社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

不完全定義多出力論理関数を表現するBDDとその応用について(041124)

松浦　宗寛†　　笹尾　　勤†

† 九州工業大学情報工学部電子情報工学科　〒 820–8502 福岡県飯塚市大字川津 680–4

あらまし　多出力論理関数を表現する二分決定グラフ (Binary Decision Diagram: BDD) の一つに, 特性関数 (Characteristic Function) を表現する BDD(BDD_for_CF) がある. 本稿では, 不完全定義多出力論理関数を BDD_for_CF で表現する方法を提案する. 次に, 不完全定義多出力論理関数を表現する BDD_for_CF の幅を小さくする方法について述べる. 最後に, この手法を基数変換回路, 加算回路, 乗算回路, および剰余数変換回路に適用した際の実験結果について述べる. この手法は関数分解や LUT カスケードの合成に有用である.

キーワード　不完全定義関数, 多出力論理関数, 特性関数, 二分決定グラフ, 関数分解

BDD Representation for Incompletely Specified Multiple-Output Logic Functions and Its Applications

Munehiro MATSUURA† and Tsutomu SASAO†

† Department of Computer Science and Electronics, Kyushu Institute of Technology
680–4, Kawazu, Iizuka, Fukuoka, 820–8502 Japan

Abstract　A multiple-output function can be represented by a binary decision diagram (BDD) for characteristic function (CF). This paper considers a method to represent multiple-output incompletely specified functions using BDD_for_CF. An algorithm to reduce the widths of BDD_for_CFs is presented. This method is useful for functional decomposition and synthesis of LUT cascade.

Key words　Incompletely specified function, Multiple-output function, Characteristic function, Binary decision diagram, Functional decomposition

1.　は じ め に

単一出力の不完全定義論理関数を二分決定グラフ (Binary Decision Diagram: BDD) で表現する方法は種々存在する [8]. このうち,

（1）　関数値として, 0, 1 の他にドント・ケアを持つ三値関数として表現する方法 [8].

（2）　三値を表現するために二つの BDD を用いる方法 [4].

（3）　ドント・ケアを表現するための変数を使用する方法 [3], [8].

等が使用されている. これらの方法を用いることで, 不完全定義多出力論理関数を共有二分決定グラフ (Shared Binary Decision Diagram: SBDD) として表現可能である.

今までのほとんどの研究は BDD の総節点数の最小化を目的としていた [3], [6], [10], [11]. しかしながら, これらの手法は多出力関数の効果的な関数分解を検出するには適していない. BDD を用いて関数分解を行う場合, BDD の幅を小さくすることが重要になる. BDD を用いて多出力論理関数を効率的に分解したい場合, 多端子二分決定グラフ MTBDD(Multi terminal BDD) や多出力関数の特性関数 (Characteristic function) を表現する BDD(BDD_for_CF) を用いる必要がある.

BDD_for_CF では, MTBDD に比べて, 少ない節点数で関数を表現できる場合が多い. ただし, 出力を表す変数をその出力に影響を与える変数より終端側に配置する必要がある. また, BDD_for_CF では入力変数の一部のみで特定の出力関数の値が定まる場合, MTBDD よりも幅を小さくできる可能性がある.

本稿では, BDD_for_CF を用いて不完全定義多出力論理関数を表現する方法と, BDD_for_CF の幅を削減する方法について述べる.

2.　諸定義ならびに基本的性質

［定義 1］　関数 f が変数 x に依存するとき, x を f の**サポート変数**という.

［定義 2］　**不完全定義関数**を $f : \{0,1\}^n \rightarrow \{0,1,d\}$ とする. ここで, d はドント・ケアを表し, 関数値が 0 でも 1 でも良いことを示す. また, 集合 $f^{-1}(0)$, $f^{-1}(1)$, $f^{-1}(d)$ を表す関数をそれぞれ f_0, f_1, f_d とする. 明らかに, $f\_0 \vee f\_1 \vee f\_d = 1$, $f\_0 \cdot f\_1 = 0$, $f\_1 \cdot f\_d = 0$, $f\_0 \cdot f\_d = 0$ である.

［定義 3］ [2] 入力変数を $X = (x_1, x_2, \ldots, x_n)$, 多出力関数を $F = (f_1(X), f_2(X), \ldots, f_m(X))$ とする. **完全定義多出力関数の特性関数**を

$$\chi(X, Y) = \bigwedge_{i=1}^{m} (y_i \equiv f_i(X))$$

とする. ここで y_i は出力を表す変数である.

完全定義多出力関数の特性関数は入力の組合せと出力の組合せのうち, 許されているものを表す. $f_{i\_0}(X) = \bar{f}_i(X)$, $f_{i\_1}(X) = f_i(X)$ とすると, 特性関数 χ は次のように表現できる.

$$\chi(X, Y) = \bigwedge_{i=1}^{m} \{\bar{y}_i \cdot f_{i\_0}(X) \vee y_i \cdot f_{i\_1}(X)\}.$$

不完全定義関数では関数値がドント・ケアの場合, その関数値は 0 でも 1 でもよい. 従って, 特性関数では, 出力変数 y_i の値が 0 でも 1 でも真となる. ドント・ケアを表現する関数を $f_{i\_d}(X)$ とすると,

$$\bar{y}_i(f_{i\_0} \vee f_{i\_d}) \vee y_i(f_{i\_1} \vee f_{i\_d}) = \bar{y}_i f_{i\_0} \vee y_i f_{i\_1} \vee f_{i\_d}$$

が成立する. これより次の定義を得る.

［定義 4］ **不完全定義多出力関数の特性関数** χ を

$$\chi(X, Y) = \bigwedge_{i=1}^{m} \{\bar{y}_i f_{i\_0}(X) \vee y_i f_{i\_1}(X) \vee f_{i\_d}(X)\}$$

とする.

［例 1］ 表 1 に示す 4 入力 2 出力の不完全定義関数を考える.

$$\begin{aligned}
f_{1_0} &= \bar{x}_1\bar{x}_2x_3 \vee x_1\bar{x}_2\bar{x}_3 \\
f_{1_1} &= \bar{x}_1x_2x_3 \vee x_1\bar{x}_2x_3 \vee x_1x_2\bar{x}_3 \\
f_{1_d} &= \bar{x}_1\bar{x}_3 \vee x_1x_2x_3 \\
f_{2_0} &= \bar{x}_1\bar{x}_2x_3 \vee x_1\bar{x}_2x_3 \vee x_1x_2\bar{x}_4 \\
f_{2_1} &= \bar{x}_2\bar{x}_3 \vee x_2x_3x_4 \\
f_{2_d} &= x_2\bar{x}_3
\end{aligned}$$

なので, この関数の特性関数は

$$\begin{aligned}
\chi = \{&\bar{y}_1(\bar{x}_1\bar{x}_2x_3 \vee x_1\bar{x}_2\bar{x}_3) \vee \\
&y_1(\bar{x}_1x_2x_3 \vee x_1\bar{x}_2x_3 \vee x_1x_2\bar{x}_3) \vee (\bar{x}_1\bar{x}_3 \vee x_1x_2x_3)\} \cdot \\
&\{\bar{y}_2(\bar{x}_1\bar{x}_2x_3 \vee x_1\bar{x}_2x_3 \vee x_1x_2\bar{x}_4) \vee \\
&y_2(\bar{x}_2\bar{x}_3 \vee x_2x_3x_4) \vee (x_2\bar{x}_3)\}
\end{aligned}$$

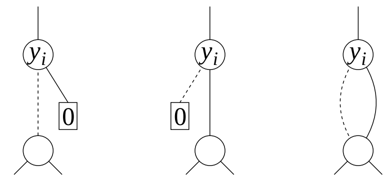
となる. (例終り)

次に, 不完全定義多出力関数の特性関数を表現する BDD とその構造について述べる.

［定義 5］ 多出力関数 $F = (f_1, f_2, \ldots, f_m)$ の **BDD_for_CF** とは, F の特性関数 χ を表現する BDD である. 但し, BDD で根節点を最上位としたとき, 出力 f_i を表現する変数 y_i は, f_i のサポート変数の下に置く.

表 1 不完全定義関数の真理値表

x_1	x_2	x_3	x_4	f_1	f_2
0	0	0	0	d	1
0	0	0	1	d	1
0	0	1	0	0	0
0	0	1	1	0	0
0	1	0	0	d	d
0	1	0	1	d	d
0	1	1	0	1	0
0	1	1	1	1	1
1	0	0	0	0	1
1	0	0	1	0	1
1	0	1	0	1	0
1	0	1	1	1	0
1	1	0	0	1	d
1	1	0	1	1	d
1	1	1	0	d	0
1	1	1	1	d	1



(a) $f_i = 0$ (b) $f_i = 1$ (c) $f_i = d$

図 1 不完全定義関数を表現する BDD_for_CF

図 1 に不完全定義多出力関数を表現する BDD_for_CF の概念図を示す. ここで, 実線は 1 枝を破線は 0 枝を表す. 出力を表現する節点 y_i の 1 枝が定数 0 節点に接続しているとき, $f_i = 0$ となる (図 1(a)). また, 節点 y_i の 0 枝が定数 0 節点に接続しているとき, $f_i = 1$ となる (図 1(b)). さらに, 節点 y_i の 0 枝と 1 枝の両方が同じ定数 0 以外の節点に接続しているとき, $f_i = d$ (ドント・ケア) となる (図 1(c)). $f_i = d$ の場合 y_i の節点は冗長なので, BDD の簡単化の際削除する.

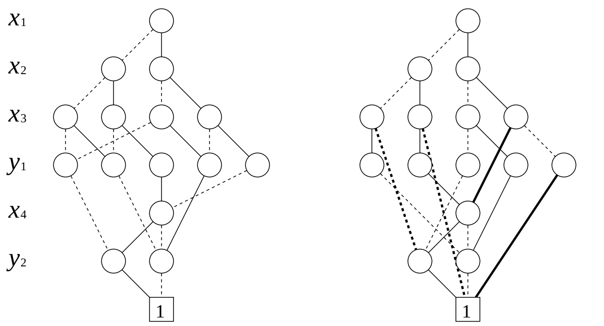
完全定義関数を表現する BDD_for_CF の場合, 根節点から定数 1 の終端節点への全ての経路上に必ず全ての出力変数の節点が現れ, 出力変数のどちらか一方の枝は必ず定数 0 の終端節点に接続されている. 一方, 不完全定義関数を表現する BDD_for_CF の場合は, 根節点から定数 1 の終端節点への経路上に現われない出力変数の節点もあり得る. 出力変数の節点が存在しない経路に対しては, その出力はドント・ケアとなる.

［例 2］ 例 1 の関数を表現する BDD_for_CF を図 2 に示す. 図 2 では簡単のために, 定数 0 の終端節点とそれに接続する枝は省いている. 図 2(a) は全てのドント・ケアに 0 を割り当てた完全定義関数を表現する BDD_for_CF で, 図 2(b) は不完全定義関数を表現する BDD_for_CF である. 太線の枝は出力変数の節点が無く, その出力値がドント・ケアであることを意味する. 図 2(a) では, 根節点から終端節点までの全ての経路に全ての出力変数 $\{y_1, y_2\}$ が現れるが, 図 2(b) の場合は, 出力 f_i がドント・ケアの場合, その経路に出力変数 y_i は現れない. (例終り)

3. BDD_for_CF の幅の削減

3.1 BDD_for_CF を用いた関数分解

特性関数を表現する BDD_for_CF を用いると, 多出力論理関数を効率的に分解できる [9]. BDD を用いて関数分解を行う場



(a) 全てのドント・ケアに 0 を割り当てた関数の BDD_for_CF　(b) 不完全定義関数の BDD_for_CF

図 2　多出力関数を表現する BDD_for_CF

表 2　不完全定義関数の分解表

		$X_1 = \{x_1, x_2\}$			
		00	01	10	11
$X_2 = \{x_3, x_4\}$	00	0	0	d	1
	01	1	1	d	d
	10	d	1	0	d
	11	0	d	0	0
		Φ_1	Φ_2	Φ_3	Φ_4

表 3　列複雑度の削減

		$X_1 = \{x_1, x_2\}$			
		00	01	10	11
$X_2 = \{x_3, x_4\}$	00	0	0	1	1
	01	1	1	d	d
	10	1	1	0	0
	11	0	0	0	0
		Φ_1^*	Φ_2^*	Φ_3^*	Φ_4^*

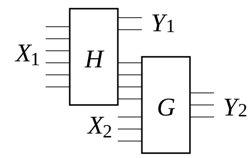


図 3　多出力関数の分解

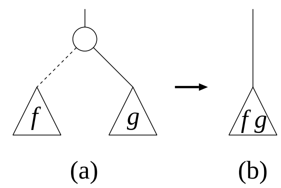


図 4　文献 [12] の簡単化法

合, BDD の幅が小さいほど分解後の回路も小さくなる. 不完全定義関数の場合, ドント・ケア割り当てを工夫することによって, BDD_for_CF の幅を減らせる場合がある. ここでは, 不完全定義関数を表現する BDD_for_CF の幅を削減する方法を考える.

［定義 6］　$(z_{n+m}, z_{n+m-1}, \ldots, z_1)$ を BDD の変数順序とする。ここで、z_{n+m} は、根節点に対応する。BDD_for_CF の**高さ k での幅**とは, 変数 z_k と z_{k+1} の間の枝の本数をいう. ここで, 同じ節点に接続している枝は一本と数える. BDD_for_CF の高さ 0 での幅を 1 と定義する。

［定義 7］　論理関数 $f(X)$ において, 変数の分割 (X_1, X_2) に対して, $2^{|X_1|}$ 行 $2^{|X_2|}$ 列の表で, 各行, 各列に 2 進符号のラベルを持ち, その要素が f の対応する真理値であるような表を**分解表**という. ここで, $|X|$ は集合 X の要素数である. また, 分解表において, 異なる列パターンの数を**列複雑度**といい, 記号 μ で表す. 列パターンが表す関数を**列関数**と呼ぶ.

［例 3］　表 2 に 4 入力 1 出力不完全定義関数の分解表を示す. 全ての列パターンが異なるので列複雑度 μ は 4 となる. (例終り)

BDD を用いた関数分解において, BDD の幅は列複雑度 μ に等しい. 変数の分割 (X_1, X_2) において, 変数 X_1 の節点に直接接続されている変数 X_2 の節点は, 分解表の列パターンに対応する.

関数分解では, $f(X_1, X_2) = g(h(X_1), X_2)$ と表現する. $\lceil \log_2 \mu \rceil < |X_1|$ が成立する時, X_1 を入力とし列関数の符号を生成する $h(X_1)$ の回路と, $g(H, X_2)$ の回路を個別に実現することによって, 関数 f を実現する. 関数分解は分解後の関数 g の入力数が少ない程効果的である. BDD の幅を削減できれば列複雑度が減り, 関数分解の効果が上がる.

［定義 8］　二つの不完全定義関数 f_a と f_b にドント・ケア割当を行うことで, 同じ関数にできる場合, f_a と f_b は**両立する**といい, $f_a \sim f_b$ と表す.

［補題 1］　二つの不完全定義関数の特性関数を χ_a, χ_b とする. $\chi_a \sim \chi_b$ かつ $\chi_c = \chi_a \chi_b$ のとき, $\chi_c \sim \chi_a$ かつ $\chi_c \sim \chi_b$ が成立する.

［例 4］　表 2 の分解表で, 列関数の対 $\{\Phi_1, \Phi_2\}$, $\{\Phi_1, \Phi_3\}$, $\{\Phi_3, \Phi_4\}$ は両立する. 列 Φ_1 と Φ_2 の論理積をとり, Φ_1^* と Φ_2^* と置き換え, 列 Φ_3 と Φ_4 の論理積をとり, Φ_3^* と Φ_4^* と置き換えると表 3 の様になり, $\mu = 2$ となる. (例終り)

［定理 1］　X_1, X_2 を入力変数の集合, Y_1, Y_2 を出力変数の集合とする. 不完全定義関数を表現する BDD_for_CF の変数順序を (X_1, Y_1, X_2, Y_2) とするとき, BDD_for_CF の (X_1, Y_1) における幅を μ とする. ここで, μ を数える際, 出力を表現する変数から定数 0 に向かう枝は無視する. 多出力関数を図 3 の回路で実現する場合, 二つの回路 H と G の間の接続線は $\lceil \log_2 \mu \rceil$ あれば十分である.

3.2　BDD_for_CF の幅を削減するアルゴリズム

不完全定義関数を表現する BDD の節点数削減法として種々の方法が知られている [3], [6], [11]〜[13]. 文献 [12] の方法では, 一つの節点に着目し, その節点の二つの子が両立する場合, 二つの子を併合して一つにすることを繰り返し, 節点数を削減する. 例えば, 図 4(a) に示す節点の子 f, g が両立する時, 図 4(b) のように, 節点を置き換えることで簡単化を行う.

今回の実験に用いたアルゴリズムを以下に示す.

［アルゴリズム 1］　[12]

BDD の根から再帰的に以下の処理を行う.

（1）　節点 v がドント・ケアを含まなければ終了.

（2）　v の二つの子 v_0, v_1 が両立するか調べる.

- 両立しなければ, v_0, v_1 に対して再帰する.
- 両立すれば, $v_{new} \leftarrow v_0 \cdot v_1$ とし, v_0, v_1 と置き換える. v_{new} に対して再帰する.

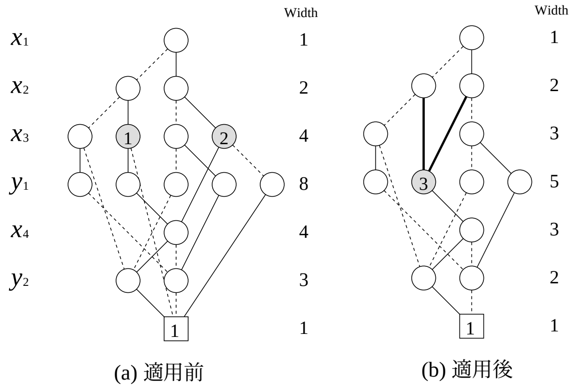


図 5　アルゴリズム 1 を適用した BDD_for_CF

［例 5］　図 2 の BDD_for_CF にアルゴリズム 1 を適用した結果を図 5 に示す. 図 5(a) がアルゴリズム適用前, 図 5(b) が適用後である. 図 5(a) の網掛の節点 1, 2 が両立する二つの子を持っている. この関数では, 節点 1 と置き換える節点と, 節点 2 を置き換える節点が同じになるので, 図 5(b) では, 節点 1, 2 が節点 3 に置き換えられている. 図の左にある Width は各高さでの幅である. 最大幅は 8 から 5 に, 非終端節点数は 15 から 12 に減っている.　(例終り)

文献 [12] の方法は, 局所的な節点数の削減には効果があるが, 一つの節点の子同士でしか両立性を考えないので, BDD の幅の削減には効果的ではない. そこで, 本研究では, BDD の幅を構成する全ての部分関数の両立性を調べ, マッチングを行うことで, 直接幅を削減する.

［定義 9］　関数を節点とし, 両立する関数同士を枝で接続したグラフを**両立グラフ**という.

関数分解において, 全ての列関数の両立性を調べ, 両立グラフを作成し, 最小数の完全部分グラフ (クリーク) による, 節点被覆を行うことで, 列複雑度 μ の最小化が可能である [13]. しかしながら, この問題は NP 困難であることが知られている [5]. そこで, 我々は発見的手法を用いる.

［アルゴリズム 2］　(クリーク集合による節点被覆)

与えられた両立グラフの全ての節点の集合を S_a とする. $\mathcal{C}$ を節点集合の集合とする. 枝を持たない節点を S_a から除去し, 要素が 1 つの節点集合 (クリーク) として $\mathcal{C}$ の要素とする.

$S_a \neq \phi$ の間以下の操作を繰り返す.

（1）　S_a 中で枝数最小の節点を調べ, これを v_i とする. $S_i \leftarrow \{v_i\}$ とする. S_a の中で v_i に接続している節点の集合を S_b とする.

（2）　$S_b \neq \phi$ の間以下の操作を間繰り返す.

（a）　S_b の中で枝数最小の節点 v_j を調べ, $S_i \leftarrow S_i \cup \{v_j\}$ とする.

（b）　v_j に接続していない節点を S_b から除去する.

（3）　$\mathcal{C} \leftarrow \mathcal{C} \cup \{S_i\}$, $S_a \leftarrow S_a - S_i$.

［アルゴリズム 3］　(BDD_for_CF の幅削減)

根節点の高さを t, 定数節点の高さを 0 とする. 高さ $t-1$ から 1 までの間, 以下の操作を繰り返す.

（1）　各高さにおける全ての列関数の集合を作る.

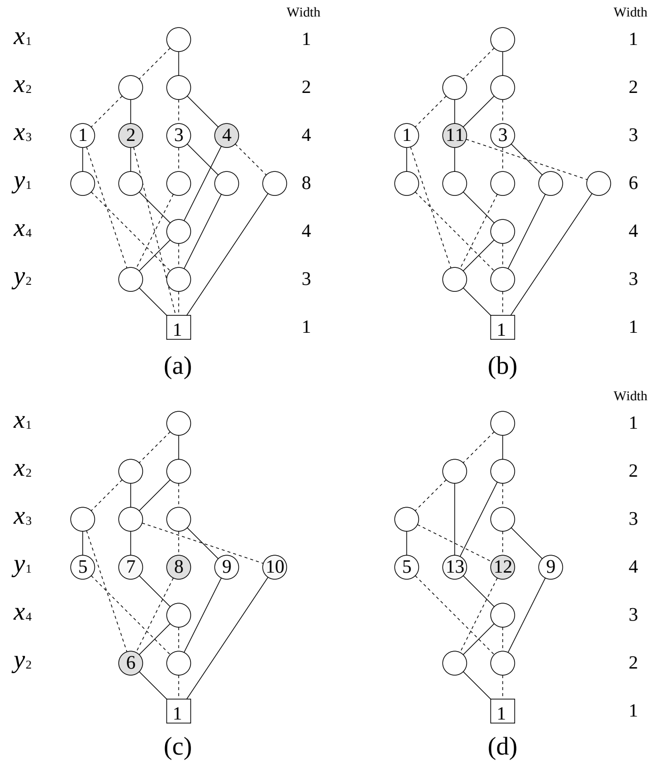


図 6　アルゴリズム 3 を用いた BDD_for_CF の幅削減

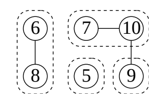


図 7　両立グラフ

（2）　列関数の集合から両立グラフを作成する.

（3）　クリーク集合による, 節点被覆 (アルゴリズム 2) を行う.

（4）　各クリークに被覆されている全ての節点に対応する関数にドント・ケアを割当て, 一つの関数にする.

（5）　各列関数を 4. で作成した関数に置き換えて, BDD を再構成する.

［例 6］　図 2 の BDD_for_CF にアルゴリズム 3 を適用した結果を図 6 に示す. 変数 x_3 の高さで, 図 6(a) の網掛の節点 2, 4 は両立するので, 図 6(b) で節点 11 に置き換えている. 次に, 変数 y_1 の高さでは, 両立グラフは図 7 となり, 図 6(c) の節点 6, 8 と節点 7, 10 を, 図 6(d) で節点 12, 13 に置き換えている. 図 6(a) と (d) では, 最大幅は 8 から 4 に, 非終端節点数は 15 から 12 に減っている. アルゴリズム 1 を適用した場合と比べて, 最大幅がより減っていることが分かる.　(例終り)

4. 実 験 結 果

4.1　ベンチマーク関数

今までの多くの論文では, 既存のベンチマーク関数に乱数を用いて生成したドント・ケアを加えて実験している. ここでは, アルゴリズム 3 の評価のために新たに生成した多出力回路について述べる. 以下に示す関数を不完全定義関数を表現する BDD_for_CF で表現し, 幅の削減を行う.

- 剰余数表現 (Residue Number System, RNS) [7] を 2 進表

現に変換する回路

（1） 5-7-11-13 RNS (14 入力 13 出力, DC 69.5%) [基数を (5,7,9,11) とした 4 桁の RNS 表現を 2 進表現に変換する回路]

（2） 7-11-13-17 RNS (16 入力 15 出力, DC 74.0%)

（3） 11-13-15-17RNS (17 入力 16 出力, DC 72.2%)

- i 桁の k 進数を 2 進数に変換する回路

（1） 4 桁 11 進 2 進変換 (16 入力 14 出力, DC 77.7%)

（2） 4 桁 13 進 2 進変換 (16 入力 15 出力, DC 56.4%)

（3） 5 桁 10 進 2 進変換 (20 入力 17 出力, DC 90.5%)

（4） 6 桁 5 進 2 進変換 (18 入力 14 出力, DC 94.0%)

（5） 6 桁 6 進 2 進変換 (18 入力 16 出力, DC 82.2%)

（6） 6 桁 7 進 2 進変換 (18 入力 17 出力, DC 55.1%)

（7） 10 桁 3 進 2 進変換 (20 入力 16 出力, DC 94.4%)

- i 桁の 10 進数の加算回路, 乗算回路

（1） 3 桁 10 進加算 (24 入力 16 出力, DC 94.0%)

（2） 4 桁 10 進加算 (32 入力 20 出力, DC 97.7%)

（3） 2 桁 10 進乗算 (16 入力 16 出力, DC 84.7%)

4.2 BDD の幅削減

4.1 で述べた不完全定義関数に対して, アルゴリズム 1 及びアルゴリズム 3 を適用して, BDD_for_CF の幅の削減を行った.

多出力関数の全ての出力を 1 つの BDD_for_CF で表現した場合, アルゴリズム 1 を用いた場合, アルゴリズム 3 の方法を用いた場合共に, 全てのドント・ケアに定数を割り当てた場合と比べて, ほとんど差がなかった. アルゴリズム 1 を用いた場合は節点数が 8.5%程度, アルゴリズム 3 を用いた場合は幅の合計が 2.4%程度しか削減できなかった.

次に BDD_for_CF を構成する前に, 各出力関数に対して, ドント・ケアの割当を工夫してサポート変数の数を削減した. BDD_for_CF の最大幅, 幅の合計, 節点数に関して, ドント・ケアに定数を割り当てた場合に比べ, 不完全定義関数 (3 値関数) を表現する BDD_for_CF では, 平均 55~57%, アルゴリズム 1 を適用した場合は平均 55~60%, アルゴリズム 3 を適用した場合は平均 56~57%削減できた. しかし, これらの BDD を一つの論理回路で実現するには, まだ幅が大き過ぎた.

多出力関数を単一の BDD_for_CF で表現した場合, 部分関数が両立するためには, 全ての出力値が両立する必要がある. 多出力関数をいくつかの出力集合に分割し, それぞれを BDD_for_CF で表現すれば, 部分関数が両立しやすくなり, 幅を削減できる可能性が上がる.

多出力関数の出力を二つに分割し, 各々を BDD_for_CF で表現して, 幅の削減を行った結果を表 4-6 に示す. 表中の $DC = 0$ は全てのドント・ケアに 0 を割り当てた場合, $DC = 1$ は全てのドント・ケアに 1 を割り当てた場合, ISF は不完全定義関数 (3 値関数) を表現する場合, Alg1 はアルゴリズム 1 を適用した場合, Alg3 はアルゴリズム 3 を適用した場合である. 表 4 は BDD_for_CF の最大幅, 表 5 は BDD_for_CF の幅の合計, 表 6 は BDD_for_CF の節点数を示している. 削減率は $DC = 0$ を 1.000 として正規化した平均値である. 今回用いた分割法は, 多出力関数 $F = (f_1, \ldots, f_m)$ を $F_1 = (f_1, \ldots, f_{\lceil m/2 \rceil})$, $F_2 = (f_{\lceil m/2 \rceil+1}, \ldots, f_m)$ と単純に二分割した.

表 4 二分割した場合の BDD_for_CF の最大幅

Name	DC=0	DC=1	ISF	Alg1	Alg3
5-7-11-13RNS	503 461	503 461	503 461	502 460	416 363
7-11-13-17RNS	471 1089	471 1089	471 1089	470 1088	337 896
11-13-15-17RNS	1574 2253	1574 2253	1574 2254	1573 2253	1573 1229
4 桁 11 進 2 進変換	117 257	117 257	129 264	123 264	117 128
4 桁 13 進 2 進変換	226 257	226 257	236 264	232 264	225 128
5 桁 10 進 2 進変換	393 257	393 257	393 78	392 76	391 64
6 桁 5 進 2 進変換	134 257	134 257	155 260	148 260	139 128
6 桁 6 進 2 進変換	185 257	185 257	189 89	188 64	184 32
6 桁 7 進 2 進変換	464 513	464 513	485 516	482 516	472 256
10 桁 3 進 2 進変換	265 513	265 513	305 514	304 514	294 256
3 桁 10 進加算	29 200	25 101	15 14	14 13	10 10
4 桁 10 進加算	85 1400	73 649	15 14	14 13	10 10
2 桁 10 進乗算	945 767	946 769	955 327	954 269	945 224
削減率	1.000	0.950	0.825	0.811	0.636

表 5 二分割した場合の BDD_for_CF の幅の合計

Name	DC=0	DC=1	ISF	Alg1	Alg3
5-7-11-13RNS	2306 2036	2319 2036	2319 2048	2302 2032	1979 1743
7-11-13-17RNS	3064 4966	3075 4966	3076 4980	3054 4960	2392 4250
11-13-15-17RNS	7718 10495	7737 10495	7738 10510	7715 10488	6167 7600
4 桁 11 進 2 進変換	1280 2170	1285 2170	1322 2278	1284 2277	1245 1414
4 桁 13 進 2 進変換	2340 2242	2346 2242	2435 1989	2394 1989	2336 1179
5 桁 10 進 2 進変換	3278 2339	3285 2339	3287 623	3261 589	3252 444
6 桁 5 進 2 進変換	1457 1889	1460 1889	1504 1838	1429 1819	1408 1224
6 桁 6 進 2 進変換	1324 1863	1331 1863	1407 474	1374 386	1328 308
6 桁 7 進 2 進変換	5252 4762	5258 4762	4995 4554	4950 4554	4844 2920
10 桁 3 進 2 進変換	3007 4691	3013 4691	3114 4466	3089 4465	3003 3292
3 桁 10 進加算	229 2056	261 1244	171 143	138 124	125 107
4 桁 10 進加算	561 12071	607 7259	232 208	189 178	172 163
2 桁 10 進乗算	3218 3130	3228 3143	3305 2137	3267 1957	3199 1714
削減率	1.000	0.979	0.830	0.805	0.680

二分割を行うことによって, 全ての関数で BDD を大きく削減できた. 関数によっては, BDD の幅の最大値が 2000 分の 1 以下, 総節点数が 70 分の 1 以下になったものもある. 二分割を行うことによって, 現実的な大きさの論理回路で実現できるようになった.

全体として, アルゴリズム 1 を用いた場合は, アルゴリズム 3 を用いた場合よりも節点数の削減に効果がある. しかし, 幅の削

表 6　二分割した場合の BDD_for_CF の節点数

Name	DC=0	DC=1	ISF	Alg1	Alg3
5-7-11-13RNS	2297	2310	2303	2039	1980
	2027	2027	2033	1784	1744
7-11-13-17RNS	3051	3062	3055	2664	2393
	4954	4954	4961	4496	4251
11-13-15-17RNS	7704	7723	7716	7039	6168
	10482	10482	10489	8948	7601
4 桁 11 進 2 進変換	1252	1257	1270	1133	1231
	2157	2157	2231	2100	1414
4 桁 13 進 2 進変換	2325	2331	2410	2225	2337
	2231	2231	1974	1942	1180
5 桁 10 進 2 進変換	3260	3267	3260	2794	3251
	2322	2322	593	527	439
6 桁 5 進 2 進変換	1442	1445	1459	1150	1396
	1875	1875	1816	1718	1218
6 桁 6 進 2 進変換	1310	1317	1367	1185	1328
	1849	1849	445	299	307
6 桁 7 進 2 進変換	5243	5249	4944	4625	4845
	4748	4748	4534	4406	2921
10 桁 3 進 2 進変換	2991	2997	3073	2510	2994
	4674	4674	4444	3150	3289
3 桁 10 進加算	197	226	134	100	109
	1643	1045	125	95	108
4 桁 10 進加算	517	560	181	135	152
	10051	6098	177	136	160
2 桁 10 進乗算	3020	3027	3072	2743	3007
	3063	3073	2024	1635	1616
削減率	1.000	0.981	0.815	0.711	0.682

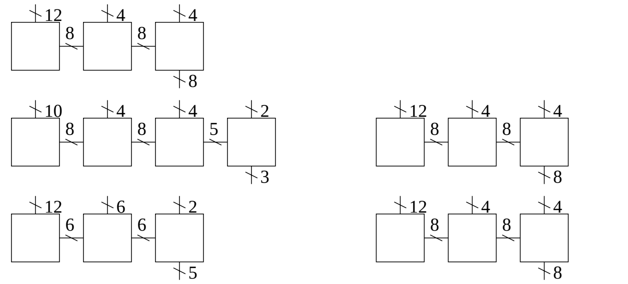
減に関してはアルゴリズム 3 ほどの効果はなかった. アルゴリズム 1 では両立するかどうかを調べる対象が, 一つの節点の子同士に限られるため, 節点数の局所的な削減には効果があるが, 幅の大域的な削減には効果が出にくい. ただし, 探索範囲が狭いため, 非常に高速である. アルゴリズム 3 では, 幅全体の両立性を調べるため, 幅削減の効果は高い. しかし, 幅を w とすると, $\frac{w^2-w}{2}$ の両立性を調べる必要があるため, アルゴリズム 1 よりも計算時間が長くなる.

4.3　LUT カスケードの合成

実際の応用では, BDD の幅が 2^k より少し大きい場合に, ドント・ケア割当を工夫することにより, 図 3 の二つのブロック H と G の間の配線を削減できる. この効果を調べるために, LUT カスケード合成システム [9] にアルゴリズム 3 を組み込み, 10 桁 3 進数を 2 進数に変換する回路を合成した. セルの入力数の最大を 12, 出力数の最大を 8 として合成を行なった. 図 8 に合成した回路を示す. 図 8(a) は, 全てのドント・ケアに 0 を割り当てた場合, 図 8(b) は, アルゴリズム 3 を適用した場合である. アルゴリズム 3 により, セルの出力数の総和は 65 から 48 に, 総段数は 10 から 6 に削減できた. 次に, 4 桁 10 進数加算回路の LUT カスケードをセルの入力数の最大を 9, 出力数の最大を 8 として合成した. アルゴリズム 3 により, セルの出力数の総和は 95 から 20 に, 総段数は 20 から 4 に削減できた.

5.　あとがき

本論文では, BDD_for_CF を用いて不完全定義多出力関数を表現する方法と, その幅を削減する方法について述べた. また, この手法を剰余数変換回路, 基数変換回路, および 10 進の加算回路と乗算回路に適用した. 多出力関数の全ての出力を単一の



(a) 全てのドント・ケアに 0 を割り当てた場合　(b) アルゴリズム 3 を適用した場合

図 8　10 桁の 3 進数を 2 進数に変換する LUT カスケード

BDD_for_CF で表現した場合には, ドント・ケアを用いても幅を削減できなかったが, 出力を二分割し, 2 個の BDD_for_CF で表現すると, 各々の BDD の幅を削減できた. 本手法は LUT カスケードの合成に有効である.

謝　辞

本研究は, 一部, 科学研究費補助金, および, 北九州地域知的クラスター創成事業の補助金による.

文　献

[1] R. L. Ashenhurst, "The decomposition of switching functions," *In Proceedings of an International Symposium on the Theory of Switching*, pp. 74-116, April 1957.

[2] P. Ashar and S. Malik, "Fast functional simulation using branching programs," *Inter. Conf. on CAD*, pp. 408-412, Nov. 1995.

[3] S. Chang, D. Cheng, and M. Marek-Sadowska, "Minimizing ROBDD size of incompletely specified multiple output functions," *In European Design & Test Conf.*, pp. 620-624, 1994.

[4] K. Cho and R. E. Bryant, "Test pattern generation for sequential MOS circuits by symbolic fault simulation," *Proc. 26th Design Automation Conf.*, pp. 418-423, June 1989.

[5] M. R. Garey and D. S. Johnson, *Computers and Intractability*, Freeman, SanFransisco, 1979.

[6] Y. Hong, P. Beerel, J. Burch, and K. McMillan, "Safe BDD minimization using don't cares," *Proc. Design Automation Conference*, pp. 208-213, 1997.

[7] I. Koren ,*Computer Arithmetic Algorithms (2nd Edition)*, A. K. Peters, Natick, MA, 2002.

[8] S. Minato, N. Ishiura, and S. Yajima, "Shared binary decision diagram with attributed edges for efficient Boolean function manipulation," *27th Design Automation Conf.*, pp. 52-57, June 1990.

[9] T. Sasao and M. Matsuura, "A method to decompose multiple-output logic functions," *41st Design Automation Conference*, pp. 428-433, San Diego, USA, June 2-6, 2004.

[10] M. Sauerhoff and I. Wegener, "On the complexity of minimizing the OBDD size for incompletely specified functions," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 15, No. 11, pp. 1435-1437, 1996.

[11] C. Scholl, S. Melchior, G. Hotz, and P. Molitor, "Minimizing ROBDD sizes of incompletely specified functions by exploiting strong symmetries," *European Design & Test Conf.*, pp. 229-234, 1997.

[12] T. R. Shiple, R. Hojati, A. L. Sangiovanni-Vincentelli, and R. K. Brayton, "Heuristic minimization of BDDs using don't cares," *Design Automation Conference 1994*, pp. 225-231, 1994.

[13] W. Wan and M. A. Perkowski, "A new approach to the decomposition of incompletely specified functions based on graph-coloring and local transformations and its application to FPGA mapping," *Proc. of the IEEE EURO-DAC'92*, pp. 230-235, Hamburg, Sept. 7-10, 1992.