関数分解に基づくLUT型FPGA用ブーリアンマッチングアルゴリズムについて

松 永 裕 介†

LUT型のFPGAは1つの基本ブロックで定められた入力数（通常4または5）以下の任意の論理関数を実現できるという特徴を持つ．そのため，従来は対象回路の論理関数を考慮せずに構造のみに注目したテクノロジマッピング手法が用いられてきた．ところが，実際のFPGAの基本ブロックの中にはXilinx社のXC4000のように5入力以下の任意の論理関数だけでなく，6入力以上の一部の論理関数を実現できるものが存在する．そのような特殊な場合のマッピングを考慮するためには，マッピング対象の回路の論理関数を考慮したブーリアンマッチングを行う必要がある．本稿ではそのための関数分解に基づくブーリアンマッチングアルゴリズムについて述べ，FPGA用の深さ最小の回路を求めるテクノロジマッピングに適用した結果を示す．

On Boolean Matching Algorithm for LUT-type FPGA Based on Functional Decomposition

Yusuke Matsunaga†

LUT-type FPGA can implement any logic function with up to a certain number of inputs (generally 4 or 5). Thus, the existing technology mappers take care about only the structure of a circuit to be mapped, and do not consider its logic function for efficiency. While, there are FPGA architectures whose basic block consists of a tree of LUT's like Xilinx XC4000. In such a case, the basic block can implement some logic functions whose inputs are more than 6. This paper proposes Boolean matching algorithm for such an FPGA architecutre. The proposed algorithm is based on an efficient implementation of simple disjoint decomposition algorithm using binary decision diagrams. Experimental results for depth-minimum technology mapping are also shown.

1. はじめに

LUT（look up table）型のFPGA（field programmable array）は1つの基本ブロックで定められた入力数（通常4または5）以下の任意の論理関数を実現できるという特徴を持つ．そのため，従来は対象回路の論理関数を考慮せずに構造のみに注目したテクノロジマッピング手法が用いられてきた．より具体的には，対象回路をいったん2入力NANDゲートとインバータのみからなる回路に分解し，1つのLUTで実現できるゲートの固まり（クラスタ）を列挙して，そのクラスタを組み合わせて回路全体を覆うという手法である．このクラスタが1つのLUTで実現できるかどうかの判断はそのクラスタの入力数がLUTの入力数以下かどうかで行えるため，非常に効率が良い．この性質を利用していくつかのテクノロジマッピングアルゴリズムが提案されている[1)~3)]．

ところが，複数のLUTを組み合わせて論理関数を実現する場合には，その入力数からだけでは判断できない場合がある．たとえば，Xilinx社のXC4000シリーズの基本ブロック（PLB: Programmable Logic Block）の組合せ回路部分の構造は**図1**に示すように，2つの4-LUTと1つの3-LUTから構成されている☆．この1つのPLBでは，

(1) 2つの任意の4入力1出力関数
(2) 1つの任意の5入力1出力関数
(3) 最大9入力の特定の1出力論理関数

を実現することが可能である．このうち，(1)と(2)は実現できるかどうかの判定として入力数を調べるだけで行えるので論理関数を考慮する必要がない．しか

† 九州大学
Kyushu University

☆ 最近のXilinx社のFPGAはこのような構造をとらず，2つの4入力LUTを組み合わせたものが多い．

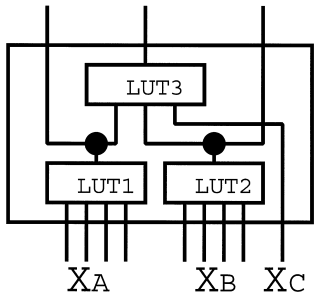

図 1 XC4000 の基本ブロック
Fig. 1 PLB of XC4000.

し，最後の 3 の場合には与えられた論理関数が 1 つの PLB で実現可能なのかどうかを判定するためには論理関数を考慮した処理が必要となる．Cong と Hwang はこの判定を行うブーリアンマッチングアルゴリズムを提案した[4)~6)]．しかし，彼らのアルゴリズムはすべての変数の分割をしらみつぶしに列挙して PLB の構造に合った関数分解が存在するか調べる，どちらかというとナイーブなもので効率的とはいいがたい．本稿では，このような FPGA のテクノロジマッピングのブーリアンマッチング問題に対する高速化手法に対して述べる．また，このブーリアンマッチングを用いて遅延最適化テクノロジマッピングを行った結果についても示す．このようなテクノロジマッピング技術は特定の FPGA アーキテクチャのみに有効なわけではなく，LUT をベースにした FPGA デバイスならば多少の変更で応用可能である．たとえば文献 7) で提案されているような，LUT に加算の桁あげ信号を伝播する機構を付加したような基本ブロック構造を持つ FPGA デバイスに対するマッパに用いることもできると思われる．

2. ブーリアンマッチングアルゴリズム

2.1 用語の定義

ここでは本稿で用いられる用語の定義を行う．

$\{0,1\}^n \to \{0,1\}$ の写像を表す論理関数 $F(X)$ に対して，ある入力変数 $x \in X$ を 0 または 1 に固定した関数 $F|_{x=0}$，$F|_{x=1}$ を F の x による**コファクタ（cofactor）**と呼ぶ．$F|_{x=0}$，$F|_{x=1}$ は $F_{\overline{x}}$,F_x と表されることもある．また，コファクタをとる変数 x が自明な場合には F_0，F_1 と表すことにする．相異なる 2 つの変数 $x_1, x_2 \in X$ および，任意のブール値 $b_1, b_2 \in \{0,1\}$ に対して，$(F|_{x_1=b_1})|_{x_2=b_2} = (F|_{x_2=b_2})|_{x_1=b_1}$ が成り立つので，論理関数 F に対する複数の入力変数のコファクタも同様に定義することができる．論理関数 F に対して x による 2 つのコファクタが異なるとき（$F_{\overline{x}} \neq F_x$），$F$ は変数 x に**依存**しているという．論理関数 $F(X)$ の**サポート（support）**とは F が依存している変数の集合である．

論理関数 $F(X)$ を次のような 2 つの関数 G，H を用いて表すことができるとき，これを関数 F の**関数分解（functional decomposition）**と呼ぶ．

$$F(X) = G(X_1, H(X_2)) \quad (1)$$

論理関数 F が式 (1) の形の分解を持つとき，関数 H のサポート X_2 を**束縛集合（bound set）**と呼ぶ．また，関数 G のサポートから H を除いたもの（$= X_1$）を**自由集合（free set）**と呼ぶ．

2 つの（変数）集合 A および B が共通な要素を持たないとき，すなわち，$A \cap B = \phi$ のとき，2 つの集合は**直交**するといい，$A \perp B$ と表記する．

式 (1) の関数 H が二値の論理関数の場合を単純な分解（simple decomposition）呼ぶ．H が多値の場合，もしくは，複数の二値論理関数のベクタの場合を複雑な分解（complex decomposition）と呼ぶ．変数集合 X_1 と X_2 が互いに素な場合（$X_1 \perp X_2$）を直交な分解（disjoint decomposition）と呼び，そうでない場合を直交でない分解（non-disjoint decomposition）と呼ぶ．

2.2 直交分解と分解グラフ

直交な分解は以下のような性質を持つ．

(1) もしも F が $G(X_1, H(X_2))$ と $G'(X_1', H'(X_2'))$ という 2 つの直交分解を持ち，かつ H および H' はこれ以上分解されない最小の関数だとすると，$X_1 \supseteq X_2'$ であるかまたは $X_1' \supseteq X_2$ である．つまり，複数の直交分解が存在した場合，それらをいかなる順序で適用しても最終的にはこれ以上分解できないユニークな分解が存在する．

(2) 論理関数 F が次のような分解を持つものとする．

$$F = G(X_1, H(X_2))$$

ここで X_1 に属する変数 $x (x \in X_1)$ に対する F のコファクタを考えると，

$$F_0 = G_0(X_1 - \{x\}, H(X_2)) \quad (2)$$

$$F_1 = G_1(X_1 - \{x\}, H(X_2)) \quad (3)$$

のようになる．ここで，F_0 は F の $x = 0$ に対するコファクタを表す（$F_0 = F|_{x=0}$）．同様に，G_0 は G の $x = 0$ に対するコファクタを表すものとする．

逆に，もしも F のコファクタが上記の式を満たすのならば，$G = \overline{x} \cdot G_0 + x \cdot G_1$ とすることで，次のような関数分解を得ることができる．

$$F = G(X_1, H(X_2)) \quad (4)$$

同様に，x が X_2 に属するとすると，

$$F_0 = G(X_1, H_0(X_2 - \{x\})) \quad (5)$$

$$F_1 = G(X_1, H_1(X_2 - \{x\})) \quad (6)$$

となり，こちらも $H = \overline{x} \cdot H_0 + x \cdot H_1$ とすることで，式 (4) を得ることができる．

Bertacco らは論理関数を二分決定グラフ（Binary Decison Diagram: BDD）[8] で表し，その二分決定グラフを再帰的にたどることによって直交分解を求めるアルゴリズムを提案した[9]．著者も同様のアルゴリズムを提案している[10),11)]．

ここでは文献 10)，11) のアルゴリズムで用いられる分解グラフを簡単に説明しておく．分解グラフ $DG(V, E)$ は節点の集合 V と枝の集合 E から成る．節点には以下の種類がある．

定数 0： 定数 0 を表す．入枝は持たない．このタイプは元の関数が定数関数である場合以外には用いられることはない．

リテラル： リテラル関数を表す．変数番号を持つ．入枝は持たない．

OR： OR 関数を表す．任意の数の入枝を持つ．

XOR： XOR 関数を表す．任意の数の入枝を持つ．

Other： それ以外の（単純でない）関数を表す．任意の数の入枝を持つ．

各々の節点は分解の構造と同時に論理関数も表している．そこで，各々の節点はその関数を表す BDD の節点へのポインタを持つ．

各々の枝は極性の属性を持つ．文献 12) の BDD の実装と同様に，否定の属性を持った枝の表す論理関数は，その枝の指している節点の表す論理関数の否定である．

分解グラフをカノニカルに保つために，以下のような規則を設ける．

- **XOR** 節点の入力の枝には否定の属性を付けない．
- **Other** 節点の入力の枝には否定の属性を付けない．
- **Other** 節点の出力の枝の極性は BDD の枝の極性と同一にする．

2.3 直交分解の集合の列挙

前節で説明した分解グラフは与えられた論理関数に対するすべての直交分解を内在的に表しているものと見なすことができる．ここでは分解グラフから直交分解を列挙するアルゴリズムについて説明を行う．

基本的には分解グラフ上のある節点を根とした部分グラフ（部分木）がある分解の関数 H に相当する（**図 2**）ので，分解グラフ上の各節点を訪れて，対応する分解を列挙すればよい．ただし，節点のタイプが OR と XOR の場合には注意が必要となる．たとえば，

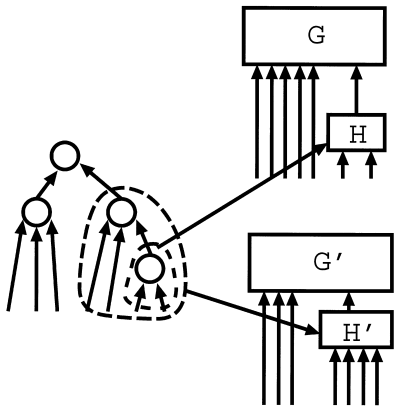

図 2 分解グラフと直交分解
Fig. 2 Decomposition graph and corresponding disjoint decomposition.

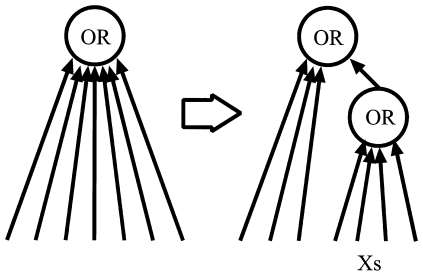

図 3 OR ノードの分解
Fig. 3 Decomposition of OR-node.

変数集合 X を入力とする OR 節点を考えると，X の任意の部分集合 X_S を束縛変数とする分解が存在する（**図 3**）．つまり，入力数 n の OR 節点は非明示的に 2^n 通りの異なる分解を表していると見なすことができる．以上を考慮したアルゴリズムが**図 4** である．直交分解の場合，関数に対して変数の分割を与えれば唯一の分解を得ることができる☆ので，ここでは変数の分割 (X_1, X_2) を用いて関数分解を表している．分解グラフの節点から入力側にたどることで自分が依存している変数の集合を得ることができる．それを X_2 として分解を登録している．X_1 は全変数から X_2 を引くことで得ることができる．

2.4 関数分解の標準形

一般に関数分解は，分解された子関数 G，H に対して再帰的に適用できるので式 (1) よりも複雑な形になりうるが，本稿では図 1 の形の関数分解を持つかどうか判定することを目的としているので，以降では式 (7) の形を関数分解の標準的な形として扱うものと

☆ 厳密には束縛集合に依存した関数 H の出力の極性の選び方で 2 通り存在する．

```
EnumDec(node, dec_list)
{
1:    node を根とする部分木の表す分解を
      dec_list に登録
2:    if (node のタイプが OR か XOR) {
3:        foreach node のファンインの部分集合 S {
4:            S の表す分解を dec_list に登録
5:        }
6:    }
7:    foreach node のファンイン fi_node {
8:        EnumDec(fi_node, dec_list)
9:    }
}
```

図 4 直交分解を列挙するアルゴリズム
Fig. 4 An algorithm enumerating all the disjoint decomposition.

する．

$$F(X) = C(A(X_A), B(X_B), X_C) \quad (7)$$

明らかなように関数 $A(X_A)$ が LUT1 に，関数 $B(X_B)$ が LUT2 に，関数 $C(a, b, X_C)$ が LUT3 に対応していたら，関数 $F(X)$ は図 1 の形の FPGA で実現できることになる．ただし，変数集合 X_A，X_B，X_C は以下の制約を満たす必要がある．

$$X = X_A \cup X_B \cup X_C \quad (8)$$

$$|X_A| \leq 4 \quad (9)$$

$$|X_B| \leq 4 \quad (10)$$

$$|X_C| \leq 1 \quad (11)$$

逆に，関数 $F(X)$ が式 (7) の形に関数分解できないとき，図 1 の形の FPGA で実現することはできない．

2.5 マッチングの場合分け

図 1 の 9 つの入力は互いに独立であるが，このうちのいくつかをブリッジ接続することも可能である．ここではこのブリッジ接続によって場合分けを行う．

まず，LUT3 の入力 X_C が他の変数集合と交わっているかどうかで次のように 3 種類に分類を行う．

Type-A： $X_A \perp X_C$ かつ $X_B \perp X_C$．つまり LUT3 の入力は他とブリッジしない．

Type-B： $X_A \perp X_C$ かつ $X_B \not\perp X_C$．つまり LUT2 の入力と LUT3 の入力がブリッジ．

Type-C： $X_A \not\perp X_C$ かつ $X_B \not\perp X_C$．つまり LUT1，LUT2 両方の入力と LUT3 の入力がブリッジ．

次に，LUT1 の入力 X_A と LUT2 の入力 X_B が交わっているかどうかについて，次のように 4 種類の分類を行う．ただし LUT3 の入力 X_C とのブリッジは考慮しない．

表 1 分類法の比較
Table 1 Comparison of classification of decomposition type.

文献 5) の分類	本稿の分類
A.1, B.1	A-0
A.2, B.2	A-1, A-2, A-3
C.1	C-0, C-1
C.2	B-0, B-1, B-2

Type-0： $|(X_A \cap X_B) \setminus X_C| = 0$.

Type-1： $|(X_A \cap X_B) \setminus X_C| = 1$.

Type-2： $|(X_A \cap X_B) \setminus X_C| = 2$.

Type-3： $|(X_A \cap X_B) \setminus X_C| = 3$.

上記の 2 種類の分類法は互いに独立であり直交しているので任意の組合せが可能である（たとえば A-0, C-3 など：各々Type-A と Type-0，Type-C と Type-3 の組合せを表すものとする）．しかし，このうちのいくつかは冗長なため除外することができる．具体的には，たとえば C-3 は独立な入力を 4 つしか持たない．4 入力の論理関数なら 1 つの LUT で実現可能なためわざわざこのように複雑な場合を考慮する必要はない．同様に，ブリッジ接続の結果残った入力数が 5 以下の場合には自明に実現可能なため考慮する必要がない．このことを考慮すると，考慮すべきケースは以下の 9 種類となる．

- A-0，A-1，A-2，A-3
- B-0，B-1，B-2
- C-0，C-1

参考までに本稿における分類と Cong と Hwang のアルゴリズムにおける分類 5) の差異を**表 1** に示す．

文献 5) では X_c が空かどうかで分類を A 群と B 群に分けているのに対して，本稿では X_c の有無にかかわらず同一のアルゴリズムを適用できるので区別をしていない．また，文献 5) では LUT2 と LUT3 の間に何本のブリッジがあるのかを区別していないこと，および本稿のタイプ B とタイプ C を大きく C 群とひとまとめにしていることが異なっている．実際には C.1 と C.2 では適用するアルゴリズムが異なるし，A 群と B 群ではブリッジがない場合とある場合とでサブグループを区別しているのに対して C 群では異なったサブグループの定義をしているなど，文献 5) の分類法は統一性を欠いていると思われる．

これに対して，本稿で提案している分類法は 2 種類の直交する分類法を組み合わせた分かりやすいものであり，また，個々の分類とそのためのマッチングアルゴリズムが 1 対 1 に対応している点も特長となって

いる．

以降ではまず LUT1 と LUT2 の間にブリッジの存在しない Type-0 の 3 種類の場合に対するマッチングアルゴリズムについて述べたあとで，それ以外のマッチングへの拡張を述べる．

2.6 タイプ A-0 に対するマッチング

A-0 の場合にはブリッジ接続がないため，式 (7) の関数分解は直交分解となる．つまり，与えられた関数 $F(X)$ が式 (7) の形の直交分解を持つかどうか調べればよい．そのために文献 10) の直交分解アルゴリズムを用いて $F(X)$ に対する分解グラフを作り，さらに直交分解の束縛集合と自由集合の対の集合を列挙する（図 4 の EnumDec アルゴリズム）．求められた束縛集合と自由集合の対の集合を $D = \{(B_0, F_0), \ldots, (B_i, F_i), \ldots\}$ とする．B_i と F_i はそれぞれ束縛集合と自由集合を表すものとする．この中で次のような条件を満たす 2 つの対 (B_i, F_i) と (B_j, F_j) が存在すれば A-0 のマッチングが存在する．

$$B_i \perp B_j$$
$$|X - B_i - B_j| \leq 1$$

この場合，B_i に対応した束縛関数を $A(X_A)$ に，B_j に対応した束縛関数を $B(X_B)$ することでもとの関数 F を実現することができる．

2.7 タイプ B-0 に対するマッチング

B-0 の場合には LUT1 を束縛関数と見なした形の直交分解が存在する．つまり，

$$F(X) = G(X_1, H(X_2))$$
$$|X_1| \leq 4$$
$$|X_2| \leq 4$$
$$X_1 \perp X_2$$

の形の関数分解が存在する．そこで，EnumDec アルゴリズムを用いて，$|B_i| \leq 4$ かつ $|F_i| \leq 4$ となる分解が存在するかどうかを調べる．そのような分解が存在した場合には（明らかに $B_i = X_2$，$F_i = X_1$ となる），その分解の自由関数 $G(X_1, h)$ が $U(V(X_1), x, h)$ の形の形の分解を持つかどうか調べればよい．ただし，この分解は直交分解ではない（$x \in X_1$）ので単純に求めることはできない．そこで，重複して用いられる入力を仮定して分解が存在するか確かめる方法を用いる．具体的には，関数 $G(X_1, h)$ および，変数 $x \in X_1$ に対して式 (13)，(14) を満たす分解が存在するか調べればよい．ただし，$X' = X_1 - \{x\}$ とする．

$$G(X_1, h) = U(V(X_1), x, h) \quad (12)$$

式 (12) の両辺を変数 x でコファクタリングすると次式を得る．

$$G_0(X', h) = U_0(V_0(X'), h) \quad (13)$$
$$G_1(X', h) = U_1(V_1(X'), h) \quad (14)$$

つまり，もとの関数 $G(X_1, h)$ に対してある変数 x でコファクタリングしたときに式 (13)，(14) の形のように，変数 h との二項分解（bi-decomposition）を持つことがこのタイプの分解が存在するための必要条件となる．一方，式 (13)，(14) の関係を満たしている場合には，

$$U = x \cdot U_1(v, h) + \bar{x} \cdot U_0(v, h)$$
$$V = x \cdot V_1(X') + \bar{x} \cdot V_0(X')$$

とすることで（v は関数 V の出力を表す変数である），$G(X_1, h) = U(V(X_1), x, h)$ の形に分解することができるので，式 (13)，(14) は十分条件でもある．よって，これらの式の形の分解（つまり h との二項分解）を持つことがこのタイプの分解が存在するための必要十分条件となる．実際にはどの変数でコファクタリングすればよいかは試さなければ分からないので，X_1 に含まれるすべての変数に対してコファクタリングして，式 (13)，(14) の分解を持つか調べる必要があるが，X_1 の要素数はたかだか 4 なので計算効率の面では問題ではない．また，ある関数 F が与えられた変数 h との二項分解を持つかどうかは h でコファクタリングすれば容易に判定可能である．

2.8 タイプ C-0 に対するマッチング

この形の分解は

$$F(X) = C(A(X_A, x_1), B(X_B, x_1), x_1) \quad (15)$$

と書き表すことができる（ただし，$X_A \perp X_B$，$|X_A| \leq 3$，$|X_B| \leq 3$）．そこで，この式の両辺を x_1 でコファクタリングすると，

$$F_0(X_1) = C_0(A_0(X_A), B_0(X_B)) \quad (16)$$
$$F_1(X_1) = C_1(A_1(X_A), B_1(X_B)) \quad (17)$$

となる．タイプ B-0 の場合と同様の議論により，この式 (16)，(17) の形の分解を持つことがタイプ C-0 の形の分解が存在することの必要十分条件となる．この判定にはタイプ A-0 の場合と同様に，まず，すべての単純直交分解をもとめ，その中で式 (16)，(17) の条件を満たす 2 つの変数集合 X_A と X_B を見つければよい☆．

このタイプの関数の最大入力数は 7 なので，最大 7 回，変数を選んでコファクタリングを行う必要がある．

2.9 タイプ A-1，A-2，A-3 に対するマッチング

タイプ A-0 の場合と異なり，これらは直交分解ではないので，そのまま単純直交分解アルゴリズムを適用

☆ これに対し Cong と Hwang のアルゴリズムではすべての変数分割（これは変数集合の要素数の指数乗，存在する）を列挙して，それが上式を満たすか調べている．

することはできない．そこで，ブリッジしている変数でコファクタリングすることによって各 LUT 間で共通に用いられる変数を削除したあとで単純直交分解を適用する．

まず，タイプ A-1 を考える．2 つの LUT 間で共通に用いられる変数を x_1 とするとタイプ A-1 の分解は

$$F(X) = C(A(X_A, x_1), B(X_B, x_1), x_c) \quad (18)$$

と書き表すことができる（ただし，$X_A \perp X_B$，$|X_A| \leq 3$，$|X_B| \leq 3$）．この式の両辺を x_1 でコファクタリングすると，

$$F_0(X_1) = C(A_0(X_A), B_0(X_B), x_c) \quad (19)$$

$$F_1(X_1) = C(A_1(X_A), B_1(X_B), x_c) \quad (20)$$

を得る．もとの分解の形はタイプ C-0 と似ているが，変数 x_1 が関数 C の直接の入力となっていないため，x_1 によるコファクタリングでは根本の関数 C が変化しないことがタイプ C-0 との大きな差異となっている．今までのケースと同様の議論で，式 (19)，(20) の形の分解を持つことが必要十分条件となる．ただし，今まで述べてきたケースの場合には単に変数集合のみが同一な分解を持つかどうかが条件になっていたが，このタイプの場合には根本の関数 C が同一であるという，より厳しい条件になっている．さらに，一般に関数分解は，分解を行う変数集合を固定しても，束縛関数の出力をどう符号化（codeing）するかよって異なる形の分解になりうるため，根本の関数が同一かどうかの判定は単純には行えない．式 (19)，(20) の場合には符号化を考慮すべき束縛関数は $A_0(A_1)$ と $B_0(B_1)$ の各々2 つである．これらは 2 値関数なので符号化の仕方は 2 通り（そのままと否定）あり，全部で $2^2 = 4$ 通りのバリエーションが存在することになる．さらに A_0 などが定数になる場合も考慮する必要があるので，具体的な判定処理は複雑なものとなる．とはいえ，同一か調べる関数 C はたかだか 3 入力の関数なので，入力変数を反転したり，定数に固定したりすることで同一になるかどうかのチェックは表引きで容易に行うことができる．

タイプ A-2，A-3 に対しても同様にまずブリッジしている 2 つもしくは 3 つの変数を仮定してコファクタリングすることによって分解が存在するかを判定することができる．ただし，タイプ A-2，A-3 の場合にはこの方法はあまり効率が良いとはいえないので Cong らが用いているような Partially-Dependent Functional Decomposition[4] の手法を応用するのが有効と思われる．

タイプ A-1，A-2，A-3 の関数の最大入力数はそれぞれ 8，7，6 であり，ブリッジする変数の選び方はそれぞれ 8，21，20 通りとなる．

2.10 タイプ B-1，B-2 に対するマッチング

基本的にはタイプ A-1 と同様のアプローチを用いる．たとえば，タイプ B-1 の場合に変数 x_1 が 2 つの LUT をブリッジしている変数とするとこの分解は，

$$F(X) = C(A(X_A, x_1), B(X_B, x_1, x_c), x_c) \quad (21)$$

と書き表すことができる（ただし，$X_A \perp X_B$，$|X_A| \leq 3$，$|X_B| \leq 2$）．この式の両辺を x_1 でコファクタリングすると，

$$F_0(X_1) = C(A_0(X_A), B_0(X_B, x_c), x_c) \quad (22)$$

$$F_1(X_1) = C(A_1(X_A), B_1(X_B, x_c), x_c) \quad (23)$$

を得る．式 (22)，(23) の形の分解が存在するかどうかの判定は各関数の最大入力数が異なる以外はタイプ B-0 の判定と同様に行える．その後で根本の関数 C が同一かどうかをタイプ A-1 と同様の手法で判定すればよい．タイプ B-2 もコファクタリングする変数が 2 つに増える以外は同様に行うことができる．

タイプ B-1 の関数の最大入力数は 7 であり，ブリッジする変数の選び方は 7 となる．さらにその中で最大 3 つの変数を X_c とのブリッジ変数として選ぶことができるので，トータルで 21 通りの変数選択を試す必要がある．

タイプ B-2 の関数の最大入力数は 6 であり，ブリッジする変数の選び方は 15 となる．さらにその中で最大 2 つの変数を X_c とのブリッジ変数として選ぶことができるので，トータルで 30 通りの変数選択を試す必要がある．

2.11 タイプ C-1 に対するマッチング

これもタイプ A-1 と同様のアプローチを用いる．変数 x_1 が 2 つの LUT をブリッジしている変数とすると，この分解は，

$$F(X) = C(A(X_A, x_1, x_c), B(X_B, x_1, x_c), x_c)$$

と書き表すことができる（ただし，$X_A \perp X_B$，$|X_A| \leq 2$，$|X_B| \leq 2$）．この式の両辺を x_1 でコファクタリングすると，

$$F_0(X_1) = C(A_0(X_A, x_c), B_0(X_B, x_c), x_c) \quad (24)$$

$$F_1(X_1) = C(A_1(X_A, x_c), B_1(X_B, x_c), x_c) \quad (25)$$

を得る．式 (24)，(25) の形の分解が存在するかどうかの判定は各関数の最大入力数が異なる以外はタイプ C-0 の判定と同様に行える．その後で根本の関数 C が同一かどうかをタイプ A-1 と同様の手法で判定すればよい．

タイプ C-1 の関数の最大入力数は 6 であり，ブリッジする変数の選び方は 6 となる．そらにその中で X_c とのブリッジ変数の選び方が 5 通りあるのでトータルの選び方は 30 通りとなる．

2.12 マッチング全体のアルゴリズム

上記の個々のタイプに対するマッチングの判定アルゴリズムは互いに独立に適用可能なので，任意の順に適用すればよい．ただし，なかには同一の関数が複数のタイプにマッチ可能な場合があるので，その場合には判定の容易なタイプから適用するほうが効率的である．これらのアルゴリズム中で主に計算時間を左右するのはいくつの変数でコファクタリングするかどうかである．つまり Type-0 よりは Typ-1 が，Type-1 よりは Type-2 のほうが判定のコストは高い．そこで，最初に A-0，B-0，C-0 に対する判定を行い，その後で A-1，B-1，C-1 の判定を行なうようにしている．

3. 実験結果

以上のアルゴリズムを C++を用いて実装し，ベンチマーク回路に適用して実験を行った．使用計算機は Pentium-III（1GHz），OS は FreeBSD-4.6 である．

3.1 ブーリアンマッチングアルゴリズムの計算時間の評価

残念ながら文献 6) はブーリアンマッチング単体での処理時間を公表しておらず，また，実験で用いたプログラムも入手できないため，本稿で提案したブーリアンマッチングアルゴリズムと処理時間の比較を直接行うことはできない．ただし，深さ最小化テクノロジマッピングの処理時間が論文中に記されているので比較を行った（**表 2**）．

表中の 2 列目，3 列目の値は文献 6) に記されたものを用いている．そこでは以下のような実験を行っている．

(1) MCNC のベンチマークデータ（二段論理回路および多段論理回路）の一部の回路に対して sis の rugged スクリプト[13] を適用し，回路を簡単化する．
(2) 回路全体の最大深さが最小になるように回路中の各ノードを 2 入力ゲートに分解する．文献 3) の DMIG アルゴリズムと同様のアルゴリズムを用いる．
(3) 各ゲートを根とする入力数 5 以下の部分回路（**クラスタ**と呼ぶことにする）を全部列挙する．
(4) このクラスタのみを用いて深さ最小マッピングを行う（FlowMap アルゴリズム[14] と同様の処理．表中の最大深さの括弧内の値）．
(5) 各ゲートの 2 つのファンインにおけるクラスタ（その入力数が 5 入力以下であることに注意）をマージして 9 入力以下のクラスタを作り，そのクラスタの関数が XC4000 シリーズの基本ブロックにマッチするか調べる．
(6) このブーリアンマッチングを（各ゲートに対して☆）50 個ずつ適用し，深さ最小マッピングを行う．

表 2 文献 6) との比較
Table 2 Comparison with Ref. 6).

	文献 6)(BM-MAP(50))		提案手法 (B[5])	
回路名	最大深さ	計算時間 (s)	最大深さ	計算時間 (s)
5xp1	2(3)	5.1	4(6)	0.1
9sym	4(5)	20.0	3(5)	0.1
9symml	4(5)	23.6	4(5)	0.1
C499	4(4)	34.2	4(4)	0.7
C880	7(8)	89.7	7(8)	0.7
alu4	8(9)	181.6	4(6)	0.3
apex6	4(5)	34.6	4(4)	0.3
apex7	3(4)	15.4	3(4)	0.1
des	4(5)	1180.8	5(6)	3.6
duke2	4(4)	47.3	5(6)	0.3
misex1	2(2)	2.1	3(3)	0.0
rd84	3(4)	14.7	4(5)	0.1
rot	6(7)	44.4	6(7)	0.5
vg2	2(3)	20.1	3(4)	0.0
z4ml	2(2)	0.6	3(4)	0.0
計	59(70)	1714.2	62(77)	6.9

表中の 2 列目の最大深さの値はこのブーリアンマッチングを適用したときの値と 5 入力以下のクラスタのみを用いたときの値（括弧内の値）を示している．3 列目の値はブーリアンマッチングを適用したときの全体の計算時間（単位秒，計算機は SUN Ultra2）を示している．

表中の 4 列目および 5 列目は提案手法に対する実験結果であり，以下のような処理を行っている．

(1) 文献 6) と同一の回路に対して sis の rugged スクリプト[13] を適用し，回路を簡単化する．
(2) 回路全体の最大深さが最小になるように回路中の各ノードを 2 入力ゲートに分解する．
(3) 各ゲートを根とする入力数 5 以下のクラスタを全部列挙する．
(4) このクラスタのみを用いて深さ最小マッピングを行う（実装は異なるが FlowMap アルゴリズム[14] と同様．表中の最大深さの括弧内の値）．
(5) 各ゲートの 2 つのファンインにおけるクラスタ（その入力数が 5 入力以下であることに注意）をマージして 9 入力以下のクラスタを作り，そのクラスタの関数が XC4000 シリーズの基本ブロックにマッチするか調べる．ここではこれを B[5] と名付ける（後述）．

☆ 原論文中には各ゲートに対してという記述はないが，文意からそう推測した．

(6) このブーリアンマッチングをすべてのクラスタに対して行い，最大深さを最小とするマッピングを行う．ただし，実際には深さの計算だけを行い，回路は生成しない．

ブーリアンマッチングに関しては文献 6) が試すクラスタの数を 50 と制限し，限られたパターンのみのマッチングを試しているのに対して，提案手法では価値のあるすべてのクラスタに対してすべてのパターンに対してマッチングを試みている．つまり，得られるマッチの集合は文献 6) のものが提案手法のつねに部分集合となっているはずである．また，文献 14) で証明されているように，FlowMap アルゴリズム（およびそこから派生した BM-Map や本提案手法）は与えられたサブジェクトグラフ（2 入力に分解された回路）に対しては厳密に深さ最小解を与えるものである．つまり，定性的には本提案手法の結果（深さ）はつねに文献 6) と同一か優れているものになるはずである．ところが，文献 6) の値と本実験の値がいくつかの回路で異なっている．さらにブーリアンマッチングを用いない純粋の FlowMap アルゴリズムの結果（括弧内の値）も異なっているので，これは初期回路として与えている 2 入力ゲートに分解された回路が異なっていることが原因と考えられる．FlowMap の値が異なっている回路のうち des および z4ml 以外はもともとが 2 段論理式で与えられているものであり，これを多段の 2 入力回路へ変換する際に構造の異なる回路になった可能性が考えられる．また，z4ml は形式は多段回路の形式であるが中身は 2 段論理回路となっている．des に関しては内部に多入力ノードを多く持つので 2 入力ゲートへの分解の仕方が異なったものと思われる．これらの問題も FPGA 用のテクノロジマッピングとしては解析し解決しなければならないものであるが，本稿の主題であるブーリアンマッチングアルゴリズムとは無関係なのでこれ以上の考察は割愛する．

さて，本題の処理時間の比較に関してであるが，ブーリアンマッチングを用いたテクノロジマッピング処理ではマッチング処理に多くの時間が割かれていると考えられるので，この処理時間からブーリアンマッチングアルゴリズムの性能を見積もることは可能であると思われる．これは以下のような理由から推測される．本稿では掲載していないが 5 入力以下のクラスタのみを用いた FlowMap アルゴリズムの計算時間は最も遅い des で 32 秒であり他の回路の場合は 1 秒以下から数秒となっている．このことから BM-Map（50）の計算時間の大半がブーリアンマッチングによるものと考えられる．使用計算機は同一でなく，さらに SUN Ultra 2 もいくつかの周波数のプロセッサを搭載したモデルがあるため，厳密な比較は行えないが，SUN Ultra 2 と 1 GHz の Pentium III との速度差はおよそ 3 倍から 4 倍程度と考えられる☆．以上のことを考慮すると，計算機の速度差が不明であることを考慮にいれても提案するアルゴリズムがはるかに高速であることが分かる．

また，今回の実験はブーリアンマッチングの計算時間の比較のために行っているので最大深さの計算のみで実際のマッピング結果の生成を行っていないため，CLB 数の比較は行っていない．

3.2 深さ最小化マッピングの結果

次にブーリアンマッチングを試す対象を広げて計算時間と回路の質（最大深さ）のトレードオフを評価するために以下のような実験を行った．

(1) MCNC'89 のベンチマーク回路に対して sis の rugged.script [13] を適用し，回路を簡単化する．
(2) 回路を 2 入力ゲートに分解する．これをサブジェクトグラフとする．
(3) 各ゲートにおける l 入力以下のクラスタをすべて列挙する（$5 \leq l \leq 7$）．
(4) 各ゲートの 2 つのファンインのクラスタをマージして 9 入力以下のクラスタを作り，XC4000 シリーズの基本ブロックにマッチするか調べる．
(5) 求められたマッチを用いて深さ最小マッピングを行う．

実験結果を**表 3** および**表 4** に示す．表 4 の計は 2 つの表の総計となっている．

各々のコラムの中の 'D' および 'T' はそれぞれ深さ（基本ブロックの段数）および計算時間（単位は秒）を表している．'G' は FlowMap アルゴリズムと同様にサブジェクトグラフの構造のみを考慮して 5 入力以下のクラスタが 1 つの CLB にマッチするものとした場合の処理結果である．残りの 'B[5]'，'B[6]'，'B[7]' はそれぞれ $l = 5, 6, 7$ の場合の処理結果である．つまり 'G' と 'B[5]' の一部の結果は表 2 と重複している．

XC4000 シリーズの基本ブロックに対するマッピングにブーリアンマッチングを適用した結果は文献 6) にも示されており，この結果でも同様に大幅な段数削減が達成できることを確認できる．文献 6) には C6288

☆ http://www.specbench.org/cpu95/のベンチマーク結果によれば，SUN Ultra2 と 500 MHz の PentimumIII との速度差は約 2 倍弱となっている．500 MHz の PentiumIII と 1 GHz の PentimumIII の速度差は理想的には 2 倍だが，メモリアクセス速度は CPU の周波数に比例しているわけではないのでこちらも 2 倍弱だと考えられる．合わせて 3 倍から 4 倍と判断した．

表 3 深さ最小化マッピング結果 (1)
Table 3 Results of depth-minimum mapping (1).

	G		B[5]		B[6]		B[7]	
回路名	D	T	D	T	D	T	D	T
5xp1	6	0.1	4	0.1	4	0.4	2	0.2
9sym	5	0.1	3	0.1	3	0.3	3	1.0
9symml	5	0.1	4	0.1	4	0.4	4	1.2
C1355	4	0.6	4	0.7	4	4.6	3	50.4
C1908	8	0.4	7	0.5	6	2.0	6	9.2
C2670	9	0.5	7	0.7	6	2.9	6	13.0
C3540	12	1.4	9	2.3	8	9.4	8	58.4
C432	14	0.1	10	0.5	10	0.8	9	2.3
C499	4	0.6	4	0.7	4	4.5	3	48.8
C5315	8	1.3	7	2.3	6	15.1	5	98.1
C6288	22	5.7	21	19.8	17	57.2	15	496.7
C7552	9	3.8	7	7.4	7	57.8	6	283.0
C880	8	0.2	7	0.7	6	1.5	6	3.4
alu4	6	0.1	4	0.3	4	0.6	4	2.2
apex6	4	0.2	4	0.3	3	0.6	3	1.7
apex7	4	0.1	3	0.1	3	0.2	3	0.9
b9	3	0.0	2	0.0	2	0.1	2	0.1
clip	5	0.1	4	0.1	4	0.3	4	1.0
con1	2	0.0	1	0.0	1	0.0	1	0.0

表 4 深さ最小化マッピング結果 (2)
Table 4 Results of depth-minimum mapping (2).

	G		B[5]		B[6]		B[7]	
回路名	D	T	D	T	D	T	D	T
des	6	2.0	5	3.6	4	30.9	4	295.2
duke2	6	0.2	5	0.3	4	1.0	4	2.0
e64	16	0.0	13	0.0	11	0.0	9	0.0
f51m	4	0.0	3	0.1	3	0.5	3	4.1
misex1	3	0.0	2	0.1	2	0.1	2	0.8
misex2	4	0.0	3	0.0	3	0.1	3	0.3
misex3	7	0.3	5	0.5	5	1.3	5	3.9
misex3c	10	0.2	8	0.3	7	1.1	7	4.4
rd73	4	0.0	3	0.1	3	0.3	3	0.8
rd84	5	0.1	4	0.1	4	0.4	4	1.2
rot	7	0.2	6	0.5	5	1.0	5	3.3
sao2	6	0.1	5	0.1	5	0.3	5	1.1
seq	6	0.7	5	1.1	4	3.4	4	8.0
vg2	4	0.0	3	0.0	3	0.1	3	0.1
z4ml	4	0.0	3	0.0	3	0.1	2	0.6
計	230	19.2	185	43.5	168	199.3	156	1397.4

や C7552 などの比較的規模の大きな例に対する結果は載っていないが，これは表 2 の des の計算時間から考えると計算時間がかかりすぎるからではないかと推測される．本手法では 10 段以上の大規模な回路に対しての削減率が著しく，実用上の効果は十分あると期待できる．また表 2 の文献 6) の結果と異なり ‘B[5]’ の計算時間は ‘G’（FlowMap 相当）と比べてそれほど長いわけではないので，現在の標準的な計算機環境では十分に実用に耐えうるものと思われる．

ただし，‘B[5]’ の結果と ‘B[7]’ の結果を見比べると計算時間の増加が著しい．回路によってはその分より良い回路が得られている場合もあるが，場合によっては計算時間が増えるだけで最大深さは同じものも多く見られる．処理の高速化のためにはブーリアンマッチングを試すクラスタを制限する効果的なヒューリスティックが必要である．

4. おわりに

本稿では 2 つの 4 入力 LUT と 1 つの 3 入力 LUT から構成された回路を基本ブロックとする FPGA のためのブーリアンマッチングアルゴリズムを提案した．そのためにまず，LUT 間のブリッジ構造に着目して

基本ブロックのマッチングパターンを 9 種類に分類を行った．この分類は Cong と Hwang の分類よりも単純で分かりやすいものであり，また，マッチングアルゴリズムと 1 対 1 に対応したものである．提案したマッチングアルゴリズムは単純な直交分解アルゴリズムを基本としたもので，対象となるマッチングパターンが直交分解でない場合には，ブリッジしている変数でコファクタリングしてから直交分解アルゴリズムを適用し，結果をマージするアプローチをとっている．この場合でもブリッジを仮定する変数はすべて列挙する必要があるが，最大の入力数が限られているため，たかだか 30 通りの組合せしか存在しない．単純な直交分解は，対象となる論理関数を二分決定グラフを用いてコンパクトに表現できればきわめて高速に求めることができるため全体としても効率良くマッチングを求めることができている．

さらに，ここで提案した分類方法およびマッチングアルゴリズムを XC4000 以外の構成の基本ブロックに応用することも比較的容易に行えると思われる．たとえば図 1 の LUT1 の入力数を 4 とした構成を考えてみた場合も X_c とのブリッジの種類をさらに 2 種類追加するだけで同様のアルゴリズムで対応することが可能である．

このアルゴリズムを用いて XC4000 などの FPGA 用のテクノロジマッピングを行うには，マッピング対象の回路の適当な部分回路を切り出してきて，それが 1 つの基本ブロックとして実現可能かを調べる必要がある．従来の論理を考慮しないマッピングの場合には考慮すべき部分回路の入力数は 5 入力以下であり，可能なすべての部分回路を列挙することも可能であったが，論理を考慮した場合には最大で 9 入力の部分回路もマッチする可能性があるため，考慮すべき部分回路は莫大な数となる．これは最悪の場合，部分回路の数がその入力数の指数に比例して増加するためである．そのため，このようなブーリアンマッチングを用いたテクノロジマッピングアルゴリズムを構築するためには，マッチングを試す対象の部分回路をある程度絞る工夫が必要と思われる．

謝辞 本研究の一部は日本学術振興会科学研究費基盤研究（B）(2)「論理関数処理を用いた FPGA テクノロジマッパの開発」（課題番号 15300019）による．

参 考 文 献

1) Francis, R.J., Rose, J. and Vranesic, Z.: Chortle-crf: Fast technology mapping for lookup table-based FPGAs, *28th ACM/IEEE Design Automation Conference*, pp.613–619 (1991).
2) Murgai, R., Shenoy, N., Brayton, R.K. and Sangiovanni-Vincentelli, A.: Improved logic synthesis algorithms for table look up architectures, *International Conference on Computer-Aided Design*, pp.564–567 (1991).
3) Chen, K.C., Cong, J., Ding, Y., Kahng, A.B. and Trajmar, P.: DAG-map: Graph-based FPGA technology mapping for delay optimization, *IEEE Design and Test of Computer*, pp.7–20 (1992).
4) Cong, J. and Hwang, Y.-Y.: Partially-Dependent Functional Decomposition with Applications in FPGA Synthesis and Mapping, *ACM 5th International Symposium on FPGA*, pp.35–42 (1997).
5) Cong, J. and Hwang, Y.-Y.: Boolean Matching for Complex PLBs in LUT-based FPGAs with Application to Architecture Evaluation, *ACM 6th International Symposium on FPGA*, pp.27–34 (1998).
6) Cong, J. and Hwang, Y.-Y.: Boolean Matching for LUT-Based Logic Blocks with Applications to Architecture Evaluation and Technology Mapping, *IEEE Trans. Computer-Aided Design*, Vol.20, No.9, pp.1077–1090 (2001).
7) 梶原裕嗣, 堀山貴史, 中西正樹, 木村晋二, 渡邊勝正：論理関数の畳み込みを考慮した Look Up Table の設計と実現，DA シンポジウム 2002, pp.223–228 (2002).
8) Bryant, R.: Graph-based algorithms for boolean function manipulation, *IEEE Trans. Comput.*, Vol.C-35(8), pp.677–691 (1986).
9) Bertacco, V. and Damiani, M.: The disjunctive decomposition of logic functions, *International Conference on Computer-Aided Design* (*ICCAD'97*), pp.78–82 (1997).
10) Matsunaga, Y.: An Exact and Efficient Algorithms for Disjunctive Decomposition, *Workshop on Synthesis And System Integration of Mixed Technologies* (*SASIMI'98*), pp.44–50 (1998).
11) Matsunaga, Y.: An Efficient Algorithm Finding Simple Disjoint Decompositions Using Bdds, *IEICE trans. fund.*, Vol.E85-A, No.12, pp. 2715–2724 (2002).
12) Brace, K., Rudell, R. and Bryant, R.: Efficient implementation of a BDD package, *27th Design Automation Conference*, pp.40–45 (1990).
13) Savoj, H., Touati, H. and Brayton, R.K.: Improved Scripts in MIS-II for Logic Minimization of Combinational Circuits, *International Workshop on Logic Synthesis* (1991).

14) Cong, J. and Ding, Y.: FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA design, *IEEE Trans. Computer-Aided Design*, Vol.13, No.1, pp.1–12 (1994).

(平成 15 年 10 月 22 日受付)
(平成 16 年 3 月 5 日採録)

松永 裕介（正会員）

1962 年生．1987 年早稲田大学大学院理工学研究科電気工学専攻修士課程修了．同年（株）富士通研究所入社．LSI の CAD の研究に従事．1991 年より 1 年間米国カリフォルニア大学バークレイ校客員研究員．2001 年より九州大学大学院システム情報科学研究院助教授．現在に至る．論理合成，論理検証，動作合成，テストパターン自動生成の研究に従事．博士（工学）．1994 年情報処理学会山下記念研究賞受賞．1999 年情報処理学会システム LSI 設計技術研究会優秀論文賞受賞．電子情報通信学会，IEEE，ACM 会員．