多出力関数のBDDによる表現法とその最適化法

松浦 宗寛 [1]　笹尾 勤 [1,2]　井口 幸洋 [3]　永山 忍 [3]

[1] 九州工業大学 情報工学部
[2] 九州工業大学 マイクロ化総合技術センター
[3] 明治大学 理工学部

あらまし: 本論文では, 多出力関数の新しい表現方法 : ECFN (Encoded Characteristic Function for Non-zero outputs) を提案する. ECFN は, n 入力 m 出力の関数を表現するために $(n+u)$ 個の 2 値変数を使う. ここで, $u = \lceil \log_2 m \rceil$ である. ECFN を表現する BDD(Binary Decision Diagram) は, SBDD(Shared BDD) より大きくなることはなく, 多くの場合小さくできる. ECFN では出力の符号化を工夫することによって BDD を縮小できるが, これは変数順序の変更と同様に効果的である. 最適な符号化をした ECFN を表現する BDD の節点数が, $2n+2$ 個であり, 最悪の符号化をした節点数が 2^{n+1} となるような n 入力 2^n 出力の関数が存在することを示す. 次に ECFN の符号化問題を定式化し, 発見的手法を提案する. 標準的なベンチマーク関数を使用した実験結果により, 符号化を考慮することによって BDD の大きさを大幅に削減できることを示す. また, ECFN の組み込みシステムへの応用も述べる.

和文キーワード: 多出力関数, 符号化問題, BDD, 特性関数.

Compact Representaions of BDDs for Multiple-Output Functions and Their Optimization

Munehiro MATSUURA[1], Tsutomu SASAO[1,2], 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

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, and can be often smaller than SBDDs. The size of a BDD depends on the encoding of the outputs as well as the ordering of the variables. We show an n-input 2^n-output function, where the optimal encoding produces a BDD with $2n+2$ nodes, while the worst encoding produces a BDD 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. We also show an application of ECFNs for the embedded system.

Key words: Multiple-output function, encoding problem, BDD, characteristic function.

1 はじめに

論理回路は, 通常, 多出力である. 各出力を独立に表現すると, ほとんどの場合, 表現が大きくなり過ぎ, 効率が悪い. 多出力関数を表現する方法は多数存在する. ここで多出力関数を, $F = (f_0, f_1, \ldots, f_{m-1}) : B^n \rightarrow B^m$, $B = \{0, 1\}$ と定義する. 本論文では, コンパクトな BDD(Binary Decision Diagram) を用いた多出力関数の表現法を考える. BDD による多出力関数の表現法には以下の 3 つの方法が知られている.

1) MTBDD(Multi-Terminal Binary Decision Diagram) [13]: MTBDD の各終端節点は, m ビットの 2 進ベクトルである. n 入力 m 出力の関数を $O(n+\log m)$ 時間で評価できる. しかし MTBDD は, サイズ (節点数) が大きくなりすぎる傾向がある.

2) 多出力関数の特性関数 CF(Characteristic Function) を表現する BDD: CF は, 写像 $F : B^n \times B^m \rightarrow B$ を表す. ここで $F(\vec{a}, \vec{b}) = 1 \Leftrightarrow (f_0(\vec{a}), f_1(\vec{a}), \ldots, f_{m-1}(\vec{a})) = \vec{b}$. CF では多出力関数を表現するために, m 個の 2 値補助変数 $\{z_0, z_1, \ldots, z_{m-1}\}$ を必要とする. F は入力と出力の有効な組み合わせ全ての集合を示す. 例えば, 4 出力の CF は次のように表現できる.

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

CF では変数が $(n+m)$ 個必要となるため, その BDD は非常に大きくなる傾向にある. CF の有利な点は短い評価時間にある. n 入力, m 出力関数の場合, CF を表現する BDD は $O(n+m)$ 時間で評価できる. CF は論理シミュレーション [1] や多段論理回路の最適化 [6] に使われる.

3) SBDD(Shared Binary Decision Diagram)[13]: 多くの場合, SBDD は対応する MTBDD や CF を表現する BDD より小さくなる. SBDD を使った関数の評価には, $O(n \cdot m)$ の時間が必要である.

本論文では, 4 の表現方法として ECFN(Encoded Characteristic Function for Non-zero outputs) を提案する. これは, 次のような写像 F を表現したものである. $F : B^n \times B^u \rightarrow B$, ここで $u = \lceil \log_2 m \rceil$. $F(\vec{a}, \vec{b}) = 1 \Leftrightarrow f_{\nu(\vec{b})}(\vec{a}) = 1$, また, $\nu(\vec{b})$ は 2 進ベクトル $\vec{b}$ によって表現される整数である. 例えば, 4 出力関数の ECFN は次のように表現できる.

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

ECFN は FPGA の設計 [10], 論理回路エミュレーション [2], 組み込みシステム [14] 等に使用できる.

後述するが, ECFN を表現する BDD は, SBDD を一般化したものであり, SBDD より小さくできる. 従って, 節点数が重要視されるような応用分野で有用である.

2 ECFN と符号化問題

本節では, ECFN(Encoded Characteristic Function for Non-zero outputs) を定義し符号化問題 [15] を定式化する.

定義 2.1 $x^0 = \bar{x}$, $x^1 = x$.

表 2.1: 4 出力関数の符号化法.

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

定義 2.2 m 出力関数 f_i $(i = 0, 1, \ldots, m-1)$ の *ECFN (Encoded Characteristic Function for Non-zero outputs)* は,

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

である. ここで $\vec{b} = (b_{u-1}, b_{u-2}, \ldots, b_0)$ は整数 i の *2* 進表現で, $u = \lceil \log_2 m \rceil$ である.

z_0, z_1, $\ldots$, z_{u-1} は出力を表現する補助変数である. 上の定義で, 整数 i は 2 進ベクトル $\vec{b}$ をそのまま符号化しているが, 符号化法を工夫することで, 表現を簡単化することも可能である.

定義 2.3 *DD(Decision Diagram)* において, 終端節点を含んだ節点の総数を *DD* の大きさという.

例 2.1 4 出力関数 $F = (f_0, f_1, f_2, f_3)$, $f_0 = 0$, $f_1 = x_0$, $f_2 = x_1$, $f_3 = x_1 \vee x_0$ を考える. 表 2.1 の *Encoding*1 を用いると, 次の *ECFN* を得る.

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

この場合,

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

を得る.

一方, 表 2.1 の *Encoding*2 を用いると, 次の *ECFN* を得る.

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

この場合,

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

となる. *Encoding*1 を使った *BDD* では 6 つの節点で十分であるが, *Encoding*2 を使った *BDD* では 7 つの節点を必要とする. *(例題終)*

例 2.2 8 出力関数 $F = (f_0, f_1, \ldots, f_7)$, ここで $f_0 = 0$, $f_1 = x_0$, $f_2 = x_1$, $f_3 = x_1 \vee x_0$, $f_4 = x_2$, $f_5 = x_2 \vee x_0$, $f_6 = x_2 \vee x_1$, $f_7 = x_2 \vee x_1 \vee x_0$ を考える.

表 2.2: 8 出力関数の符号化法.

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

この場合, 8 出力を表現するために z_0, z_1 と z_2 の 3 つの補助変数が必要となる. 表 2.2 の *Encoding*1 では次の *ECFN* を得る.

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

従って,

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

を得る. 一方, 表 2.2 の *Encoding*2 では次の *ECFN* を得る.

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

従って,

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

となる. *Encoding*1 を使った *BDD* では 8 つの節点で十分であるが *Encoding*2 を使った *BDD* では 16 個の節点を必要とする. *(*例題終*)*

これらの 2 つの例は, 良い符号化を探すことが重要であると示す.

m 出力関数では, m 個の出力を表現するために $u = \lceil \log_2 m \rceil$ 個の補助変数が必要である. 従って符号化法は

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

通り存在する. しかし, BDD の大きさは, 補助変数の否定や名前の付け変えでも不変であるので, 最小の BDD となる ECFN は

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

表 2.3: 定理 2.1の ECFN を表現する BDD の大きさ.

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

通りの符号化を考えれば十分である. $m = 4$ とすると, $u = 2$ となり, $N = \frac{3!}{1!2!} = 3$ 個の符号化を考えればよい. 表 2.1にこれらの 3 つの符号化を示す.

従って, 本問題は次のように定式化ができる.

問題 2.1 *(ECFN* における符号化問題*)* 与えられた多出力関数 $F : B^n \to B^m$ に対して, 節点数最小の *BDD* となるように $u = \lceil \log_2 m \rceil$ 個の補助変数を使って F を表現せよ.

著者らの知る限り, 問題 2.1を定式化した論文はない. 最適解を求めるには, 網羅的な探索方法以外, 知られていない.

定理 2.1 n 入力 2^n 出力関数 $f_i(\vec{x}) = \bigvee_{j=0}^{n-1} a_{ij} x_j (i = 0, 1, \ldots, 2^n - 1)$ の *ECFN* は, *BDD* で表現すると $2n+2$ の節点で表現できる. ここで $\vec{x} = (x_{n-1}, x_{n-2}, \ldots, x_0)$, $\vec{a}_i = (a_{in-1}, a_{in-2}, \ldots, a_{i0})$ は整数 i の *2* 進表現である.

例 2.3 $n = 3$ で, $f_0 = 0$, $f_1 = x_0$, $f_2 = x_1$, $f_3 = x_1 \vee x_0$, $f_4 = x_2$, $f_5 = x_2 \vee x_0$, $f_6 = x_2 \vee x_1$, $f_7 = x_2 \vee x_1 \vee x_0$ とする. これらの関数は例 *2.2*のものと同一である. *ECFN* は次式で与えられる.

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

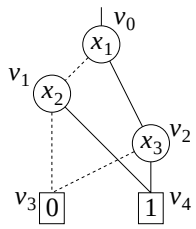
表2.3は定理2.1中で定義した多出力関数を表現する ECFN の BDD の大きさを示している. *Worst* の BDD は 100 回のランダムな符号化の中から見つけた. この表から, 次のように推測できる.

推測 2.1 n 入力 2^n 出力関数で, その *ECFN* を表現する *BDD* が最適符号化の場合 $2n + 2$ 個の節点, 最悪符号化の場合 2^{n+1} 個の節点を必要とする関数が存在する.

3 組み込みシステムへの応用

3.1 ブランチング・プログラム

ここでは, 組み込みシステムへの応用を考える. 組み込みシステムではメモリ量が非常に重要となる [3].



(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)

図 3.1: ブランチング・プログラム.

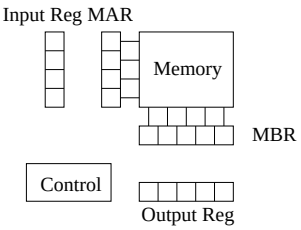


図 3.2: ブランチング・プログラムを実行するアーキテクチャ.

ブランチング・プログラムは, 次に示すような手順により, 順序回路で実現できる.

1) 与えられた論理関数を BDD[4] で表現する (図 3.1(a) で点線は 0 枝を表し, 実線は 1 枝を表す).
2) BDD の各非終端節点を *If then else* 文に置き換え, f を表現するブランチング・プログラムを得る (図 3.1(b)).
3) プログラムを汎用マイクロプロセッサで実装する.

命令読みだし時間を削減するために, BDD 構造を辿る専用マシンが提案されている [8, 9, 5].

ブランチング・プログラムは BDD の節点数に比例してメモリを必要とする. ECFN を表現する BDD を構成する際, 出力の符号化法を工夫すれば, 最適な変数の順序を探すのと同様に BDD を小さくできる. 実験結果で示すように, 本方法で得られた BDD は多くの場合, 他の DD よりも小さくなる.

3.2 再構成可能なアーキテクチャ

多出力関数は図 3.2に示すアーキテクチャで実現できる. 本アーキテクチャでは, メモリ部には BDD のデータを格納し, コントロール部は BDD を辿る.

例 3.1 次の 4 出力関数を考える. $f_0 = x_1x_2$, $f_1 = x_1x_2 \vee x_4$, $f_2 = x_1x_2 \vee x_3$, $f_3 = x_1x_2 \vee x_3 \vee x_4$.

ECFN を

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

とする. 変数順序が $(z_1, z_0, x_1, x_2, x_3, x_4)$ のとき, 図 3.3 に示すように節点数が 16 の *BDD* を得る. 一方, 変数の順序を $(x_1, x_2, z_1, x_3, z_0, x_4)$ とすると, 図 3.4 に示すように節点数 8 の *BDD* を得る.

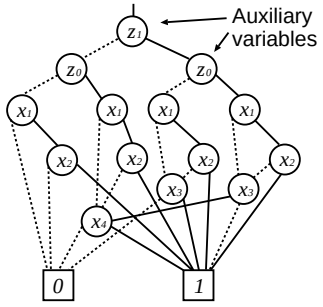


図 3.3: 元の変数順序で表現した ECFN.

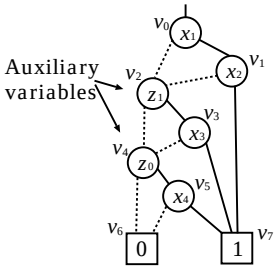


図 3.4: 最適化した変数順序で表現した ECFN.

図 3.5 は図 3.4 の *BDD* データを表す. 各節点は変数のインデックスと, *0* 枝と *1* 枝の 2 つのポインタの 3 つの属性から成る. また, $index = 0$ は終端節点を意味する. $f_0(1,1,1,1)$ の値を評価するためには, 変数に $(x_1, x_2, z_1, x_3, z_0, x_4) = (1,1,0,1,0,1)$ を代入すればよい. *(例題終)*

4 実験結果

4.1 ECFN の符号化法

ECFN を表現する積和形 (SOP) の積項数を符号化を工夫することにより, 減らす発見的アルゴリズムが開発されている [15]. しかし残念ながら SOP 用符号化アル

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

図 3.5: 図 3.4の ECFN の BDD データ.

ゴリズムは必ずしも, BDD を最小化するとは限らない. そのため, [15] のアルゴリズムに以下の改良を加えた.

アルゴリズム 4.1 *(ECFN の符号化)*

1) 多出力関数の *SBDD* を簡単化する.

2) 自然な符号化を用いた *ECFN (*例えば, f_0 には $00\cdots00$ を, f_1 には, $00\cdots01$ を, f_2 には, $00\cdots10$ を割り当てたもの*)* を生成し, *BDD* を最小化する.

3) *SOP* 用の発見的アルゴリズム *[15]* を用いて符号化を見つけ *ECFN* を構成する. そして *BDD* を最小化する.

4) *BDD* が最も小さくなるまで上記 3 ステップを繰り返す.

SBDD の簡単化には, 発見的アルゴリズムを使用しているため, 1) で得た SBDD は 2) で得た BDD よりも小さくなり得る.

4.2 ベンチマーク関数を用いた実験結果

アルゴリズム 4.1を実装し, 様々なベンチマーク関数に対する BDD を簡単化した.

表 4.1は, DD の大きさを比較している. 表中の *Name* は関数名を, *In* は, 入力変数の個数を, *Out* は, 出力変数の個数を, MTBDD, BDD for CF, SBDD, BDD for ECFN は, それぞれの DD の大きさを示す. BDD for ECFN は, アルゴリズム 4.1を用いて求めた. –は, MTBDD または BDD for CF が大きくなりすぎ構成できなかった関数を表している. 表 4.1より MTBDD と BDD for CF が非常に大きいことがわかる.

入力変数と補助変数を混合することによって ECFN を表現する BDD を最適化した. CF を表現する BDD では $O(n+m)$ 時間で評価するためには, 出力変数の上部に, 依存する入力変数がくる必要がある. しかしながら, ここでは, BDD を小さくするためにこの制約を無視して BDD を作成した [16]. CF を表現する BDD は, 変数順序の制約を無視して構成したものでさえ, 多くの場合が大きくなりすぎてしまった. 表で空白が入っているのは, そのような DD を表している.

表 4.2は, 大規模なベンチマーク関数の DD の大きさの比較をしている. MTBDD, BDD for CF, SBDD, ECFN はそれぞれの DD の大きさを示す. これらのベンチマーク関数では, CF を表現する BDD よりも MTBDD のほうがより多く構成することができた. *Heu* の列は, 別の発見的アルゴリズムによって得られた ECFN の節点数を示す. アルゴリズム 4.1は中間のデータ構造として SOP を使うため, 利用できなかった. そこで, データ構造に BDD を使う方法を開発した. しかし残念ながら, 小さい問題にしか適用できない. また, 表 4.1の関数においてアルゴリズム 4.1より遅い.

ECFN を表現する BDD は FBDD(Free BDD) より小さくなり得る場合もある. Gunther と Drechsler[7] は FBDD の発見的最小化アルゴリズムを発案し, 大きな FBDD に対する実験結果を発表している. 表 4.3は, ECFN を表現する BDD が簡単化された FBDD より小さくなり得ることを示している. 表中の FBDD では否定枝を使っているが, ECFN を表現する BDD は使っていない.

表 4.1: 様々な DDs の大きさ.

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

4.3 再構成可能なハードウェアの試作

3節で示したアーキテクチャの性能を確認するため, 市販の FPGA ボードを使って再構成可能なハードウェアを開発した. FPGA ボードの仕様は次の通りである.

- FPGA: Altera FLEX10K100
- Clock 周波数: 20MHz
- RAM: Static 4M Bit

この試作機では, BDD の 1 つの節点を処理するために $\alpha = 2$ クロック必要である. 従って, 1 つの入力パターンに対する m 出力関数の評価には $n \cdot m \cdot \alpha$ クロック必要である. ここで n は入力変数の数で m は関数の出力数である. 制御部は容易に入手可能な, Altera 社の FPGA: FLEX 10K100 を使用した. しかし, 制御部は非常に単純な構成なので, どんな論理回路, 例えば CPLD を用いても実現できる.

表 4.2: 大規模な関数を表現する DD の大きさ.

Name	In	Out	MTBDD	BDD for CF	SBDD	BDD for ECFN	
						Org	Heu
C432	36	7	1198	885	1075	921	854
C499	41	32			27876	24216	
C880	60	26			4166	4006	3958
C1908	33	25			7456	7430	7425
C2670	233	140			2847	2724	2697
C3540	50	22	9564117		34710	34710	
C5315	178	123			2564	2560	
C7552	207	108			2944	2941	
b9	41	21	32720	1582	177	177	176
dalu	75	16	522749	9042	1178	805	801
des	256	245			3975	2855	
ex4	128	28	4722244		540	539	532
rot	135	107			8501	8475	8463

表 4.3: FBDD との比較.

Name	FBDD	BDD for ECFN
C432	1063	854
C499	25866	24216
des	2941	2855

5 結論とコメント

本論文で, 多出力関数の新しい表現方法: ECFN (Encoded Characteristic Function for Non-zero outputs) を提案した. ECFN は, 2 値変数だけを使い, 符号化を工夫する事で, 変数順序を考えるのと同様に BDD を簡単化できる. また, 符号化問題を定式化し, 発見的手法を提案した. また, n 入力 2^n 出力関数のうちである符号化では $2n+2$ 個の節点をもつ BDD になり, 別の符号化では 2^{n+1} 個の節点になる BDD が存在することも示した.

また, メモリとシーケンサから構成された, 再構成可能なハードウェアも開発した. 特長は次のとおりである.

1. システムの構成はメモリとシーケンサからなる: ハードウェアの構成は単純で, 設計は ECFN を表現する BDD の簡単化に対応する.
2. ECFN を表現する BDD は対応する SBDD より小さくできる.

現在, ECFN の符号化アルゴリズムを改良中である.

謝辞

本研究は一部, 日本学術振興会, 科学研究費補助金による.

参考文献

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

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

[14] 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*, pp. 225-230, Lake Tahoe, June 2001.

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

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

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