IEICE TRANS. FUNDAMENTALS, VOL.E82–A, NO.11 NOVEMBER 1999

PAPER *Special Section on VLSI Design and CAD Algorithms*

Representations of Multiple-Output Functions Using Binary Decision Diagrams for Characteristic Functions

Hafiz Md. HASAN BABU[†], ***Nonmember and*** **Tsutomu SASAO**[†], ***Member***

SUMMARY This paper proposes a method to construct smaller binary decision diagrams for characteristic functions (BDDs for CFs). A BDD for CF represents an n-input m-output function, and evaluates all the outputs in $O(n+m)$ time. We derive an upper bound on the number of nodes of the BDD for CF of n-bit adders (adr$n$). We also compare complexities of BDDs for CFs with those of shared binary decision diagrams (SBDDs) and multi-terminal binary decision diagrams (MTBDDs). Our experimental results show: 1) BDDs for CFs are usually much smaller than MTBDDs; 2) for adr$n$ and for some benchmark circuits, BDDs for CFs are the smallest among the three types of BDDs; and 3) the proposed method often produces smaller BDDs for CFs than an existing method.

key words: *binary decision diagram (BDD), characteristic function (CF), multiple-output function, variable ordering, logic simulation, adder, bit-counting function, multiplier*

1. Introduction

Binary decision diagrams (BDDs) are compact representations of logic functions, and are useful for logic synthesis, time-division multiplexing (TDM) realization, test, verification, etc. [1],[6],[7],[12],[14],[16]. Shared binary decision diagrams (SBDDs), multi-terminal binary decision diagrams (MTBDDs), and BDDs for characteristic functions (BDDs for CFs) represent multiple-output functions. SBDDs are compact, but they require $O(m \cdot n)$ time to evaluate an n-input m-output function. MTBDDs evaluate all the outputs simultaneously, but they usually blow up in memory for large benchmark circuits. BDDs for CFs use CFs of multiple-output functions, and evaluate all the outputs in $O(n+m)$ time. A CF is a switching function representing the relation of inputs and outputs. Figure 1 shows the general structure of a BDD for CF. BDDs for CFs are usually much smaller than MTBDDs. The main applications of BDDs for CFs are logic simulation of digital circuits [2],[4],[5] and implicit state enumeration of finite state machines [3]. In this paper, we consider a method to construct compact BDDs for CFs. We present an algorithm to find a good ordering of input and output variables. We also derive an upper bound on the number of nodes of the BDD for CF of n-bit adders (adr$n$). The rest of the paper is organized as follows: Section 2 presents basic definitions. Section 3 introduces SBDDs, MTBDDs and BDDs for CFs of multiple-output functions, and compares their complexities. Section 4 describes an optimization algorithm for BDDs for CFs. Finally, Sect. 5 presents experimental results for adr$n$, bit-counting circuits (wgt$n$), n-bit multipliers (mlp$n$), and various benchmark circuits.

Manuscript received March 10, 1999.
Manuscript revised June 11, 1999.

[†]The authors are with the Department of Computer Science and Electronics, Kyushu Institute of Technology, Iizuka-shi, 820–8502 Japan.

2. Basic Definitions

This section contains some important definitions.

Definition 1: ***support***(f) is the set of input variables that the function f depends on. The ***size of the support*** is the number of variables in the support(f).

Example 1: Let $f(x_1, x_2, x_3) = x_1x_2x_3 \vee x_1x_2\bar{x}_3$. Then, support$(f) = \{x_1, x_2\}$, since f is also represented as $f = x_1x_2$. Thus, the size of the support is two. □

Definition 2: Let f_{i1} and f_{i2} be two output functions. ***The size of the union of the support for*** f_{i1} ***and*** f_{i2} is the number of support variables for f_{i1} and f_{i2}.

Example 2: Consider the 4-input 2-output function:

$$f_0(x_1, x_2, x_3, x_4) = \bar{x}_1x_2 \vee x_1\bar{x}_3 \vee \bar{x}_2x_3, \text{ and}$$
$$f_1(x_1, x_2, x_3, x_4) = x_1x_3 \vee \bar{x}_3x_4.$$

The size of the union of the support for f_0 and f_1 is 4, since x_1, x_2, x_3 and x_4 are the support variables for f_0 and f_1. □

Definition 3: Let $\boldsymbol{f} : \{0,1\}^n \rightarrow \{0,1\}^m$. The ***size of a decision diagram (DD)*** for a multiple-output function $\boldsymbol{f}$, denoted by size(DD, $\boldsymbol{f}$), is the total number of terminal and non-terminal nodes in the minimal DD.

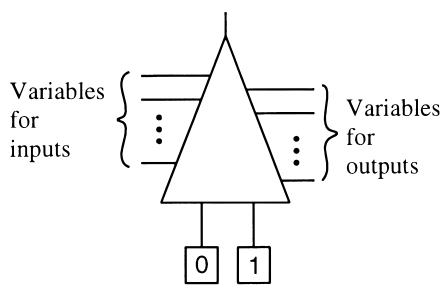


Fig. 1 General structure of a BDD for CF.

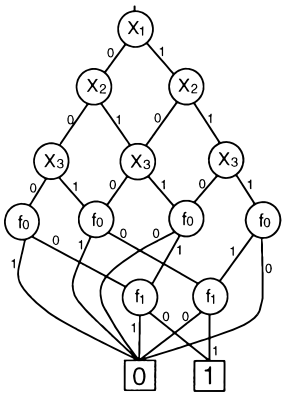


Fig. 2 BDD for CF of a 3-input 2-output bit-counting function (wgt3).

In the case of SBDDs, the size also includes the nodes for output selection variables.

Example 3: The size of the BDD for CF in Fig. 2 is 14. □

3. Binary Decision Diagrams for Multiple-Output Functions

In this section, we present shared binary decision diagrams (SBDDs), multi-terminal binary decision diagrams (MTBDDs) and binary decision diagrams for characteristic functions (BDDs for CFs). We also compare complexities of BDDs for CFs with those of SBDDs and MTBDDs.

3.1 SBDDs and MTBDDs

Shared binary decision diagrams (SBDDs) and multi-terminal binary decision diagrams (MTBDDs) represent multiple-output functions. An SBDD is a set of BDDs combined by a tree for output selection, while an MTBDD is a BDD with many terminal nodes. MTBDDs evaluate all the outputs simultaneously, but they are usually much larger than SBDDs [13].

3.2 BDDs for CFs

In this part, we will present definition and properties of BDDs for CFs.

Definition 4: Let $B = \{0, 1\}$. Let $\boldsymbol{a} \in B^n$, and $\boldsymbol{f}(\boldsymbol{a}) = (f_0(\boldsymbol{a}), f_1(\boldsymbol{a}), \ldots, f_{m-1}(\boldsymbol{a})) \in B^m$. Let $\boldsymbol{b} \in B^m$. The ***characteristic function*** (CF) F of a multiple-output function $\boldsymbol{f} = (f_0, f_1, \ldots, f_{m-1})$ is an $(n+m)$-variable switching function such that

$$F(\boldsymbol{a}, \boldsymbol{b}) = \begin{cases} 1 & \text{if } \boldsymbol{b} = \boldsymbol{f}(\boldsymbol{a}) \\ 0 & \text{otherwise.} \end{cases}$$

□

Table 1 2-input 2-output function.

Input		Output	
x_1	x_2	f_0	f_1
0	0	0	0
0	1	0	0
1	0	1	0
1	1	1	1

A CF of an n-input m-output function is a switching function with $n+m$ variables. In the CF, besides the input variables, one binary variable is used for each output function. A ***BDD for CF*** is a BDD representing the CF. In order to guarantee a fast evaluation, the output variables can appear on any path of the BDD for CF only after all the supports have appeared. Figure 2 shows the BDD for CF of a 3-input 2-output bit-counting function (wgt3), where x_1, x_2 and x_3 are input variables, and f_0 and f_1 are output variables. The BDD for CF in Fig. 2 shows that each path from the root to the terminal 1 corresponds to an input-output combination. The advantages of BDDs for CFs are: 1) they can represent large multiple-output functions; and 2) they can evaluate all the outputs in $O(n+m)$ time.

Definition 5: Let F be the characteristic function of an n-input m-output function. An input-output combination for F is ***valid*** if the output vector in the combination is produced when the input vector of the combination is applied.

Lemma 1: Let F be a characteristic function of $\boldsymbol{f} : \{0,1\}^n \to \{0,1\}^m$. Then, the number of valid input-output combinations for F is 2^n.

Example 4: Consider the 2-input 2-output function in Table 1. The valid input-output combinations are $(0,0,0,0)$, $(0,1,0,0)$, $(1,0,1,0)$ and $(1,1,1,1)$. □

Lemma 2: If the output variables appear only after all the supports have appeared, then an arbitrary n-input m-output function can be evaluated by a BDD for CF in $O(n+m)$ time. □

Note that if we drop the above restriction of the ordering of the variables, then we can't guarantee the evaluation time of $O(n+m)$.

3.2.1 Size of BDDs for CFs

In this part, we will present the sizes of BDDs for CFs of multiple-output functions. Since the CF of a multiple-output function is a special case of an $(n+m)$-variable switching function, we have the followings:

Theorem 1: Let $\boldsymbol{f}$ be an n-input m-output function. Then,

$$size(BDD \text{ for CF}, \boldsymbol{f}) \leq \min_{k=1}^{n+m} \{2^{(n+m)-k} - 1 + 2^{2^k}\}.$$

Table 2 Comparison of SBDDs, MTBDDs and BDDs for CFs.

	SBDD	MTBDD	BDD for CF
Upper bound on the size	$\min_{k=1}^{n}\{m \cdot 2^{n-k} - 1 + 2^{2^k}\}$	$\min_{k=1}^{n}\{2^{n-k} - 1 + r^{2^k}\}$	$min_{k=1}^{n+m}\{2^{(n+m)-k} - 1 + 2^{2^k}\}$
Asymptotic bound on the size	$O(\frac{m2^n}{n})$	$O(\frac{r^n}{n})$	$O(\frac{2^{n+m}}{n+m})$
Size for $\boldsymbol{f}_s$	$2m+1$	$2^{m+1}-1$	$3m+2$
Evaluation time	$O(m \cdot n)$	$O(n)$	$O(n+m)$

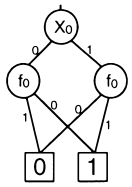


Fig. 3 BDD for CF of function $f_0 = x_0$.

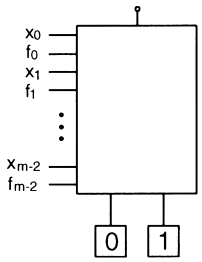


Fig. 4 BDD for CF of $(m-1)$ functions.

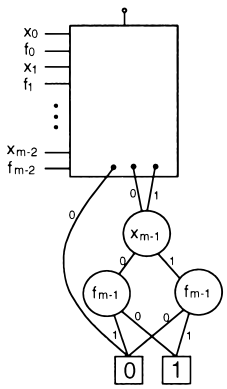


Fig. 5 BDD for CF of m functions.

Example 5: Let $\boldsymbol{f} : \{0,1\}^7 \rightarrow \{0,1\}^{10}$. Then, by Theorem 1, we have $size(BDD \text{ for CF}, \boldsymbol{f}) \leq 16639$. On the other hand, from Table 2, we have $size(SBDD, \boldsymbol{f}) \leq 335$. □

Theorem 2: Let $\boldsymbol{f}_s = (f_0, f_1, \ldots, f_{m-1})$ represent m functions, where $f_i = x_i$ $(i = 0, 1, \ldots, m-1)$. Then, $size(BDD \text{ for CF}, \boldsymbol{f}_s) \leq 3m + 2$.

Proof: The proof is done by mathematical induction.
1) Base: For $m = 1$, the CF of the function $f_0 = x_0$ is realized by a BDD for CF with five nodes as shown in Fig. 3.
2) Induction: Assume that the hypothesis is true for $k = m - 1$ functions. That is, the CF of $m - 1$ functions is realized by the BDD for CF in Fig. 4 with $3m-1$ nodes. In Fig. 4, first remove the constant 0 and constant 1. Second, attach variables x_{m-1}, f_{m-1}, and corresponding three non-terminal nodes, as well as nodes for constant 0 and constant 1. Then, we have the diagram in Fig. 5. It is clear that Fig. 5 shows the BDD for CF of m functions with $3m + 2$ nodes which has three more non-terminal nodes than Fig. 4. Thus, from 1) and 2), we have the theorem. □

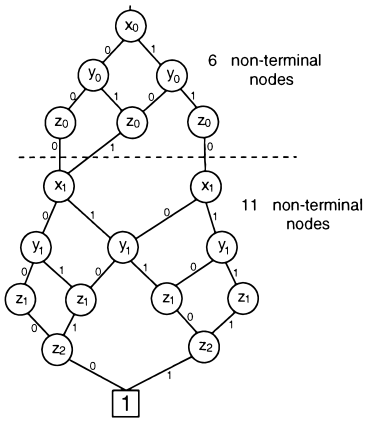


Fig. 6 BDD for CF of adr2.

Note that the size of the MTBDD for the functions f_s is exponential, while that for the BDD for CF and the SBDD are linear (Table 2).

Definition 6: Let ***adr****n* be a $2n$-input $(n+1)$-output function that computes the sum of two n-bit numbers.

Theorem 3: $size(BDD \text{ for CF}, \text{adr}n) \leq 9n+1$ $(n \geq 2)$.

Proof: Suppose that the variables of the adr$n$ are assigned as follows:

$$\begin{array}{rr} & x_{n-1}\ x_{n-2},\ldots,x_2\ x_1\ x_0 \\ +) & y_{n-1}\ y_{n-2},\ldots,y_2\ y_1\ y_0 \\ \hline & z_n\ z_{n-1}\ z_{n-2},\ldots,z_2\ z_1\ z_0 \end{array}$$

We will use mathematical induction to prove the theorem.
1) Base: As shown in Fig. 6, adr2 is represented by using 17 non-terminal nodes and two terminal nodes. In the figure, only 1-paths are shown, and constant 0 and 0-paths are omitted for simplicity. Note that x_0 and y_0 are near to the root node, and z_2 (the output variable representing the most significant bit of adr2) is the nearest to the constant 1 node.
2) Induction: Suppose that adr$n$ is represented by using $(9n - 1)$ non-terminal nodes and two terminal nodes. Also, assume that the variable z_n is the nearest to the constant 1 node. Let v_0 and v_1 be the nodes of z_n, where edges e_0 and e_1 are connecting to the constant 1, respectively. This situation is shown in Fig. 7. In Fig. 7, first remove the variable z_n, e_0, e_1, and constants. Second, attach variables x_n, y_n, z_n, and z_{n+1}, and cor-

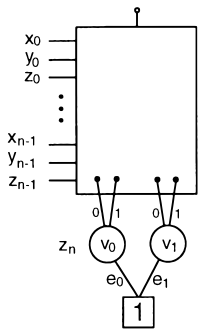


Fig. 7 BDD for CF of adr$n$.

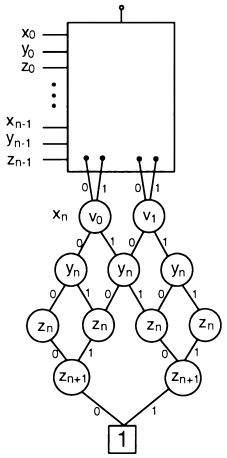


Fig. 8 BDD for CF of adr$(n+1)$.

responding 9 non-terminal nodes, as well as constant nodes. Then, we have the diagram shown in Fig. 8. Note that Fig. 8 has 9 more nodes than Fig. 7. It is clear that Fig. 8 represents the characteristic function of adr$(n+1)$, and has $(9n-1)+9+2 = 9(n+1)+1$ nodes. In this case, the ordering of the variables is $(x_0, y_0, z_0, x_1, y_1, z_1, \ldots, x_n, y_n, z_n, z_{n+1})$. Thus, from 1) and 2), we have the theorem. □

3.3 Comparison of Various BDDs

BDDs are useful for various applications [2]–[5],[12],[14]. Sometimes different BDDs can be used for the same application. So, it is necessary to know the properties of BDDs. The size of the BDD is important to represent functions compactly, while the evaluation time for the BDD is useful for logic simulation. Table 2 compares the sizes and the evaluation time of SBDDs, MTBDDs and BDDs for CFs of an n-input m-output function. In the table, r denotes the number of distinct output vectors for the outputs, and $\boldsymbol{f}_s = (f_0, f_1, \ldots, f_{m-1})$, where $f_i = x_i$ $(i = 0, 1, \ldots, m-1)$.

4. Construction of Compact BDDs for CFs

The construction of compact BDDs for CFs is useful for efficient representations of multiple-output functions. In this section, we will present a method to construct compact BDDs for CFs.

4.1 Formulation of the Problem

Definition 7: A BDD for CF is ***minimum*** iff it contains the minimum number of nodes.

Problem 1: Let $u_1, u_2, \ldots, u_k$ be the variables. Let $Order[k] = (u_{e_1}, u_{e_2}, \ldots, u_{e_k})$ be a permutation of the k variables. Let $size(BDD \text{ for CF}, \boldsymbol{f})$ be the total number of nodes in the BDD for CF for a certain $Order[k]$ of the variables. Find a variable ordering $Order[k] = (u_{e_1}, u_{e_2}, \ldots, u_{e_k})$ for a given multiple-output function $\boldsymbol{f}$ such that the $size(BDD \text{ for CF}, \boldsymbol{f})$ is the minimum.

In general, it is very time-consuming to find the best ordering of variables of the BDD for CF. So, we will compute a good variable ordering from the initial one by using the modified sifting algorithm (Sect. 4.5). To generate a good initial ordering, we will apply the following methods: i) ordering of output variables (Sect. 4.2); ii) interleaving based sampling schemes for ordering of input variables (Sect. 4.3); and iii) interleaving method for input variables and output variables (Sect. 4.4).

4.2 Ordering of Output Variables

Output functions are ordered so that the outputs with many support variables in common are adjacent. We will use the following strategies:

Strategy 1: f_i and f_j are the candidates of a pair of output functions if $support(f_i) \bigcap support(f_j) \neq \phi$.

Strategy 2: Let $s(f_{i1}, f_{i2})$ be the size of the union of the support for f_{i1} and f_{i2}. Then, (f_{i1}, f_{i2}) is a candidate of a pair of output functions if $s(f_{i1}, f_{i2})$ for (f_{i1}, f_{i2}) is the smallest among all $s(f_{i1}, f_{i2})$. Apply the same idea to the rest of functions recursively to find a good partition of the output functions.

Example 6: Consider the 4-input 4-output function:

$$f_0(x_1, x_2, x_3, x_4) = \bar{x}_1 x_2 \vee x_1 \bar{x}_3 \vee \bar{x}_2 x_3,$$
$$f_1(x_1, x_2, x_3, x_4) = x_3 \bar{x}_4,$$
$$f_2(x_1, x_2, x_3, x_4) = x_1 x_3 \vee \bar{x}_3 x_4, \text{ and}$$
$$f_3(x_1, x_2, x_3, x_4) = x_4.$$

There are six pairs of output functions. The sizes of the supports for these pairs of output functions are: $s(f_0, f_1) = s(f_0, f_2) = s(f_0, f_3) = 4$, $s(f_1, f_2) = s(f_2, f_3) = 3$, and $s(f_1, f_3) = 2$. Since $s(f_1, f_3) = 2$ is the smallest among all $s(f_i, f_j)$, (f_1, f_3) is the candidate of a pair. The remaining outputs are f_0 and f_2. Thus, (f_0, f_2) is the another pair. Therefore, we have the partition of output functions as follows: $\{(f_1, f_3), (f_0, f_2)\}$. □

Algorithm 1: (Ordering of output functions)

1. Find a good partition of output functions using Strategies 1 and 2.
2. Order the output functions with the pairs of outputs of a good partition.

Example 7: Consider the functions in Example 6. Since $\{(f_1, f_3), (f_0, f_2)\}$ is a good partition of output functions, the ordering of outputs is (f_1, f_3, f_0, f_2). □

4.3 Interleaving Based Sampling Schemes for Ordering of Input Variables

The sizes of BDDs are sensitive to orderings of input variables. Dynamic reordering methods are useful to order the input variables [11]. However, such methods are extremely time-consuming, and can fail to construct the BDDs for many functions [8],[9]. In the real life, many practical logic circuits are multiple outputs [12]. So, it is important to find the same good variable ordering for different output functions, since most of the BDD-based CAD tools handle multiple-output functions at the same time. In this part, we will present a method to order the inputs of multiple-output functions. We consider the sampling methods [9],[10] for computing variable orderings of SBDDs, where a sample corresponds to a group of output functions, and each SBDD represents a sample. Then, we use an interleaving method [8] to find a good ordering of the input variables for output functions from the variable orderings of compact SBDDs. The algorithm for the proposed method is shown in Fig. 12. The input variables that greatly affect the size of the BDD are called ***influential***. The influential variables should be the higher positions in a good variable ordering.

Definition 8: A ***sample*** is a multiple-output functions in which outputs with the common support variables are usually adjacent. These functions form a part of total functions. The ***size of a sample*** is the number of outputs in the sample.

Example 8: Consider the functions in Example 6. (f_0, f_2) can be a sample, since $support(f_0) = \{x_1, x_2, x_3\}$ and $support(f_2) = \{x_1, x_3, x_4\}$. The size of (f_0, f_2) is 2. □

Definition 9: Let G and H be two samples. The ***support correlation*** between G and H is the number of common support variables.

4.3.1 Generating Samples from Output Functions

In this part, we will present a technique to generate samples from output functions.

Algorithm 2: (Generating samples)

1. Order the outputs using Algorithm 1, and make an initial sample with the ordered outputs.
2. Check the size of the sample, and do the process of generating samples by using Step 3 only if the size of the sample is larger than the expected one, otherwise stop the process for this sample.
3. Check the supports of the outputs of the sample. If all the outputs depend on all the inputs, then goto Step 4, otherwise goto Step 5.
4. Randomly divide the sample into some such that the construction of the SBDD for each sample is easy† to handle.
5. Divide the sample into two such that the outputs with common support variables are in the same sample, and the support correlation between samples is small†. Return to Step 2 for each sample.

Example 9: Consider the functions in Example 6. (f_1, f_3) and (f_0, f_2) are two samples, since $support(f_0) = \{x_1, x_2, x_3\}$, $support(f_1) = \{x_3, x_4\}$, $support(f_2) = \{x_1, x_3, x_4\}$, and $support(f_3) = \{x_4\}$. □

4.3.2 Interleaving the Variable Orderings of Samples

In the previous part, we have presented a method to generate samples from the output functions. Now, we construct the compact SBDD for each sample by using the sifting algorithm starting with the initial variable ordering, and obtain the variable ordering for the sample from the SBDD. Then, we interleave the variables of the variable orderings from the highest to the lowest priority of the samples as shown in Fig. 12. Note that a sample has the highest priority if the size of the SBDD for the sample is the largest.

Example 10: Consider the functions in Example 6. (x_4, x_3, x_1, x_2) and (x_3, x_4, x_2, x_1) are the variable orderings in Figs. 9 and 10 for samples (f_1, f_3) and (f_0, f_2), respectively. The sample (f_0, f_2) has the highest priority, since the size of the SBDD for this sample is the largest. Figure 11 shows that (x_3, x_4, x_2, x_1) is a good variable ordering for (f_0, f_1, f_2, f_3) which is computed from the variable orderings of samples (f_1, f_3) and (f_0, f_2) by using an interleaving method [8]. □

4.4 Interleaving Method for Input Variables and Output Variables

In Sects. 4.2 and 4.3, we have presented methods to find good orderings of the input variables and the output variables. In Example 7 and in Fig. 11, we have shown that (f_1, f_3, f_0, f_2) is a good ordering of the outputs, and (x_3, x_4, x_2, x_1) is a good ordering of the inputs. In this part, we will present a method to find the relative position of inputs and outputs. To find the relative position of variables, we use the following strategy:

Strategy 3: For any output function, immediately after all the support variables appear, we place the variables for this output.

†Note that the size of a sample should not be too small so that the number of samples is large. To solve this problem, we finally combine some smaller samples together to get a considerable number of samples.

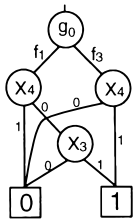


Fig. 9 SBDD with the variable ordering for sample (f_1, f_3) obtained by the sifting algorithm.

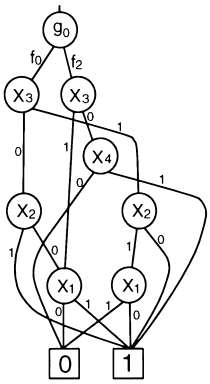


Fig. 10 SBDD with the variable ordering for sample (f_0, f_2) obtained by the sifting algorithm.

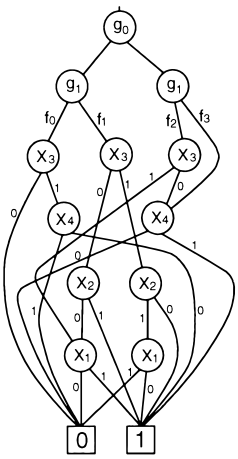


Fig. 11 SBDD with the variable ordering for $\boldsymbol{f} = (f_0, f_1, f_2, f_3)$ obtained from the variable orderings of samples (f_1, f_3) and (f_0, f_2) by using an interleaving method [8].

```
Procedure sift_m(f : {0,1}^n → {0,1}^m) {
   S ← Set of samples obtained by using Algorithm 2;
   Count ← 0; z ← φ;
for (each sample in S) do {
    Construct the compact SBDD by using the sifting
    algorithm starting with the initial variable ordering;
    Count ← Number of nodes in the SBDD;
    z ← Ordering of inputs in the SBDD;
 return Count;
 return z;
   }
 }
Procedure Interleave_sift_m(f : {0,1}^n → {0,1}^m) {
  Z ← φ; /*  "Z" denotes a set of orderings for
             input variables of the ordered samples */
  I ← φ; /*  "I" denotes the resulting order of inputs */
     sift_m(f);
Make an order of the samples in descending order of the
numbers of nodes in the "Count";
  Z ← Orderings of the input variables for the ordered
  samples;
while Z ≠ φ do {
    for (each variable order z from the beginning of
     Z) do {
       Choose the input variable x from the top of z;
       if (x is not in I) then {
         if (x is top in z) then
           Insert x into top of  I;
              }
         else
           x is already in I, and let y be a variable just before
           x in z and (y is not in I);
           Insert y before x in I; /* y is inserted before x, since
                y is more influential than x in z */
       }
    }
  return I;
  }
```

Fig. 12 Pseudocode for interleaving based sampling schemes for the ordering of input variables.

Example 11: Consider the two-bit adder (adr2) shown below:

$$\begin{array}{rr} & x_1\ x_0 \\ +) & y_1\ y_0 \\ \hline & z_2\ z_1\ z_0 \end{array}$$

The support for z_0 is $\{x_0, y_0\}$, and the supports for z_1 and z_2 are $\{x_0, y_0, x_1, y_1\}$. Also, z_2, z_1, and z_0 are partially symmetric with respect to $\{x_0, y_0\}$ and $\{x_1, y_1\}$. Thus, the reasonable ordering for the input and output variables would be $(x_0, y_0, z_0, x_1, y_1, z_1, z_2)$. □

4.5 Algorithm for Ordering the Variables

In this part, we will present a method to optimize BDDs for CFs using the modified sifting algorithm.

Algorithm 3: (Optimization of BDDs for CFs)

1. Make an initial ordering for the variables of the BDD for CF using Algorithm 1, Procedures in Fig. 12 and Strategy 3.
2. Select a variable from the initial ordering, and use the sifting algorithm [11] to find the position of the variable that fits Strategy 3 to minimize the size of the BDD for CF.
3. Do Step 2 until all the variables from the initial ordering have been checked, and choose the smallest BDD for CF.

Example 12: Consider the 4-input 4-output function in Example 6. In this example, (x_3, x_4, x_2, x_1) is a good ordering of the inputs, and (f_1, f_3, f_0, f_2) is a good ordering of the outputs. Since $support(f_1) = \{x_3, x_4\}$, f_1 appears after $\{x_3, x_4\}$. Next, $support(f_3) = \{x_4\}$, f_3 appears after f_1. Finally, $support(f_0) = \{x_1, x_2, x_3\}$ and $support(f_2) = \{x_1, x_3, x_4\}$, f_0 and f_2 appear in the last. Thus, an initial ordering for the input and output variables is $(x_3, x_4, f_1, f_3, x_2, x_1, f_0, f_2)$. □

Table 3 Sizes of DDs to represent adr$n$.

Circuit name	In	Out	SBDD	MTBDD	BDD for CF
adr2	4	3	15	18	19
adr3	6	4	25	44	28
adr4	8	5	35	98	37
adr5	10	6	45	208	46
adr6	12	7	55	430	55
adr7	14	8	65	876	64
adr8	16	9	75	1770	73

In: number of inputs; Out: number of outputs.

Table 4 Sizes of DDs to represent wgt$n$.

Circuit name	In	Out	SBDD	MTBDD	BDD for CF
wgt2	2	2	7	6	10
wgt3	3	2	11	10	14
wgt4	4	3	19	15	22
wgt5	5	3	27	21	28
wgt6	6	3	37	28	38
wgt7	7	3	47	36	44
wgt8	8	4	64	45	57
wgt9	9	4	80	55	67
wgt10	10	4	98	66	79

5. Experimental Results

We implemented C programs to construct SBDDs, MTBDDs and BDDs for CFs. SBDDs and MTBDDs were simplified by the sifting algorithm [11], while BDDs for CFs were simplified by Algorithm 3. The numbers of terminal nodes in the SBDD, MTBDD and BDD for CF are at most 2, 2^m, and 2, respectively, where m is the number of output functions.

Table 3 compares sizes of DDs of adr$n$ (Definition 6). From this table, we can observe that, for $2 \leq n \leq 8$,

$$size(SBDD, \text{adrn}) = 10n - 5,$$
$$size(MTBDD, \text{adrn}) = 7 \cdot 2^n - 2n - 6, \text{ and}$$
$$size(BDD \text{ for CF}, \text{adrn}) = 9n + 1.$$

Table 3 also shows that, for $n = 7$ to 8, the BDD for CF is the smallest.

Table 4 compares sizes of DDs of ***wgt****n*, where wgt$n$ is an n-input $(\lfloor \log_2 n \rfloor + 1)$-output function that counts the number of 1's in the inputs, and represents it by a binary number. From this table, we can observe that, for $2 \leq n \leq 10$, MTBDD is the smallest. This table also shows that, for $7 \leq n \leq 10$,

$$size(BDD \text{ for CF}, \text{wgtn}) < size(SBDD, \text{wgtn}).$$

Table 5 compares sizes of DDs of n-bit multipliers (***mlp****n*), where mlp$n$ is a $2n$-input $2n$-output function computing $X \times Y$. From this table, we can observe that, for $2 \leq n \leq 6$, SBDD is the smallest, and for $n = 6$,

$$size(BDD \text{ for CF}, \text{mlpn}) < size(MTBDD, \text{mlpn}).$$

Table 6 compares sizes of DDs of various benchmark circuits. In this table, BDDs for CFs are usually much smaller than MTBDDs. However, for some circuits, MTBDDs are the smallest, e.g. mainpla. For apex5, cps, c499, c880, c1908, soar, x1, and x3, MTBDDs are too large to be constructed, while BDDs for CFs are smaller and easy to be constructed. Furthermore, for clip, c499, f51m, misex3, and 5xp1, BDDs for CFs are the smallest among the three types of DDs. Sometimes BDDs for CFs are much smaller than other

Table 5 Sizes of DDs to represent mlp$n$.

Circuit name	In	Out	SBDD	MTBDD	BDD for CF
mlp2	4	4	15	19	28
mlp3	6	6	51	82	100
mlp4	8	8	150	330	360
mlp5	10	10	423	1332	1346
mlp6	12	12	1200	5270	5059

Table 6 Sizes of DDs to represent various benchmark circuits.

Circuit name	In	Out	SBDD	MTBDD	BDD for CF
apex5	117	88	1167	–	5618
chkn	29	7	281	220	264
clip	9	5	111	139	96
cordic	23	2	78	45	49
cps	24	109	1095	–	4920
count	35	16	98	1298	147
c432	36	7	1298	7511	1693
c499	41	32	31732	–	16840
c880	60	26	4168	–	388462
c1908	33	25	8741	–	115934
duke2	22	29	366	639	815
f51m	8	8	76	511	60
in7	26	10	107	299	185
mainpla	27	54	1869	629	1178
misj	35	14	56	4656	109
misex1	8	7	44	28	55
misex3	14	14	557	2910	531
sao2	10	4	90	69	81
signet	39	8	1449	7347	2950
soar	83	94	632	–	3719
tial	14	8	701	697	825
x1	51	35	586	–	2061
x3	135	99	715	–	4396
xparc	41	73	1949	3861	2349
5xp1	7	10	79	255	74

" – " indicates memory overflow.

Table 7 Sizes of BDDs for CFs of various benchmark circuits generated by Scholl-Drechsler-Becker [5] and the Proposed method.

Circuit name	In	Out	Scholl *et al.* [5]	Proposed method
apex7	49	37	2479	2260
b9	41	21	735	680
c432	36	7	1730	1693
c499	41	32	17017	16840
c880	60	26	393633	388462
c1908	33	25	117231	115934
clip	9	7	102	96
count	35	16	184	162
f51m	8	8	65	60
misex1	8	7	59	55
misex2	25	18	230	189
rd73	7	3	42	44
rd84	8	4	52	57
sao2	10	4	86	81
vg2	25	8	140	127
5xp1	7	10	74	74

DDs. For example, the sizes of the BDD for CF and the SBDD for c499 are 16840 and 31732, respectively, while the MTBDD for c499 is too large to be constructed. Note that c880 is one of the most complex benchmark circuits in Table 6, and our program required 86.2 CPU seconds to order the variables, and to construct the BDD for CF on a JU1/170 with 160 MB of main memory (Sun Ultra1-170 compatible workstation).

Table 7 compares the sizes of BDDs for CFs of various benchmark circuits generated by Scholl-Drechsler-Becker [5] and the proposed method. It shows that the proposed method often generates smaller BDDs for CFs than Scholl-Drechsler-Becker [5].

6. Concluding Remarks

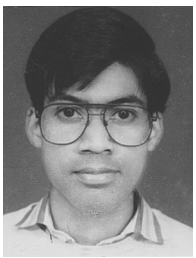
In this paper, we have proposed a method to construct smaller binary decision diagrams for characteristic functions (BDDs for CFs) to represent multiple-output functions. We have compared the sizes of SBDDs, MTBDDs and BDDs for CFs. SBDDs evaluate outputs in $O(m \cdot n)$ time, while MTBDDs and BDDs for CFs evaluate outputs in $O(n)$ time and $O(n+m)$ time, respectively. Experimental results show that, in most cases, BDDs for CFs are much smaller than MTBDDs. However, BDDs for CFs are usually larger than the corresponding SBDDs. For some benchmark circuits (e.g. c499), BDDs for CFs are the smallest among the three types of BDDs. We have shown three types of circuits: 1) n-bit adders (adr$n$), where BDDs for CFs are the smallest; 2) bit-counting circuits (wgt$n$), where MTBDDs are the smallest; and 3) n-bit multipliers (mlp$n$), where SBDDs are the smallest. We have also derived upper bounds on the sizes of SBDDs, MTBDDs and BDDs for CFs of adr$n$. Furthermore, the experimental results have shown that the proposed method often produces smaller BDDs for CFs than an existing method. BDDs for CFs are used for logic simulation [2],[4],[5], e.g., c880 is one of the most complex benchmark circuits in Scholl-Drechsler-Becker [5], and their simulator required 29.37 CPU seconds for the evaluation of the 500,000 random input vectors using a BDD for CF on a Sun Sparc 20 workstation (256 MB physical memory). An SBDD-based simulator can be faster for some functions. However, the simulator based on the BDD for CF should be faster than the SBDD-based one when there are no page faults in the physical memory and no misses in the Translation Lookaside Buffer (TLB) during function evaluation [4].

Acknowledgements

This work was supported in part by a Grant in Aid for Scientific Research of the Ministry of Education, Science, Culture and Sports of Japan. The authors thank to the constructive comments of the reviewers. This paper is based on [15].

References

[1] S.B. Akers, "Binary decision diagrams," IEEE Trans. Comput., vol.C-27, no.6, pp.509–516, June 1978.
[2] P. Ashar and S. Malik, "Fast functional simulation using branching programs," Proc. International Conference on Computer-Aided Design, pp.408–412, Nov. 1995.
[3] H. Touati, H. Savoj, B. Lin, R.K. Brayton, and A.L. Sangiovanni-Vincentelli, "Implicit state enumeration of finite state machines using BDDs," Proc. International Conference on Computer-Aided Design, pp.130–133, Nov. 1990.
[4] P.C. McGeer, K.L. McMillan, A. Saldanha, A.L. Sangiovanni-Vincentelli, and P. Scaglia, "Fast discrete function evaluation using decision diagrams," International Workshop on Logic Synthesis, pp.6.1-6.9, May 1995. Also, Proc. International Conference on Computer-Aided Design, pp.402–407, Nov. 1995.
[5] C. Scholl, R. Drechsler, and B. Becker, "Functional simulation using binary decision diagrams," Proc. International Conference on Computer-Aided Design, pp.8–12, Nov. 1997.
[6] R.E. Bryant, "Graph-based algorithms for Boolean function manipulation," IEEE Trans. Comput., vol.C-35, no.8, pp.677–691, Aug. 1986.
[7] T. Sasao and J.T. Butler, "A method to represent multiple-output switching functions by using multi-valued decision diagrams," Proc. IEEE International Symposium on Multiple-Valued Logic, pp.248–254, May 1996.
[8] H. Fujii, G. Ootomo, and C. Hori, "Interleaving based variable ordering methods for ordered binary decision diagrams," Proc. International Conference on Computer-Aided Design, pp.38–41, Nov. 1993.
[9] J. Jain, W. Adams, and M. Fujita, "Sampling schemes for computing OBDD variable orderings," Proc. International Conference on Computer-Aided Design, pp.631–638, Nov. 1998.
[10] A. Slobodová and C. Meinel, "Sample method for minimization of OBDDs," Proc. International Workshop on Logic Synthesis, pp.311–316, May 1998.
[11] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," Proc. International Conference on Computer-Aided Design, pp.42–47, Nov. 1993.
[12] T. Sasao, ed., Logic Synthesis and Optimization, Kluwer Academic Publishers, Boston, 1993.
[13] Hafiz Md. Hasan Babu and T. Sasao, "Shared multi-terminal binary decision diagrams for multiple-output functions," IEICE Trans. Fundamentals, vol.E81-A, no.12, pp.2545–2553, Dec. 1998.
[14] Hafiz Md. Hasan Babu and T. Sasao, "Time-division multiplexing realizations of multiple-output functions based on shared multi-terminal multiple-valued decision diagrams," IEICE Trans. Inf. & Syst., vol.E82-D, no.5, pp.925–932, May 1999.
[15] Hafiz Md. Hasan Babu and T. Sasao, "Representations of multiple-output functions by binary decision diagrams for characteristic functions," Proc. The Eighth Workshop on Synthesis And System Integration of MIxed Technologies (SASIMI'98), pp.101–108, oct. 1998.
[16] Hafiz Md. Hasan Babu and T. Sasao, "Shared multiple-valued decision diagrams for multiple-output functions," Proc. IEEE International Symposium on Multiple-Valued Logic (ISMVL'99), pp.166–172, May 1999.



Hafiz Md. Hasan Babu was born in Bangladesh. He received the M.Sc. degree in Computer Science and Engineering from the Technical University of Brno, Czechoslovakia, in 1992. He has been with the Department of Computer Science and Engineering, Khulna University, Bangladesh since 1993. In 1995, he was at the Asian Institute of Technology (AIT), Thailand under the DAAD Fellowship from the Federal Republic of Germany. He is currently working towards the Ph.D. degree with the Japanese Government Scholarship at the Kyushu Institute of Technology, Iizuka, Japan. His research interests include logic synthesis and representations of logic functions.

Tsutomu Sasao received the B.E., M.E., and Ph.D. degrees in Electronic Engineering from Osaka University, Osaka Japan, in 1972, 1974, and 1977, respectively. He was with Osaka University Japan, IBM T.J. Watson Research Center and Naval Postgraduate School in Monterey, California. Now, he is a Professor of 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 many books on logic design including, "Logic Synthesis and Optimization," "Representation of Discrete Functions," and "Switching Theory for Logic Synthesis," Kluwer Academic Publishers 1993, 1996, and 1999, respectively. He has served Program Chairman for the IEEE International Symposium on Multiple-Valued Logic (ISMVL) many times. Also, he was the Symposium Chairman for the ISMVL-98 held in Fukuoka, Japan in 1998. He received the NIWA Memorial Award in 1979, and Distinctive Contribution Awards from IEEE Computer Society MVL-TC in 1987 and 1996. Now, he is an associate editor of IEEE Transactions on Computers. He is a Fellow of IEEE.