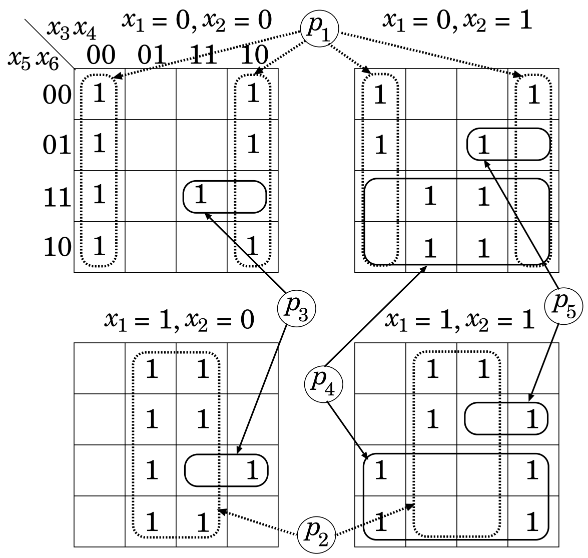

Fig. 7 Karnaugh map of an output of *adr3* (output phase optimized).

adr4: $1_4, 2_2, 3_2$
adr5: $1_4, 2_2, 3_2, 4_2$
adr6: $1_4, 2_2, 3_2, 4_2, 5_2$
adr7: $1_4, 2_2, 3_2, 4_2, 5_2, 6_2$
adr8: $1_4, 2_2, 3_2, 4_2, 5_2, 6_2, 7_2$
adr9: $1_4, 2_2, 3_2, 4_2, 5_2, 6_2, 7_2, 8_2$
adr10: $1_4, 2_2, 3_2, 4_2, 5_2, 6_2, 7_2, 8_2, 9_2$
adr11: $1_4, 2_2, 3_2, 4_2, 5_2, 6_2, 7_2, 8_2, 9_2, 10_2$

Fig. 8 Distribution of the clusters of output phase optimized SOPs for adders with four-valued inputs.

5.2 Adders with Two-Bit Decoders

We obtained functions with four-valued inputs from their two-valued counterparts by pairing two variables using Espresso-MV [25]. Figure 8 shows the distribution of the clusters of output phase optimized SOPs for adders with two-bit decoders, where an entry c_k represents k clusters each having c cubes. It shows that the output phase optimized SOP for n-bit ($4 \le n \le 11$) adder with two-bit decoders have $2n$ clusters. Note that the number of cubes in the clusters for adders with two-bit decoders also have a regular structure. We use the following method to partition the clusters into two covers F_A and F_B:

1. Sort the clusters in descending order of the number of cubes in them.
2. Starting from the beginning of the sorted list of the clusters, at first add a pair of clusters to F_A, then alternately add a cluster to F_A and F_B.

6. Number of Products in Adders

In this section we derive an upper bound on the number of products in an EX-SOP for an n-bit adder with two-valued inputs.

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

$$
\begin{array}{rcccc}
 & x_{n-1} & x_{n-2} & \cdots & x_0 \\
+) & y_{n-1} & y_{n-2} & \cdots & y_0 \\
\hline
z_n & z_{n-1} & z_{n-2} & \cdots & z_0 \\
 & c_{n-1} & c_{n-2} & \cdots & c_0
\end{array}
$$

where z_i's are sums and c_i's are carries. Note that $z_n = c_{n-1}$. For *adrn*, we have the following relations:

$$
\begin{aligned}
z_i &= (x_i \oplus y_i) \oplus c_{i-1} \\
&= p_i \oplus c_{i-1} \\
&= \bar{p}_i \oplus \bar{c}_{i-1}, \\
c_i &= x_i y_i \oplus (x_i \oplus y_i) c_{i-1} \\
&= g_i \oplus p_i c_{i-1} \\
&= \bar{g}_i \oplus (\bar{p}_i \vee \bar{c}_{i-1}), \\
c_i &= x_i y_i \vee c_{i-1}(x_i \vee y_i), \\
\bar{c}_i &= \bar{x}_i \bar{y}_i \vee \bar{c}_{i-1}(\bar{x}_i \vee \bar{y}_i) \\
&= r_i \vee s_i \bar{c}_{i-1},
\end{aligned}
$$

where $p_i = x_i \oplus y_i$, $g_i = x_i y_i$, $r_i = \bar{x}_i \bar{y}_i$, $s_i = \bar{x}_i \vee \bar{y}_i$. Also, $z_0 = p_0 = x_0 \oplus y_0$, and $c_0 = g_0 = x_0 y_0$.

Let $t(SOP, f)$ be the number of products in a minimum SOP for f. Let $t(EX\text{-}SOP, f)$ be the number of products in a minimum EX-SOP for f.

Lemma 1:

$$
\begin{aligned}
&t(SOP, \bar{g}_{i-1} \oplus p_{i-1} g_{i-2}) = 5. \\
&t(SOP, \bar{p}_{i-1} \oplus g_{i-2}) = 6. \\
&t(SOP, \bar{p}_{i-1}) = 2. \\
&t(SOP, \bar{p}_{i-2} \vee \bar{c}_{i-3}) = 2 + t(SOP, \bar{c}_{i-3}).
\end{aligned}
$$

Lemma 2: $t(EX\text{-}SOP, z_0) = 2$.

Lemma 3: $t(EX\text{-}SOP, z_1) = 3$.

Lemma 4: $t(SOP, \bar{c}_i) = 3 \cdot 2^i - 1$.

Proof: Note that $t(SOP, \bar{c}_0) = 2$. From $\bar{c}_i = r_i \vee s_i \bar{c}_{i-1}$, we have $t(SOP, \bar{c}_i) = 1 + 2t(SOP, \bar{c}_{i-1})$. From the recurrence relation, we have the lemma.

Lemma 5: $t(EX\text{-}SOP, z_i) \le 8 + t(SOP, \bar{c}_{i-2})$.

Proof:

$$
\begin{aligned}
z_i &= p_i \oplus c_{i-1} \\
&= p_i \oplus g_{i-1} \oplus p_{i-1} c_{i-2} \\
&= (p_i \oplus g_{i-1}) \oplus (\bar{p}_{i-1} \vee \bar{c}_{i-2}).
\end{aligned}
$$

Since $t(SOP, p_i \oplus g_{i-1}) = 6$ and $t(SOP, \bar{p}_{i-1}) = 2$, we have the lemma.

Lemma 6: Two functions c_{i-1} and z_{i-1} can be realized with an EX-SOP at the same time by using $15 + t(SOP, \bar{c}_{i-3})$ products.

Proof:

$$\begin{aligned}
c_{i-1} &= g_{i-1} \oplus p_{i-1}c_{i-2} \\
&= g_{i-1} \oplus p_{i-1}(g_{i-2} \oplus p_{i-2}c_{i-3}) \\
&= (g_{i-1} \oplus p_{i-1}g_{i-2}) \oplus (p_{i-1}p_{i-2}c_{i-3}) \\
&= (\bar{g}_{i-1} \oplus p_{i-1}g_{i-2}) \oplus (\bar{p}_{i-1} \vee \bar{p}_{i-2} \vee \bar{c}_{i-3}), \\
z_{i-1} &= p_{i-1} \oplus c_{i-2} \\
&= p_{i-1} \oplus g_{i-2} \oplus p_{i-2}c_{i-3} \\
&= (\bar{p}_{i-1} \oplus g_{i-2}) \oplus (\bar{p}_{i-2} \vee \bar{c}_{i-3}).
\end{aligned}$$

From Lemmas 1 to 5, we have this lemma.

Theorem 2: An n-bit adder without carry input can be represented by an EX-SOP with at most $3 \cdot 2^{n-2} + 7n - 5$ products for $n \geq 3$.

Proof: Let W be the number of products necessary in an EX-SOP. Then, we have

$$\begin{aligned}
W &= \sum_{i=0}^{n-2} t(SOP, z_i) + 15 + t(SOP, \bar{c}_{n-3}) \\
&\leq 2 + 3 + \sum_{i=2}^{n-2} [8 + t(SOP, \bar{c}_{i-2})] + 15 \\
&\quad + t(SOP, \bar{c}_{n-3}) \\
&= 5 + \sum_{i=2}^{n-2} [7 + 3 \cdot 2^{i-2}] + 15 + t(SOP, \bar{c}_{n-3}) \\
&= 5 + 7(n-3) + 3(2^0 + 2^1 + \cdots + 2^{n-4}) \\
&\quad + 15 + 3 \cdot 2^{n-3} - 1 \\
&= 3 \cdot (2^0 + 2^1 + \cdots + 2^{n-4} + 2^{n-3}) \\
&\quad + 7n - 16 + 15 - 1 \\
&= 3 \cdot (2^{n-2} - 1) + 7n - 2 \\
&= 3 \cdot 2^{n-2} + 7n - 5.
\end{aligned}$$

7. Experimental Results

We implemented the proposed method to simplify EX-SOPs for adders in C by using Espresso-MV [25] routines on a 2.40 GHz Pentium 4 PC running Linux. For the experiments we prepared minimized SOPs and output phase optimized SOPs by using Espresso-MV with default options. We obtained adders with four-valued inputs from their two-valued counterparts by pairing two variables using Espresso-MV.

Tables 1, 2, and 3 summarize the experimental results, which are obtained by using: a) output phase optimized SOPs as the input for the EX-SOP minimizer; b) two different techniques to partition the clusters of cubes: partitioning method for adders from Sect. 5 and random partitioning method from AOXMIN [10]; and c) *Simplify_Local* for *Simplify_Single* and Espresso-MV for *Simplify_Double* in Fig. 4.

In Table 1, the columns with heading 'SOP', 'OPO SOP', 'EX-SOP', and 'OPO EX-SOP' indicate the number

Table 1 Number of products and execution time in seconds for adders with two-valued inputs.

									Dubrova-Miller-Muzio Partition [10]			
						Proposed Partition			20 Iterations		50 Iterations	
Data	In	Out	SOP	Time	OPO SOP	EX-SOP	Time	OPO EX-SOP	EX-SOP	Time	EX-SOP	Time
adr3	6	4	31	0.01	25	12	0.01	11	17	0.10	13	0.29
adr4	8	5	75	0.01	61	21	0.01	18	32	0.40	32	1.23
adr5	10	6	167	0.04	137	37	0.03	36	50	2.32	50	5.64
adr6	12	7	355	0.15	293	67	0.13	66	146	10.29	133	30.29
adr7	14	8	735	0.45	609	122	0.40	120	128	38.71	128	94.57
adr8	16	9	1499	2.04	1245	233	1.81	233	423	149.90	380	360.14
adr9	18	10	3031	7.38	2521	454	6.73	454	840	594.58	840	1504.74
adr10	20	11	6099	34.63	5077	967	28.39	967	2168	2627.12	1898	6754.61
adr11	22	12	12239	153.81	10193	1993	129.73	1993	4136	15465.33	3677	38734.42

OPO: Output phase optimized.

Table 2 Number of connections to the inputs of gates for adders with two-valued inputs.

			SOP		OPO EX-SOP			
Data	In	Out	AND	OR	AND	OR	EXOR	OPO EX-SOP/SOP
adr3	6	4	116	31	30	20	8	0.39
adr4	8	5	340	75	58	35	10	0.25
adr5	10	6	892	167	128	72	12	0.20
adr6	12	7	2196	355	282	136	14	0.17
adr7	14	8	5196	735	620	254	16	0.15
adr8	16	9	11972	1499	1422	505	18	0.14
adr9	18	10	27068	3031	3234	998	20	0.14
adr10	20	11	60340	6099	7846	1767	22	0.15
adr11	22	12	133036	12239	18096	3307	24	0.15

OPO: Output phase optimized.

Table 3 Number of products and execution time in seconds for adders with four-valued inputs.

						Dubrova-Miller-Muzio Partition [10]			
			Proposed Partition			20 Iterations		50 Iterations	
Data	SOP	OPO SOP	EX-SOP	Time	OPO EX-SOP	EX-SOP	Time	EX-SOP	Time
adr4	17	14	13	0.01	12	13	0.15	13	0.38
adr5	26	22	18	0.03	18	18	0.53	18	1.47
adr6	37	32	25	0.07	25	25	1.67	25	4.63
adr7	50	44	33	0.18	33	33	5.35	33	12.46
adr8	65	58	42	0.60	42	43	16.48	43	39.03
adr9	82	74	52	2.41	52	51	60.15	51	138.20
adr10	101	92	63	6.78	63	65	223.18	61	535.91
adr11	122	112	75	48.63	75	75	721.77	75	2142.84

OPO: Output phase optimized.

of products in the corresponding expression, where 'OPO' is an abbreviation for 'output phase optimized'. The fifth column with heading 'Time' indicates the CPU seconds spent by the Espresso-MV [25] to minimize SOPs. The other columns with heading 'Time' indicate the CPU seconds spent by our program to simplify EX-SOP and they do not include the time to prepare minimized SOPs or output phase optimized SOPs.

Table 1 shows that, for an n-bit adder with two-valued inputs and with sufficiently large n, the proposed partitioning method produces solutions with a half products of the random partitioning method in about 250 times shorter time. We used *adr6* to see how the choice of the two-level minimizers in Fig. 4 influence the quality of the solution and execution time. By using random partitions and 1000 iterations, we found that when Espresso-MV is used for both *Simplify_Single* and *Simplify_Double* the algorithm requires 567.15 seconds and produces a solution with 122 products; however, when we use *Simplify_Local* for *Simplify_Single* and Espresso-MV for *Simplify_Double*, the algorithm produces a solution with 81 products and requires 541.44 seconds. We found similar tendencies for other adders too. It should be noted that in Table 1 data on the 7th and 9th columns are the same for the last four rows. This is because of the memory overflow of Espresso-MV as outlined in Sect. 3. In spite of this as Table 1 shows, adders based on EX-SOPs require far fewer gates than those based on SOPs.

Table 2 shows the number of connections to the inputs of gates for adders with two-valued inputs. For large n, three-level AND-OR-EXOR PLAs achieve about 85 percent saving in the cost of connections.

Table 3 shows that the proposed partitioning method also produces good solutions quickly for adders with four-valued inputs. However, in most cases, these solutions can be obtained by random partitioning method by a reasonable increase in the computation time. The experimental data also reveals that the minimization time for EX-SOPs with four-valued inputs is much smaller than that for the corresponding EX-SOPs with two-valued inputs, because the former requires many fewer products than the later. Note that an EX-SOP for an n-bit adder with two-bit decoders requires at most $(n^2 + n + 2)/2$ products [28].

8. Conclusions and Comments

Adders are important because they form the basic building blocks of numerous digital systems, and EX-SOPs are promising because they often require many fewer products than SOPs. We presented partitioning methods, which are effective in optimizing EX-SOPs for adders. Our experimental result shows that random partitioning method is unsuitable for designing adders when n is large, because it requires excessive amount of CPU time to obtain a moderately optimized design. We found that the choice of two-level minimizers in AOXMIN-like algorithm have a great influence on the number of products in EX-SOPs and that a powerful minimizer is not always a good choice. We proved that an n-bit adder with two-valued inputs requires at most $3 \cdot 2^{n-2} + 7n - 5$ products in an EX-SOP while an SOP requires $6 \cdot 2^n - 4n - 5$ products. We obtained adders with four-valued inputs from their two-valued counterparts by pairing two variables using Espresso-MV code [25], which reduces the number of products in SOPs [26]. A different pairing algorithm targeting EX-SOPs may lead to better solutions. Investigations are underway for integrating the proposed AND-OR-EXOR design techniques with three-level OR-AND-OR synthesis methods [8] and for adapting the integrated design systems to synthesize logic circuits for commercial CPLDs that have four-level OR-AND-OR-EXOR architecture [2]. Logic synthesis for such a four-level architecture is a challenging problem and very little has been published on the topic [27].

A limitation of the proposed method is its inability to handle large adders. However, in the practical LSI applications, optimization of only small adders is sufficient in implementing large adders. The fan-in of the gates of an AND-OR-EXOR three-level realization for an n-bit adder increases with n. In the LSI realization, gates with large fan-in are difficult to fabricate and tend to be slow [24], [33]. Therefore, monolithic implementations of n-bit adders for large n are impractical. When n is large, fast adders are implemented by combining well-designed adders of smaller sizes [17], and the design strategies are primarily guided by the overall speed of the adders. Various schemes for such design have been developed. One of them is carry-skip adders which use $\sqrt{n/2}$-bit adders for implementing an n-bit adder [17, p.117]. Therefore, large carry-skip adders such as one to add 128-bit numbers can be implemented by using only 8-bit adders. A 64-bit hybrid carry lookahead adder also uses 8-bit adders as its building blocks [18]. Another variant of carry-skip scheme uses 2- to 6-bit adders for implementing a 128-bit adder [14]. Various module-based designs are used in practice [12], [17], [21].

Acknowledgement

This work was supported in part by the Japan Society for the Promotion of Science and in part by the Ministry of Education, Science, Culture, and Sports of Japan. We thank Prof. E.V. Dubrova and Prof. D.M. Miller for valuable discussions on AOXMIN.

References

[1] Altera Corporation, "MAX EPM7128 celebrates 50 million units," News & Views: Newsletter for Altera Customers, p.28, Fourth Quarter, 2000.
[2] Altera Corporation, MAX 9000 Programmable Logic Device Family Data Sheet, June 2003.
[3] R.K. Brayton, G.D. Hachtel, C.T. McMullen, and A. Sangiovanni-Vincentelli, Logic Minimization Algorithms for VLSI Synthesis, Kluwer Academic Publishers, 1984.
[4] D. Debnath and T. Sasao, "An optimization of AND-OR-EXOR three-level networks," Proc. Asia and South Pacific Design Automation Conference, pp.545–550, Jan. 1997.
[5] D. Debnath and T. Sasao, "Exclusive-OR of two sum-of-products expressions: Simplification and an upper bound on the number of products," 3rd International Workshop on Applications of the Reed-Muller Expansion in Circuit Design, pp.45–60, Sept. 1997.
[6] D. Debnath and T. Sasao, "Minimization of AND-OR-EXOR three-level networks with AND gate sharing," IEICE Trans. Inf. & Syst., vol.E80-D, no.10, pp.1001–1008, Oct. 1997.
[7] D. Debnath and T. Sasao, "A heuristic algorithm to design AND-OR-EXOR three-level networks," Proc. Asia and South Pacific Design Automation Conference, pp.69–74, Feb. 1998.
[8] D. Debnath and Z.G. Vranesic, "A fast algorithm for OR-AND-OR synthesis," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol.22, no.9, pp.1166–1176, Sept. 2003.
[9] 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," Electron. Lett., vol.31, no.7, pp.541–542, March 1995.
[10] E.V. Dubrova, D.M. Miller, and J.C. Muzio, "AOXMIN: A three-level heuristic AND-OR-XOR minimizer for Boolean functions," 3rd International Workshop on Applications of the Reed-Muller Expansion in Circuit Design, pp.209–218, Sept. 1997.
[11] E.V. Dubrova, D.M. Miller, and J.C. Muzio, "AOXMIN-MV: A heuristic algorithm for AND-OR-XOR minimization," 4th International Workshop on Applications of the Reed-Muller Expansion in Circuit Design, pp.37–53, Aug. 1999.
[12] M.D. Ercegovac and T. Lang, Digital Arithmetic, Morgan Kaufmann, 2004.
[13] 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," Proc. IEEE/ACM International Conference on Computer-Aided Design, pp.203–205, Nov. 1985.
[14] A. Guyot, B. Hochet, and J.-M. Muller, "A way to build efficient carry-skip adders," IEEE Trans. Comput., vol.C-36, no.10, pp.1144–1152, Oct. 1987.
[15] A. Jabir and J. Saul, "Heuristic AND-OR-EXOR three-level minimization algorithm for multiple-output incompletely-specified Boolean functions," IEE Proc., Comput. Digit. Tech., vol.147, no.6, pp.451–461, Nov. 2000.
[16] A. Jabir and J. Saul, "Minimization algorithm for three-level mixed AND-OR-EXOR/AND-OR-EXNOR representation of Boolean functions," IEE Proc., Comput. Digit. Tech., vol.149, no.3, pp.82–96, May 2002.
[17] I. Koren, Computer Arithmetic Algorithms, Second ed., A K Peters, 2002.
[18] T. Lynch and E.E. Swartzlander, Jr., "A spanning tree carry lookahead adder," IEEE Trans. Comput., vol.41, no.8, pp.931–939, Aug. 1992.
[19] Monolithic Memories Inc., PAL/PLE DEVICE: Programmable Logic Array Handbook, Fifth ed., 1986.
[20] S. Muroga, Logic Design and Switching Theory, John Wiley & Sons, 1979.
[21] B. Parhami, Computer Arithmetic: Algorithms and Hardware Designs, Oxford University Press, 2000.
[22] D. Pellerin and M. Holley, Practical Design Using Programmable Logic, Prentice Hall, 1991.
[23] RICOH, CMOS Electrically Programmable Logic, Series 20, no.85–02, 1985.
[24] J.M. Rabaey, A. Chandrakasan, and B. Nikolić, Digital Integrated Circuits: A Design Perspective, Second ed., Prentice Hall, 2003.
[25] R.L. Rudell and A. Sangiovanni-Vincentelli, "Multiple-valued minimization for PLA optimization," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol.CAD-6, no.5, pp.727–750, Sept. 1987.
[26] 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.
[27] T. Sasao, "Logic synthesis with EXOR gates," in Logic Synthesis and Optimization, ed. T. Sasao, Kluwer Academic Publishers, 1993.
[28] T. Sasao, "A design method for AND-OR-EXOR three-level networks," IEEE/ACM International Workshop on Logic Synthesis, pp.8:11–8:20, May 1995.
[29] T. Sasao, Switching Theory for Logic Synthesis, Kluwer Academic Publishers, 1999.
[30] K. Shu, H. Yasuura, and S. Yajima, "Optimization of PLDs with output parity gates," National Convention, Information Processing Society of Japan, March 1985.
[31] Texas Instruments Inc., The TTL Data Book for Design Engineers, 1973.
[32] A. Weinberger, "High-speed programmable logic array adders," IBM J. Res. Dev., vol.23, no.2, pp.163–178, March 1979.
[33] N.H.E. Weste and D. Harris, CMOS VLSI Design: A Circuits and Systems Perspective, Third ed., Addison-Wesley, 2005.

Debatosh Debnath received the B.Sc.Eng. and M.Sc.Eng. degrees from the Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, in 1991 and 1993, respectively, and the Ph.D. degree from the Kyushu Institute of Technology, Iizuka, Japan, in 1998. He held research positions at the Kyushu Institute of Technology from 1998 to 1999 and at the University of Toronto, Ontario, Canada, from 1999 to 2002. In 2002, he joined the Department of Computer Science and Engineering at the Oakland University, Rochester, Michigan, as an Assistant Professor. His research interests include logic synthesis, design for testability, multiple-valued logic, and CAD for field-programmable devices. He was a recipient of the Japan Society for the Promotion of Science Postdoctoral Fellowship.

Tsutomu Sasao received the BE, ME, and PhD degrees in electronics engineering from Osaka University, Osaka, Japan, in 1972, 1974, and 1977, respectively. He has held faculty/research positions at Osaka University, Japan, the IBM T.J. Watson Research Center, Yorktown Heights, New York, and the Naval Postgraduate School, Monterey, California. He is now a Professor of the Department of Computer Science and Electronics at the Kyushu Institute of Technology, Iizuka, Japan. His research areas include logic design and switching theory, representations of logic functions, and multiple-valued logic. He has published more than nine books on logic design, including *Logic Synthesis and Optimization, Representation of Discrete Functions, Switching Theory for Logic Synthesis*, and *Logic Synthesis and Verification*, Kluwer Academic Publishers, 1993, 1996, 1999, and 2001, respectively. He has served as Program Chairman for the IEEE International Symposium on Multiple-Valued Logic (ISMVL) many times. Also, he was the Symposium Chairman of the 28th ISMVL held in Fukuoka, Japan, in 1998. He received the NIWA Memorial Award in 1979, Distinctive Contribution Awards from the IEEE Computer Society MVL-TC in 1987, 1996, and 2004 for papers presented at ISMVLs, and Takeda Techno-Entrepreneurship Award in 2001. He has served as an Associate Editor of the *IEEE Transactions on Computers*. He is a fellow of the IEEE.