Compact SOP Representations for Multiple-Output Functions — An Encoding Method using Multiple-Valued Logic —

Tsutomu Sasao
Department of Computer Science and Electronics
Kyushu Institute of Technology
Center for Microelectronic Systems
Iizuka 820-8502, Japan

Abstract

This paper shows a method to represent a multiple-output function: Encoded characteristic function for non-zero outputs (ECFN). The ECFN uses $(n+u)$ binary variables to represent an n-input m-output function, where $u = \lceil \log_2 m \rceil$. The size of the sum-of-products expressions (SOPs) depends on the encoding method of the outputs. For some class of functions, the optimal encoding produces SOPs with $O(n)$ products, while the worst encoding produces SOPs with $O(2^n)$ products. We formulate encoding problem and show a heuristic optimization method. Experimental results using standard benchmark functions show the usefulness of the method.

Index term: Multiple-output function, encoding problem, multiple-valued logic, TDM, SOP, characteristic function.

1. Introduction

Logic networks usually have many outputs. In most cases, independent representation of each output is inefficient. So, efficient methods to represent multiple-output functions are important. An n-input m-output combinational logic network is represented by a multiple-output function $F_1 = (f_0, f_1, \ldots, f_{m-1}) : B^n \rightarrow B^m$, where $B = \{0, 1\}$.

The first method to represent multiple-output functions is truth tables. Table 1 is a truth table for a 2-input 3-output function. For an n-input function, the table requires 2^n rows, and is too large for large n.

The second method is the characteristic function (CF) of the multiple-output function. It is a mapping $F_2 : B^n \times B^m \rightarrow B$, where $F_2(\vec{a}, \vec{b}) = 1$ iff $(f_0(\vec{a}), f_1(\vec{a}), \ldots, f_{m-1}(\vec{a})) = \vec{b}$. In this representation, the function requires m auxiliary binary variables $\{z_0, z_1, \ldots, z_{m-1}\}$ that represent outputs. F_2 shows that set of all the valid combinations of the inputs and the outputs. For example, the CF of the multiple-output function shown in Table 1 is

$$F_2 = \bar{x}_1\bar{x}_0\bar{f}_0\bar{f}_1\bar{f}_2 \vee \bar{x}_1x_0\bar{f}_0f_1f_2 \vee x_1\bar{x}_0\bar{f}_0f_1f_2 \vee x_1x_0f_0\bar{f}_1f_2.$$

The CF uses only binary variables, but the size of the representations tends to be very large, since it involves $(n+m)$ binary variables. CFs are used in logic simulation [1], and multi-level logic optimization [6].

The third method is a characteristic function for non-zero outputs (CFN). It is a mapping $F_3 : B^n \times M \rightarrow B$, where $M = \{0, 1, \ldots, m-1\}$, and $F_3(\vec{a}, j) = 1$ iff $f_j(\vec{a}) = 1$, $j \in M$. The CFN has one auxiliary m-valued variable Y that represents the output part. For example, the CFN of the multiple-output function shown in Table 1 is

$$\begin{aligned} F_3 \;=\; & \bar{x}_1x_0Y^1 \vee \bar{x}_1x_0Y^2 \vee x_1\bar{x}_0Y^1 \vee \\ & x_1\bar{x}_0Y^2 \vee x_1x_0Y^0 \vee x_1x_0Y^2. \end{aligned}$$

A sum-of-products expression (SOP) for a CFN is realized by a programmable logic array (PLA) or an AND-OR two-level network. SOPs for CFNs are extensively used in logic synthesis [11]. They can be simplified by using multiple-valued logic minimizers such as MINI [7] and ESPRESSO-MV [9]. The CFN represents only non-zero input-output combinations, and SOPs for CFNs tend to be more compact than SOPs for CFs.

In this paper, we will consider the fourth method to represent multiple-output function: Encoded characteristic function for non-zero outputs (ECFN). It uses only binary variables and represents a mapping $F_4 : B^n \times B^u \rightarrow B$, where $u = \lceil \log_2 m \rceil$. $F_4(\vec{a}, \vec{b}) = 1$ iff $f_{\nu(\vec{b})}(\vec{a}) = 1$, where $\nu(\vec{b})$ denotes the integer represented by the binary vector $\vec{b}$. For example, the ECFN of the multiple-output function shown in Table 1 is

$$\begin{aligned} F_4 \;=\; & \bar{x}_1x_0\bar{z}_1z_0 \vee \bar{x}_1x_0z_1\bar{z}_0 \vee x_1\bar{x}_0\bar{z}_1z_0 \vee \\ & x_1\bar{x}_0z_1\bar{z}_0 \vee x_1x_0\bar{z}_1\bar{z}_0 \vee x_1x_0z_1\bar{z}_0. \end{aligned}$$

Note that F_4 is also represented as

$$F_4 = \bar{z}_1\bar{z}_0f_0 \vee \bar{z}_1z_0f_1 \vee z_1\bar{z}_0f_2 \vee z_1z_0f_3.$$

ECFNs are useful for FPGA design [8], logic emulation [2], and logic simulation [12].

Table 1. Truth table for a 2-input 3-output function.

x_1	x_0	f_0	f_1	f_2
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	0	1

2. ECFN and Encoding Problem

In this section, we will define encoded characteristic functions for non-zero outputs (ECFN's) and formulate their encoding program.

Definition 2.1 $x^0 = \bar{x}$, *and* $x^1 = x$.

Definition 2.2 *For an m-output function f_i ($i = 0, 1, \ldots, m-1$), an encoded characteristic function for non-zero outputs (ECFN) is defined as*

$$F = \bigvee_{i=0}^{m-1} z_{u-1}^{b_{u-1}} z_{u-2}^{b_{u-2}} \cdots z_0^{b_0} f_i,$$

where $\vec{b} = (b_{u-1}, b_{u-2}, \ldots, b_0)$ is a binary representation of an integer i, and $u = \lceil \log_2 m \rceil$.

Note that z_0, z_1, ... , and z_{u-1} are auxiliary variables that represent outputs. In the above definition, the integer i is encoded by a binary vector $\vec{b}$ in a natural way. However, by changing the encoding, we can often simplify the representation.

Example 2.1 *Consider a four-output function $F = (f_0, f_1, f_2, f_3)$. Encoding 1 in Table 2 produces the ECFN*

$$F_1 = \bar{z}_1\bar{z}_0 f_0 \vee \bar{z}_1 z_0 f_1 \vee z_1\bar{z}_0 f_2 \vee z_1 z_0 f_3.$$

Consider the case where $f_0 = 0$, $f_1 = x_0$, $f_2 = x_1 \vee x_0$, and $f_3 = x_1$. Then, we have

$$\begin{aligned} F_1 &= \bar{z}_1\bar{z}_0 0 \vee \bar{z}_1 z_0 x_0 \vee z_1\bar{z}_0(x_1 \vee x_0) \vee z_1 z_0 x_1 \\ &= \bar{z}_1 z_0 x_0 \vee z_1\bar{z}_0 x_0 \vee z_1\bar{z}_0 x_1 \vee z_1 z_0 x_1 \\ &= (\bar{z}_1 z_0 \vee z_1\bar{z}_0)x_0 \vee (z_1\bar{z}_0 \vee z_1 z_0)x_1 \\ &= \bar{z}_1 z_0 x_0 \vee z_1\bar{z}_0 x_0 \vee z_1 x_1. \end{aligned}$$

However, Encoding 2 in Table 2 produces the ECFN

$$F_2 = \bar{z}_1\bar{z}_0 f_0 \vee \bar{z}_1 z_0 f_1 \vee z_1\bar{z}_0 f_3 \vee z_1 z_0 f_2.$$

In this case, we have

$$\begin{aligned} F_2 &= \bar{z}_1\bar{z}_0 0 \vee \bar{z}_1 z_0 x_0 \vee z_1\bar{z}_0 x_1 \vee z_1 z_0(x_1 \vee x_0) \\ &= \bar{z}_1 z_0 x_0 \vee z_1\bar{z}_0 x_1 \vee z_1 z_0 x_0 \vee z_1 z_0 x_1 \\ &= z_0 x_0 \vee z_1 x_1. \end{aligned}$$

Note that Encoding 1 requires three products, while Encoding 2 requires only two products. (End of Example)

Table 2. Encoding methods for four-output function.

z_1	z_0	Encoding 1	Encoding 2	Encoding 3
0	0	f_0	f_0	f_0
0	1	f_1	f_1	f_3
1	0	f_2	f_3	f_2
1	1	f_3	f_2	f_1

Example 2.2 *Consider an 8-output function $(f_0, f_1, \ldots, f_7)$, where $f_0 = 0$, $f_1 = x_0$, $f_2 = x_1$, $f_3 = x_2$, $f_4 = x_0 \vee x_1$, $f_5 = x_0 \vee x_2$, $f_6 = x_1 \vee x_2$, $f_7 = x_0 \vee x_1 \vee x_2$.*

In this case, we need three auxiliary variables z_0, z_1, and z_2 to represent 8 outputs. Encoding 1 in Table 3 produces the ECFN

$$\begin{aligned} F_1 = {} & \bar{z}_2\bar{z}_1\bar{z}_0 f_0 \vee \bar{z}_2\bar{z}_1 z_0 f_1 \vee \bar{z}_2 z_1\bar{z}_0 f_2 \vee \bar{z}_2 z_1 z_0 f_3 \vee \\ & z_2\bar{z}_1\bar{z}_0 f_4 \vee z_2\bar{z}_1 z_0 f_5 \vee z_2 z_1\bar{z}_0 f_6 \vee z_2 z_1 z_0 f_7. \end{aligned}$$

Thus, we have

$$\begin{aligned} F_1 &= \bar{z}_2\bar{z}_1\bar{z}_0 0 \vee \bar{z}_2\bar{z}_1 z_0 x_0 \vee \bar{z}_2 z_1\bar{z}_0 x_1 \vee \bar{z}_2 z_1 z_0 x_2 \vee \\ &\quad z_2\bar{z}_1\bar{z}_0(x_0 \vee x_1) \vee z_2\bar{z}_1 z_0(x_0 \vee x_2) \vee \\ &\quad z_2 z_1\bar{z}_0(x_1 \vee x_2) \vee z_2 z_1 z_0(x_0 \vee x_1 \vee x_2) \\ &= \bar{z}_1 z_0 x_0 \vee z_1\bar{z}_0 x_1 \vee z_1 z_0 x_2 \vee z_2\bar{z}_1 x_0 \vee z_2\bar{z}_0 x_1 \vee \\ &\quad z_2 z_0 x_2 \vee z_2 z_1 x_2 \vee z_2 z_0 x_0 \vee z_2 z_1 x_1. \end{aligned}$$

However, Encoding 2 in Table 2 produces the ECFN

$$\begin{aligned} F_2 = {} & \bar{z}_2\bar{z}_1\bar{z}_0 f_0 \vee \bar{z}_2\bar{z}_1 z_0 f_1 \vee \bar{z}_2 z_1\bar{z}_0 f_2 \vee \bar{z}_2 z_1 z_0 f_4 \vee \\ & z_2\bar{z}_1\bar{z}_0 f_3 \vee z_2\bar{z}_1 z_0 f_5 \vee z_2 z_1\bar{z}_0 f_6 \vee z_2 z_1 z_0 f_7. \end{aligned}$$

Thus, we have

$$\begin{aligned} F_2 &= \bar{z}_2\bar{z}_1\bar{z}_0 0 \vee \bar{z}_2\bar{z}_1 z_0 x_0 \vee \bar{z}_2 z_1\bar{z}_0 x_1 \vee \\ &\quad \bar{z}_2 z_1 z_0(x_0 \vee x_1) \vee z_2\bar{z}_1\bar{z}_0 x_2 \vee z_2\bar{z}_1 z_0(x_0 \vee x_2) \vee \\ &\quad z_2 z_1\bar{z}_0(x_1 \vee x_2) \vee z_2 z_1 z_0(x_0 \vee x_1 \vee x_2) \\ &= (\bar{z}_2\bar{z}_1 z_0 \vee \bar{z}_2 z_1 z_0 \vee z_2\bar{z}_1 z_0 \vee z_2 z_1 z_0)x_0 \vee \\ &\quad (\bar{z}_2 z_1\bar{z}_0 \vee \bar{z}_2 z_1 z_0 \vee z_2 z_1\bar{z}_0 \vee z_2 z_1 z_0)x_1 \vee \\ &\quad (z_2\bar{z}_1\bar{z}_0 \vee z_2\bar{z}_1 z_0 \vee z_2 z_1\bar{z}_0 \vee z_2 z_1 z_0)x_2 \\ &= z_0 x_0 \vee z_1 x_1 \vee z_2 x_2. \end{aligned}$$

Note that Encoding 1 requires 9 products, while Encoding 2 requires only three products. (End of Example)

As shown in the previous examples, good encodings produce simpler SOPs.

For an m-output function, we need $u = \lceil \log_2 m \rceil$ auxiliary variables $\{z_0, z_1, \ldots, z_{u-1}\}$ to represent m outputs. So, the number of different encodings is

$$\frac{2^u!}{(2^u - m)!}.$$

Table 3. Encoding methods for 8-output function.

z_2	z_1	z_0	Encoding 1	Encoding 2
0	0	0	f_0	f_0
0	0	1	f_1	f_1
0	1	0	f_2	f_2
0	1	1	f_3	f_4
1	0	0	f_4	f_3
1	0	1	f_5	f_5
1	1	0	f_6	f_6
1	1	1	f_7	f_7

However, the number of products in the SOP is invariant under the complement and/or permutation of the auxiliary variables. Thus, to find the encoding for an ECFN that requires the least number of products in an SOP, we have only to consider

$$N = \frac{2^u!}{(2^u - m)!2^u u!} = \frac{(2^u - 1)!}{(2^u - m)!u!}$$

different encodings. For $m = 4$, we have $u = 2$, and we need only to consider $N = \frac{3!}{1!2!} = 3$ different encodings. Note that Encoding 3 in Table 2 also requires three products to represent the function in Example 2.1.

Given a CFN $F : B^n \times M \rightarrow B$, where $M = \{0, 1, \ldots, m-1\}$, we can formulate

Problem 2.1 *(Encoding problem for an ECFN) Given a function $B^n \times M \rightarrow B$, where $M = \{0, 1, 2, \ldots, m-1\}$. Represent the m-valued variable by using $u = \lceil \log_2 m \rceil$ binary variables so that the resulting SOP has the least number of products.*

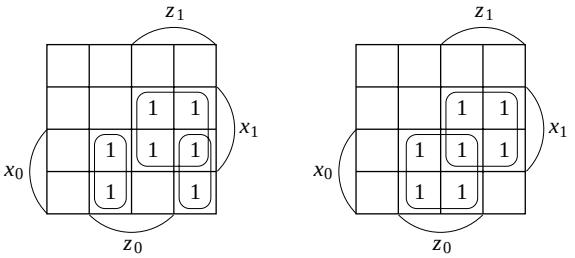
This problem is similar to

Problem 2.2 *(Input encoding problem) Given a function $B^n \times M \rightarrow B$, where $M = \{0, 1, 2, \ldots, m-1\}$. Represent the m-valued variable by using the sufficient number of auxiliary binary variables, so that the resulting SOP has the least number of products.*

Note that in Problem 2.1, the number of the auxiliary variables is fixed to $u = \lceil \log_2 m \rceil$, while in Problem 2.2, the number of auxiliary variables can be more than u. Both of these problems are intractable and require heuristic algorithms to solve them. Several algorithms have been developed for Problem 2.2 [5, 13, 14]. However, to our knowledge, Problem 2.1 is formulated for the first time, and no good algorithm is known.

3. Encoding Algorithm for ECFN

In this section, we will consider a heuristic algorithm to encode an m-valued variables by using $u = \lceil \log_2 m \rceil$ binary variables, so that the resulting SOP has the least number of products. To show the idea of the heuristic, we will use



(a) Encoding 1. (b) Encoding 2.

Figure 1. SOPs for ECFNs with different encodings.

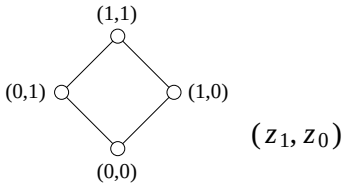


Figure 2. 2-dimensional cube.

Example 3.3 *Consider the 4-output function $f_0 = 0$, $f_1 = x_0$, $f_2 = x_0 \vee x_1$, $f_3 = x_1$. The positional cubes [7, 11, 6] for the ECFN are*

x_1		x_0		f_0	f_1	f_2	f_3
11	–	01	–	0	1	0	0
11	–	01	–	0	0	1	0
01	–	11	–	0	0	1	0
01	–	11	–	0	0	0	1

They are simplified to

x_1		x_0		f_0	f_1	f_2	f_3
11	–	01	–	0	1	1	0
01	–	11	–	0	0	1	1

Encodings 1 and 2 produce maps in Fig. 1(a) and (b), respectively. Note that Encoding 1 requires three products, while Encoding 2 requires only two products.

In general, an encoding for an ECFN corresponds to assign m output functions to the nodes of the u-dimensional cube. In this example, $m = 4$ and $u = 2$. Thus, we have to assign four functions f_0, f_1, f_2, f_3 to four nodes of the 2-dimensional cube in Fig. 2.

We can find a good assignment from the output part of the minimized positional cubes:

f_0	f_1	f_2	f_3
0	1	1	0
0	0	1	1

This is called a **constraint matrix** *[6]. The first row of the constraint matrix shows that f_1 and f_2 must be adjacent to be represented by a single product. And the second row of the constraint matrix shows that f_2 and f_3 must be adjacent to be represented by a single product. Encoding 1 in Table 2 (Fig. 3(a)) does not satisfy the conditions, while Encoding 2 (Fig. 3(b)) satisfies the conditions. So, we can expect that Encoding 2 produces simpler SOP than Encoding 1.*
(End of Example)

As shown in Example 3.3, the constraint matrix is useful to find a good encoding. To find a good encoding, we will use the **Merit matrix**. The value of $Merit(i,j)$, where $i,j \in M$ and $M = \{0,1,2,\ldots,m-1\}$ is large when f_i and f_j should be assigned to the adjacent nodes in the u-dimensional cube.

Algorithm 3.1 *(Derivation of the Merit Matrix)*

1. *From the minimized SOP of the CFN, obtain the constraint matrix. Ignore the rows with all 1's. Ignore the rows with single 1's.*
2. *Let $Merit(j,k) \leftarrow 0.0$, where $j,k \in M$ and $M = \{0,1,\ldots,m-1\}$.*
3. *For each row i in the constraint matrix, let S_i be the set of indexes of columns that have 1's in the row i. Let $|S_i|$ be the number of elements in S_i. For each pair $(j,k) \in S_i$, do $Merit(j,k) \leftarrow Merit(j,k) + \frac{1}{|S_i|-1}$.*
4. *If $Merit(j,k) = 0.0$ and $j,k \leq m$ and $j \neq k$, then let $Merit(j,k) \leftarrow -1 + \frac{u}{2^n}$.*

Example 3.4 *Consider the four-output function in Example 3.3, where $m = 4$. By using Algorithm 3.1, we will obtain the merit matrix.*

1) $S_1 = \{1,2\}$, $S_2 = \{2,3\}$, and $|S_1| = |S_2| = 2$.

2) For $i = 1$, we have

$$Merit(1,2) \leftarrow 1.0,\ Merit(2,1) \leftarrow 1.0.$$

For $i = 2$, we have

$$Merit(2,3) \leftarrow 1.0,\ Merit(3,2) \leftarrow 1.0.$$

3) In Step 3, we have

$$Merit = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \end{bmatrix} \end{array}.$$

4) Since $n = 2$ and $u = 2$ in Step 4, we have

$$-1.0 + \frac{u}{2^n} = -1.0 + 0.5 = -0.5.$$

Thus, we have

$$Merit = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0.0 & -0.5 & -0.5 & -0.5 \\ -0.5 & 0.0 & 1.0 & -0.5 \\ -0.5 & 1.0 & 0.0 & 1.0 \\ -0.5 & -0.5 & 1.0 & 0.0 \end{bmatrix} \end{array}.$$

(End of Example)

Algorithm 3.2 *(Encoding of an ECFN)*

1. *As an initial solution, assign functions f_0, f_1,...,f_{m-1} to distinct nodes of the u-dimensional cube. Let f_0 be assigned to the node $(0,0,\ldots,0)$. When $m < 2^u$, assign dummy functions to the remaining nodes.*
2. *Gain $\leftarrow \sum Merit(j,k)$, where the sum is obtained for the adjacent nodes (j,k) in the u-dimensional cube.*
3. *Fix the function f_0 to the node $(0,0,\ldots,0)$. For other $2^u - 1$ functions, choose a pair of functions. If Gain increases by the exchange of the functions in the pair, then exchange the functions. Otherwise, do not exchange the functions. Repeat this operation while Gain increases.*
4. *Fix the function f_0 to the node $(0,0,\ldots,0)$. For other $2^u - 1$ functions, choose a pair of functions. If Gain do not decrease by the exchange of the functions in the pair, then exchange the functions. Otherwise, do not exchange the functions. Repeat this operation while Gain increases.*
5. *Do the same thing as Step 3.*
6. *If Gain increased in Step 5, then go to Step 3. Otherwise stop.*

Example 3.5 *By using Algorithm 3.2, we will obtain a good encoding for the function in Example 3.3.*

1) The initial assignment is given in Fig. 3(a).

2) Gain= $Merit(0,1) + Merit(0,2) + Merit(1,3) + Merit(2,3) = -0.5 - 0.5 - 0.5 + 1.0 = -0.5$.

3) Choose the pair (f_2, f_3), and interchange the functions. Then, we have the assignment in Fig. 3(b).

4) In this case, $Gain = Merit(0,1) + Merit(0,3) + Merit(1,2) + Merit(2,3) = -0.5 - 0.5 + 1.0 + 1.0 = 1.0$. Since, Gain has increased, we keep this exchange.

5) Choose the pair (f_1, f_3) in Fig. 3(a), and interchange the functions. Then, we have the assignment in Fig. 3(c).

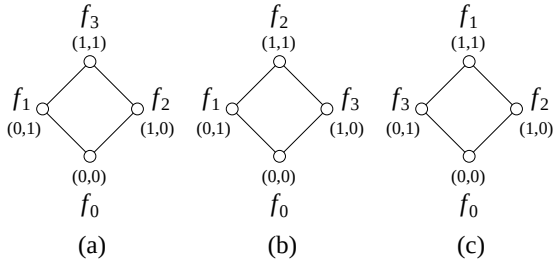


Figure 3. Assignments of four outputs to 2-dimensional cubes.

6) In this case, $Gain = Merit(0,2) + Merit(0,3) + Merit(1,2) + Merit(1,3) = -0.5 - 0.5 + 1.0 - 0.5 = -0.5$, which is the same as the case for Fig. 3(a). It is clear that Figs. 3(b) gives the best encoding.
(End of Example)

Note that Algorithm 3.2 does not always produces the optimal solution.

Theorem 3.1 *The n-input 2^n-output function $f_i(\vec{x}) = \bigvee_{j=0}^{n-1} a_{ij}x_j (i = 0, 1, \ldots, 2^n - 1)$, is represented by an SOP for an ECFN with n products, where $\vec{x} = (x_{n-1}, x_{n-2}, \ldots, x_0)$ and $\vec{a}_i = (a_{in-1}, a_{in-2}, \ldots, a_{i0})$ is a binary representation of the integer i.*

Example 3.6 *When $n = 3$, we have $f_0 = 0$, $f_1 = x_0$, $f_2 = x_1$, $f_3 = x_1 \vee x_0$, $f_4 = x_2$, $f_5 = x_2 \vee x_0$, $f_6 = x_2 \vee x_1$, $f_7 = x_2 \vee x_1 \vee x_0$. In this case the ECFN is represented by*

$$\begin{aligned} F &= \bar{z}_2\bar{z}_1\bar{z}_0 f_0 \vee \bar{z}_2\bar{z}_1 z_0 f_1 \vee \bar{z}_2 z_1\bar{z}_0 f_2 \vee \bar{z}_2 z_1 z_0 f_3 \vee \\ &\quad z_2\bar{z}_1\bar{z}_0 f_4 \vee z_2\bar{z}_1 z_0 f_5 \vee z_2 z_1\bar{z}_0 f_6 \vee z_2 z_1 z_0 f_7 \\ &= z_0x_0 \vee z_1x_1 \vee z_2x_2 \\ &= \bigvee_{i=0}^{2} z_ix_i. \end{aligned}$$

(End of Example)

Theorem 3.2 *The n-input 2^n-output function $f_i(\vec{x}) = \bigvee_{j=0}^{n-1} x_j^{\bar{a}_{ij}} \ (i = 0, 1, 2, \ldots, 2^n - 1)$, is represented by an SOP for an ECFN with $2n$ products, where $\vec{x} = (x_{n-1}, x_{n-2}, \ldots, x_0)$ and $\vec{a}_i = (a_{in-1}, a_{in-2}, \ldots, a_{i0})$ is a binary representation of the integer i.*

Example 3.7 *When $n = 3$, we have*

$$\begin{aligned} f_0 &= x_2 \vee x_1 \vee x_0, \\ f_1 &= x_2 \vee x_1 \vee \bar{x}_0, \\ f_2 &= x_2 \vee \bar{x}_1 \vee x_0, \\ f_3 &= x_2 \vee \bar{x}_1 \vee \bar{x}_0, \\ f_4 &= \bar{x}_2 \vee x_1 \vee x_0, \\ f_5 &= \bar{x}_2 \vee x_1 \vee \bar{x}_0, \\ f_6 &= \bar{x}_2 \vee \bar{x}_1 \vee x_0, \\ f_7 &= \bar{x}_2 \vee \bar{x}_1 \vee \bar{x}_0. \end{aligned}$$

In this case, the ECFN is represented by

$$\begin{aligned} F &= \bar{z}_2\bar{z}_1\bar{z}_0 f_0 \vee \bar{z}_2\bar{z}_1 z_0 f_1 \vee \bar{z}_2 z_1\bar{z}_0 f_2 \vee \bar{z}_2 z_1 z_0 f_3 \vee \\ &\quad z_2\bar{z}_1\bar{z}_0 f_4 \vee z_2\bar{z}_1 z_0 f_5 \vee z_2 z_1\bar{z}_0 f_6 \vee z_2 z_1 z_0 f_7 \\ &= (z_0 \oplus x_0) \vee (z_1 \oplus x_1) \vee (z_2 \oplus x_2) \\ &= \bigvee_{i=0}^{2} (z_i \oplus x_i). \end{aligned}$$

(End of Example)

Definition 3.3 *Let $\tau(MSOP, CFN)$ denote the number of products in a minimum sum-of-products expression (MSOP) for characteristic function for non-zero outputs (CFN), i.e., the number of products for minimized PLA for the multiple-output function. $\tau(MSOP, ECFN)$ and $\tau(MSOP, CF)$ are defined similarly.*

Theorem 3.3 $\tau(MSOP, CFN) \leq \tau(MSOP, ECFN)$.

Definition 3.4 *Let $\tau(MSOP, CFN(n))$ denote the number of products in a MSOP for CFN of an n-variable function. $\tau(MSOP, CF(n))$ is defined similarly.*

Theorem 3.4 $\tau(MSOP, CFN(n)) \leq 2^n$, $\tau(MSOP, CF(n)) \leq 2^n$.

4. Experimental Results

We developed programs for Algorithms 3.1 and 3.2, and minimized SOPs for various benchmark functions.

Table 4 shows the experimental results, where *Name* denotes the function name; *In* denotes the number of input variables; *Out* denotes the number of output variables; *CFN* denotes the number of products in the SOP for CFN (*i.e.*, the number of products in the minimized PLA); *ECFN* denotes the number of products in the SOP for ECFN; *Good* denotes the ECFN obtained by Algorithm 3.2; *Org* denotes the ECFN obtained by the straightforward encoding; *Bad* denotes the ECFN obtained by Algorithm 3.2 modified so that the Gain becomes small as possible; *CF* denotes the number of products in the SOP for CF. Table 4 shows that we can reduce the number of products by considering the encoding. Also, we can verify that Theorem 3.3 holds. In many cases, we could not obtain SOPs for CF due to memory overflow.

Table 5 shows the numbers of products in the SOPs for ECFNs for the multiple-output functions defined in Theorem 3.1. This table shows that the number of products in *Bad* encoding increases exponentially as n increases. Thus, we have

Table 4. Number of Products in SOPs.

Name	In	Out	CFN	ECFN			CF
			PLA	Good	Org	Bad	
5xp1	7	10	63	67	69	74	128
amd	14	24	66	108	125	158	137
apex1	45	45	206	522	719	898	
apex3	54	50	280	475	496	627	
apex4	9	19	429	772	833	983	438
b12	15	9	44	46	48	53	208
clip	9	5	118	126	132	141	430
duke2	22	29	86	138	175	198	
in4	32	20	211	255	305	350	
misex1	8	7	12	18	23	26	18
misex2	25	28	28	28	29	29	472
misex3	14	14	690	904	962	1121	1938
misg	56	23	69	70	73	75	
mish	94	43	82	84	89	91	
misj	35	14	35	35	48	48	
mlp4	8	8	126	130	133	144	227
opa	17	69	80	194	244	298	
risc	8	31	27	33	43	46	29
seq	41	35	350	730	1100	1398	
soar	83	94	357	378	448	482	
ti	47	72	214	335	388	511	
x2dn	82	56	104	106	114	116	

Table 5. Number of Products in SOPs for ECFNs in Theorem 3.1.

n	Good	Bad
3	3	8
4	4	20
5	5	48
6	6	112
7	7	274
8	8	633

Conjecture 4.1 *An n-input 2^n-output function exists that requires $O(n)$ and $O(2^n)$ products in SOPs for the ECFN with the encoding optimized and an ECFN with the encoding un-optimized, respectively.*

5. Conclusion and Comments

In this paper, we presented a new method to represent a multiple-output function: An encoding characteristic function for non-zero outputs (ECFN). An ECFN uses only binary variables, and its SOP can be simplified by considering the encoding. We formulated the encoding problem, and presented a heuristic optimization method. We also showed an n-input 2^n-output function that requires $O(n)$ products in an MSOP for one encoding, and $O(2^n)$ products for other encoding.

In this paper, we only considered the encoding for SOPs. However, the more interesting problem is the encoding for BDDs, which is useful for logic simulator [12].

Acknowledgments

This work was supported in part by a Grant in Aid for Scientific Research of the Ministry of Education, Culture, Sports, Science and Technology of Japan. Mr. M. Matsuura prepared the LaTeX files.

References

[1] P. Asher and S. Malik, "Fast functional simulation using branching programs," *ICCAD*, pp. 408-412, Oct. 1995.

[2] J. Babb, R. Tessier, M. Dahl, S. Z. Hanono, D. M. Hoki, and A. Agarwal, "Logic emulation with virtual wires," *IEEE Trans on CAD*, Vol. 16, No. 6, pp. 609-626, June 1997.

[3] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, Vol. C-35, No. 8, pp. 677-691(Aug. 1986).

[4] E. Cerny and M. A. Martin, "An approach to unified methodology of combinational switching circuits," *IEEE TC*, Vol. C-26, No. 8, pp. 745-756, Aug. 1977.

[5] G. De Micheli, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Optimal state assignment of finite state machine," *IEEE Trans. on CAD*, vol. CAD-4, No. 3, pp. 262-285, July 1985.

[6] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, 1994.

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

[8] J.-H. R. Jian, J.-Y. Jou, and J.-D. Huang, "Compatible class encoding in hyper-function decomposition for FPGA synthesis," *Design Automation Conference*, pp. 712-717, June 1998.

[9] R. L. Rudell and A. L. Sangiovanni-Vincentelli, "Multiple-valued minimization for PLA optimization," *IEEE TCAD*, Vol. CAD-6, No. 5, pp. 727-750, Sept. 1987.

[10] T. Sasao, "An application of multiple-valued logic to a design of programmable logic arrays," *ISMVL-78*, pp. 65-72, May 1978.

[11] T. Sasao, *Switching Theory for Logic Synthesis*, Kluwer Academic Publishers, 1999.

[12] T. Sasao, M. Matsuura, and Y. Iguchi, "Cascade realization of multiple-output function and its application to reconfigurable hardware," (draft).

[13] T. Villa and A. Sangiovanni-Vincentelli, "NOVA: State assignment for finite state machines for optimal two-level logic implementation," *IEEE TCAD*, Vol. CAD-9, No. 9, pp. 905-924, Sept. 1990.

[14] S. Yang and M. Ciesielski, "Optimum and suboptimum algorithm for input encoding and its relationship to logic minimization," *IEEE Trans. CAD*, Vol. CAD-10, No. 1 pp. 4-12, Jan. 1991.