論　文

許容関数に基づいた表参照型 FPGA の最適化手法

山下　茂[†]　　上林　彌彦[†]　　室賀　三郎[††]

Optimization Methods for Look-Up Table Type FPGAs Based on Permissible Functions

Shigeru YAMASHITA[†], Yahiko KAMBAYASHI[†], and Saburo MUROGA[††]

あらまし　近年書き換え可能なゲートアレイ (FPGA:Field Programmable Gate Array) はその性能の技術的な発展が著しく, また, 柔軟でかつ手軽に論理変更ができるため, ハードウェアの試作や再構成可能な計算機アーキテクチャーへの適用が進んできている. そのため, FPGA の論理設計手法を確立する要求が高まってきている. 本論文では, 表参照型 FPGA にマッピングされた回路を最適化する手法について述べる. この手法は設計改良による論理回路最適化手法であるトランスダクション法の許容関数の概念を適用して冗長度を減らそうとするもので, 次の 2 つの方法を検討した. 1) ある論理ブロックで他の論理ブロックの代用をすることにより, ブロック数を減少させる. 2) 出力に影響を与えない範囲で論理ブロックの実現する内部論理を変化させることを併用して, 論理ブロックの代用をする. 後者の方が一般に能力が高いが, 前者の方の結果の方が良い場合もあり, また前者の方が計算時間が短くて済む. また, 表参照型 FPGA にマッピングされた回路に対してこれらの手法を適用した結果, 約 9% の冗長度を削減できたことを示し, 有効性を示している. 本論文の手法はスタンダードセルやゲートアレイを用いた設計にも適用することができる.

キーワード　論理設計, トランスダクション法, 表参照型 FPGA, 論理最適化

1. まえがき

書き換え可能なゲートアレイ (FPGA : Field Programmable Gate Array) はユーザー側でプログラム可能な素子であり, システムのプロトタイプの設計などに使われている. 近年, 比較的大規模な回路を商用の FPGA で実現できるようになり, また再構成可能なシステムの設計などへの応用へも進んできている. 商用の FPGA はいろいろなタイプのものがあり, それにより設計技法も多少変わってくるが, 本論文では表参照型の FPGA を対象としている. 表参照型 FPGA は二次元に配列された論理ブロックとそれらの間を縦横に走る配線領域で相互に接続できるようにしたものである. 論理ブロックは LUT(Look Up Table) と呼ばれ, 任意の k(4 から 6 のものが多い) 変数関数を実現できる.

FPGA は多段論理構造をしており, 基本回路や設計の評価関数が複雑であるため, その設計技法は完全には確立されていないのが現状であるが, 典型的な設計フローとしては以下の通りである. [7]

- 論理最適化
- テクノロジーマッピング
- 配置
- 配線

まず HDL などで記述された回路の記述を通常はブール関数などに変換した後, そのブール関数の冗長性を**論理最適化**によって削除する [1]. 次に**テクノロジーマッピング**のツールにより, 回路を FPGA の論理ブロックに分割する [4] [5] [6]. そして, それらの論理ブロックを**配置**ツールによって FPGA のどの位置に配置するかを決定する. 最後に**配線**ツールによって必要な結線を実現するように内部のスイッチの状態を決定する.

上述した設計フローの中において, 従来のテクノロジーマッパーは論理ブロック間の関数の依存関係等よりも回路の形状等の情報を考慮して回路を分割するため, テクノロジーマッピングされた後の回路にいくらかの冗長性が残る可能性がある.

本論文では表参照型 FPGA にテクノロジーマッピングされた回路に対してトランスダクション法 [2] の許容

[†] 京都大学工学部, 京都市
Faculty of Engineering, Kyoto University, Kyoto Unversity, Kyoto-shi, 606-01 Japan
[††] イリノイ大学計算機科学科, イリノイ州アーバナ市
Department of Comput er Science, University of Illinois, Urbana, Illinois

関数の概念を導入して, 冗長性を削減する手法について述べる.

2 つの手法, **論理ブロック置換**と**内部論理変更**を提案し, 実際に MIS テクノロジーマッパー [4] によって表参照型 FPGA にマッピングされた回路に対して適用した. 内部論理変更は従来の NOR や NAND 素子での設計では考えられなかったものである.

平均して, **論理ブロック置換**では約 8%, **内部論理変更**では約 9% 論理ブロック数の削減が可能であった.

また, 本手法はゲートアレイやセルライブラリーを用いて設計された回路の冗長性を削減することにも応用できると考えられる.

2. 基本的事項

本論文では, 表参照型 FPGA にマッピングされた回路を扱う. そのため以下では, 節点が論理ブロックを表し, 枝が論理ブロック間の結線を表すような有向非巡回グラフ (DAG) で回路を表現することにする. ここで以下の用語を定義する. 回路の入力数を n とする. N を回路中の論理ブロックの数とし, $L = \{l_1, l_2, \cdots, l_N\}$ を論理ブロックの集合とする. $C = \{c_{ij}\}$ は結線の集合で, l_i から l_j への結線が存在すれば c_{ij}がC に含まれるものとする. もし c_{ij}がC に含まれれば, l_i は l_j の immediate predecessor であるといい, l_j は l_i の immediate successor であるという. $IP(l_i)$ および $IS(l_i)$ を, それぞれ, l_i のすべての immediate predecessor の集合および l_i のすべての immediate successor の集合とする.

以下では, 回路内の要素 c(結線, 又は論理ブロック) で実現される論理関数を $f(c)$ と表現する. $f(c)$ は 2^n 次元ベクトルで, $f(c) = (f(c)^{(1)}, f(c)^{(2)}, \cdots, f(c)^{(2^n)})$ と表現される. ここで $f(c)^{(j)}$ は, $f(c)$ を表す真理値表の j 番めの欄の値で, 0 か 1 である.

回路中の要素 c(結線, 論理ブロック) の実現する関数 f を f' で置き換えても, 回路の出力に変化がなければ, f' は c の**許容関数** [2] であるという. 許容関数は一つとは限らないので, 一般には許容関数の集合が考えられる. 許容関数集合の内で, 回路中の全要素に対して同時に複数箇所で変更可能な許容関数のみからなる集合を CSPF(Compatible Set of Permissible Functions) と呼ぶ. 以下では, $G(c)$ で回路中の要素 c(結線, 又は論理ブロック) の CSPF を表すものとする. $G(c)$ は $0, 1, *(don't care)$ の 3 値を要素として持つ 2^n 次元ベクトルで, $G(c)$ の要素に * を含む場合, * の代わりに 0 と 1 とをすべての可能な組合せで代入して生成されるすべてのベクトルの集合が c の CSPF であることを表している. 例えば c の CSPF が {(1,0,0,0),(1,0,0,1),(1,0,1,0),(1,0,1,1)} である場合, $G(c) = (1, 0, *, *)$ と表現される.

表 1 二項演算 ♯1

First element \ Second element	0	1
0	0	*
1	*	1
*	*	*

表 2 二項演算 ♯2

First element \ Second element	0	1
0	*	1
1	0	*
*	*	*

表 3 二項演算 ♯3

First element \ Second element	0	1
0	*	*
1	*	*
*	*	*

3. CSPF の計算方法

本節では CSPF の計算方法について述べる. これは [2] で述べられている手法を表参照型 FPGA に適用するものである.

論理ブロック l_i の内部論理は積和形で表現されているとし, その積項数を n_i とする. また, l_i の内部論理を積和形で表現した時の積項の集合を $T_i = \{t_{i1}, t_{i2}, \cdots, t_{in_i}\}$ で表すことにする. また, $f(t_{ij})$ で論理ブロック l_i の内部論理の j 番めの積項で実現される回路の入力による論理関数を表すことにする. 従って, $f(l_i) = f(t_{i1}) + f(t_{i2}) + \cdots f(t_{in_i})$ となる.

また, L から正整数集合 $1, 2, \cdots, N$ への 1 対 1 写像を r とし, T_i から正整数集合 $1, 2, \cdots, n_i$ への 1 対 1 写像を q_i とする. $r(l_i)$ によって論理ブロック l_i の順序づけを, $q_i(t_{ij})$ によって l_i の内部論理を表す積項 t_{ij} に順序づけをつけることができる. これらの順序づけをあらかじめ決定しておき, それに従って CSPF は計算される.

また, 二項演算 ♯1, ♯2, ♯3, ⊘, △ を, それぞれ表 1, 2, 3, 4, 5のように定義する. ここで, '-' は未定義を示す. 2 つのベクトル間にこれらの二項演算を適用する場合は, 対応する要素間に適用するものとする. $G(l_k)$ と $IP(l_k)$ に含まれるすべての l_j に対して, $f(c_{jk})$ が与え

られている時, $G(c_{jk})$ は次式で計算される.

$$G(c_{jk}) = \bigcap_{t_{ki} \in T_k} [\ \{\ G(l_k) \oslash \left(\bigcup_{q_k(t_{ko}) > q_k(t_{ki})} f(t_{ko}) \right) \sharp 1\ f(t_{ki})\ \} \triangle \left(\bigcap_{l_m \in IP(l_k)} V_{ij}(f(c_{mk})) \right) \diamond\ f(c_{jk})\ ]$$

ここで, 二項演算 $\diamond$ は, 積項 t_{ki} が結線 c_{jk} に対応する変数を持つ場合 $\sharp 1$, 結線 c_{jk} に対応する変数の否定を持つ場合 $\sharp 2$, 結線 c_{jk} に対応する変数を持たない場合 $\sharp 3$ とみなされるとする. また, $q_k(t_{ko}) > q_k(t_{ki})$ を満足する t_{ko} が存在しないような場合には, $(\bigcup_{q_k(t_{ko}) > q_k(t_{ki})} f(t_{ko}))$ は恒等的に 0 である関数であるとみなされるとする. また, $V_{ij}(f(c_{mk}))$ は以下のようにみなされるとする.

- $r(l_m) > r(l_j)$ かつ t_{ki} が結線 c_{mk} に対応する変数を持つ場合, $f(c_{mk})$ とみなされる.
- $r(l_m) > r(l_j)$ かつ t_{ki} が結線 c_{mk} に対応する変数の否定を持つ場合, $\overline{f(c_{mk})}$ とみなされる.
- その他の場合は, 恒等的に 1 である関数であるとみなされる.

論理ブロック l_k と $IS(l_k)$ に含まれるすべての論理ブロック l_j に対して, $G(c_{kj})$ が与えられている時, $G(l_k)$ は次式で計算される.

$$G(l_k) = \bigcap_{l_j \in IS(l_k)} G(c_{kj})$$

表 4　二項演算 $\oslash$

		Second element	
	$\oslash$	0	1
	0	0	-
First element	1	1	*
	*	*	*

表 5　二項演算 $\triangle$

		Second element	
	$\triangle$	0	1
	0	*	0
First element	1	-	1
	*	*	*

4. CSPF を用いた最適化手法

本節では, 表参照型 FPGA にマッピングされた回路に対して前節で述べた CSPF を用いて冗長性を削減する 2 つの手法について述べる. まず, ある論理ブロックに対して代用できる論理ブロックを探し, その論理ブロックで置き換えていく**論理ブロック代用**について述べ, 続いて論理ブロックの内部論理を変更することを併用する**内部論理変更**について述べる.

4.1 論理ブロック代用

例えば, 図 1(a) のように回路中に論理ブロック l_i と l_j があり, $f(l_j)$ が $G(l_i)$ に含まれる場合, l_i の出力がつながっている結線 (この例では l_m と l_n につながれている結線) を図 1(b) のように l_j からの結線で置き換えることによって, l_i を取り除くことが可能である. このようにある論理ブロックの出力を他の論理ブロックの出力で置き換えることによって回路を最適化していく手法を

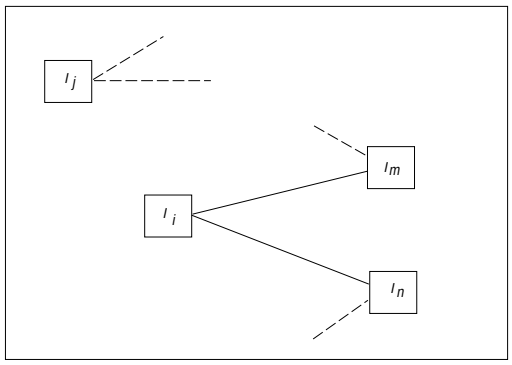

図 1　論理ブロック代用の適用例
Fig. 1　An example of Logic Block Substitution

論理ブロック代用と呼ぶことにする．以下にその手順の概略を示す．

手続き 論理ブロック代用

step1　回路の出力側から順に論理ブロックを選び, それを l_i として step2 へ．そのような論理ブロックがなくなれば終了．
step2　回路の入力側から順に, l_i より回路の入力側からの段数が小さく, f(l_j) が G(l_i) に含まれるような論理ブロック l_j を選び step3 へ．そのような論理ブロックがなければ step1 へ．
step3　l_i が回路全体の出力となっていたら l_j を新しく回路の出力とする．
step4　$IS(l_i)$ の中から論理ブロックを順に選びそれを l_k として, step5 へ．そのような論理ブロックがなくなれば, step7 へ．
step5　$IP(l_k)$ に l_j が含まれていれば, step6 へ．そうでなければ, l_k の l_i との入力を l_j につけかえて, step4 へ．
step6　l_i と l_k の間の結線を切り, それにともない l_k の内部論理の変更を行ない (以下で例を用いて説明する．), step4 へ．
step7　l_i とそのすべての入力との間の結線を取り除き, step1 へ．

step2 において, 回路の入力側からの段数が l_i より小さいものの中から l_j を選ぶ理由は, l_j の方が入力側からの段数が大きいと, l_i の代わりに l_j をつなぐことによって回路の最長パスの段数が増える可能性があるためである．また, l_j の方が l_i より入力側からの段数が小さいため step4 で選ばれる l_k の方が必ず l_j より入力側からの段数が大きくなるので, step5 において l_j から l_k に結線をつないでもループを作ることがないということが保証される．

次に step6 のケースを例を上げて説明する．　例えば,

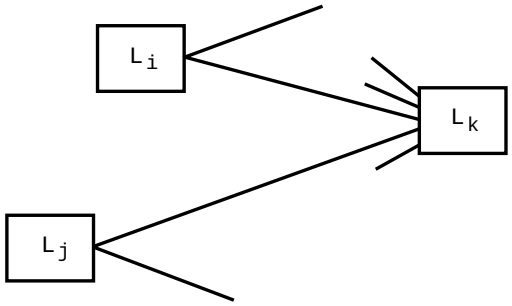

図 2　内部論理を変更する必要例
Fig. 2　An example of the case requiring to change the internal logic

図 2のように l_j が $IP(l_k)$ に含まれている場合には, l_k の入力は l_j と既につながっているので, step5 で l_k の l_i との入力を l_j につけかえるのではなく, 単に l_i と l_k の間の結線を切れば良い．ただし, それにともない変更後の $f(l_k)$ が $G(l_k)$ に含まれているように, l_k の内部論理の変更を適切に行なう必要がある．　例えば, 図 3のように l_k の内部論理が l_k の入力 $x_1 \cdots x_5$ に対して ($x_1 \cdot x_3 \cdot x_4 + x_2 \cdot x_3 \cdot x_5$) と表現されており, l_i が x_3, l_j が x_4 に接続されているような場合に, l_i の出力を l_j の出力で置き換える場合には, 図 4のように l_i と l_k の間の結線を切り, 内部論理を $(x_1 \cdot x_4 + x_2 \cdot x_4 \cdot x_5)$ と変更

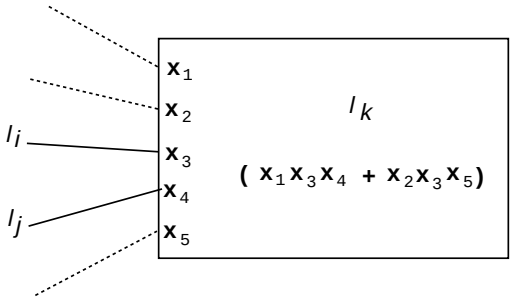

図 3　内部論理の変更例 (1)
Fig. 3　An example of changing the internal logic(1)

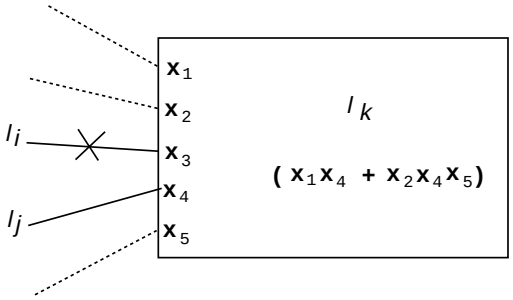

図 4　内部論理の変更例 (2)
Fig. 4　An example of changing the internal logic(2)

すれば良い. すなわち, l_i の出力を l_j の出力で置き換えるということは l_k の内部論理を表現する論理式中の x_3 を x_4 で置き換えることであるから, もとの内部論理の ($x_1 \cdot x_3 \cdot x_4 + x_2 \cdot x_3 \cdot x_5$) は ($x_1 \cdot x_4 \cdot x_4 + x_2 \cdot x_4 \cdot x_5$) と変更され, これを簡単化して $(x_1 \cdot x_4 + x_2 \cdot x_4 \cdot x_5)$ となるのである.

4.2　内部論理変更

前小節で述べた**論理ブロック代用**は基本的に論理ブロックの内部論理が固定されているという前提で変換を行なっている. しかし, 表参照型 FPGA の場合, 論理ブロックの内部論理を自由に変更することができるので, それを利用することによって, より回路変換の自由度が上がると考えられる. 本小節では, 論理ブロックの内部論理を変更することを併用することによって, **論理ブロック代用**よりもより多くの場合に代用できる論理ブロックを見つけ, その論理ブロックで置き換える**内部論理変更**について述べる.

論理ブロック代用では, $f(l_j)$ が $G(l_i)$ に含まれていなければ l_i の出力を l_j の出力で置き換えることはできない. しかし, **内部論理変更**では, そのような場合でも以下の条件が成立すれば, l_i の出力ををを l_j の出力で置き換える.

- $G(l_i) \cap G(l_j)$ が空集合でない.
- l_j の内部論理を変更するだけで, l_j で実現される関数を $(G(l_i) \cap G(l_j))$ に含まれるような関数に変更できる.

以下に**内部論理変更**の手順の概略を示す.

手続き 内部論理変更

step1　回路の出力側から順に論理ブロックを選びそれを l_i として step2 へ. そのような論理ブロックがなくなれば終了.

step2　回路の入力側から順に, l_i より回路の入力側からの段数が小さく, $G(l_i) \cap G(l_j)$ が空集合でないような論理ブロック l_j を選び step3 へ. そのような論理ブロックがなければ step1 へ.

step3　手続き SOP(後述) によって, l_j の内部論理を変更して l_j で実現される関数を $(G(l_i) \cap G(l_j))$ に含まれるような関数に変更して, step4 へ. 手続き SOP によってそのような変更ができない場合には step2 へ.

step4　l_i が回路全体の出力となっていたら l_j を新しく回路の出力とする.

step5　$IS(l_i)$ の中から論理ブロックを順に選びそれを l_k として, step6 へ. そのような論理ブロックがなくなれば, step8 へ.

step6　$IP(l_k)$ に l_j が含まれていれば, step7 へ. そうでなければ, l_k の l_i との入力を l_j につけかえて, step5 へ.

step7　l_i と l_k の間の結線を切り, それにともない l_k の内部論理の変更を行ない, step5 へ.

step8　l_i とそのすべての入力との間の結線を取り除き, step1 へ.

内部論理変更は, step3 を除いて**論理ブロック代用**と同様の処理を行なうものである. そのため, ここでは step3 の手続き SOP について述べることにする. $f_1, f_2, \cdots, f_i$ をある論理関数とし, F をある CSPF とすると F に含まれるある論理関数を表す $f_1, f_2, \cdots, f_i$ の積和形を作るのが $SOP(F, f_1, f_2, \cdots, f_i)$ である. そのような積和形が作れない場合には $SOP(F, f_1, f_2, \cdots, f_i)$ は Error を返すものとする. $SOP(F, f_1, f_2, \cdots, f_i)$ の手続きを図 5に示す. 図 5において, True, および False はそれぞれ恒真関数, 恒偽関数を表す. '+', および'·' はそれぞれ論理和, 論理積を表す. また, ● は表 6で定義されるよう

```
SOP(F, f1, f2, ··· , fi)
    if(i = 1){
        if(f1 ∈ F) return f1
        else if(f̄1 ∈ F) return f̄1
        else return Error
    }
    else{
        F1 = F • f1
        if(False ∈ F1) F1' = False
        else if(True ∈ F1) F1' = True
        else {
            F1' = SOP(F1, f2, ··· , fi)
            if(F1' = Error) return Error
        }
        F0 = F • f̄1
        if(False ∈ F0) F0' = False
        else if(True ∈ F0) F0' =True
        else {
            F0' = SOP(F0, f2, ··· , fi)
            if(F0' = Error) return Error
        }
        return (F1'·f1 + F0'·f̄1)
    }
```

図5 SOP のアルゴリズム
Fig. 5 The Algorithm of SOP

表6 二項演算 •

First element \ Second element	0	1
0	*	0
1	*	1
*	*	*

な二項演算子である. これらの演算子による演算結果はオペランドのうちの 1 つでも Error なら Error とする. $SOP(F, f_1, f_2, \cdots, f_i)$ は, F を f_1 から順にシャノン展開していくことによって F を表す積和形を見つけ出そうというものである. 以下では, 図 6の場合の例について説明する. 図 6の論理ブロック l_j は 3 つの入力 f_1, f_2, f_3 を持ち, それらで実現されている論理関数をそれぞれ, 1011, 0101, 0111 とする (簡単のため 4 ビットで考えている). 今 l_j の内部論理が $(f_1 \cdot \overline{f_2} \cdot f_3)$ となっているとすると, $f(l_j) = (0010)$ となる. また, $G(l_j) = (**10)$ とする. この時ある論理ブロック l_i が $G(l_i) = (*1*0)$ を満たしているとすると, $G(l_i) \cap G(l_j) = (*110)$ であるから, l_j の内部論理を変更することによって $f(l_j)$ を (0011) から (∗110) に含まれるような論理関数に変更できれば, l_i の出力を l_j の出力で置き換えることができることになる. そこで, SOP により (∗110) に含まれる関数を実現する f_1, f_2, f_3 の積和形を見つける. まず, 図 6に示してある通り, (∗110) を $(F_1 \cdot f_1 + F_0 \cdot \overline{f_1})$ の形に展開する. ここで, F_1 は (∗110) • f_1 によって計

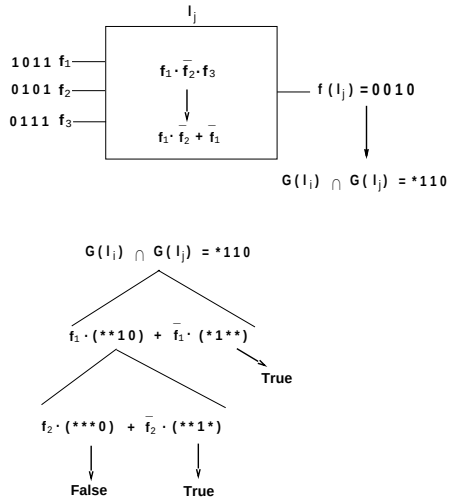

図6 SOP の適用例
Fig. 6 An example of application of SOP

算され (∗∗10) となる. F_0 は (∗110) • $\overline{f_1}$ によって計算され (∗11∗) となる. 次に, 同様に F_1 を f_2 で展開し, $(f_2 \cdot (***0) + \overline{f_2} \cdot (***1))$ となる. このように入力変数で以下の条件が満たされるまで展開していく.

- 展開した結果の F_1 および F_0 に True または False が含まれる場合. この場合, それ以上展開する必要がない.
- 展開するための入力変数がなくなってしまった場合. この場合, 要求を満たすような関数に展開できなかったことを意味するため, Error を返す.

図 6の例では, 展開に成功し, 内部論理を $(f_1 \cdot \overline{f_2} + \overline{f_1})$ と変更すればいいことになる.

5. 実験結果および考察

前節で述べた手法を実装し, MCNC ベンチマーク回路に対して実験を行なった. 内部での関数表現には, 現 NTT 湊氏による SBDD パッケージ [3] を用いている. 論理ブロックは 5 入力のものを仮定して実験を行なった. 次のような MIS [1] のコマンドにより, 5 入力の論理ブロックに分割を行なった.

- xl_split -n 5
- xl_partion -n 5
- xl_cover

この手法により生成された回路を初期回路として, 前節で述べた 2 つの手法, **論理ブロック代用**および**内部論理**

表 8　内部論理変更と論理ブロック代用の比較

回路名	論理ブロック代用				内部論理変更			
	LB	conn	lev	CPU	LB	conn	lev	CPU
C432	109	282	13	25.7	109	**277**	13	68.0
C499	**211**	376	11	621.3	217	**370**	11	821.4
alu2	138	533	22	4.7	**137**	**526**	22	4.3
alu4	260	879	24	19.4	**258**	**870**	24	18.5
apex7	128	313	7	2.7	128	313	7	4.0
b9	88	191	3	2.1	**87**	**188**	3	2.7
c8	**84**	**224**	4	1.8	85	228	4	2.2
cordic	**50**	**94**	7	1.8	51	96	6	2.1
example2	**189**	**422**	4	2.4	192	433	4	5.5
i9	443	1407	**7**	50.2	**420**	**1262**	9	253.3
lal	**73**	**179**	5	1.5	75	183	5	1.8
sct	**61**	**160**	4	1.4	62	162	4	1.8
term1	149	460	8	11.8	**141**	**424**	8	21.3
too_large	295	1004	13	294.5	**280**	**960**	**11**	7821.3
vda	**355**	**1458**	5	6.8	367	1518	5	11.9

変更を適用した．**論理ブロック代用**を適用した結果を表7に，**内部論理変更**と**論理ブロック代用**をそれぞれ適用した結果の比較を表8に示す．　表中では，LB, conn, lev,

表7　論理ブロック代用の実験結果

回路名	初期回路			論理ブロック代用		
	LB	conn	lev	LB	conn	lev
C432	122	317	13	**109**	**282**	13
C499	243	408	11	**211**	**376**	11
alu2	141	540	22	**138**	**533**	22
alu4	264	891	25	**260**	**879**	**24**
apex7	129	318	7	**128**	**313**	7
b9	90	197	3	**88**	**191**	3
c8	87	235	4	**84**	**224**	4
cordic	55	107	7	**50**	**94**	7
example2	195	445	5	**189**	**422**	**4**
i9	478	1490	9	**443**	**1407**	**7**
lal	83	213	5	**73**	**179**	5
sct	65	169	4	**61**	**160**	4
term1	173	553	8	**149**	**460**	8
too_large	352	1250	13	**295**	**1004**	13
vda	400	1683	5	**355**	**1458**	5

CPU はそれぞれ回路中の論理ブロックの数，回路中の結線の数，回路の最長パス上の論理ブロックの数，および Sparc Station 10 での実行時間 (秒) を表す．また，表中では良い方の結果を太字で示している．

表7より，上述したような MIS のコマンドにより生成された初期回路に対して**論理ブロック代用**は平均して 8% 論理ブロック数を削減していることがわかる．これにより，**論理ブロック代用**によりある程度の冗長度を削減することが可能であることが示された．ただ，**論理ブロック代用**は回路の段数はほとんど削減することができていない．その理由の 1 つとして，表参照型の FPGA にマッピングされた回路では回路内に多数の最長パスがあるため，簡単に回路の段数を減らすことができないということが考えられる．

次に表8より，**内部論理変更**は**論理ブロック代用**と比べて計算時間が少し余分にかかっているが，**論理ブロック代用**に比べて平均して 1% 論理ブロック数を削減していることがわかる．ただ，すべての回路において**内部論理変更**の方が結果がいいというわけではなかった．**内部論理変更**の方が，内部論理を変更することも許して論理ブロックの代用を行なうため，ある時点でみるとより多くの論理ブロックを代用することができることは明らかである．しかし，最終結果が悪くなっている場合がある原因としては，以下のような場合があるからだと考えられる．例えば，ある時点で**論理ブロック代用**なら論理ブロック l_i を他の論理ブロックで代用できないのだが，**内部論理変更**なら可能であったとする．しかし，l_i を他の論理ブロックで代用したために，回路の冗長度が減りその後の回路変形がうまくできなくなり，最終的な結果は逆に悪くなる場合があるのだと考えられる．すなわち，代用を行なう論理ブロックの順序の選択が正しく行なわれないと**内部論理変更**の方が最終結果が悪くなることもあり得るのだと考えられる．本論文では，代用を行なう論理ブロックの候補が複数ある場合にどれを選択するかについては考慮にいれなかったが，このことについて考察することは今後の課題である．

6. むすび

本論文ではトランスダクション法の許容関数を表参照型 FPGA に導入した．また，表参照型 FPGA にマッピングされた回路を許容関数を用いて最適化する 2 つの手法，**論理ブロック代用**および**内部論理変更**を提案し，その実験結果も示した．実験結果より，MIS のコマンドにより生成された初期回路に対して**論理ブロック代用**は平均して 8% 論理ブロック数を削減することができ，**内部論理変更**はそれよりもさらに 1% 論理ブロック数を削減できることがわかった．これによって我々の提案した手法により，ある程度の冗長度が減らせることがわかった．今後はより最適化能力の高い最適化手法の開発，および配置，配線ツールとの結合を行なうつもりである．

謝辞　SBDD パッケージの使用を快諾していただいた京都大学工学部矢島脩三教授並びに矢島研究室の皆様に深謝致します．なお，本研究は文部省の科研国際共同研究による援助を受けたものである．

文　献

[1] R. Brayton, E. Detjens, S. Krishna, T.Ma, P. McGeer, L.Pei, N. Phillips, R. Rudell, R. Seagal, A. Wang, R. Yung and A. Sangiovanni-Vincentelli: "Multiple-Level Logic Optimization System" ,Proc. IEEE Internatinal Conference on Computer Aided Design, pp.356–359, Nov. 1986.

[2] S. Muroga, Y. Kambayashi, H. C. Lai, J. N. Culliney: " The Transduction Method-Design of Logic Networks Based on Permissible Functions", *IEEE Trans. Comput.*, Vol.38, No.10, October 1989.

[3] S. Minato, N. Ishiura, S. Yajima: "Shared binary decision diagram with attributed edges for efficient boolean function manipulation", Proc. 27th Design Automat. Conf., pp.52–57, June 1990.

[4] Murgai R., Nishizaki Y., Shcnoy N., Brayton R., Sangiovanni-Vinccentelli A.: " Logic Synthesis for Programmable Gate Arrays", Proc. 27th Design Automat. Conf., pp.620–625, June 1990.

[5] R.J.Francis, J.Rose, Z.Vranesic: "Techonology Map-

ping for Delay Optimization of Lookup Table-Based FPGAs", International Workshop on Logic Syntesis, 1991.

[6] P.S.Sawar, D.E. Thomas: "Area and Delay Mapping for Table-Look-Up Based FPGAs", Proc. 29th Design Automat. Conf., pp.368–373, June 1992.

[7] Stephen D.Brown, Robert J.Francis, Jonathan Rose, Zvonko G.Vranesic: "FIELD-PROGRAMMABLE GATE ARRAYs", Kluwer Academic Publishers, 1992.

（平成 7 年 4 月 3 日受付）

山下　茂

平 5 京大・工・情報卒．同年同大大学院修士課程入学．論理回路の設計の研究に従事．情報処理学会会員．

上林　彌彦　（正員）

昭 40 京大・工・電子卒．昭 45 同大大学院博士課程了．工博．京大助手，イリノイ大リサーチアソシエイト，京大助教授，九大教授を経て平 2 より京大工学部教授．論理回路，オートマトン，データベースの研究に従事．カナダマッギル大，クウェート大，中国武漢大の客員教授．「Database-A Bibliography」(Computer Science Press)，「データベース」(昭晃堂) など．昭 50 本会米沢賞，昭 58 丹羽賞，昭 62 本会著述賞．情報処理学会，日本ソフトウェア科学会，ACM 各会員，IEEE シニア会員．

室賀　三郎　（正員）

昭 22 東大・工・電気卒．日本国有鉄道研究所，電波庁電波管理管理局，日本電信電話公社電気通信研究所，IBM T.J. WATSON 研究所を経て，昭 39 よりイリノイ大コンピュータ・サイエンス科教授 (電気工学科兼任)．音声認識，多重化通信方式，情報理論，閾値論理，論理回路の自動合成の研究に従事．Threshold Logic and Its Applications, Logic Desigin and Swiching Theory, VLSI System Design など．IEEE Fellow, ACM など各会員．