関数分解を用いたFPGAのブーリアンマッチングの高速化手法について

松永, 裕介
九州大学大学院システム情報科学研究院情報工学部門

関数分解を用いたFPGAのブーリアンマッチングの高速化手法について

松永 裕介
九州大学大学院システム情報科学研究院情報工学部門

概 要

LUT 型の FPGA は一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．本稿では関数分解に基づくブーリアンマッチングを利用して効率よく LUT 型 FPGA 用の深さ最小の回路を求めるテクノロジマッピングアルゴリズムについて述べる．

1 はじめに

LUT(look up table) 型の FPGA(field programmable array) は一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，対象回路を一旦，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なえるため，非常に効率がよい．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 [3, 4, 5].

ところが，実際の FPGA の基本ブロックの中には Xilinx 社の XC4000 シリーズのように 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．図 1 に示すように Xilinx 社の XC4000 シリーズの基本ブロック (PLB:Programmable Logic Block) は，2 つの 4-LUT と 1 つの 3-LUT から構成される．この 1 つの PLB では，

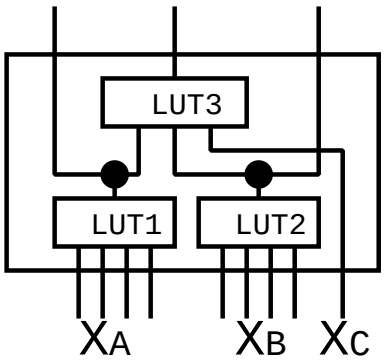

図 1: XC4000 の基本ブロック

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なえるので論理関数を考慮する必要がない．しかし，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 はこの判定を行なうブーリアンマッチングアルゴリズムを提案した [6, 7, 8]. しかし，彼らのアルゴリズムは全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言いがたい．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．ここで提案している高速化手法の主な技術は，

- 2 分決定グラフ (Binary Decision Diagram:BDD[9]) を用いた論理関数の直交分解 (disjoint functional decomposition) アルゴリズムを応用したブーリアンマッチングアルゴリズム
- 2 分決定グラフを用いた NPN 同値類判定アルゴリズムによるブーリアンマッチング結果のキャッシュ手法
- 部分回路の構造に基づくマッチング対象候補のフィルタリング手法

の 3 つである．また，このブーリアンマッチングを用いて深さ優先テクノロジマッピングを行な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 XC4000 シリーズ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．例えば文献 [14] 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もつ FPGA デバイスに対するマッパーに用いることもできると思われる．

2 LUT 型 FPGA 用ブーリアンマッチング手法

2.1 直交分解を用いたブーリアンマッチングアルゴリズム

ここでは LUT 型 FPGA 用ブーリアンマッチングアルゴリズムについて簡単に述べる．詳細は文献 [12] を参照されたい．

まず，マッチングのための場合わけを行う．図 1 の 9 つの入力 (X_A(4 本), X_B(4 本), X_C(1 本) の 3 つのグループからなる) 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 (同一の信号線につなぐ) 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わけを行なう．場合わけは 2 種類の独立した分類法からなる．

ひとつめは，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分類するものである．

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．

ふたつめは，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よる分類である．

Type-0: $|X_A \cap X_B| = 0$.

Type-1: $|X_A \cap X_B| = 1$.

Type-2: $|X_A \cap X_B| = 2$.

Type-3: $|X_A \cap X_B| = 3$.

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わせが可能である (例えば A-0, C-3 など: 各々 Type-A と Type-0，Type-C と Type-3 の組合わ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，例えば 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

A-0 の場合，各々LUT の入力は互いに共通部分を持たないので，この LUT の接続構造と同じ形の単純な直交分解が存在するはずである．そこで，文献 [11] のアルゴリズムを用いて論理関数の直交分解を行い，その形が A-0 にマッチするかを調べれば良い．他のケースの場合，ブリッジしている入力があるために簡単ではないが，ブリッジしている入力を 0 および 1 に固定した関数を考えてみると，残りの入力は互いに共通部分を持たないので，関数の直交分解を行ってマッチングを調べることができる．問題はどの入力がブリッジ入力かわからないということであるが，これは単純なしらみつぶしで調べる．XC4000 シリーズのパタンの場合，最大でも 3 つの変数しかブリッジしないのでしらみつぶしの組み合わせも高々30 通りである ([12])．また，直交分解を求めるアルゴリズムの計算複雑度は論理関数を表す二分決定グラフの節点数の二乗であるが，今回扱っている論理関数は最大で 9 入力であり節点数は多くて数百なので問題はない．

2.2 NPN 同値類判定によるキャッシュ手法

テクノロジマッピングにこのブーリアンマッチングを用いた場合，異なるクラスタであってもその論理関数が同一になることがあり，結果として同一の関数に対して何回もブーリアンマッチングを試みることになる．このようなものは論理関数とブーリアンマッチングの結果を記憶しておけば 2 回め以降のブーリアンマッチングの手間を省くことができる．さらに，FPGA の場合，入力および出力の極性反転と入力の順序の入れ替えは自由に行なえるため，ある論理関数 F に対するブーリアンマッチングが成功した場合 (この論理関数 F を 1 つの基本ブロックで実現可能ということ)，この論理関数の入力変数および出力変数の極性を反転させたり，入力の順序を入れ替えた論理関数 F' もまたブーリアンマッチング可能なことが分かっている．このように，入力変数または出力変数の極性を反転させたり，入力の順序を入れ替えることで一致する論理関数 (上の例の F と $'F$) の集合を NPN 同値類と呼ぶ．すでにブーリアンマッチング結果の記憶されている論理関数と NPN 同値類であれば，新たにブーリアンマッチングを行なう必要はないので処理の高速化を行なうことができる．与えられた 2 つの関数が同一の NPN 同値類に属しているかどうかを判定には筆者の開発した効率のよいアルゴリズム [10] を用いている．

ただし，前述の関数の直交分解アルゴリズムが最悪でも BDD のサイズの 2 乗の手間で実行できるのに対して，この NPN 同値類の判定アルゴリズムは最悪の場合，入力数の階乗に比例した手間を必要とする．そのため，一般的には Type A-0 のマチングを判定するよりも NPN 同値類の判定のほうが時間を要する．これでは高速化にならないため，9 入力の関数に対しては (9 入力関数は Type A-0 にしかマッチしない)，NPN 同値類の判定を用いたキャッシュを行なわない．

2.3 部分回路の構造に基づく対象候補のフィルタリング

ブーリアンマッチングを用いたテクノロジマッピングでもっとも計算時間を要するのはクラスタが 1 つの基本ブロックで実現可能か判定する部分である．回路の構造にもよるが，多くの場合，1 つのゲートを根とするクラスタは数十から数百となり，全体として回路規模×数百個のクラスタが存在することになる．ただし，それらは全く無関係なわけではなく，もともとは同じ回路から切り出されたクラスタであり，なかには共通の部分をもつクラスタも多数存在する．そこで，単独のブーリアンマッチングの高速化とは別に，複数のブーリアンマッチング処理を全体として高速化するための手法を提案する．これは以下のような直感的な考察に基づくものである．

“もしも，あるクラスタが実現可能でなければそのクラスタを部分グラフとして含むようなクラスタも実現不可能である．”

つまり，クラスタ相互の包含関係を記録しておいて，実現不可能なクラスタを部分クラスタに持つクラスタはブーリアンマッチングを試す候補から除外すれば無駄なブーリアンマッチングを起動する回数を減らすことができる．ただし，実際には実現不可能なクラスタを包含するクラスタでも実現可能な場合があるため，この判定には少し複雑な処理を必要とする [13].

3 遅延最小化テクノロジマッピングアルゴリズム

ここでは，遅延最小解を求めるマッピングアルゴリズムに必要な用語および概念の定義を行う．

マッピング対象の回路の各ゲートを 2 入力ゲートに分解したものをサブジェクトグラフ (subject graph) と呼ぶ．分解の仕方は一通りではないし，分解の仕方によってマッピング結果が変わりうるが，マッピング前にどのような分解が適切なのかは予測不可能なので，ここでは任意の分解を用いるものとする．この 2 入力ゲートは NAND や NOR のみでなく任意の 2 入力論理関数を実現することができるものとする．この 2 入力ゲートの実現している論理関数を局所関数 (local function) と呼ぶことにする[1]．以降，マッピング対象の回路とはサブジェクトグラフのことを指すものとする．また，2 入力ゲートの代わりにサブジェクトグラフ上の節点という表現を用いる．節点 v のファンインとは節点 v の入力となっている節点の集合であり，$FI(v)$ と表す．節点 v のファンアウトとは節点 v の出力となっている節点の集合であり，$FO(v)$ と表す．節点 v の推移的ファンインとは節点 v の入力からたどることのできる節点の集合であり，$TFI(v)$ と表す．$TFI(v)$ は再帰的に次式のようにも表される．

$$TFI(v) = \cup_{u \in FI(v)} TFI(u)$$

節点 v の推移的ファンアウトとは節点 v の出力からたどることのできる節点の集合であり，$TFO(v)$ と表す．

[1] もちろん，ブーリアンマッチングを行うのでなければこの局所関数は用いられない．

$TFO(v)$ は再帰的に次式のようにも表される．

$$TFO(v) = \cup_{u \in FO(v)} TFO(u)$$

グラフ $G(V,E)$(V は節点の集合，E は枝の集合) に含まれる任意の節点対 $(v_i \in V, v_j \in V)$ に対して，その節点を結ぶ経路が存在する時，グラフ G を連結グラフと呼ぶ．グラフ G から節点の部分集合 $S \subset V$ とそれに接続する枝の部分集合を取り除いた結果，グラフ G が連結でなくなる時，そのような節点の部分集合 S をセパレータと呼ぶ．

サブジェクトグラフ $G(V,E)$ 上の節点 v に対して，節点 v とその推移的ファンインのみからなる部分グラフ G_v を考える．この部分グラフ G_v 上のセパレータ S が与えられたとき，G_v から S および S に接続している枝を取り除くと G_v は 2 つ以上の非連結な部分グラフに分解される．これらの部分グラフのうち v を含むものを v を根とするクラスタ (cluster) と呼ぶことにする (図 2)．このようにクラスタは根の節点 v とセパレータ S によって定義されるので，以降，$c(v,S)$ でクラスタを表記することとする．また，根の節点 v が自明な場合にはセパレータ S でクラスタを表すこととする．クラスタ $c(v,S)$ の度数 (degree) を $|S|$ と定義する．すなわち，クラスタの入力側に隣接しているセパレータの節点数である．図 2 のようにクラスタの複数の枝がセパレータの一つの節点に接続していることもあるのでクラスタの度数とクラスタの入力側の枝数とは必ずしも一致しないことに注意．ただし，本稿では誤解のない場合には，クラスタの入力数という表記でクラスタの度数を表すものとする．クラスタ $c(v,S)$ に対して，v の出力の論理関数を S の各節点を入力として表したものをクラスタ関数 (cluster function) と呼ぶ[2]．

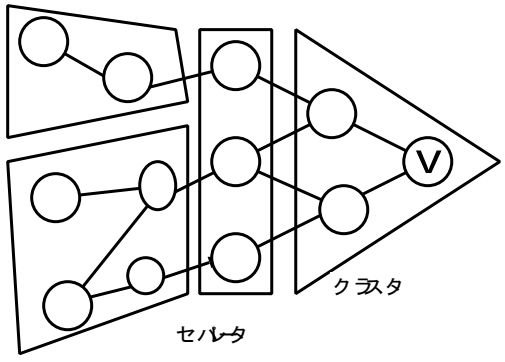

図 2: クラスタ

以上の定義から k 入力 LUT が度数 k 以下のクラスタにマッチすることは明らかである．そこで，FPGA のマッピング問題とはサブジェクトグラフ $G(V,E)$ を適当なクラスタの集合で被覆する問題と見なすことができる．ただし，ここでいう被覆とは一般的な被覆と少し意味が異なる．まず，サブジェクトグラフ上の全ての節点がいずれかのクラスタに含まれていなければならない，という通常の被覆条件は当然，満たされなければならないが，さらに，あるクラスタ $c(v,S)$ が解に含まれるならば，$u_i \in S_1$ であるような節点 u_i を根とするクラスタ $c(u_i, S_{u_i})$ も解に含まれなければならないという条件も満たされなければならない．このような条件の被覆問題は DAG covering 問題と呼ばれ，一般的には厳密に解くことが難しい問題と言われている [1] が，目的関数が遅延時間の場合には動的計画法を用いて入力側から局所最適解を求めれば最終的に全体の最適解が得られることが知られている [2].

論理を考慮しない場合は，遅延最小解を求めるアルゴリズムとしては Cong と Ding の FlowMap アルゴリズム [2] を用いれば良いが，ブーリアンマッチングを行う場合にはクラスタの度数だけではなく，クラスタ関数も考慮しなければならない．そこで，各節点 v に対して，その v を根とするすべてのクラスタを列挙し，そのクラスタ関数が XC4000 シリーズの PLB で実現可能かどうかブーリアンマッチングで調べる必要があるが，さいわいなことに，遅延最小解を求めるためにはすべてのクラスタに対してマッチングを試す必要はなく，最小解を得ることができるマッチを一つ求めてしまえば残りのクラスタに対してはマッチングを行わなくても良い．

マッチングの候補を減らすために次のように節点とクラスタのレベルという概念を定義する．サブジェクトグラフの節点を $l(v)$ で表す．節点 v が外部入力節点の場合 $l(v) = 0$ と定める．v を根とするクラスタ S のレベルを $cl(S)$ で表し，次式のように定義する．

$$cl(S) = \max_{u \in S} l(u) + 1$$

つまり，そのクラスタの入力となっている節点のレベルの最大値 $+1$ がそのクラスタのレベルである．内部節点 v のレベル $l(v)$ を次式のように定義する．

$$l(v) = \min_{\{S | S \text{ は } v \text{ を根とするマッチ可能なクラスタ}\}} cl(S)$$

以上の定義から外部出力節点 v に出力している節点のレベルの最大値がこの回路を FPGA にマッピングしたときの深さ (段数) の最小値となることは明らかである．つまり，サブジェクトグラフの各節点に対して，外部入力側から最小値を与えるマッチ可能なクラスタを選択して

[2]局所関数と同様にブーリアンマッチングでのみ用いられる．

ゆけば，回路全体の深さが最小の解を求めることができる．節点のレベルの他にレベルの下限 $ll(v)$ を以下のように定義する．

$$ll(v) = \min_{\{S|S \text{ は } v \text{ を根とするクラスタ}\}} cl(S)$$

ここで，節点 v の 2 つのファンインを u_1 と u_2 とすると，次式が成り立つ [2].

$$\max(ll(u_1), ll(u_2)) \leq l(v) \quad (1)$$

$$\max(l(u_1), l(u_2)) + 1 \geq l(v) \quad (2)$$

(1) 式：$l(v)$ を与えるクラスタ $c(v, S)$ から v を取り除き，$u_1(u_2)$ の推移的ファンインのみを残したものは $u_1(u_2)$ を根とするクラスタになっており (ただしマッチ可能とは限らない)，かつそのレベルは $l(v)$ と等しいか小さいはずである．その値よりも $ll(u_1)(ll(u_2))$ は等しいか小さい．

(2) 式：v に対して $S = \{u_1, u_2\}$ という節点集合は必ずセパレータになっており，このセパレータから作り出されたクラスタのレベルは $\max(l(u_1), l(u_2)) + 1$ であるから $l(v)$ がこの値よりも大きいはずはない．

(1),(2) 式からわかることは，節点 v のレベルは自分の入力となっている節点のレベルの下限と等しいか大きい．また，自分の入力となっている節点のレベルよりも大きくても 1 しか大きくないということである．そこで，まず，入力の節点のレベルの下限と等しいレベルを持つクラスタからマッチングを試して行き，マッチすることがわかれば残りのクラスタのマッチを試す必要はない．

ただし，マッチが求まったからと言ってクラスタの列挙まで省略することはできない．これは節点 v のクラスタ集合を作り出すのに，節点 u_1 と節点 u_2 のクラスタ集合をマージしているからである．つまり，u_1 の最初のクラスタが XC4000 の PLB にマッチすることがわかったとしてもそのあとで v のクラスタに対するマッチングは試す必要があり，さらにそのときには u_1 のマッチングを試さなかった残りのクラスタを部分として含むマッチが存在する可能性があるからである．

以上のことから，遅延最小化テクノロジマッピングアルゴリズムは以下のようになる．

(1): 外部入力に対して空のクラスタをマッチさせる．レベルは 0 としておく．

(2): 外部入力側から節点 v を一つ取り出し以下の処理を行う．

(2-a): 入力節点 u_1 と u_2 のクラスタをマージして，入力数が 9 入力以下のもののみを残す．

(2-b): そのクラスタ集合をレベルにしたがって分類する．

(2-c): レベルの低いクラスタからマッチングを試す．ひとつでもマッチが見つかったら処理を終わる．そのクラスタのレベルを $l(v)$ とする．とする．

(2-d): マッチが見つからなかったら

$$l(v) = \max(l(u_1), l(u_2)) + 1$$

3: 外部出力側から最小レベルを与えるクラスタを選択してゆく．

Cong と Hwang のアルゴリズムではブーリアンマッチングが高速でない (と推測される) のですべてのパタンのマッチングは試していない [8] が，提案手法のブーリアンマッチングは高速なのでマッチングを試すパタンを制限する必要はない．しかし，回路によってはクラスタ数が膨大になってしまうので，これを間引くために 2 つの入力のクラスタ集合をマージするときに各々のクラスタの度数に制限を設けている (たとえば 5 入力以下のクラスタ同士だけをマージする)．

4 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．

4.1 ブーリアンマッチングアルゴリズムの評価

残念ながら文献 [8] はブーリアンマッチング単体での処理時間を公表していないため，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なうことはできないが，深さ最小化テクノロジマッピングの処理時間で比較してみると以下のようになった (表 1)．

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もっとも計算時間を要するのはマッチング処理であり，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り，使用計算機が同一でないことを考慮しても本稿で述べたブーリアンマッチングアルゴリズムが遥かに高速であることがわかる．

表 1: 文献 [8] との比較

回路名	文献 [8] (Sun ULTRA2)	提案手法 (1GHz PentiumIII)
9symml	23.6	1.2
C499	34.2	13.0
C880	98.7	1.6
apex6	34.6	1.4
apex7	15.4	0.3
des	1180.8	31.3
rot	44.4	1.7
z4ml	0.6	0.2

4.2 高速化手法の評価

次にブーリアンマッチングを高速化する手法の評価を行なった．ベンチマーク回路を 2 入力ゲートに分解し，入力数が 9 以下のすべてのクラスタを列挙し，そのすべてのクラスタに対してブーリアンマッチングを行なった．実験結果を表 2 に示す．

最初のコラムは全クラスタ数を表している．2 番目のコラムはそれらのクラスタのうち XC4000 シリーズの 1 つの基本ブロックで実現可能なクラスタの数 (ブーリアンマッチングでマッチした数) を表している．残りのコラムはそのブーリアンマッチングに要したすべての処理時間を表している．まず，I と記されたコラムは関数分解に基づくブーリアンマッチングアルゴリズムのみを適用したときの時間， II は I に加えて NPN 同値類の判定に基づくキャッシュ手法を用いたときの時間，III は II に加えて構造に基づくフィルタを用いたときの時間を表している．特に NPN 同値類によるキャッシュは効果が大きく約 5 倍の高速化を行なっている．また，構造フィルタも約 1 割の高速化に役立っておりともに有効な手法であると言える．

5 深さ優先マッピングの結果

実験の概要は以下の通りである．

- MCNC'89 の多段回路ベンチマークに対して sis の rugged.script を適用し，回路を簡単化する．
- 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．
- XC4000 シリーズの基本ブロックにマップさせる．今回はレベル (ブロック段数) 最小化のみを目的関数にした．

前述のように全ての回路に対して 9 入力のクラスタをすべて列挙することはできなかったので，2 つの入力のクラスタ集合をマージする際に，その度数が 5 以下，6 以下，7 以下の 3 種類の制限値でクラスタ集合を間引きした．これは 1 つの入力のクラスタの度数が 5, 6, 7 以下ということであって，マージされた結果は 9 入力以下という制限のままであることに留意されたい．実験結果を表 3 に示す．

各々のコラムのなかの 'D' および 'T' はそれぞれレベル (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', 'B[5]', 'B[6]', 'B[7]' はそれぞれ以下のように適用したマッチングの手法を表している．

G: 単純にクラスタの度数が 5 以下の場合のみマッチする．

B[5]: ブーリアンマッチングを行う．クラスタ列挙のためのマージの際に入力の節点のクラスタのうち度数が 5 以下のクラスタのみを用いる．

B[6]: ブーリアンマッチングを行う．クラスタ列挙のためのマージの際に入力の節点のクラスタのうち度数が 6 以下のクラスタのみを用いる．

B[7]: ブーリアンマッチングを行う．クラスタ列挙のためのマージの際に入力の節点のクラスタのうち度数が 7 以下のクラスタのみを用いる．

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 [8] 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．さらに文献 [8] の実験ではマージの際の制限値を 5 にして，さらに一部のブーリアンマッチングのみを調べているので提案手法を用いた結果よりも悪いものが見られた．実験に用いた初期回路の分割が同一ではないため厳密な比較はできないが，たとえば文献 [8] では C499 が 4(3)，C880 が 7(5) となっている (括弧内は本手法の結果)．また，文献 [8] には C6288 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．本手法では 10 段以上の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．ただし，B[5] の結果と B[7] の結果を見比べると明らかに探索範囲の大小で計算時間と段数のトレードオフが存在することがわかる．処理の高速化のために中途半端に列挙するクラスタ数を制限することはマッピング

表 2: 実験結果

全クラスタ数	マッチ数	CPU 時間		
		I: ブーリアンマッチングのみ	II: I + NPN 同値類	III: II + 構造フィルタ
139153	97095	402.8	78.1	70.5

表 3: 深さ優先マッピング結果

回路名	G		B[5]		B[6]		B[7]	
	D	T	D	T	D	T	D	T
9symml	6	0.4	5	1.2	4	3.0	4	3.8
C1355	4	1.0	4	13.1	4	46.2	3	232.6
C1908	10	0.7	8	5.0	7	12.9	6	31.1
C2670	13	0.7	8	6.1	7	15.5	6	36.1
C3540	13	1.9	10	21.4	9	50.5	8	112.5
C432	15	0.2	12	1.4	11	3.5	11	5.5
C499	4	0.8	4	13.0	4	45.8	3	232.1
C5315	7	2.0	7	22.6	5	67.5	5	233.3
C6288	22	8.3	21	82.8	17	378.6	13	2022.8
C7552	11	5.4	9	85.7	8	233.5	8	953.1
C880	9	0.3	7	1.6	6	3.6	5	7.6
apex6	5	0.4	4	1.4	4	3.4	4	5.9
apex7	5	0.1	4	0.3	3	0.3	3	1.1
b9	3	0.1	3	0.2	2	0.3	2	0.7
des	7	3.2	6	31.3	5	76.6	4	245.2
f51m	4	0.1	4	0.4	2	1.7	2	8.6
rot	8	0.4	7	1.7	6	4.5	5	8.7
z4ml	3	0.0	3	0.2	2	0.5	1	1.3
計	149	26.0	126	289.4	106	947.9	93	4142.0

結果の品質に大きな影響を与えることになるのでヒューリスティックを考えるときには注意が必要である．

6 おわりに

本稿では，LUT 型 FPGA 用のブーリアンマッチング手法およびその高速化手法について述べた．このアルゴリズムは既存の FPGA デバイス専用のものであるが，提案手法はそれ以外のタイプの FPGA デバイスにも応用可能である．従来はテクノロジマッピングを容易にするために単純な LUT 構造を用いたものが多く提案されていたが，本稿で提案したブーリアンマッチングの技術を用いればより複雑な構造を持つ FPGA デバイスでも比較的容易にテクノロジマッピングが可能であり，高速処理や低消費電力などの特徴を持った FPGA デバイスの開発に際してより柔軟な構造を考慮することを可能にするものと考える．

参考文献

[1] R. Rudell, “Logic synthesis for VLSI design”, Ph.D. thesis, University of California, Berkeley, 1989.

[2] J. Cong and Y. Ding, “FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA design”, *IEEE*

Transactions on Computer-Aided Design, vol. 13, no. 1, pp. 1–12, Jun. 1994.

[3] R. J. Francis, J. Rose, and Z. Vranesic, "Chortle-crf: Fast technology mapping for lookup table-based FPGAs", *in Proceedings of the 28th ACM/IEEE Design Automation Conference*, pp. 613–619, June 1991.

[4] R. Murgai, N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Improved logic synthesis algorithms for table look up architectures", *in Proceedings of the International Conference on Computer-Aided Design*, pp. 564–567, Nov. 1991.

[5] K. C. Chen, J. Cong, Y. Ding, A. B. Kahng, and P. Trajmar, "DAG-map: Graph-based FPGA technology mapping for delay optimization", *IEEE Design and Test of Computer*, pp. 7–20, Sept. 1992.

[6] J. Cong, and Y.-Y. Hwang, "Parially-Dependent Functional Decomposition with Applications in FPGA Synthesis and Mapping", *In Proceedings of the ACM 5th International Symposium on FPGA*, pp. 35 – 42, Feb. 1997.

[7] J. Cong, and Y.-Y. Hwang, "Boolean Matching for Complex PLBs in LUT-based FPGAs with Application to Architecture Evaluation", *In Proceedings of the ACM 6th International Symposium on FPGA*, pp. 27 – 34, Feb. 1998.

[8] J. Cong, and Y.-Y. Hwang, "Boolean Matching for LUT-Based Logic Blocks with Applications to Architecture Evaluation and Technology Mapping", *IEEE Transactions on Computer-Aided Design*, vol. 20, no. 9, pp. 1077–1090, Sep. 2001.

[9] R.E. Bryant, "Graph-based algorithms for boolean function manipulation", *IEEE Transactions on Computer*, C-35(8), pp. 677–691, Aug. 1986.

[10] Y. Matsunaga, "A New algorithm for Boolean Matching Utilizing Structural Information", IEICE Trans. Inf. & Syst., E78-D, No. 3, pp. 219–223, Mar. 1995.

[11] Y. Matsunaga, "An Exact and Efficient Algorithms for Disjunctive Decomposition", In Proccedings of the Workshop on Synthesis And System Integration of Mixed Technologies (SASIMI'98), pp. 44 – 50, Oct. 1998.

[12] 松永 裕介, "関数分解を用いた LUT 型 FPGA 用ブーリアンマッチングアルゴリズムについて", 信学技法 VLD2000–96, pp. 161–166, Nov. 2000.

[13] 松永 裕介, "再しゅうれん構造に着目した FPGA 用ブーリアンマッチングの高速化手法について", 信学技報 VLD2002–2, pp. 7–12, May 2002.

[14] 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, Jul. 2002.