設 計 自 動 化 60−16
（１９９１．１２．１３）

最大ＣＳＰＦを用いた大規模組み合わせ回路の最適化手法

藤本 徹哉、本庄 浩、神戸 尚志
シャープ株式会社

あらまし　最大ＣＳＰＦを用いて大規模組み合わせ回路の最適化を行なう手法を示す。本手法はノードのＣＳＰＦをそのノードの論理変換にのみ用い、同時に論理変換の候補を探索する空間を限定することにより、ＣＳＰＦを用いた手法が大きな記憶領域と長い計算時間を要するという問題を解決する。本手法が十分に実用的なメモリと計算時間で１０００～２０００ゲートの回路を最適化し得るものであり、よい解を与えることを実験的に示す。

Large Network Optimization Using Maximal CSPF

Tetsuya FUJIMOTO, Hiroshi HONJO and Takashi KAMBE

SHARP Corporation
632, Nara, JAPAN

Abstract　A Logic optimization method for large combinational circuit is described. The transduction method is basically used and maximal CSPF is used for representing implicit don't cares. We cope with the major problems in such approarch, huge memory requirements and long computational time, by introducing restricted network transformation, in which CSPF of a node is only used for the transformation on the node. The experimental results show that the presented method handles large circuits with up to 2000 gates in a reasonable CPU time and memory, and gives smaller area cost than that by conventional methods.

1 はじめに

論理回路の自動設計技術の普及はまず８０年代後半から組み合わせ回路の合成の分野で大きく前進した[3]。さらに９０年代に入ってレジスタ転送レベル（ＲＴレベル）での設計がＶＨＤＬやＵＤＩ／Ｉなどのハードウェア記述言語の確立と共に実用化されつつある[11]。

ＲＴレベルの設計が一般的になると、組み合わせ回路の合成にも大きな影響が及ぶと見られる。すなわち、これまでの人手による論理式や真理値表を用いた設計に比べて、より大規模な組み合わせ論理回路が設計され、従って大規模回路の最適化が重要な課題となってくる。最適化の程度と計算コストのトレードオフにも再考の必要が生じると思われる。

このような背景から、我々は大規模回路に対する最適化手法の確立が急がれると考え、許容関数集合の概念を用いた最適化を行なって品質のよい論理回路の合成を試みた。

代数的手法[1, 2]は計算機資源を多く消費せずある程度までの大規模回路に適用可能であることが知られていたが、これに対しブール的手法は論理関数の表現のために多量の記憶領域を必要とすることから大規模回路に不向きであると考えられていた。

しかし、共有二分決定グラフ（ＳＢＤＤ）[8]の利用によって、記憶領域の問題が大きく前進した。個々の論理関数の表現自体がコンパクトになることのみならず、プライマリ出力を表現したＢＤＤが、論理回路の内部で実現される論理関数を表現したＢＤＤとの間で多くのノードを共有する結果、必要となるＳＢＤＤ全体の大きさは個々の関数に必要な記憶領域の総和より削減できるからである。

しかし、ブール的手法で最適化を行なうためにはＳＢＤＤだけではなお不十分である。回路内部の論理関数以外に内部ドントケアを表現する必要があり、かつ、一般的に内部ドントケアの表現は論理関数以上に複雑で、そのためより大きな記憶領域を必要とするからである。

従って、これまでのアプローチでは「ドントケアフィルタ」と呼ばれる概念を導入し、複雑な表現を持つ最大の内部ドントケアのかわりに、より簡略な表現を持つその部分集合を用いることによってこの問題に取り組んできた[5]。その結果、ブール的手法は高い簡単化能力を持ち、かつ計算時間に関しても実用性を持つものであることが示され始めた[6, 12]。しかし、これらの手法では、内部ドントケアをより小さな部分集合で近似するため回路変換と簡単化の自由度が減少し、解の品質が低下する危険がある。

そこで、解の品質を一層高めるため、本論文で提案する手法では、最大の内部ドントケアを用い、記憶領域の問題を別な角度から解決する。最大の内部ドントケアとは、文献[5]でmaximal CSPFと呼ばれるものである。最適化手法としてトランスダクション法[9]の一つを適用する。論理関数の表現には前記のＳＢＤＤ[8]を用いる。

本論文の手法では、あるノードに注目して論理変換を行なう際には、そのノード以外のＣＳＰＦを用いない。逆にいえばノードのＣＳＰＦは、それを得る際に可能な論理変換と冗長除去のみに用いる。これによって、同時にメモリ中に保持せねばならないＣＳＰＦをなるべく少なくすることで必要な記憶領域を小さくする。これは回路変換の自由度をある程度奪ってしまうが、この制限が解の品質にほとんど影響しないことを実験的に示す。

トランスダクション法は、必要とする記憶領域の大きさとともに計算時間も大きいという問題を持っている。同法では論理変換の基本戦略として、回路の出力を不変に保ちながら新たな接続の追加と冗長な接続の削除を繰り返すが、このとき、接続可能な関数をほとんど総当たりで探索する。また、接続可能性を判定するためには論理関数やＣＳＰＦの間で論理演算が必要である。従って、回路が大規模化すると総当たりの探索領域の増加と論理関数の複雑化による論理演算コストの増加の２要因の積によって、莫大な計算時間が必要になってしまう。この問題については、解に全く影響を与えない探索領域の削減手法を用いて、接続可能性判定回数を削減し、計算時間の短縮をはかる手法について述べる。

提案する手法を実装したプログラムをＩＳＣＡＳ’８５とＩＷＬＳ’８９に含まれる大規模回路に適用した実験によって、（１）最大ＣＳＰＦを用いた最適化が可能であり、かつ実用性を持つこと（２）本手法がこれまで提案されてきた手法に較べ、同等以上の解を与えること（３）ＲＴ合成システムの組合せ回路部の合成に大きく寄与すること、などが示される。

2 用語と記号

組み合わせ論理回路のプライマリ入力数をNとする。論理回路を非巡回有向グラフ（ＤＡＧ）で表し、各論理ゲート、プライマリ入出力をノードnで表す。n_iの出力からn_jの入力への接続があるとき、それをエッジe_{ij}で表す。

n_iからn_jへの接続があるとき、n_iをn_jのファンイン、n_jをn_iのファンアウトと呼ぶ。n_jのファンイン、ファンアウトの集合をそれぞれFI_j、FO_jと表

す。n_j へ、または n_j から到達可能なノードの集合をそれぞれ TFI_j、TFO_j と表す。

ノード n_j 上で実現される論理関数を f_j、許容関数集合（ＣＳＰＦ）を $G_j = (G_j^{on}, G_j^{dc}, G_j^{off})$ とかく。論理関数 f が G_j の要素ならば、すなわち

$$f \supseteq G_j^{on} \tag{1}$$

$$\overline{f} \supseteq G_j^{off} \tag{2}$$

が満たされていれば、f_j を f と交換してもプライマリ出力上の論理関数は不変である。エッジ e_{ij} 上のＣＳＰＦを同様に G_{ij} と書く。

3 トランスダクション法の概要

次に本論文で用いるトランスダクション法について簡単に説明する。これは文献[9]に示されているいくつかの方法のうち connectable/disconnectable と呼ばれるものとほぼ同一で、接続可能条件を利用するものである。以下では特に断らない限りノードの論理はＮＯＲであるとする。ノード n_i が n_j に接続可能であるとは、次の関係を満たすことである。

$$G_j^{on} \subseteq \overline{f_i} \tag{3}$$

$$n_i \notin TFO_j \tag{4}$$

式（3）では、新たな接続 e_{ij} の追加によって変更された論理関数 f_j が G_j の要素であることを保証する。また、式（4）は閉ループをつくらないことを保証するためのものである。

さらに、接続可能なノード n_i が f_j のカバーに寄与することが必要である。すなわち、

$$G_j^{off} \cap f_i \neq \phi \tag{5}$$

式（5）を満たす関数ならば、e_{ij} を追加することによって n_j の元のファンインを冗長化できる可能性がある。逆に、これを満たさない関数を接続しても簡単化には無効である。

さらに、二つ以上の接続可能ノード n_i、n_h に対し以下の関係

$$f_i \supseteq f_h \tag{6}$$

がなりたつとき、われわれは n_h を接続無効とする。f_j のカバーに対する e_{hj} の寄与が e_{ij} のそれに包含されているからである。

ノード n_j に対するトランスダクションの概略（以下手続き１と呼ぶ）を次に示す。ここで、FI_j は手続き１を適用する以前の n_j のファンインの集合とする。

step 1 ノード n_j の許容関数を計算する。
$G_j^{dc} = \cap G_{jk}^{dc}$
ここで $n_k \in FO_j$ である。

step 2 ノード n_j に接続可能かつ無効でないノードの集合 C_j を見つける。

step 3 n_j のファンインのうちの幾つか（$R_j \subset FI_j$）からのエッジを切断する。

step 4 ファンインの切断によって失われた f_j のカバーを、いくつかの接続可能ノード（$U_j \subset C_j$）によって補償する。ただし、U_j が存在するとは限らない。

step 5 U_j が存在すれば U_j からのエッジを n_j に接続する。存在しなければ元の接続に戻す。

step 6 FI_j の全ての要素に対して step 3 ～ step 5 を適用する。

step 7 冗長なファンインがあれば切断する。更新された各ファンイン n_i に対して G_{ij} を計算する。

さらに、ネットワーク全体の最適化を行なう手続き（以下手続き２とよぶ）は次のようなものである。

step 1 プライマリ出力にＣＳＰＦを与える。これは外部ドントケアとして論理回路の仕様から決定されるものである。

step 2 ネットワーク中の全ノードをスケジューリングする。各ノードは全てのファンアウトより後、全てのファンインより前になければならない。

step 3 手続き１を step 2 で得られた順序にしたがって適用する。

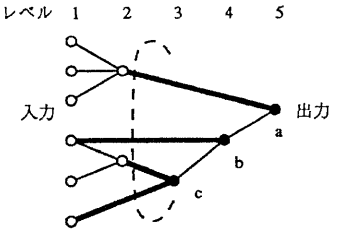
トランスダクションの全体は手続き２をコストの改善が得られなくなるまで繰り返し適用するものである。

4 記憶領域と計算時間の削減

4.1節では、回路変換時にＣＳＰＦの利用を制限することによる記憶領域の削減手法を、4.2節では接続可能ノードの探索領域を削減して計算時間の短縮をはかる手法を、それぞれ述べる。

4.1 回路変換の限定

トランスダクション法が多量の記憶領域を消費する最大の理由はＣＳＰＦを表現するためである。しかし、手続き１で、もし「ノード n_j のＣＳＰＦ G_j を、n_j に対して適用される手続き１の中でしか用いない」と仮定すると、これをメモリ中に保持するのは手続き１の中のみでよい。また、エッジのＣＳＰＦ G_{ij} についても、エッジのソース側のＣＳＰＦ G_i を計算した後は不要となる。従って、図1に示すように保持しなくてはならないＣＳＰＦは、ネットワーク中で、ＣＳ



○: ＣＳＰＦ未計算 ●: ＣＳＰＦ計算済み

図 1: エッジ上ＣＳＰＦの保持（ＡＬＡＰ）

ＰＦの計算を終えたノードの集合とまだ終えていないノードの集合を分割するカットセットに含まれるエッジ（太線で示す）のＣＳＰＦのみでよい。この仮定が最適化アルゴリズムの中で強い制約とならなければ、保持すべきＣＳＰＦの数を小さくでき、必要な記憶領域を削減することができる。

そこで、本仮定を用いたときの解の品質低下について考察する。手続き１の中では、他のノードのＣＳＰＦを必要とするのは step 2 のみである。すなわち、すでに手続き１の適用を受けたノード n_i を C_j に加える場合、f_i の値がドントケア部分で保証されていないため、接続可能条件として式（３）の代わりに

$$G_j^{on} \subseteq G_i^{off} \tag{7}$$

を用い、このノードの f_j に対するカバーへの寄与として式（５）の代わりに

$$G_i^{on} \cap G_j^{off} \neq \phi \tag{8}$$

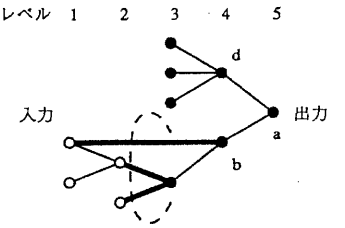
をとらなくてはならない。従って、ノードのＣＳＰＦを用いないことは接続可能ノードの探索範囲を限定することに等しく、手続き１が適用されたノードから順に回路変換の対象外となることがわかる。つまり仮定の導入はノードに対し手続き１を適用する順序を通じて最適化結果に影響を及ぼす。そこで次の二つの順序付けを考えよう。

（１）ＡＬＡＰ

計算順序をプライマリ入力側すなわち計算順序の後ろから決めていくことにより、ＣＳＰＦの計算をなるべく遅らせる。ネットワークを図1のようにプライマリ入力側からレベル付けし、出力側のレベルに属するノードから手続き１を適用する（図1ではａ、ｂ、ｃの順）。この順序付けによると、step5 で新たに導入される接続によって既存の最長パスより長いパスが生じない。従って、遅延時間最適化に近い回路変換が行なわれる。いくつかの文献 [4, 13] では、面積と同時に遅延時間を最適化しても面積コストの犠牲は非常に小さいことが報告されていることから、仮定を導入することによる解への影響は小さいと考えられる。

（２）ＡＳＡＰ

（１）とは逆に、プライマリ出力側からレベル付けする（図２）。この場合、接続可能ノードの探索領域の削減がＡＬＡＰに較べてより入力に近い側で起こる（図2の例ではノードｄ以下の部分回路はすでに変換対象ではない）ため、、解に悪影響が及ぶ可能性があると考えられる。しかし、ノードはより出力に近いレベルに属する傾向を持ち、そのためプライマリ入力に近い側ではＣＳＰＦを保持せねばならないカットセットは小さくなる傾向を持つ。ＣＳＰＦの表現は入力側へ向かってより複雑になっていくので [10]、ＡＳＡＰによればＣＳＰＦを保持するために必要な記憶領域がＡＬＡＰに較べてさらに少なくなると考えられる。



○: ＣＳＰＦ未計算 ●: ＣＳＰＦ計算済み

図 2: プライマリ出力からのレベル付け

4.2 接続可能性判定の高速化

ノード n_j への接続可能ノードの集合を求めるときには、ループフリー条件式（４）を満たす各ノードに対して接続可能性と接続有効性すなわち式（３）と（５）を判定する。この過程はすでに述べたようにトランスダクション法の中で総当たりの部分である。しかし、接続可能、または不可能なことがわかっているノードがあれば、論理演算を行なうことなく、回路の局所的な構造を利用して接続不可能性、または接続無効性を持つ他のノードを見つけることができる。

n_j に接続可能なノードを探索する過程で、ノード n_i の接続可能性を判定したとする。n_i が接続不可能な場合と可能な場合に分けて、以下で論じる。

（１）まず、n_i が n_j に接続不可能な場合について、探索範囲の枝刈りができることを図3の例によって示す。接続可能条件を満たすためには n_j は f_i に対し

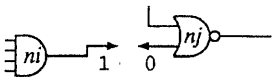


図 3: 接続不可能ノードの例（1）

G_j^{on} 上で0を要求するが、接続不可能な場合にはこの要求に対し1を与える最小項が存在する。従って、n_i がＡＮＤならば、そのファンインは全て接続不可能であることがわかる。また、図4のように、n_i のファンアウトにＯＲゲート n_k があれば、n_i の論理の種類によらず n_k は n_j に接続不可能である。さらに、回路構

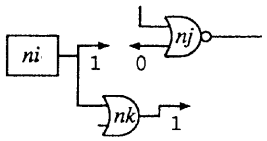


図 4: 接続不可能ノードの例（2）

造によっては図5における n_h のファンインのようにさらに遠くまで接続を辿って得られるものもある。

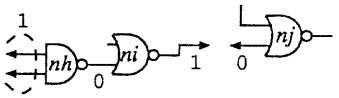


図 5: 接続不可能ノードの例（3）

（2）次に、n_i が n_j に接続可能な場合を考える。簡単のため n_j、n_i ともにＮＯＲゲートであるとしよう。このとき、関係式（3）～（5）が満たされている。従って、n_i のファンアウトゲート n_k に対し、次の関係が成り立つ。

$$G_j^{off} \subseteq f_k \quad (n_k : NAND\ or\ NOT) \qquad (9)$$

$$f_j \supseteq f_k \qquad (n_k : AND) \qquad (10)$$

すなわち、式（9）は n_k が n_j に接続可能ではないことを、式（10）は n_k は n_j に接続可能だが、カバーへの寄与が f_i の部分集合であるため接続が無効であることをそれぞれ示している。同様のことはノードの論理がこれらの組み合わせ以外の時にもいえる場合があり、n_j と $n_k \in FO_i$ の間に表1の関係が成り立つ。n_i のファンインについても同様のことがいえ、表2の関係がある。

	nj の論理	
nk の論理	NOR 又は OR	NAND 又は AND
NOT	接続不可能	接続不可能
NOR	不明	接続不可能
NAND	接続不可能	不明
OR	不明	接続無効
AND	接続無効	不明

表 1: 接続可能ノードのファンアウト接続可能性

	nj の論理	
ni の論理	NOR 又は OR	NAND 又は AND
NOT	接続不可能	接続不可能
NOR	不明	接続不可能
NAND	接続不可能	不明
OR	接続無効	不明
AND	不明	接続無効

表 2: 接続可能ノードのファンイン接続可能性

以上二つの関係による枝刈りは再帰的に適用することができる。アルゴリズムの総当たり部分に適用でき、かつ解に全く影響を与えないので回路構造によっては大きな効果があると考えられる。

5 実験結果

これまでに述べた手法をわれわれの論理合成プログラムに実装し、以下に示す一連の評価実験を行なった。なお、回路のコストはＭＣＮＣＩＩのライブラリを想定しノード数とエッジ数の和で評価した（マッピングは行なっていない）。表中の計算時間は全てＭＩＰＳ社の計算機（５０Ｍｉｐｓ）によるもので単位は秒である。ただし一部のデータでは後ろにｍを付して分であることを示した。

ノードのスケジューリングを行ない、同時にノードのＣＳＰＦを保持せずトランスダクションの適用範囲を限定することによる効果を示す。表3に、トランスダクション１回の適用に要した時間とメモリを、その結果得られたノード数、エッジ数と共に示す。ノードの許容関数を保持した場合に較べ、計算時間は約１／２～１／３、記憶領域は約１０～５０％削減できており、また、解の品質の劣化が小さいことが確かめられた。同時に、スケジューリング手法ＡＳＡＰとＡＬＡＰとの特長が現れていることがわかる。すなわち、ＡＳＡＰはＡＬＡＰに比べ、１～１５％程度高速であり、０～１０％少ない記憶領域しか要しない。しかし解の品質ではほぼ同等であり、予想されたＡＬＡＰの

network name	全探索				ALAP				ASAP			
	node	edge	cpu	memory	node	edge	cpu	memory	node	edge	cpu	memory
rot	498	1067	802	4.31	485	1070	265	2.63	500	1084	232	2.44
k2	836	2420	716	1.43	892	2684	333	1.31	901	2719	308	1.38
too_large	420	1081	1800	8.00	428	1161	636	5.46	425	1103	557	4.00
frg2	700	1656	516	2.25	708	1713	252	1.50	683	1608	249	1.38
vda	427	1242	149	0.63	439	1331	69.4	0.50	435	1323	68.1	0.44

表 3: スケジューリングの効果

優位性はこの実験では認められなかった。

この結果から、トランスダクション法ではノードのＣＳＰＦを保持する必要は小さく、また、一旦ＣＳＰＦの計算およびトランスダクションを適用したノードは以降の回路変換の対象から除外しても、大きく解を悪くすることがないといえる。

表4は、ＭＣＮＣ多レベル回路ベンチマークおよびＩＳＣＡＳ'８５ベンチマークに対し提案する手法を適用した結果である。我々の結果は、多レベルのベンチマーク回路を初期解とし、手続き２を改善がなくなるまで繰り返し適用して得られたものである。ＢＤＤノード数のピークは示されていない。論理関数やＣＳＰＦの表現に要するもの以外に、ファンイン交換時にカバー問題を解き、そのため多くのＢＤＤノードが消費されるため求めることができなかったためである。ＢＤＤ入力変数の順序付けについては一部のものを除き文献 [7] のものの逆順を使用した。ＡＳＡＰスケジューリングの結果については面積縮小効果を最適化適用前後の面積コスト（ノード数＋エッジ数）の比（Ｒ１）で示した。

接続ノード探索を限定することで、計算時間はほぼ半分になっているが、中には、収束速度が非常に遅くなってしまったため関係が逆転しているものもある（term1）。しかし、解の品質の低下は非常に小さく、数％以下である。回路によっては限定探索による方がよい結果を与えることがあるが、これは前述のようにこの限定手法がアルゴリズムに遅延時間最適化の性質を加えるためであり、これは、「遅延時間最適化が面積コストに悪影響を及ぼすとはかぎらない」という従来から経験的に知られている事実に一致する。

許容関数集合の部分集合（より小さな内部ドントケア）を用いて大規模回路を最適化した試み [6, 12] との比較を同表4の右端のコラムに示した。これらの結果はいずれもリテラル数で示されているので、最適化の効果を最適化前後のリテラル数の比（Ｒ２）で表した。

ただし、右端列のうち＊を付したものは「ドントケアフィルタ」を用いていない。我々の手法では、最大

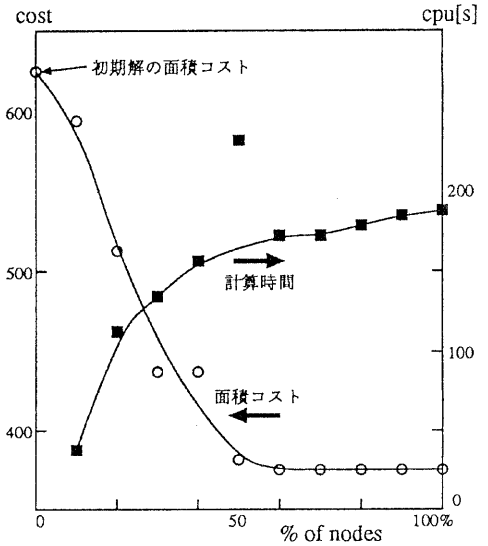


図 6: ＣＳＰＦの不完全伝搬

の内部ドントケアを用いているにもかかわらず [12] で簡単化できた全ての回路を取り扱えている。また、最適化能力についても多くの例で優れている。ただし、計算時間の面では１０～２０倍悪い場合もあり、今後の課題の一つである。また、二つの文献の結果が本論文の手法による結果と有意に差がない場合、逆にすぐれている場合がみられるが、その原因の一つにわれわれの実験ではテクノロジマッピングの工程が続くことを考慮してゲートのファンイン数を４に抑えているためであると考えられる。

表4に示した結果のうち、Ｃ８８０に対しては、ネットワーク中の全ノードに手続き１を適用することができなかった。プライマリ出力側から始めて簡単化を行なう途中でＢＤＤの記憶領域があふれ、計算を続行できなくなったため次の繰り返しループにはいって手続き２を適用している。このように、回路のプライマ

network		初期解		全探索			ASAP				initial	ref.[12]
name	in/out	node	edge	node	edge	cpu	node	edge	cpu	R1	literals	R2
C432	36/7	209	421	108	273	284	111	274	115	61.1	272	71.3*
C499	41/32	579	993	unable			491	1041	243m	97.5	568	100
C880	60/26	393	748	unable			339	741	245m	94.7	451	94.7
C1908	33/25	718	1335	405	920	560m	402	935	236m	65.1	833	68.2
C2670	233/140	1016	1893	unable			640	1355	250m	68.6	1182	72.2
C5315	178/123	1929	4006	unable			1358	3321	350m	78.8	2589	79.6
apex6	135/99	682	1382	601	1416	1311	612	1423	639	98.6	832	90.6*
apex7	49/37	199	456	161	372	33.8	161	373	27.8	81.5	352	79.3*
rot	135/107	563	1204	not tried			476	1045	1857	86.1	1443	82.7*
												ref.[6]
example2	85/66	261	546	226	512	53.7	222	511	75.4	90.8	568	79.4
frg2	143/139	961	2457	619	1488	2799	648	1602	885	65.8	2528	72.8
k2	45/43	1182	3885	not tried			700	2252	1563	58.3	3887	77.0
my_adder	33/17	221	381	166	316	19.8	162	308	20.7	78.0	346	73.9
term1	34/10	345	827	114	264	35.5	117	257	139	31.9	903	41.5
too_large	38/3	574	1576	not tried			290	779	3240	49.7	1614	66.3
ttt2	24/21	195	467	114	264	49.7	123	282	29.6	61.2	448	72.5
vda	17/39	570	1870	345	1016	342	386	1235	170	66.4	1860	73.4
x1	51/35	299	706	216	487	426	220	495	91.3	71.1	687	69.4
x3	135/99	681	1705	560	1382	939	568	1415	432	83.1	1820	78.7
x4	94/71	377	912	283	690	204	284	655	76.7	72.8	979	72.7
pair	173/137	1375	3012	not tried			1209	2819	271m	91.8	not available	

表 4: ベンチマーク結果およびドントケアフィルタとの比較

リ入力側に位置する一部のノードに論理変換を施せなかった場合の解の振舞いを図6に示す。横軸にはプライマリ出力側から始めて何％のノードに対して手続き1を適用したかを示す。実験に用いた回路はC432である。計算時間が妥当な曲線に乗らないものがあるが、たまたま収束が遅れ繰り返し回数が増えたものであろう。このデータからは、あるしきい値（この例では50％）までは解の品質への影響はほとんどないと考えてよい。すなわち、記憶領域を節約するためにトランスダクションを部分的に適用することが可能であることがわかる。

最後に、分割／一括最適化の違いを比較する。最初に述べたように、RTレベルの論理合成で生成される制御回路は全体のプライマリ入力としてシステムへの外部入力と状態レジスタを持つが、その機能によっていくつかの部分回路の集まりとして設計されると考えられる。一例として我々のRTレベル論理合成システム[11]では命令デコード部、データパス制御部、次状態算出部およびどれにも属さない若干の論理回路に4分割されている。表5は同システムより得た回路について、一括最適化の効果を示すため、4つの部分回路を個別に最適化した結果の合計との比較を示すものである。用いた回路はある8ビットCPUのデータパスのための制御回路である。一括最適化の結果は部分回路

回路	in/out	node	edge	cpu
命令デコード	8/181	207	247	10.9
データパス制御	195/96	752	2029	1164
次状態算出	113/8	604	1744	2222
その他	58/39	196	555	77.0
合計		1759	4575	3474
一括最適化	39/103	896	2641	6688

表 5: RTレベル合成の結果への適用例

の最適化によるものより約44％小さくなっている。ただし計算コストは2倍近い。一般的に複数の機能ブロックを一括最適化することが効果的であるとはいえないが、RTレベル論理合成の場合、部分回路相互の結合度が高く、全部分回路を一つにしたことにより解空間が大きく拡大し、最適化が進んだと考えられる。

6 おわりに

解の品質を落とすことなく大規模組み合わせ論理を最適化するための手法を提案し、最大ＣＳＰＦを用いた最適化が実用的なものであることを示した。従来ドントケアフィルタを用いてのみ可能であったような大規模回路に対しても有効であること、および、前記の大規模回路対策の定性的な性質が実験的に確認された。さらに、ＲＴ合成システムの組み合わせ回路合成部分として使用することの効果の大きさが示された。現在、処理時間の短縮が最大の課題であり、実用化を進めるためにはＢＤＤ変数の順序付け、ＢＤＤパッケージの最適化などが必要であると考えている。

謝辞

有益な御示唆をいただいたイリノイ大学・室賀三郎教授に深謝します。

参考文献

[1] R. K. Brayton and C. McMullen, "The Decomposition and Factorization of Boolean Expressions", *Proc. IEEE ISCAS,* pp. 49-54, 1982.

[2] R. K. Brayton, R. Rudell, A. Sangiovanni - Vincentelli, and A. R. Wang, "MIS: Multi-level Interactive Logic Optimization System", *IEEE Trans. on CAD,* vol. CAD-6, pp. 1062-1081, November 1987.

[3] R. K. Brayton, A. Sangiovanni-Vincentelli, and G. D. Hachtel, "Multi-Level Logic Synthesis", *Proc. of the IEEE,* vol. 78, pp. 264-300, February 1990.

[4] K-C. Chen and S. Muroga, "Timing Optimization for Multi-Level Combinational Networks", *Proc. 27th ACM/IEEE DAC.,* 1990.

[5] H. Savoj and R. K. Brayton, "The Use of Observability and External Don't Cares for the Simplification of Multi-Level Networks", *Proc. 27th ACM/IEEE DAC.,* 1990.

[6] K-C. Chen, Y. Matsunaga, S. Muroga, and M. Fujita, "A Resynthesis Approach for Network Optimization", *Proc. 28th ACM/IEEE DAC.,* 1991.

[7] S. Malik, A. R. Wang, R. K. Brayton, A. Sangiovanni-Vincentelli, "Logic Verification using Binary Decision Diagrams in a Logic Synthesis Environment", *Proc. IEEE ICCAD,* pp. 6-9, 1988.

[8] S. Minato, N. Ishiura, and S. Yajima, "Shared Binary Decision Diagram with Attributed Edges for Efficient Boolean Function Manipulaation", *Proc. 27th ACM/IEEE DAC.,* 1990.

[9] S. Muroga, Y. Kambayashi, H. C. Lai, and J. N. Culliney, "The Transduction Method - Design of Logic Networks based on Permissible Functions", *IEEE Trans. Comput.,* vol. C-38, pp. 1404-1424, October 1989.

[10] 松永 裕介, 藤田 昌宏, "トランスダクション法の評価と改良", 情処研報 Vol. 89, No. 108, 89-DA-50, pp. 55-62, Dec. 1989.

[11] 野田 浩明, 高橋 瑞樹, 横山 昌生, 神戸尚志 "レジスタ転送レベル合成システム（1）、（2）", 電子情報通信学会春季全国大会 1991.

[12] 藤田 昌宏, 松永 裕介, 角田 多苗子, "大規模回路の多段論理簡単化について", 第 41 回情報処理全国大会 4N-5, 1990.

[13] 藤本 徹哉, 神戸 尚志, "許容関数集合に基づく組み合わせ論理回路の遅延最適化", 情処研報 Vol. 90, No. 100, 90-DA-55, pp. 73-78, Dec. 1990.