設計自動化　５０－８
（1989.12.14）

トランスダクション法の評価と改良

松永　裕介　　藤田　昌宏

（株）富士通研究所

あらまし　本稿では、２分決定グラフと呼ばれるデータ構造を利用したトランスダクション法の改良について述べる。トランスダクション法はNORゲートのみからなる回路を対象に考えられたものであり、比較的容易に一般的な回路を扱うように拡張可能であるが、その場合には効率的な最適化が行えるとは限らない。そこで、そのような点を考慮した最適化処理の改良を行った。合成品質、処理速度のどちらも満足の行く結果が得られている。また２分決定グラフの処理についてもいくつかの改良を行い高速化をはかったのでここに述べる。

Evaluation and Improvement of Transduction Method

Yusuke MATSUNAGA Masahiro FUJITA

FUJITSU LABORATORIES LTD.

1015, Kamikodanka Nakahara-ku,
Kawasaki 211, JAPAN

Abstract　This paper introduces improvement of transduction method using ordered binary decision diagrams. The original transduction method was designed to handle NOR-gate networks, and can be easily extended to handle general circuits. In such a case, however, transduction method can't always optimize circuit efficiently because of its naive extension. We propose a new procedure called 'Boolean division', which is suitable for optimization of general circuits. Good results were taken with this procedure.

1　はじめに

筆者らは、2分決定グラフと呼ばれるデータ構造を論理関数の表現方法として利用した多段論理最適化手法の開発を行っている[8,9]。本稿ではこの最適化手法に関する改良点について述べる。改良点は2点あり、1つは最適化処理に関するもの、もう1つは2分決定グラフの処理に関するものである。

2　2分決定グラフを用いたtransduction法

本章では、まず多段論理最適化の概観について述べ、筆者らが用いたtransduction[6]、2分決定グラフ[7]についての説明を行う。

2.1　従来の多段論理最適化手法

CMOSゲートアレイやスタンダードセル等を用いた組み合わせ多段論理回路を自動合成する手法の多くがweak division[1]と呼ばれる手法を元にしている。これは論理式中に繰り返し現われる部分論理を括り出したり（factorization）、中間ノードに置き換える（decomposition）ことによって、回路面積（ゲート数）の減少をはかる手法である。この手法の欠点は、論理式（ブール式）を一般の代数式として扱うために実際の論理をうまく考慮できない場合があるということである。例えば、$f = ad + bcd + e$という論理式を$g = a + b$で、代数式として割ることはできないが、論理式としては$f = g(ad + cd) + e$と表すことが可能である[2]（厳密な意味ではブール代数に除算(division)は存在しないが、このような処理をBoolean divisionと呼ぶ）。このように論理に対する考慮の欠如によりweak divisionによって生成された多段回路にはしばしば冗長性が含まれる（weak divisionによって作られた回路はPRIMEかつIRREDUNDANTという意味においては局所的最適解ではある）。そこで、そのような欠点を補うための最適化手法がいくつか提案されている[2,3][5][6]。MIS[2,3]は2段論理式の簡単化プログラムESPRESSOをベースにした簡単化処理を行っているが、多段論理回路中のドントケア条件の計算および表現に莫大なCPU時間とメモリ量を必要とするために、適当なフィルターを使ってドントケア条件を小さくしている[4]。BOLD[5]もMISに似たアプローチをとるが、ドントケア条件を明示的には計算せずに、2つの回路の等価性検証をもとに回路変換を行う。この場合はメモリはそれほどではないがやはり莫大なCPU時間を必要とする。transduction[6]はドントケア条件を許容関数によって表現し、この許容関数に基づいた回路変換を行うが、許容関数を真理値表の形で表現するため、常に入力変数の指数に比例したメモリが必要とされる。

このように多段回路内部のドントケア条件を利用した最適化手法の最大の問題点は、いかにドントケア条件を表現するか、である。MISの例をみてもわかるように2段論理式を扱うためには適切かつ効率のよいデータ構造であったカヴァー表現も、ドントケア条件を表現するには適切ではない。そこで、我々はBryantの提案した順序付き2分決定グラフ（ordered binary decision diagrams: OBDDと略す）[7]と呼ばれるデータ構造を用いた最適化手法を開発した[8,9]。この手法は前述のtransductionをもとにしており、許容関数をOBDDを用いて表現することにより、CPU時間・メモリ量の削減をはかったものである。ベンチマークを用いた実験結果から、その最適化能力・処理速度は他の手法と比較して同程度かそれ以上であることが報告されている[8,9]。

2.2　transduction法と許容関数

transductionとは、transformation & reductionの略で、回路の変形（transformation）・冗長部分の削除（reduction）を繰り返し行う回路の最適化手法であり、1970年代にイリノイ大の室賀教授によって提案されたものである。ここでは、簡単に許容関数の説明のみを行う。transductionの最適化処理については文献[6,8,9]等を参照されたい。

まず、用語の定義を行う。

Boolean Network[2]：　テクノロジ非依存の多段回路を表すDAG(directed acyclic graph)をBoolean networkと呼ぶ。グラフのノードは1）入力ピン、2）出力ピン、3）中間変数に対応する。中間変数のノードは実際の回路においてはNAND,NOR等の単一ゲートやAND-OR-INVERTERの様な複合セル、さらには1出力のPLAに対応している。ノードv_iからノードv_jへの有向枝e_{ij}は、ノードv_jの論理式中にv_iが現われていることを意味する。図1に簡単なBoolean networkの例を示す[3]。

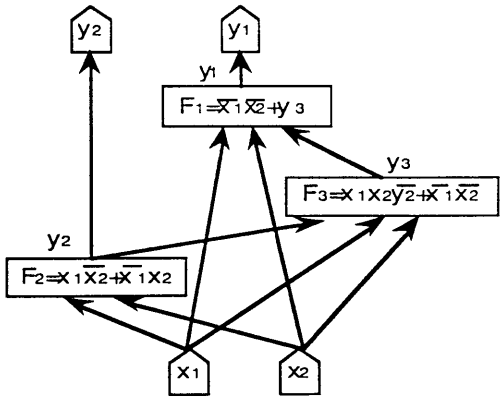

図1：Boolean network

許容関数（permissible function）[5]： 回路中のノードv_iにおいて実現されている論理関数f_iを他の論理f'_iに置き換えても、回路のどの出力の論理関数も変化しない時、このf'_iをノードv_iに対する許容関数と呼ぶ。ここで、論理関数と論理式の違いに注意されたい。ノードv_iの論理式はノードv_iの論理関数を、直接のファンインであるノードによって表現（sum of product formやfactored form）したものである。

図２に許容関数の例を示す。ここでは論理関数をベクタの形で表現している。各ベクタは$2^{(入力数)}$の要素を持つ。各要素がそれぞれの入力パタンに対応している。今、v_3の論理f_3をf'_3のように変えても、回路の出力であるv_5の論理は変化しない。そこで、論理関数f'_3はv_3における許容関数である。

通常、１つのノードに対する許容関数は複数存在する。そこで、許容関数の集合を表すためにドントケアを意味する＊を用いる。例えば、[0,1,1,1]と[0,1,1,0]の２つの許容関数をまとめて[0,1,1,*]で表す。この表記によって、許容関数の集合は１つの論理関数（ただしincompletely specified function）として表現できる。

transduction法はこのような許容関数を計算し、回路の各ノードにおいて実現されている論理関数がこの許容関数に含まれる範囲で回路変換を行うことによって回路の最適化を行うものである。回路変換の処理としては、

1　冗長なゲート、コネクションの除去
2　ゲートの併合
3　コネクションの追加削除

等がある。前述のようにtransductionはNORゲート用に開発されたものであるので、これらの処理も当然、ゲート単位で行われる。また、筆者らの行った拡張[8,9]もAND,OR,NAND,NORの単一ゲートを対象にしたものであり、回路変換処理はゲートを基本としたものになっている。図３にゲートの併合の例を示す。

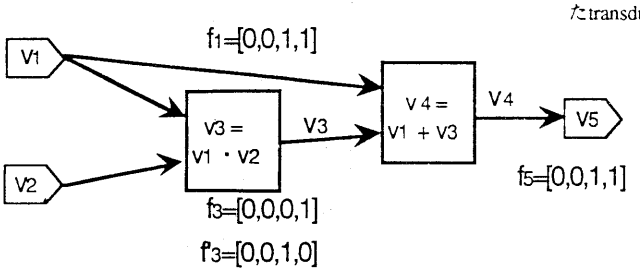

図２：許容関数の例

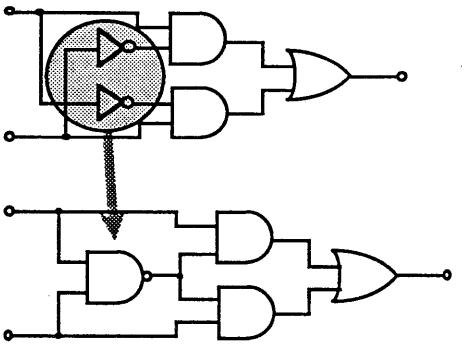

図３：ゲートの併合処理

2.3　順序付き２分決定グラフを用いた論理表現

許容関数によるドントケアの表現は、他の方法で表現できるドントケアを全てカバーしており、そのため強力な最適化処理を行えるものであるが、論理関数・許容関数をベクタの形で保持しているため、計算時間とメモリ量が入力変数の個数に対して指数的に増加し、大規模な回路へ適用することが難しい。この問題を解決するために筆者らは、論理関数を順序付き２分決定グラフ（Ordered Binary Decision Diagram: OBDDと略す）と呼ばれるデータ構造で表現してtransduction法の実装を行った。

OBDDは1986年にCMUのBryant教授が提案した論理の表現方法[7]で、通常の２分決定木のノードの現れる順序を固定し、共通な部分木を縮退させた有効グラフである。OBDDの主な特徴は、同一の論理から得られるグラフは常に同形のグラフとなる（いわゆる正規性）性質と、真理値表の形式や積和形表現ではメモリ量が爆発してしまう様な論理関数もOBDDを用いれば比較的コンパクトに表現できるという点である（もちろん最悪ケースでは指数的に爆発する）。

OBDDのもう一つの特徴は種々の論理演算が比較的簡単に実装できることである。そのため、ドントケアを表す終端節点を新たに導入して、許容関数に関する幾つかの論理演算を実装することにより、２分決定グラフを用いたtransduction法の実装は容易に行える。

3　transduction法の改良

最適化能力と処理速度の向上を目的としてtransduction法に関する以下の２点の改良を行った。１点は、最適化能力に関するもので、Boolean divisionと呼ばれる処理の追加である。この処理は従来のtransductionの処理とは異なり、一般のBoolean networkのノードを対象にし

たもので、既存の多段回路よりはweak divisionによって2段論理式から自動合成された回路に対して有効な処理である。2点めは、許容関数に対するフィルタの導入である。これはMISと同様にドントケア条件の範囲を絞ることによって処理の高速化をはかったものである。

3.1 Boolean division

weak divisionはブール式を代数式として扱うためにしばしば割れないで論理式が残ってしまうことがある。この様な論理式をブール式として割ってしまおう、というのがこの処理の目的である。Boolean divisionはMISやBOLDで用いられている回路変換処理で、一般の積和形論理式で表現されるノードからなるBoolean networkを対象としたものであり、従来のtransduction法の回路変換処理とは大きく異なっているが、大まかに言えばコネクションの追加／削除を基本としている（MIS[2]で述べられているアルゴリズムとは若干異なる）。処理の流れを図4に示す。divisionの対象となる論理式をx、divisiorの変数をyとする。

例を用いて説明を行う。x = abe'f' + cde'f' + abc'd' + c'd'ef + a'b'cd + a'b'ef、y = ab + cd + efとする。

1行目：最初の積項としてabe'f'を選ぶ。

2行目：y = ab + cd + efを追加しても論理は変わらない。

4行目：リテラルabを削除、e'f'yとする。

1行目：cde'f'を選ぶ。

2行目、4行目：yを追加、cdを削除、e'f'yとなる。

1行目、2行目、4行目：abc'd'を選択、yを追加し、abを削除、c'd'yとなる。

1行目、2行目、4行目：c'd'efを選択、yを追加し、efを削除、c'd'yとなる。

1行目、2行目、4行目：a'b'cdを選択、yを追加し、cdを削除、a'b'yとなる。

1行目、2行目、4行目：a'b'efを選択、yを追加し、efを削除、a'b'yとなる。

8行目：x = e'f'y + e'f'y + c'd'y + c'd'y + a'b'y + a'b'yなので、冗長な積項を削除し、

x = e'f'y + c'd'y + a'b'yを得る。

2行目の新たなリテラルの追加はESPRESSOで言うところのREDUCEの変形であり（REDUCEはもともとの論理式中に存在するリテラルの追加を行う）、4行目の削除はEXPANDそのものである。REDUCEを行うとその積項によってカヴァーされるmintermの数は減りこそすれ増えることはないので、もし注目している積項がEXPANDされなければ、他の積項のカヴァーしなければならない領域が増え、後の処理の自由度を奪うので、そのような追加はキャンセルする（5行目）。このような追加／削除の結果、論理式がPRIMEかつIRREDUNDANTでなくなる場合があるので、8行目の処理で冗長なリテラル／積項を取り除く。

以上のようにBoolean divisionを行う。問題はどのように追加／削除を行うか、である。MISではESPRESSOを利用して上記の処理を行っているが、transductionで用いられている許容関数でそのような処理を行うことも可能である。以下に許容関数を用いた追加／削除を述べる。

・リテラルの追加のチェック

ノードv_iの論理式中の積項cに対して新たなリテラルy = v_jを加えるものとする。v_iの許容関数の集合をPF、そのON-setをPF^{on}とする。v_iからcを取り除いた時のノードの論理関数をF'、そのON-set をF'^{on}とすると、cがカヴァーすべき領域は$PF^{on} - F'^{on}$である。ノードv_jの論理関数をG、そのOFF-setをG^{off}とすると、y = v_jを追加することによりG^{off}に含まれる領域がカヴァーされなくなるので、追加可能条件は、

$(PF^{on} - F'^{on}) \cap G^{off} = \phi$

となる。

・リテラルの削除のチェック

ノードv_iの積項cからリテラルxを取り除くものとする。v_iの許容関数の集合をPF、そのOFF-setをPF^{off}とし、cからxを取り除いた時のcの実現している論理関数をH、そのON-setをH^{on}とすれば、削除可能な条件は、

$H^{on} \cap PF^{off} = \phi$

となる。他の積項の論理を考慮する必要はない。

```
1 foreach cube c in x {
2  try to connect y or y' to c;
3   if successful {
4    try to delete other literals in c;
5     if not succesful { undo this addition;}
6  }
7 }
8 make x PRIME & IRREDUNDANT;
```

図4 ：Boolean division処理

3.2 許容関数のフィルタ

図5のようなBoolean networkを考える。ノードhの論理はaとbによって表せるが、許容関数はその他に、c,d,e,f,gの値に影響を受ける。ノードi,jの値が1とならないとhの値がマスクされてしまうからである。このように、論理関数は表現できても許容関数が容易には表現できない例が数多く存在する。そこで、多少の見落としがあっても処理が高速に行えるように、許容関数の集合を小さく抑えるためのフィルタを導入した。元々、許容関数の集合にはmaximum set of permissible function(MSPF)とcompatible set of permissible function(CSPF)の2種類があり、CSPFはMSPFの部分集合となっている。しかし、CSPFでも大きな回路に対しては適切とは言えないので、さらに、

(1) 自分の直接のファンアウト先のノードを出力と見做して許容関数を計算する。

(2) 積項に対する許容関数を求める際に他の積項の論理を参照せず、自分自身の値とノードの許容関数からのみ計算する。例えば、自分自身が0で、ノードの許容関数が1の時（必ず他の積項が1であるので）、＊となる、また、自分自身が0で、ノードの許容関数が0の時は、0、自分自身が1の時は1とする。

という制限を設けている。(1)の制限によって2段以上先のノードで値がマスクされて伝わらないようなドントケア条件を見逃すことになるが、その分許容関数の集合は小さくなる。(2)の制限によって、他の積項が1であり自分自身の許容関数は＊でよいのに1となる場合が生ずる（図6）が、計算の手間は著しく軽減される。

4 2分決定グラフに関する改良

Bryantの提案した2分決定グラフとその演算アルゴリズム[7]は効率的であるが、transductionを高速に行うためには

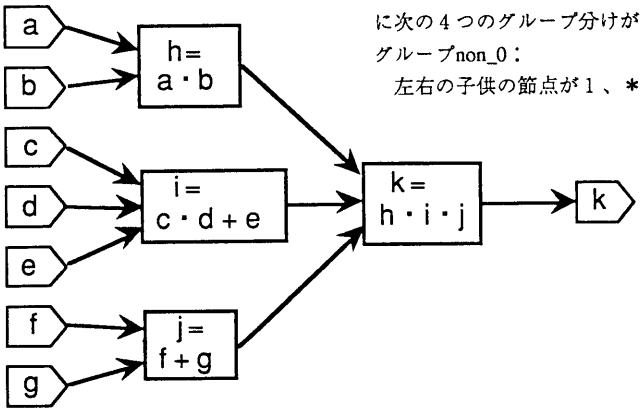

図5：許容関数が複雑になる例

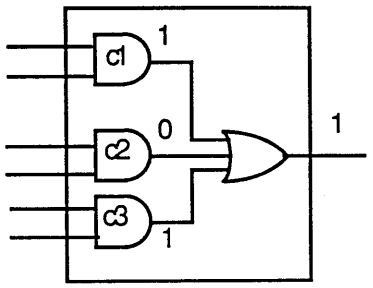

図6：(2)の制限による冗長性の見落とし
（c1、c3の両方が1である必要はない）

さらにいくつかの改良が必要であるので、以下に述べる。尚、これらの変更は純粋に処理速度のみに関係しており、回路変換処理自体には全く影響せず、同一の最適化が行えるものである。

4.1 肯定／否定の同時判定

新たな変数の追加のチェックのような処理では、肯定のリテラルと否定のリテラルの両方を試す場合が多い。この場合、その変数の論理のON-setを用いるか、OFF-setを用いるかという点が異なるだけであとは全く同じ処理となる。2分決定グラフの場合、ON-setとOFF-setは全く同型のグラフを用いて表現可能であるので、1回の演算で両方の判定を行うようになっている（後述の否定枝を使うとより効率的となる）。

4.2 非終端節点のグループ化

オリジナルの2分決定グラフでは終端値は0、1の2値であり、非終端節点の値はXとなっていたが、許容関数を表すために＊の終端節点を加えた結果、非終端節点に次の4つのグループ分けがなされるようになった。

グループnon_0：

左右の子供の節点が1、＊の終端節点もしくは、グル

ープnon_0の非終端節点のみの節点。要するに、子供をたどっても０の終端節点にはたどり着かない節点。

グループnon_1：

左右の子供の節点が０、＊の終端節点もしくは、グループnon_1の非終端節点のみの節点。

グループnon_d：

左右の子供の節点が０、１の終端節点もしくは、グループnon_dの非終端節点のみの節点。

グループX：

上記の３つのグループに属さない非終端節点。

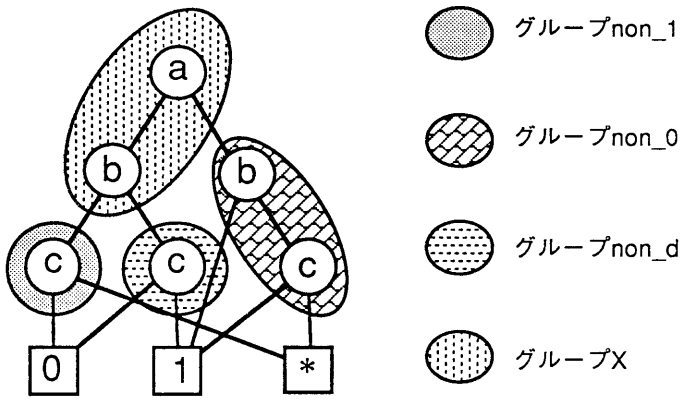

図７：非終端節点のグループ分け

図７にグループ分けの例を示す。このグループ分けが効果的なのは、例えばある許容関数のOFF-setのみを考えた時である。この場合には許容関数の値が１であろうが＊であろうが関係なく、ただ０と区別できればよい、すなわち、グループnon_0の非終端節点をそのままnon_0という値を持った終端節点と見なしてしまって構わないので、見かけ上のグラフの大きさを縮退できる。さらに、節点のグループは自分の子供の節点より決定されるので、グラフ生成時にリニアな時間でグループ分けを行うことができる。

4.3　複数グラフの共有と否定枝の導入

この２点は、transductionに特別なものではなく、文献[10]等で提案されている手法で、グラフの総節点数の節約と処理の高速化に効果がある（図８）。複数グラフを共有するには、全部のグラフの節点を一つの管理用テーブルに登録しておき、新たな節点を作る際に既に存在する節点がないかを調べることにより行える。また、否定枝はその下の部分グラフの論理を反転させるためのもので、任意の場所に挿入するとグラフの正規性が失われるので、制限が必要となる。ここではドントケアを表す終端節点＊が加わっているので、次のような規則を設けている。

(1) 終端節点は１、＊のみとする。０の値を表すときは１の節点に否定枝を付ける。

(2) 右の部分グラフ（変数を１に固定したときの論理を表す）には否定枝を付けない。

(3) いかなる場合も＊の終端節点への枝に否定は付けない。

否定枝の挿入はグラフ作成時にオンラインに行うことができる。

5　実験結果

5.1　Boolean divisionによる最適化

新しく実装したBoolean divisionの効果を見るためにベンチマーク回路を用いて実験を行った。例題はMCNC'89の論理合成のワークショップの２段論理式で与えられているものを用いた。比較のため、

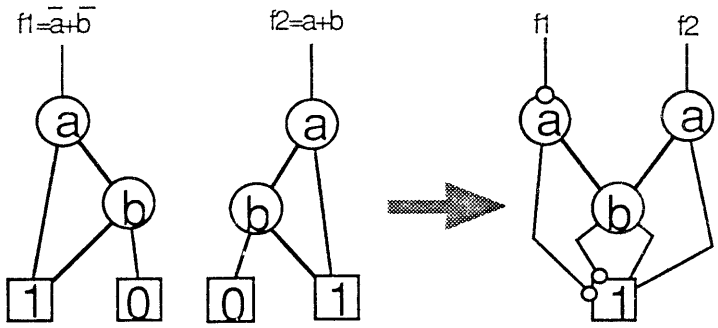

図８：グラフの共有と否定枝の導入

(1)方式1

・2段論理式を各出力論理ごとに独立に簡単化を行う。

・weak divisionを行い多段論理式（Boolean network）に変換する。

・従来のtransduction法の回路変換処理（ゲートの併合・コネクションの追加／削除の繰り返し）を用いて最適化を行う。

(2)方式2

・2段論理式を各出力論理ごとに独立に簡単化を行う。

・評価値（リテラル数のメリット）10以上の論理式でweak divisionを行う。

・各中間ノードをdivisorとして許容関数を用いたBoolean divisionを行う。

・評価値5以上の論理式でweak divisionを行う。

・各中間ノードをdivisorとして許容関数を用いたBoolean divisionを行う。

・メリットの制限を設けずにweak divisionを行う。

・各中間ノードをdivisorとして許容関数を用いたBoolean divisionを行う。

の2つの方法で多段論理式（Boolean network）の合成を行った。どちらも改良された2分決定グラフの処理を用いている。方式2で用いられる許容関数の計算時には、前述のフィルタを用いている。実験結果を表1に示す。合成結果は多段のBoolean networkとなるので、各ノードのfactored form形式のリテラル数の和を記した（factored formのリテラル数に関しては文献[2]を参照のこと）。使用計算機はSUN4/260、CPU時間の単位は秒である。右の2列はMIS2.1の同じベンチマーク例題を用いた結果[11]である（使用計算機はDEC3100）。

まず、従来の方式1と新しい方式2とを比べてみると、方式2で合成された結果のほうが大部分の例題に対して小さくなっており、全ての例題に対して高速に（約2～5倍）処理を行っている。これは、Boolean divisionが、weak divisionではうまく扱えない部分を効率的に最適化していることと、許容関数に対するフィルタが最適化能力を落とすことなしにうまく働いていることを示している。また、MIS2.1の結果と比べても、処理速度が遅い点（使用計算機のCPUパワーはほぼ同程度）はあるものの、合成品質はかなり良いものとなっている。このことから、カヴァー形式によるドントケア条件の表現とESPRESSOの組み合わせよりも、2分決定グラフを用いた許容関数の表現が多段論理式の最適化には有効であると言えよう。

表1：2段論理式からの合成結果

	方式1		方式2		MIS2.1	
回路名	リテラル数	CPU時間	リテラル数	CPU時間	リテラル数	CPU時間
5xp1	101	28	85	19	114	45
9sym	208	738	179	134	229	169
alu4	139	3948	129	1762	263	359
bw	168	32	144	23	163	33
clip	100	76	100	44	156	35
con1	18	1	18	0	19	0
duke2	338	703	339	335	442	119
e64	254	2529	254	263	253	3167
misex1	55	4	56	3	49	3
misex2	103	20	99	16	106	6
misex3	467	11339	491	4374	553	1424
misex3c	421	7740	405	1384	452	539
o64	130	168	130	23	130	480
rd53	34	4	37	2	38	3
rd73	85	48	67	15	90	22
rd84	126	148	108	62	148	114
sao2	130	115	128	36	122	74
vg2	87	28	79	15	86	25
xor5	16	1	16	0	16	1
合計	2980	27669	2864	8511	3429	6618

＊1　リテラル数はfactored form形式

＊2　使用計算機はSUN4/260（MIS2.1はDEC3100）
CPU時間の単位は秒

5.2　2分決定グラフの処理の改良

2分決定グラフの処理の違いによる計算時間の比を表2に示す。表中、A、B、C、Dの各欄は

A: 従来のもの

B: Aに肯定／否定の同時判定を加えたもの

C: Bに非終端節点のグループ分けを加えたもの

D: Cに複数グラフの共有と否定枝を加えたもの

の処理時間（2分決定グラフに関する処理ではなく、最適化処理全体の時間）の比をAを100として表したものである。個々の例題によって違いはあるが、各々の改良により1割～2割の高速化がなされ、全体で約半分の時間で処理が行えるようになっている。この結果から、この改良が2分決定グラフの処理の高速化に大きく貢献していると共に、最適化処理のなかで2分決定グラフの処理が大部分を占めていることがわかる。

6　終わりに

本稿では、筆者らの開発した多段論理回路最適化手法についての、最適化能力・処理速度に関する改良を述べた。新しい回路変換処理として実装されたBoolean divisionは従来のゲートの併合・コネクションの追加／削除の繰り返しによる回路変換処理と比べて極めて高速に、かつ、遜色のない最適化を行っている。また、2分決定グラフの処理に関する改良により、最適化処理には全く影響なく約2倍程度の高速化を行うことができた。処理速度、合成品質ともに実用的なものであるといえる。

表2：2分決定グラフの処理の高速化

	処理時間比（%）			
回路名	A	B	C	D
5xp1	100	83	77	43
misex1	100	81	77	38
misex2	100	83	67	36
rd53	100	91	68	49
rd73	100	84	67	48
rd84	100	97	66	52

参考文献

[1] R. K. Brayton, C. McMullen, "The Decomposition and Factorization of Boolean Expressions", Proceedings of the International Symposium on Circuits and Systems, Rome, 1982, pp. 49-54.

[2] R. Brayton, E. Detjens, S. Krishna, T. Ma, P. McGeer, L. Pei, N. Phillips, R. Rudell, R. Segal, A. Wang, R. Yung, A. Sangiovanni-Vincentelli, "Multiple-Level Logic Optimization System", IEEE International Conference on Computer Aided Design, November 1986, pp. 356-359.

[3] K. A. Bartlett, R. K. Brayton, G. D. Hachtel, R. M. Jacoby, R. Rudel, A. Sangiovanni-Vincentelli, A. Wang, "Mulitilevel Logic Minimization Using Implicit Don't Cares", IEEE Trans. on CAD, June, 1988, pp. 723-740.

[4] A. Saldanha, A. R. Wang, R. K. Brayton, "Mulit-Level Logic Simplification using Don't Cares and Filters", proc. of 26th Design Automation Conference, pp. 277-282, 1989.

[5] D. Bostick, G. D. Hachtel, R. Jacoby, M. R. Lightner, P. Moceyunas, C. R. Morrison, D. Ravenscroft, "The Boulder Optimal Logic Design System", IEEE International Conference on Computer Aided Design, November 1987, pp. 62-65.

[6] S. Muroga, Y. Kambayashi, H. C. Lai, J. N. Culliney, "The Transduction Method - Design of Logic Networks based on Permissible Functions", to apper IEEE Trans. on Computers, in 1989.

[7] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation", IEEE Trans. on Computers, August, 1986, pp.677-691.

[8] Y. Matsunaga, M. Fujita, "Multi-level Logic Optimization usingBinary Decision Diagrams", IEEE International Conference on Computer Aided Design, November 1989

[9] 松永　裕介、藤田　昌宏、”2分決定グラフを用いた多段論理回路の最適化手法”、信学技報　Vol. 89, No. 92, CAS89-10, pp. 19-26(Jun. 1989).

[10] 湊　真一、石浦　菜岐佐、矢島　修三、”共有二分決定図を用いた論理関数の処理手法について”、情処38回全国大会、5S-9(Mar. 1989).

[11] 、"MIS2.1 Logic Optimization Benchmark Results", MCNC International Workshop on Logic Synthesis, Research Triangle Park, North Carolina, May 1989.

盛光印刷所