Compact BDD Representations for Multiple-Output Functions and Their Application

Tsutomu SASAO[1,2], Munehiro MATSUURA[1], Yukihiro IGUCHI[3], and Shinobu NAGAYAMA[3]

[1]Department of Computer Science and Electronics, Kyushu Institute of Technology
[2]Center for Microelectronic Systems, Kyushu Institute of Technology
[3]Department of Computer Science, Meiji University

September 6, 2001

Abstract

This paper shows a new method to represent a multiple-output function: an 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 binary decision diagrams (BDDs) for ECFNs are never greater than corresponding SBDDs. The size of a BDD depends on the encoding of the outputs as well as the ordering of the variables. We conjecture that there exists an n-input 2^n-output function, where the optimal encoding produces BDDs with $2n+2$ nodes, while the worst encoding produces BDDs with 2^{n+1} nodes. We formulate an encoding problem and show a heuristic method. Experimental results using standard benchmark functions show that the sizes of BDDs can be reduced significantly by considering encodings.

Index term: Multiple-output function, encoding problem, BDD, SBDD, MTBDD, characteristic function.

1 Introduction

Logic networks usually have many outputs. In most cases, independent representations of each output are inefficient. Several methods exist to represent multiple-output functions: $F = (f_0, f_1, \ldots, f_{m-1}) : B^n \rightarrow B^m$, where $B = \{0,1\}$. In this paper, we consider methods to represent multiple-output functions by using compact binary decision diagrams (BDDs). There have been three previous methods to represent multiple-output functions by BDDs.

The first method is a multi-terminal binary decision diagram (MTBDD) [14]. In an MTBDD, each terminal is a binary vector of m bits. For an n-input m-output function, we can evaluate the function in $O(n + \log m)$ time. Unfortunately, MTBDDs tend to be too large to construct.

The second method is a binary decision diagram (BDD) for the characteristic function (CF) of the multiple-output function. The CF is a mapping $F : B^n \times B^m \rightarrow B$, where $F(\vec{a}, \vec{b}) = 1$ iff $(f_0(\vec{a}), f_1(\vec{a}), \ldots, f_{m-1}(\vec{a})) = \vec{b}$. The CF requires m auxiliary binary variables $\{z_0, z_1, \ldots, z_{m-1}\}$ that represent outputs. F shows the set of all the valid combinations of the inputs and the outputs. For example, the CF of a 4-output function is

$$F = (\bar{z}_0\bar{f}_0 \vee z_0 f_0)(\bar{z}_1\bar{f}_1 \vee z_1 f_1)(\bar{z}_2\bar{f}_2 \vee z_2 f_2)(\bar{z}_3\bar{f}_3 \vee z_3 f_3).$$

The size of a CF tends to be very large, since it involves $(n+m)$ binary variables. The advantage of the CF is its small evaluation time: For an n-input m-output function, we can evaluate the function in $O(n+m)$ time by using a BDD for CF. CFs are used in logic simulation [1], and multi-level logic optimization [6].

The third method is a shared binary decision diagram (SBDD) [14]. In many cases, SBDDs are smaller than corresponding MTBDDs and BDDs for CFs. To evaluate the function using an SBDD, we need $O(n \cdot m)$ time.

In this paper, we consider the fourth method to represent a multiple-output function, Encoded Characteristic Function for Non-zero outputs (ECFN). It represents a mapping $F : B^n \times B^u \rightarrow B$, where $u = \lceil \log_2 m \rceil$. $F(\vec{a}, \vec{b}) = 1$ 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 a 4-output function is

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

ECFNs can be used in FPGA design [10], logic emulation [2], and embedded systems [15].

As shown in later, a BDD for an ECFN is a generalization of an SBDD, and can be made smaller than the corresponding SBDD. So, it is useful for applications where the size is important.

2 ECFN and Encoding Problem

In this section, we define encoded characteristic functions for non-zero outputs (ECFNs) and formulate their encoding program [16].

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*

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

Table 2.1: 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

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.

Definition 2.3 *The size of a decision diagram (DD) is the number of nodes in the DD, including the terminal nodes.*

Example 2.1 *Consider the four-output function $F = (f_0, f_1, f_2, f_3)$, where $f_0 = 0$, $f_1 = x_0$, $f_2 = x_1$, and $f_3 = x_1 \vee x_0$. Encoding 1 in Table 2.1 produces the ECFN:*

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

In this case, we have

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

However, Encoding 2 in Table 2.1 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

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

Note that the minimum BDD using Encoding 1 requires 6 nodes, while the minimum BDD using Encoding 2 requires 7 nodes. (End of Example)

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

In this case, we need three auxiliary variables z_0, z_1, and z_2 to represent 8 outputs. Encoding 1 in Table 2.2 produces the ECFN:

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

Table 2.2: 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_7
0	1	0	f_2	f_3
0	1	1	f_3	f_2
1	0	0	f_4	f_5
1	0	1	f_5	f_4
1	1	0	f_6	f_6
1	1	1	f_7	f_1

Thus, we have

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

However, Encoding 2 in Table 2.2 produces the ECFN:

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

Thus, we have

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

Note that the minimum BDD using Encoding 1 requires 8 nodes, while the minimum BDD using Encoding 2 requires 16 nodes. (End of Example)

These two examples show the advantage of finding good encodings.

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

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

However, the size of the BDD is invariant under the complementation and/or renaming of the auxiliary variables. Thus, to find the encoding for an ECFN that has the smallest BDD, 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. Table 2.1 shows these three encodings.

Thus, we can formulate

Problem 2.1 *(Encoding problem for an ECFN) Given a multiple-output function $F : B^n \to B^m$, represent F by using $u = \lceil \log_2 m \rceil$ auxiliary binary variables so that the resulting BDD has the fewest nodes.*

Table 2.3: Sizes of BDDs for ECFNs in Theorem 2.1.

n	Encoding	
	Best	Worst
3	8	16
4	10	32
5	12	64
6	14	128
7	16	256
8	18	512
9	20	1024

To the best of our knowledge, Problem 2.1 has not been previously formulated. No good algorithm except for the exhaustive search is known.

Theorem 2.1 *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 a BDD for an ECFN with $2n + 2$ nodes, 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 2.3 *When $n = 3$, we have $f_0 = 0$, $f_1 = x_0$, $f_2 = x_1$, $f_3 = x_1 \vee x_0$, $f_4 = x_2$, $f_5 = x_2 \vee x_0$, $f_6 = x_2 \vee x_1$, $f_7 = x_2 \vee x_1 \vee x_0$. Note that these functions are the same as those in Example 2.2. The ECFN that requires the fewest nodes in the BDD is given by*

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

(End of Example)

Table 2.3 shows the sizes of the BDDs for ECFNs for the multiple-output functions defined in Theorem 2.1. The BDDs headed *Worst* were found among 100 random encodings. From this table, we have

Conjecture 2.1 *An n-input 2^n-output function exists that requires $2n+2$ and 2^{n+1} nodes in BDDs for the ECFN with the encoding optimized and un-optimized, respectively.*

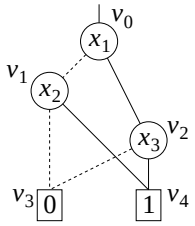
3 Application to Embedded Systems

3.1 Branching Program

Here, we consider the application to embedded systems, where the size of the memory is very important [3].

The *branching program method* realizes a logic function by a sequential network as follows:

1) Represent the given logic function by a BDD [4] (Fig. 3.1(a), where dotted lines show 0-edges and solid line show 1-edges).



(a)

v_0: if($x_1 == 0$) goto v_1;
else goto v_2;
v_1: if($x_2 == 0$) goto v_3;
else goto v_4;
v_2: if($x_3 == 0$) goto v_3;
else goto v_4;
v_3: return(0);
v_4: return(1);

(b)

Figure 3.1: Branching program method.

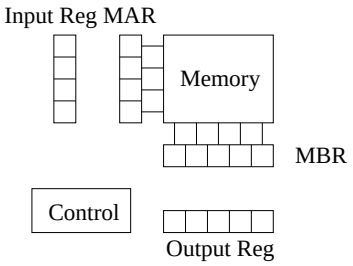


Figure 3.2: Architecture for Logic Simulator.

2) Replace each non-terminal node of the BDD with an *If then else* statement, and derive the branching program to represent f (Fig. 3.1(b)).
3) Implement the program by a general-purpose microprocessor.

To reduce the instruction fetch time, special sequential machines that traverse the BDD structure are proposed [8, 9, 5].

The branching program requires memory that is proportional to the number of nodes in the BDD. When constructing a BDD for an ECFN, we can reduce the size of the BDD by finding the optimal ordering of variables, as well as by finding the best encoding of the outputs. As will be shown in the experimental results, the sizes of the BDDs obtained in this way are, in many cases, smaller than other types of DDs. When all the auxiliary variables are adjacent to the root node, the BDD is equivalent to an ordinary SBDD.

3.2 Architecture for Reconfigurable Hardware

Multiple-output functions can be evaluated by the architecture shown in Fig. 3.2. In this architecture, the memory stores the data for the BDD, while the control part (a sequencer) traverses the BDD.

Example 3.1 *Consider the 4-output function:*

$$\begin{aligned} f_0 &= x_1 x_2 \\ f_1 &= x_1 x_2 \vee x_4 \\ f_2 &= x_1 x_2 \vee x_3 \end{aligned}$$

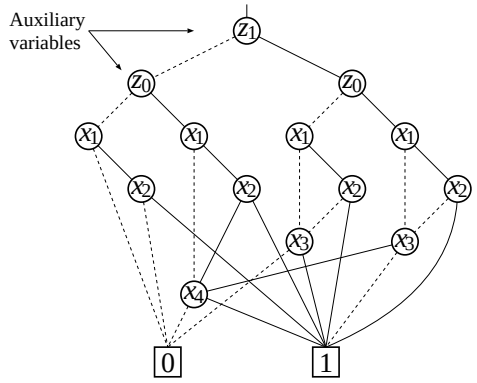


Figure 3.3: Original variable ordering for ECFN.

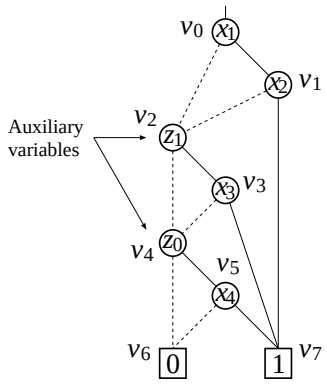


Figure 3.4: Optimized variable ordering for ECFN.

$$f_3 = x_1x_2 \vee x_3 \vee x_4.$$

Let the ECFN be

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

When the ordering of the variables is $(z_1, z_0, x_1, x_2, x_3, x_4)$*, we have the BDD with 16 nodes as shown in Fig. 3.3. However, when the ordering of the variables is* $(x_1, x_2, z_1, x_3, z_0, x_4)$*, we have the BDD with only 8 nodes as shown in Fig. 3.4.*

Table 3.1 shows the BDD data for Fig. 3.4. Each node has three basic attributes: an index of the variables and a pointer for each of the 0-edge and 1-edge. Also, index=0 denotes a terminal node. To evaluate the value of $f_0(1, 1, 1, 1)$*, we have to set the variables to* $(x_1, x_2, z_1, x_3, z_0, x_4) = (1, 1, 0, 1, 0, 1)$*. (End of Example)*

Table 3.1: BDD data for Fig. 3.4.

address	index	0-edge	1-edge
v_0	x_1	v_2	v_1
v_1	x_2	v_2	v_7
v_2	z_1	v_4	v_3
v_3	x_3	v_4	v_7
v_4	z_0	v_6	v_5
v_5	x_4	v_6	v_7
v_6	0	0	0
v_7	0	1	1

4 Encoding Algorithm for ECFN

In this part, we show a heuristic algorithm to encode m outputs by using $u = \lceil \log_2 m \rceil$ binary variables, so that the resulting BDD has the fewest nodes. It is similar to the encoding method that minimizes the number of products in sum-of-products expression (SOP) [16]. Because of space limitations, we only show the outline.

An encoding for an ECFN corresponds to an assignment of m output functions to the nodes of the u-dimensional cube.

Constraint matrix [6] is the output part of the minimized positional cubes [13]. We use the **Merit matrix** to find encodings. 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 adjacent nodes in the u-dimensional cube.

Algorithm 4.1 *(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* $Merit(j, k) \leftarrow -1 + \frac{u}{2^u}$*.*

The following algorithm produces a good encoding for SOPs.

Algorithm 4.2 *(Encoding of an ECFN for SOPs)*

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 the 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 does 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.*

Unfortunately, Algorithm 4.2 does not always work well for the minimization of BDDs. So, we modified it as follows:

Algorithm 4.3 *(Encoding for ECFNs for BDDs)*

1) Minimize the SBDD for the multiple-output function.

2) Make an ECFN for the natural encoding (i.e., f_0 is assigned to $00\cdots00$, f_1 is assigned to $00\cdots01$, f_2 is assigned to $00\cdots10$, etc.), and minimize the BDD.

3) Find an encoding by Algorithm 4.2 and make an ECFN. Minimize the BDD.

4) Return the smallest BDD among three produced from steps 1), 2), and 3).

Since we are using a heuristic algorithm [12] for BDD minimization, the minimized SBDD obtained by 1) can be smaller than the BDD obtained by 2) or 3).

5 Experimental Results

5.1 Benchmark Results

We implemented Algorithm 4.3 and minimized BDDs for various benchmark functions.

Table 5.1 compares sizes of DDs, where *Name* denotes the function name; *In* denotes the number of input variables; *Out* denotes the number of output variables; *MTBDD*, *BDD for CF*, *SBDD*, and *BDD for ECFN* denote sizes of the corresponding DDs obtained by Algorithm 4.3; entry – shows a function in which the MTBDD or BDD size was too large to be constructed. Table 5.1 shows that MTBDDs and BDDs for CF are often very large.

We optimized the BDD for ECFN by mixing the input variables and auxiliary variables. In the case of BDDs for CFs, to evaluate the logic function in $O(n+m)$ time, all the output variables must be located after the input variables they depend on. However, we dropped this restriction in the optimization of BDDs for CFs [17]. Even if we drop this restriction, in some cases, BDDs for CF are too large to construct: the entry with - shows such a DD.

For some functions such as *seq*, the size of the BDD for ECFN is less than a half of the corresponding SBDD. For other functions, such as *b9*, the sizes of the SBDDs are the same as those of BDDs for ECFNs. An SBDD is considered as the ECFN with the natural encoding. Thus, the natural encoding is the optimum encoding for the function such as *b9*. We could reduce the sizes of BDDs for 35 functions out of 43, or 81 percents of the functions.

Table 5.1: Sizes of various DDs.

Name	In	Out	MTBDD	BDD for CF	SBDD	BDD for ECFN
5xp1	7	10	255	73	79	76
amd	14	24	206	404	266	196
apex3	54	50	537	1786	983	854
apex7	49	37	–	–	302	300
b9	41	21	32720	1582	177	177
clip	9	5	139	99	99	81
cps	24	109	–	3728	1095	828
duke2	22	29	646	755	395	351
e64	65	65	131	2277	194	194
e1010	10	10	1551	2335	1423	1423
ex5	8	63	244	1125	342	302
ex7	16	5	636	239	90	88
exep	30	63	1278	2448	675	629
exps	8	38	286	1227	585	482
ibm	48	17	591586	2249	246	246
intb	15	7	735	758	608	608
jbp	36	57	509832	3129	467	466
k2	45	45	913	2860	1321	1167
mainpla	27	54	634	2836	1857	1017
mark1	20	31	4179	272	119	115
misex2	25	18	113	198	100	98
newtpla	15	5	69	69	54	50
opa	17	69	252	1369	428	364
p1	8	18	370	656	208	208
p3	8	14	247	392	134	134
pdc	16	40	19434	2733	596	590
pope	6	48	118	876	280	278
prom1	9	40	852	4252	2012	1479
prom2	9	21	578	2852	958	737
rckl	32	7	65	135	198	67
risc	8	31	56	257	99	84
seq	41	35	853	1350	1284	506
shift	19	16	196095	843	78	62
spla	16	46	11732	2121	628	605
t2	17	16	308	444	145	140
t3	12	8	33	83	66	41
t4	12	8	96	83	44	44
table3	14	14	457	1060	766	506
table5	17	15	442	1038	685	476
tms	8	16	68	221	125	99
ts10	22	16	589837	5954	163	83
vg2	25	8	134	153	90	82
xparc	41	73	3876	4745	1947	1237

The CPU time to obtain the encodings depends on the number of the outputs. The most time-consuming one was *cps*, which has 108 outputs. Given a minimized SOP of a multiple-output function, the time to obtain the encoding was about two minutes by a PC with an INTEL Pentium microprocessor (840MHz).

5.2 Prototype of Reconfigurable Hardware

In order to verify the performance of the architecture shown in Section 3, we developed a reconfigurable hardware using a commercially available FPGA board.

The specification of the FPGA board is as follows:

- FPGA: Altera FLEX10K100
- Clock frequency: 20MHz
- RAM: Static 4M Bits

In this prototype, to process one node of a BDD, we need $\alpha = 2$ clock cycles. Thus, to evaluate the function for an input pattern, we need $n \cdot m \cdot \alpha$ clocks, where n is the number of input variables, and m is the number of outputs of the function.

We used the Altera FPGA FLEX10K100, since it is readily available. However, we can use any logic circuit, e.g., CPLDs, because the control part is very simple.

6 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 BDD can be simplified by considering the encoding as well as the ordering of the variables. BDDs for ECFNs can be made smaller than the corresponding SBDDs. We formulated the encoding problem and presented a heuristic method. We also conjecture that there exists an n-input 2^n-output function that requires $2n + 2$ nodes in a BDD for one encoding, and 2^{n+1} nodes for other encoding.

We also developed a reconfigurable hardware consisting of a memory and a sequencer. The hardware is simple to implement, and the design corresponds to a minimization of a BDD for ECFN.

Currently, we are improving the encoding algorithm for ECFNs.

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. Prof. Jon T. Butler's comments improved the English presentation.

References

[1] P. Ashar and S. Malik, "Fast functional simulation using branching programs," *ICCAD'95*, pp. 408-412, Oct. 1995.

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

[3] F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, A. Sangiovanni-Vincentelli, E. M. Sentovich, and K. Suzuki, "Synthesis of software programs for embedded control applications," *IEEE Trans. CAD*, Vol. 18, No. 6, pp.834-849, June 1999.

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

[5] M. Davio, J-P Deschamps, and A. Thayse, *Digital Systems with Algorithm Implementation*, John Wiley and Sons, New York, 1983.

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

[7] W. Gunther and R. Drechsler, "Minimization of free BDDs," *Proc. of Asia and South Pacific Design Automation Conference*, Jan. 1999, pp. 323-326.

[8] Y. Iguchi, T. Sasao, M. Matsuura, and A. Iseno "A hardware simulation engine based on decision diagrams," *Asia and South Pacific Design Automation Conference (ASP-DAC'2000)*, Jan. 26-28, Yokohama, Japan.

[9] Y. Iguchi, T. Sasao, M. Matsuura, "Implementation of multiple-output functions using PQMDDs," *International Symposium on Multiple-Valued Logic*, pp.199-205, May 2000.

[10] 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.

[11] P. C. McGeer, K. L. McMillan, A. Saldanha, A. L. Sangiovanni-Vincentelli, and P. Scaglia, "Fast discrete function evaluation using decision diagrams," *ICCAD'95*, pp. 402-407, Nov. 1995.

[12] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 42-47, Santa Clara, CA, November 1993.

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

[14] T. Sasao and M. Fujita (ed.), *Representations of Discrete Functions*, Kluwer Academic Publishers 1996.

[15] T. Sasao, M. Matsuura, and Y. Iguchi, "Cascade realization of multiple-output function and its application to reconfigurable hardware," *International Workshop on Logic and Synthesis*, Lake Tahoe, June 2001, pp. 225-230.

[16] T. Sasao, "Compact SOP representations for multiple-output functions: An encoding method using multiple-valued logic," *International Symposium on Multiple-Valued Logic*, Warsaw, Poland, 2001, pp. 207-212.

[17] C. Scholl, R. Drechsler, and B. Becker, "Functional simulation using binary decision diagrams," *ICCAD'97*, pp. 8-12, Nov. 1997.

[18] S. Yang, *Logic synthesis and optimization benchmark users guide version 3.0*, MCNC, Jan. 1991.