社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2004－ＳＬＤＭ－113　(15)
2004／1／23

LUTの木構造に対するブーリアンマッチングアルゴリズムについて

松永　裕介†

† 九州大学システムLSI研究センター
〒816–8580 福岡県春日市春日公園6–1
E-mail: †matsunaga@slrc.kyushu-u.ac.jp

あらまし　本稿では，与えられた論理関数を実現するLUT(Look-up Table)の回路を生成するアルゴリズムについて述べる．提案するアルゴリズムは二分決定グラフを用いて論理関数の直交な分解を列挙するアルゴリズムを応用したもので，2個のLUTで実現できる関数に関しては必ずその構造を求めることができる．3個以上のLUTを必要とする関数に対しては必ずしも最適解を見つける保証はないヒューリスティックアルゴリズムであるが，ベンチマーク回路を用いた実験では回路の構造のみに基づくアルゴリズムよりもより少ないブロック数の回路を生成している．

キーワード　FPGA，二分決定グラフ，関数分解，テクノロジマッピング

On a Boolean matching algorithm for LUT trees

Yusuke MATSUNAGA†

† System LSI Research Center, Kyushu University
Kasuga Koen 6–1, Kasuga, Fukuoka, 816–8580 Japan
E-mail: †matsunaga@slrc.kyushu-u.ac.jp

Abstract　This paper presents an algorithm for generating a tree structure of Look-up Tables (LUTs) realizing a given logic function. The proposed algorithm, which is an application of an efficient algorithms for disjoint decomposition using Binary Decision Diagrams, can find the minimum solutions for any functions whose realization circuits require only two LUTs. For functions whose realization circuits require three or more LUTs, the algorithm does not guarantee to find the minimum solutions, however, the experimental results show that on avarage the proposed algorithm outperforms existing structural based algorithm.

Key words　FPGA, Binary Decision Diagrams, functional decomposition, technology mapping

1. はじめに

K 入力 look-up table(以後 K-LUT もしくはただ単に LUT と略す) は 2^K ビットのメモリとその選択回路から構成される．K 本のアドレス線を入力と見なし，各々のビット値をそのアドレスに対応する入力の組み合わせに対する関数の値と見なせば，任意の K 入力 (以下) の論理関数を実現することができる．そのため，K-LUT は再構成可能なハードウェアの基本素子として広く用いられている．

実現しようとしている論理関数の入力数が K よりも多い場合には複数の K-LUT を組み合わせる必要があるが，一般に，任意の論理関数に対して最適な個数の K-LUT を用いて実現するアルゴリズムは知られていない．そのため，K-LUT 用の論理合成では以下のようなヒューリスティックが用いられている．

- 回路構造に基づく手法
- 関数分解 (functional decomposition) に基づく手法

回路構造に基づく手法では，まず対象回路を K 入力以下の論理ゲートからなる回路に分解し，その後，複数の論理ゲートを1つの K-LUT にまとめる処理を行う．これらの処理は回路の構造の情報のみを調べればよいので高速に行うことができるが，結果の品質は回路構造に大きく依存する．ところが，一般の論理合成はもともとスタンダードセルによる実現を念頭に考えられたものであるために，同じ入力数の関数でもリテラル数の少ないものをより最適であるとみなす最適化を行っており，K-LUT による実現に向いた回路構造になっているとは言えない．

一方，関数分解に基づく手法では，回路全体もしくは回路の一部を表す論理関数に対して関数分解— 1つの関数をより小さな複数の関数で表すこと— を求めることで K-LUT による実現に向いた回路構造の生成を目指している．つまり，元の関数が K 入力以下の関数に分解できればよいわけである．現在のところ，関数分解の手法として用いられているのは，

（1） 関数分解に対応する入力変数の分割を求める.
（2） その分割に沿った関数分解を求める.
というものである. このうち, 2の関数分解を求める手法としては分解表 (decomposition chart) を用いるもの [1], キューブ表現を用いるもの [2], 二分決定グラフ (BDD) [3] をもちいるもの [4] などが提案されており, 比較的容易に行うことができる. ところが, 1の入力変数の分割を求める処理を効率よく行う手法は知られていない. 入力数の指数乗に比例した個数存在する変数の分割をすべて列挙して分解を試す以外に最適な分解を求める手段は知られていない.

ただし, 分解された関数の入力が変数を共有しないタイプの関数分解 (直交な関数分解) に対しては, 二分決定グラフを用いて効率よくすべての関数分解を列挙する手法が知られている [5]~[7]. また, 著者は分解された関数の入力が変数を共有する場合でも, その共有されている変数を取り除くことで問題を直交な関数分解に落すアルゴリズムを提案している [8]. このアルゴリズムでも共有される可能性のある変数集合をすべて列挙しなければならないが, 完全な変数の分割を列挙するよりも場合の数ははるかに少ない.

著者はこれらの関数分解のアルゴリズムを応用して Xilinx 社の XC4000 シリーズの FPGA 向けのテクノロジマッピングアルゴリズムを開発している [9], [10]. しかし, このアルゴリズムは特定の FPGA アーキテクチャに依存しており, また, 与えられた論理関数が XC4000 シリーズの1つの基本ブロックで実現可能かどうかを判断することを基本処理としているために, 実際のマッピングのためには数多くの部分論理関数を切り出してきてそれが基本ブロックで実現可能かを調べる必要があり, 実用性が高いとは言えない.

本稿では, 同様に関数分解のアルゴリズムを応用ならが汎用の K-LUT 向けのマッピングアルゴリズムを提案する. このアルゴリズムは与えられた論理関数に対して, それを実現する K-LUT のネットワークを返すものである. 以降, 2. 章で関数分解とそのアルゴリズムについて説明し, 3. 章で提案するマッピングアルゴリズムを述べる. 4. 章で実験結果を示し, 5. 章でまとめと今後の課題について述べる.

2. 関数分解

ここでは基本用語の定義および関数分解の理論についての簡単な説明を行う.

2.1 用語の定義

$\{0,1\}^n \rightarrow \{0,1\}$ 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**コファクター (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, コファクターをとる変数 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ー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つのコファクターが異なるとき ($F_{\overline{x}} \neq F_x$), 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 の x によるコファクターを F_0 および F_1 とするとき, F を式 (1) の形に展開することを **Shannon 展開**と呼ぶ.

$$F = x \cdot F_1 \vee \overline{x} \cdot F_0 \tag{1}$$

関数 F_0 および F_1 のサポートの要素数は必ずもとの関数 F のサポートの要素数よりも少ない.

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.

$$F(X) = G(X_1, H(X_2)) \tag{2}$$

論理関数 F が式 (2) 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 を除いた物 $(=X_1)$ 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**すると言い, $A \perp B$ と表記する.

式 (2) 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. $|X_1 \cap X_2|$ を分解の**多重度**と呼ぶことにする. 明らかに直交な分解の多重度は0である.

2.2 関数分解のアルゴリズム

与えられた関数に対する一般の関数分解をすべて列挙することは実用上は困難であるが, 直交な分解に関しては分解は唯一に定まり, それを二分決定グラフを用いて効率よく求めるアルゴリズムが存在する (文献 [5]~[7]). これは関数 F の直交な分解は F の2つのコファクター F_0 と F_1 の共通な分解から求めることができるという性質を利用したもので, 関数 F を表す二分決定グラフの節点数を N とすると $O(N^2)$ の手間で F の直交な分解を求めるものである.

直交でない場合も, 自由集合と束縛集合の共通な変数が少ない場合には, 実用的なアルゴリズムが存在する [8]. 例えば,

$$F(X) = G(X_1, H(X_2))$$
$$X_1 \cap X_2 = \{x\}$$

というように x のみが共通な関数分解を考える. F の x に対するコファクター F_0, F_1 は次式のように表される. ただし, $\hat{X} = X \setminus \{x\}$, $\hat{X_1} = X_1 \setminus \{x\}$, $\hat{X_2} = X_2 \setminus \{x\}$ とする.

$$F_0(\hat{X}) = G_0(\hat{X_1}, H_0(\hat{X_2})) \tag{3}$$

$$F_1(\hat{X}) = G_1(\hat{X_1}, H_1(\hat{X_2})) \tag{4}$$

ここで, $\hat{X_1} \cap \hat{X_2} = \Phi$ であるから式 (3),(4) は直交な分解である. つまり, 2つのコファクター F_0, F_1 が同様の変数分割

$(\hat{X_1}, \hat{X_2})$ に基づく直交な分解を持てば，F 自身は x のみが自由集合と束縛集合に共通に含まれた (X_1, X_2) という変数分割に基づく関数分解を持つことがわかる．複数の変数に対するコファクターを定義した時と同様の議論により，直交でない関数分解も共通に現れる変数が複数の場合でも同様に求めることができる．具体的には，すべてのコファクターを求め，それらがすべて同様の変数分割に基づく直交な関数分解を持つかどうか調べればよい．

3. K-LUT マッピングアルゴリズム

ここでは与えられた論理関数 F を実現する K-LUT のネットワークを生成するアルゴリズムについて述べる．まず，準備としていくつかの性質について調べた後で，与えられた関数が K-LUT2 個で実現可能かを判定するアルゴリズムを示し，そのアルゴリズムを拡張することで目的の K-LUT マッピングアルゴリズムを得る．

3.1 準　　備

L 個の K-LUT からなるネットワークで実現できる論理関数の最大の入力数は表 1 の通りである．このことから与えられた関数の入力数 n に対して，必要な K-LUT のブロック数の下限 $L(n)$ は次式のように計算される．

表 1　L 個の K-LUT で実現できる論理関数の上限

LUT 数	最大の入力数
1	K
2	$2K-1$
3	$3K-2$
...	...
L	$L(K-1)+1$

$$L(n) \times (K-1) + 1 \geq n \quad (5)$$
$$L(n) \times (K-1) \geq n-1$$
$$L(n) \geq \frac{n-1}{K-1}$$

また，$K \geq 3$ の時，$K+1$ 入力の関数 F は図 1 のような 3 つの K-LUT で実現可能である．ただし，3 つの K-LUT の関数は以下の通り．

$$F_{\mathrm{LUT1}} = x_0 \cdot y_1 \vee \overline{x_0} \cdot y_2$$
$$F_{\mathrm{LUT2}} = F_{x_0}$$
$$F_{\mathrm{LUT3}} = F_{\overline{x_0}}$$

これは，F の x_0 による Shannon 展開を用いている．

同様に $K \geq 3$ の時，$K+2$ 入力の関数は 7 つの K-LUT で実現可能である．これを一般化すると $K+m$ 入力の関数は $2 \times 2^m - 1$ 個の K-LUT を用いれば必ず実現できることがわかる．このことから n 入力関数を実現するのに必要な K-LUT の個数の上限は式 (6) で与えられる．

$$U(n) = \begin{cases} 1 & n \leq K \\ 2^{(n-K+1)} - 1 & n > K \end{cases} \quad (6)$$

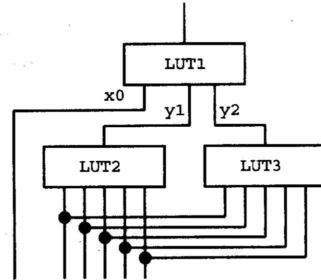

図 1　$K+1$ 入力関数の実現方法

この上限は指数的に増加するので n が $K+1$ 以上の時にはあまり実用的に意味のあるものではない(注1)．

3.2 2 個の K-LUT で実現可能な関数を求めるアルゴリズム

前述の式 (6) より $K+1 \sim 2K-1$ 入力の関数は 2 個の K-LUT で実現できる可能性がある．逆に $K+1$ 入力の関数でも 2 個の K-LUT では実現できない場合があるため，与えられた関数が 2 個の K-LUT で実現できるかどうかを判定する問題は単純ではない．

LUT2 個からなるネットワークは図 2 のような形となるので，対応した形の関数分解を持つかどうか調べればよい．図中の破線は LUT1 と LUT2 に共通に現れる変数を表している．図から明らかなように与えられた関数 F が LUT2 個で実現できるためには，関数 F が $G(X_1, H(X_2)), |X_1| \leq K, |X_2| \leq K$ の形の関数分解を持つことが必用十分条件となる．この関数分解が直交な分解である場合には文献 [7] のアルゴリズムを直接用いて，上の条件を満たす分解が存在するかどうか調べればよい．直交でない場合には，共通な変数を選び，その変数でコファクターしたものが同様の分解を持つか調べることになる．LUT は K 入力なので最大の多重度は K であるが，以下のような理由により $K-2$ 以下の多重度をもった分解のみを考慮すればよいことがわかる．図 2 の回路の多重度が 1 であった場合，独立な入力変数の個数は最大の $2K-1$ より 1 減って $2K-2$ となる．同様に多重度が l であった場合の入力数は $2K-l-1$ とな

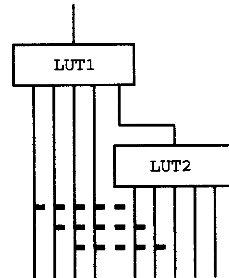

図 2　LUT2 個からなる回路

(注1)：さらに言えばこの上限はあまりタイトではない．この個数を使えば実現できると言っているだけで，最低でもこの個数だけ必要となる関数が存在していることを証明しているわけではない．

る．これが $K+1$ を下回った場合には K-LUT 一個で実現できることになるので，そのような多重度をもつ分解を考える必要はない．そこで，

$$K+1 = 2K-l-1$$
$$l = K-2$$

となり，$K-2$ 以下の多重度を持つ分解のみを考えればよいことになる．以上のことをまとめると与えられた関数が LUT2 個のネットワークで実現可能かを判定するアルゴリズムは図 3 のようになる．

```
LUT2_map(F)
{
1:   n ← F の入力数
2:   // ここでは n ≧ K, n ≦ 2K − 1 と仮定する.
3:   foreach F の直交な分解 {
4:       if |X1| ≦ K ∧ |X2| ≦ K
5:           return; // 分解が見つかった.
6:   }
7:   for(l = 1; l ≦ K − 2; l++) {
8:       foreach i(n,l) ∈ {n 個の中から l 個を選ぶ組み合わせ } {
9:           i(n,l) に含まれる変数で F をコファクタリングする.
10:          foreach すべてのコファクターに共通な直交な分解 {
11:              if |X1| ≦ K − l ∧ |X2| ≦ K − l
12:                  return; // 分解が見つかった.
13:          }
14:      }
15:  }
16:  return; // 分解は見つからなかった.
}
```

図 3　2 個の K-LUT に分解可能かを判定するアルゴリズム

このアルゴリズムは厳密に LUT2 個で実現可能かどうかを判定する．計算量は 6 行目と 7 行目のループの回数 × 直交分解の手間 となる．例えば $K=4$ の場合，n は最大でも 7 であり，ループの回数は最大で ${}_7C_1 + {}_7C_2 = 7+21 = 28$ 回である．$K=5$ の場合ループの最大の回数は 45 回である．直交分解の最悪の手間は二分決定グラフの節点数の二乗に比例するが，実際にはほぼ節点数に比例した手間で処理することができる．10 入力程度の関数の場合，二分決定グラフの節点数は数十〜数百なのでこのアルゴリズムで非常に高速に処理することができる．

3.3 与えられた関数を実現する LUT の回路を求めるアルゴリズム

K-LUT2 個の場合と同様に K-LUT が N 個の回路は図 4 のように K-LUT が $(N-1)$ 個の回路に K-LUT を 1 つ結合したものと考えることができる．つまり，束縛集合の要素数が K 以下であるような ($|X_2| \leq K$) 分解を求め，残りの部分を再帰的に処理して行けば任意の関数を K-LUT の回路で実現することができることになる．

ところが，K-LUT 2 個の場合と異なりこのアルゴリズムは

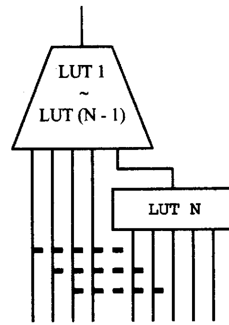

図 4　LUT N 個からなる回路

単純には機能しない．まず，直交でない分解の多重度として K まで考慮しなければならない．このことは分解を行った結果の関数の入力数がもとの関数よりも増えることを意味する．すると，単純に分解を求めてその結果の関数に対して再帰的にこのアルゴリズムを呼び出したのでは再帰処理が終了しないことになる．本来は入力数が増えても意味のある分解と意味のない分解を区別して，考えられるすべての分解を効率よく列挙する必要があるが，これは実用的には難しいことであるので，今回はヒューリスティックとして分解の結果の入力数が元の関数の入力数よりも少ないもののみを分解の候補として用いるようにしている．この制約のため，このアルゴリズムは必ず停止するが，その反面，実際には K-LUT が N 個で実現可能な関数であってもそれを発見できない場合がある．そのような場合の救済策として最後に Shannon 展開を用いた分解を試みている．この Shannon 展開による分解を行うことでこのアルゴリズムは最悪でも前述の上限値 $U(n)$ 以下の回路を生成することができる．

アルゴリズムを図 5 に示す．ここでは簡単のため **LUT_map** は与えられた関数を実現するのに必要な LUT の個数のみを値として返すものとしているが，LUT のネットワーク自体を結果として返すように変更することは容易である．

4. 実験結果

提案手法の効果を調べるために **LUT_map** アルゴリズムを C++を用いて実装し，計算機実験を行った．仕様計算機は Pentium-IV (2.5GHz) メモリ 1GB, OS とコンパイラは FreeBSD-4.8, g++-3.3.3 である．

実験は以下のような手順で行った．

（1） MCNC のベンチマークデータに論理合成プログラム sis の rugged スクリプト [11] を適用し，簡単化を行う．

（2） 8 入力以下の部分回路を列挙する．

（3） 切り出された部分回路を 2 入力ゲートに分解し，構造に基づく 4-LUT のマッピングを行う．具体的には 4 入力以下のクラスタを列挙して，クラスタによる最小被覆を求める [12], [13](このプログラムは参考文献のものではなく自作). [struct-map]

（4） 切り出された部分回路の論理関数を計算し，**LUT_map** を適用する．ただし $K=4$. [lut-map]

```
LUT_map(F)
{
1:   n ← F の入力数
2:   BestCost ← ∞;
3:   foreach F の直交な分解 {
4:     if |X_2| ≤ K {
5:       G ← 分解した残りの関数
6:       CurCost ← LUT_map(G) + 1;
7:       if BestCost > CurCost {
8:         BestCost ← CurCost;
9:         if BestCost = L(n)
10:          return BestCost;
11:      }
12:    }
13:  }
14:  for(l = 1; l ≤ K − 2; l++) {
15:    foreach i_(n,l) ∈ {n 個の中から l 個を選ぶ組み合わせ } {
16:      i_(n,l) に含まれる変数で F をコファクタリングする.
17:      foreach すべてのコファクターに共通な直交な分解 {
18:        if |X_2| ≤ K − l {
19:          G ← 分解した残りの関数
20:          CurCost ← LUT_map(G) + 1;
21:          if BestCost > CurCost {
22:            BestCost ← CurCost;
23:            if BestCost = L(n)
24:              return BestCost;
25:          }
26:        }
27:      }
28:    }
29:  }
30:  x ← 入力変数の 1 つ;
31:  F のコファクター F_0 および F_1 を計算する.
32:  CurCost ← LUT_map(F_0) + LUT_map(F_1) + 1;
33:  if BestCost > CurCost
34:    BestCost ← CurCost;
35:  return BestCost;
}
```

図 5 K-LUT のネットワークを求めるアルゴリズム

表 2 実験結果

回路名	ブロック数							
	struct-map				lut-map			
	5	6	7	8	5	6	7	8
9symml	—	—	—	14.0	—	—	—	9.0
C1355	2.0	4.3	4.0	3.0	2.0	3.0	3.0	3.0
C1908	2.0	2.0	2.4	3.1	2.0	2.0	2.0	3.0
C2670	2.0	3.5	3.3	3.8	2.0	2.0	3.0	3.0
C3540	2.5	3.0	3.8	6.9	2.0	2.0	2.5	3.2
C432	2.0	2.5	2.0	—	2.0	2.0	2.0	—
C499	3.0	4.3	4.0	3.0	2.0	3.0	3.0	3.0
C5315	2.7	3.1	3.3	4.2	2.0	2.1	2.3	3.0
C6288	2.4	3.2	3.2	3.2	2.0	2.1	2.7	3.0
C7552	3.1	3.2	3.8	5.0	2.0	2.1	2.3	3.1
C880	2.0	2.0	2.0	—	2.0	2.0	2.0	—
apex6	2.5	3.1	4.8	4.0	2.0	2.1	2.5	3.0
apex7	—	2.0	2.3	—	—	2.0	2.0	—
b9	2.0	3.0	4.4	5.2	2.0	2.0	2.6	3.2
des	2.2	8.7	24.8	26.8	2.0	3.1	9.5	13.5
f51m	5.2	18.9	27.3	32.0	2.0	3.6	7.8	12.4
rot	2.4	3.1	4.1	5.0	2.0	2.1	2.4	3.0
z4ml	—	—	5.5	5.5	—	—	2.5	3.0
Total	2.6	3.5	8.7	21.0	2.0	2.2	4.0	10.7

（5） 2 つの手法の平均のブロック数を比較する.

表 2 に結果を示す. 各回路の各入力数ごとに部分回路をまとめ, その部分回路を実現するのに必要な平均のブロック数を表している. 表中の "—" は該当する部分回路がなかったことを示している. 処理時間はこれらすべての計算を 10 分程度で行っている. マッピングを試した部分回路の総数は約 8,000 であり, 部分回路 1 つあたりの処理時間はそれほど長くないが, 実用的なテクノロジマッパーとしては低速と言える. 現在はプログラムはアルゴリズムの検証用の試作段階であり, 速度的には改善の余地があるものと思われる.

結果は概ね良好で struct-map よりもはるかに少ないブロック数で実現できることがわかる. ほとんどの回路に対しては 6 入力までの部分回路を 2 個のブロックで実現できている. これらの例では, LUT2 個で実現できる場合が多く, 提案するアルゴリズムでは LUT2 個で実現できる例を必ず発見できることが効いているものと思われる. また, 9symml, des, f51m 以外は 8 入力の部分回路のほとんどを 3 個のブロックで実現している (トータルの平均が 10 なのは des の部分回路数が多いため). しかるに構造に基づく手法では 1〜2 個の余計なブロックを必要としている. これらの回路は一旦, 論理最適化を行っているので元の回路に論理的な冗長性は残っていない. にも関わらず, 構造に基づく手法と関数分解に基づく手法でこのような差がついていることは特筆すべき事柄である. LUT 型 FPGA の合成に特化した論理最適化手法の必要性が示されていると言えよう.

9symml は人為的な対称論理, des は暗号演算回路であることが知られており, 通常の単純な制御論理と趣きを異にしていることがこの実験結果からも見て取ることができる. 構造に基づく手法に比べると 2 倍以上ブロック数の差がついており, このように複雑な論理関数に対しても関数分解に基づく手法が有効であることがわかる.

5. おわりに

本稿では, 関数分解を用いて与えられた論理関数を実現する LUT の回路を生成するアルゴリズムについて述べた. これは二分決定グラフを用いて高速に直交分解を求めるアルゴリズムを応用したもので常に構造に基づく手法よりも小さな回路を生成している. LUT を用いた FPGA に対する論理合成手法としてこのような論理関数処理を用いた手法が有効であることが示されている. 入力数が多くなると構造に基づく手法との差は顕著となるが, 一方で計算時間も増える傾向があるため, 効率的な処理手法を開発することが今後の課題である. 提案アルゴリズ

ムは現在の所，非常に簡単なヒューリスティックを用いているので，効率化とともに性能向上の余地も十分にあると思われる．

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．

文　献

[1] R. L. Ashenhurst: "The decomposition of switching functions", international symposium on the theory of switching, pp. 74–116 (1957).

[2] J. P. Roth and R. M. Karp: "Minimization over boolean graphs", IBM Journal, pp. 227–238 (1962).

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

[4] T. Sasao: "Fpga design by generalized functional decomposition", Logic Synthesis and Optimization, Kluwer Academic Publishers, pp. 233–258 (1993).

[5] V. Bertacco and M. Damiani: "The disjunctive decomposition of logic functions", International Conference on Computer-Aided Design (ICCAD'97), pp. 78–82 (1997).

[6] Y. Matsunaga: "An exact and efficient algorithms for disjunctive decomposition", the Workshop on Synthesis And System Integration of Mixed Technologies (SASIMI'98), pp. 44 – 50 (1998).

[7] Y. Matsunaga: "An efficient algorithm finding simple disjoint decompositions using bdds", IEICE trans. on fund., **E85-A**, 12, pp. 2715–2724 (2002).

[8] 松永："直交でない関数分解の効率的な列挙手法", 情処研報, 第2000-SLDM-96-9巻, pp. 57–63 (2000).

[9] 松永："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, 第VLD2000–96巻, pp. 161–166 (2000).

[10] 松永："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, 第VLD2002–2巻, pp. 7–12 (2002).

[11] H. Savoj, H. Touati and R. K. Brayton: ""Improved Scripts in MIS-II for Logic Minimization of Combinational Circuits"", International Workshop on Logic Synthesis (1991).

[12] R. J. Francis, J. Rose and Z. Vranesic: "Chortle-crf: Fast technology mapping for lookup table-based fpgas", the 28th ACM/IEEE Design Automation Conference, pp. 613–619 (1991).

[13] R. Murgai, N. Shenoy, R. K. Brayton and A. Sangiovanni-Vincentelli: "Improved logic synthesis algorithms for table look up architectures", the International Conference on Computer-Aided Design, pp. 564–567 (1991).