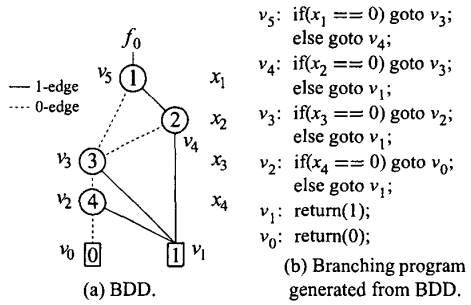
Evaluation of Multiple-Output Logic Functions using Decision Diagrams

Yukihiro IGUCHI[1], Tsutomu SASAO[2,3], and Munehiro MATSUURA[2]
[1]Department of Computer Science, Meiji University
[2]Department of Computer Science and Electronics, Kyushu Institute of Technology
[3]Center for Microelectronic Systems, Kyushu Institute of Technology

Abstract— This paper shows four different methods to evaluate multiple-output logic functions using decision diagrams: Shared BDD (SBDD), Multi-Terminal BDD (MTBDD), BDD for characteristic functions (CF), and BDDs for Encoded Characteristic Function for Non-zero outputs (ECFNs). Methods to compute average evaluation time for each type of decision diagrams are presented. By experimental analysis using benchmark functions, the number of nodes and average evaluation time are compared. Our results show that BDDs for ECFNs outperforms MTBDDs, BDDs for CFs, and SBDDs with respect to both number of nodes and computation time. The sizes of BDDs for ECFNs are smaller than for MTBDDs, BDDs for CFs, and SBDDs.



(a) BDD. (b) Branching program generated from BDD.

Fig. 1. BDD and branching program.

I. Introduction

Various kinds of decision diagrams (DDs) exist to represent multiple-output logic functions. Among them, a multi-terminal BDD (MTBDD), a shared BDD (SBDD), and a BDD representing the characteristic function (BDD for CF) are popular. For a BDD for CF or an MTBDD, the evaluation time is $O(n+m)$, where n is the number of input variables, and m is the number of output variables. For an SBDD, the evaluation time is $O(n \cdot m)$. BDDs for CFs and MTBDDs are suitable for high speed evaluation. Unfortunately, the sizes of these BDDs tend to be too large. Thus, we have to resort to SBDDs, which require longer evaluation time.

In [7], a new data structure, a BDD for encoded characteristic function for non-zero outputs (ECFN) is introduced. In this paper, we show that by using BDDs for ECFNs, logic evaluation can be more than two times faster than by using SBDDs. Also, the size of the memory is smaller than by using SBDDs.

II. Function Evaluation using BDDs

One method to represent a logic function is the branching program [1]. Fig. 1 shows a method to convert a BDD into a branching program. For a given logic function, construct a BDD, as shown in Fig. 1(a). Then, replace each non-terminal node by an *if then else* statement. The result is a branching program, as shown in Fig. 1(b). Then, by implementing this program on a computer, we can evaluate the logic function.

The time to evaluate a logic function for a given input is proportional to the number of non-terminal nodes that appear on the path from the root node to a terminal node.

Example 2.1 *Consider the BDD in Fig. 1. When* $(x_1, x_2, x_3, x_4) = (1,0,1,1)$ *is applied,* $f_0(1,0,1,1) = 1$ *is evaluated, by traversing nodes 1, 2, 3, and the constant 1 node. Note that this involves traversing along three edges. Fig. 1 also shows that the minimum path length is two, while the maximum path length is four. (End of Example)*

As shown in the above example, the evaluation time depends on the input values. To estimate the evaluation time of different DDs, we introduce a metric **average path length** (APL), which measures the average evaluation time over all possible combinations. We measure the average evaluation time by the APL in the BDD. We assume that each variable occurs as a 0 with the same probability as a 1. That is, at any node, a 0-edge is as likely to be traversed as a 1-edge.

Definition 2.1 The node traversing probability, *denoted by* $P(v_i)$, *is the probability of traversing the node* v_i *when a BDD is traversed from the root node to a terminal node.* **The edge traversing probability**, *denoted by* $P(e_{i_0})$ $(P(e_{i_1}))$ *is the probability of traversing the 0 edge (the 1 edge) from the node* v_i.

Since, the probabilities that 0 and 1 occur are assumed to be equal and 1/2, we have $P(e_{i_0}) = P(e_{i_1}) = P(v_i)/2$.

Lemma 2.1 *The node traversing probability is equal to the sum of the edge traversing probabilities of the incoming edges.*

Theorem 2.1 *The APL is equal to the sum of the node traversing probabilities of the non-terminal nodes.*

Example 2.2 *Let us calculate the APL of the BDD in Fig. 2.* $P(v_5) = 1$, *and we have* $P(e_{5_0}) = P(e_{5_1}) = 1/2$. $P(v_4) = 1/2$. $P(e_{4_0}) = P(e_{4_1}) = 1/4$. $P(v_3) = P(e_{5_0}) + P(e_{4_0}) = 1/2 + 1/4 = 3/4$. $P(e_{3_0}) = P(e_{3_1}) = 3/8$. $P(v_2) = 3/8$. $P(e_{2_0}) = P(e_{2_1}) = 3/16$. *Thus, the average path length of the BDD is given by* $P(v_5) + P(v_4) + P(v_3) + P(v_2) = 1 + 1/2 + 3/4 + 3/8 = 21/8 = 2.625$. *(End of Example)*

0-7803-7659-5/03/$17.00 ©2003 IEEE.

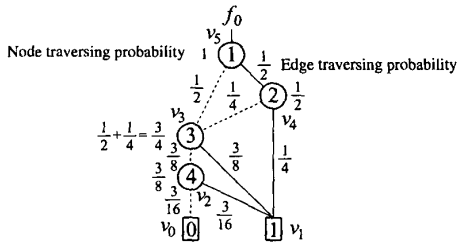


Fig. 2. APL of BDD.

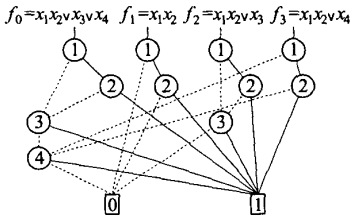


Fig. 3. Example of SBDD.

III. Function Evaluation using BDDs for ECFNs

A new method to represent multiple-output functions, using a BDD for ECFN (encoded characteristic function for non-zero outputs) [7] is faster and requires smaller memory than SBDDs. This section shows the properties of the BDD for ECFN.

A. ECFN and Output Encoding Problem

An ECFN represents the mapping: $F : B^n \times B^u \rightarrow B$, where $u = \lceil \log_2 m \rceil$. ECFNs can be used in FPGA design, logic emulation, embedded systems, etc.

Definition 3.1 *$z^0 = \bar{z}$ and $z^1 = z$.*

Definition 3.2 *For an m-output function $f_i (i = 0, 1, \ldots, m-1)$, the ECFN is $F = \bigvee_{i=0}^{m-1} z_{u-1}^{b_{u-1}} z_{u-2}^{b_{u-2}} \cdots z_0^{b_0} f_i$, where $\boldsymbol{b} = (b_{u-1}, b_{u-2}, \ldots, b_0)$ is a binary representation of the integer i, and $u = \lceil \log_2 m \rceil$.*

$z_0, z_1, \ldots, z_{u-1}$ are auxiliary variables. In the above definition, integer i is represented by a binary vector $\boldsymbol{b}$ using the natural encoding.

Example 3.1 *The ECFN for the four-output function shown in Fig. 3 is $F = \bar{z}_1\bar{z}_0 f_0 \vee \bar{z}_1 z_0 f_1 \vee z_1\bar{z}_0 f_2 \vee z_1 z_0 f_3$.* *(End of Example)*

Note that different encodings can simplify the representation.

TABLE I
Encoding Methods for Four-Output Function.

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

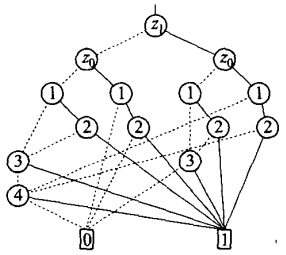


Fig. 4. BDD for multiple-output function using auxiliary variables.

Example 3.2 *Consider the four-output function $F = (f_0, f_1, f_2, f_3)$, where $f_0 = 0$, $f_1 = x_1$, $f_2 = x_2$, and $f_3 = x_1 \vee x_2$. The ECFN F_1 generated by Encoding 1 in Table I is $F_1 = \bar{z}_1\bar{z}_0 f_0 \vee \bar{z}_1 z_0 f_1 \vee z_1\bar{z}_0 f_2 \vee z_1 z_0 f_3$. In this case, we have $F_1 = \bar{z}_1\bar{z}_0 0 \vee \bar{z}_1 z_0 x_1 \vee z_1\bar{z}_0 x_2 \vee z_1 z_0 (x_1 \vee x_2) = z_0 x_1 \vee z_1 x_2$.*

The ECFN F_2 generated by Encoding2 in Table I is $F_2 = \bar{z}_1\bar{z}_0 f_0 \vee \bar{z}_1 z_0 f_1 \vee z_1\bar{z}_0 f_3 \vee z_1 z_0 f_2$. In this case, we have $F_2 = \bar{z}_1\bar{z}_0 0 \vee \bar{z}_1 z_0 x_1 \vee z_1\bar{z}_0 (x_1 \vee x_2) \vee z_1 z_0 x_2 = \bar{z}_1 z_0 x_1 \vee z_1\bar{z}_0 x_1 \vee z_1 x_2$.

This example shows that encodings influence the size of the representation. *(End of Example)*

Fig. 3 shows an example of SBDD to represent a multiple-output function. By attaching a tree to select the output we have an SBDD$^+$. Note that we use $\log_2 4 = 2$ auxiliary variables to represent the output.

B. Optimization of BDDs for ECFNs

The BDD shown in Fig. 4 can be considered as a special case of a BDD for ECFN. As shown in Fig. 5, in an SBDD$^+$, the auxiliary variables appear above the input variables. However, in a general, the auxiliary variables and the input variables can be mixed together in the BDD for ECFN. By optimizing the ordering of the input and the auxiliary variables, the BDD can be minimized.

Definition 3.3 *The number of nodes in the BDD (including both non-terminal and terminal nodes) is denoted by nodes(BDD).*

Theorem 3.1 *$nodes(BDD\ for\ ECFN) \leq nodes(SBDD^+)$*

Example 3.3 *Let the BDD in Fig. 4 be a BDD for ECFN. When we optimize the ordering of the variables, the BDD for ECFN shown in Fig. 6 is obtained.* *(End of Example)*

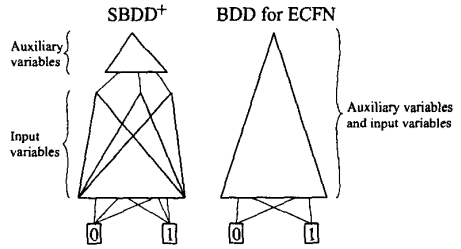


Fig. 5. SBDD+ and BDD for ECFN.

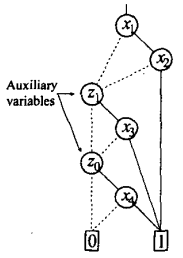


Fig. 6. BDD for ECFN considering variable ordering.

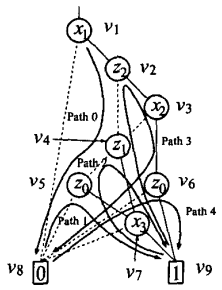


Fig. 7. Method to traverse BDD for ECFN.

In this way, a BDD for ECFN can be made smaller than the corresponding SBDD+. By optimizing both the output encoding and the variable ordering, we can obtain the BDD for ECFN having fewer nodes than the corresponding SBDD+.

C. Fast Evaluation using BDDs for ECFNs

By using the BDD for ECFN in Fig. 6, we can save memory and evaluate functions faster than the corresponding SBDD+. For example, when we apply the input $(x_1,x_2,x_3,x_4) = (1,1,0,0)$ to the BDD in Fig. 6, we can see that all the outputs f_0, f_1, f_2, and f_3 are 1. On the other hand, if we use the SBDD+ in Fig. 4, we need to traverse the BDD four times. This shows that a BDD for ECFN often evaluates several outputs in one traversal.

Example 3.4 *Consider the BDD for ECFN shown in Fig. 7. Note that not all the auxiliary variables appear in a path from the root to a terminal node. Let* $(x_1,x_2,x_3) = (1,1,1)$ *be an input to the BDD for ECFN. By traversing five paths* 0 *through path* 4, *we can evaluate eight outputs* $(0,1,1,1,0,1,0,1)$. *In fact, in the path 0,* f_0 *is evaluated; in the path 1,* f_1 *is evaluated; in the path 2,* f_2 *and* f_3 *are evaluated; in the path 3,* f_4 *and* f_6 *are evaluated; in the path 4,* f_5 *and* f_7 *are evaluated. (End of Example)*

From here, we will show a fast method to evaluate a BDD for ECFN. During BDD traversal, when we encounter an auxiliary variable, by searching both subtrees for 0-edge and 1-edge, we can evaluate all the outputs efficiently.

Algorithm 3.1 *(Evaluation of Multiple-Output Function using a BDD for ECFN)*
Let the input variables be $X = (x_1,x_2,\ldots,x_n)$ *and the auxiliary variables be* $Z = (z_0,z_1,\ldots,z_{u-1})$ *and the outputs be* $Y = (y_0,y_1,\ldots,y_{m-1})$. *Set the input vector to X, and let Z be* $(d,d,\ldots,d)$, *and let v be the root node, where d denotes don't care. Search the BDD recursively from the root node.*

1. *When v is a non-terminal node.*
   (a) *When v represents an input variable.*
   - *i is the index of the input variable for v.*
   - *If* $x_i = 0$, *then v ← 0-edge of v, and do recursion.*
   - *If* $x_i = 1$, *then v ← 1-edge of v, and do recursion.*

   (b) *When v represents an auxiliary variable.*
   - *j is the index of an auxiliary variable for v.*
   - $z_j \leftarrow 0$, $v \leftarrow$ *0-edge of v, and do recursion.*
   - $z_j \leftarrow 1$, $v \leftarrow$ *1-edge of v, and do recursion.*
   - $z_j \leftarrow d$.
2. *When v is a terminal node.*
   Set the output value to y_z. *When the value of* z_j *is d then expand to* 0 *and* 1. *For example, if* $Z = (0,d,0)$ *then expand to* $(0,0,0) = 0$ *and* $(0,1,0) = 2$, *and set the output values to* y_0 *and* y_2. *If all the output values have been obtained, stop the algorithm.*

Example 3.5 *Consider the evaluation of the BDD for ECFN shown in Fig. 7 by using Algorithm 3.1. Let the input vector be* $(x_1,x_2,x_3) = (1,1,1)$ *and* $v \leftarrow v_1$. *The node* v_1 *represents the input variable* x_1. *Since the value of* x_1 *is 1,* $v \leftarrow v_2$ *and do recursion. Since the node* v_2 *represents the auxiliary variable* z_2, $z_2 \leftarrow 0$, $v \leftarrow v_4$, *and do recursion. Since the node* v_4 *represents the auxiliary variable* z_1, $z_1 \leftarrow 0$, $v \leftarrow v_5$, *and do recursion. Since the node* v_5 *represents the auxiliary variable* z_0, $z_0 \leftarrow 0$, $v \leftarrow v_8$, *and do recursion. The node* v_8 *is the terminal node* 0. *Since* $Z = (0,0,0)$, *the value of* y_0 *is* 0. *Return to the node* v_5 *and set* $z_0 \leftarrow 1$, $v \leftarrow v_7$, *and do recursion. The node* v_7 *represents the input variable* x_3. *Since the value of* x_3 *is 1,* $v \leftarrow v_9$ *and do recursion. The node* v_9 *is the terminal node* 1. *Since* $Z = (0,0,1)$, *the value of* y_1 *is* 1. *Return to the node* v_5 *and set* $z_0 \leftarrow d$. *Return to the node* v_4 *and set* $z_1 \leftarrow 1$, $v \leftarrow v_7$, *and do recursion. The node* v_7 *represents the input variable* x_3. *Since the value of* x_3 *is 1,* $v \leftarrow v_9$ *and do recursion. Since* $Z = (0,1,d)$, *the values of* y_2 *and* y_3 *are* 1. *Return to the node* v_4 *and set* $z_1 \leftarrow d$. *Return to the node* v_2 *and set* $z_2 \leftarrow 1$, $v \leftarrow v_3$, *and do recursion. The node* v_3 *represents the input variable* x_2. *Since the value of* x_2 *is 1,* $v \leftarrow v_6$ *and do recursion. Since the node* v_6 *represents the auxiliary variable* z_0, $z_0 \leftarrow 0$, $v \leftarrow v_8$, *and do recursion. Since* $Z = (1,d,0)$, *the values of* y_4 *and* y_6 *are* 0. *Return to*

TABLE II
AVERAGE NUMBER OF NODES AND EVALUATION TIME OF DECISION DIAGRAMS.

Name	# In	# Out	MTBDD		BDD for CF		SBDD$^+$		BDD for ECFN	
			Nod	Tra	Nod	Tra	Nod	Tra	Nod	Tra
apex1	45	45	942	8.8	3594	76.3	1324	269.8	1087	31.8
apex3	54	50	537	6.6	2449	81.6	986	286.6	708	28.2
duke2	22	29	662	6.4	997	49.9	366	150.3	346	22.5
e64	65	65	131	2.0	260	99.5	194	256.0	194	256.0
exep	30	63	1170	7.8	3030	102.3	675	255.7	660	38.0
k2	45	45	929	8.8	3594	76.3	1321	269.8	1167	29.1
mainpla	27	54	632	4.2	3114	85.2	1857	277.5	1018	49.2
mark1	20	31	4138	6.4	745	52.9	119	115.7	117	109.2
misex2	25	18	118	4.9	184	31.9	100	75.6	98	18.9
opa	17	69	241	4.4	1778	107.9	428	322.2	364	37.8
pdc	16	40	19178	10.2	5852	70.2	596	215.4	590	181.1
rckl	32	7	65	2.0	135	12.5	198	100.6	67	4.8
seq	41	35	378	3.3	1197	55.8	1284	168.0	493	28.2
shift	19	16	196095	15.5	3746	39.5	78	86.0	62	55.0
spla	16	46	11100	9.7	7522	78.1	628	226.6	604	99.9
t2	17	16	304	7.5	484	31.7	145	72.9	140	44.9
table5	17	15	436	8.0	677	30.5	685	114.1	476	10.6
ts10	22	16	589837	5.5	589826	31.5	163	88.0	83	14.5
vg2	25	8	420	11.8	471	23.8	90	48.9	82	45.7
x1dn	27	6	214	9.8	245	21.2	139	41.0	139	41.0
x6dn	39	5	195	4.1	225	11.6	235	41.2	193	13.4
x9dn	27	7	292	9.8	237	20.3	139	50.4	139	50.4
xparc	41	73	3844	3.6	4773	113.1	1947	304.8	1232	21.8
Ratio			271.3	0.072	162.5	0.36	1.0	1.0	0.81	0.41

Nod: Number of terminal and non-terminal nodes
Min: When the auxiliary variables can be any place
Tra: Average number of edge traversal
Ratio: Average of ratios when the value for SBDD is 1.0

the node v_6 and set $z_0 \leftarrow 1$, $v \leftarrow v_9$, and do recursion. Since $Z = (1, d, 1)$, the values of y_5 and y_7 are 1. Since all the output values have been obtained, stop the algorithm. We have the output vector $(y_0, y_1, y_2, y_3, y_4, y_5, y_6, y_7) = (0, 1, 1, 1, 0, 1, 0, 1)$.
(End of Example)

The average evaluation time is obtained from the APL, which can be obtained similarly to the case of BDDs.

IV. Experimental Results

For the selected MCNC benchmark functions [9], we constructed MTBDDs, BDDs for CFs [1, 8], and SBDD$^+$s. Table II compares the average number of nodes and evaluation time of those decision diagrams. To construct BDDs for ECFNs, we used the encoding method presented in [6] and [7]. In the table, *Name* denotes the function's name, *In* denotes the number of inputs, *Out* denotes the number of outputs, *Nod* denotes the number of nodes, and *Tra* denotes the average number of edge traversal to evaluate all the outputs. For MTBDDs, *Tra* corresponds to the APL.

This table shows that sizes of BDDs for ECFNs are 81% of corresponding SBDD$^+$. Also, the average number of edge traversal is 41% of corresponding SBDD$^+$. However, because of the overhead for implementation the recursive traversal procedure, the time for one edge traversal in a BDD for ECFN is longer than in the SBDD$^+$. The average evaluation time for MTBDDs and BDDs for CFs are smaller, but they are very large.

V. Summary

In this paper, we presented four different methods to represent multiple-output functions by BDDs. We compared the sizes and the average number of edge traversal. Theoretically, decision-diagram (DD) based function evaluation is orders-of-magnitude faster than traditional logic simulation methods [4]. The number of input variables is typically smaller than the number of gates in the circuit, which is the complexity of levelized complied-code simulation [1]. Unfortunately, the sizes of MTBDDs and BDDs for CFs grow very quickly.

Our experiments show that the functional evaluation using a BDD for ECFN is at least two times faster than SBDD$^+$ for the functions in Table II. The applications of the method include software synthesis [3] and logic simulation of small-scale circuits.

Acknowledgments

This research is partly supported by the Grant in Aid for Scientific Research of The Japan Society for the Promotion of Science (JSPS), and the Takeda Foundation. Prof. Jon T. Butler and Dr. Alan Mishchenko's comments improved presentation.

References

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

[2] 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 embed-ded control applications," *IEEE Trans. CAD*, Vol. 18, No. 6, pp. 834-849, June 1999.

[3] Yunjian Jiang and R. K. Brayton, "Software synthesis from synchronous specifications using logic simulation techniques," *Design Automation Conference (DAC)*, New Orleans, June 2002.

[4] 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.

[5] 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.

[6] 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.

[7] T. Sasao, M. Matsuura, Y. Iguchi, and S. Nagayama, "Compact BDD Representations for Multiple-Output Functions," *VLSI-SOC 2001*, Montpellier, France, Dec. 3-5, 2001.

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

[9] S. Yang, *Logic Synthesis and Optimization Benchmark User Guide Version 3.0*, MCNC, Jan. 1991.