論理合成技術

湊 真一

NTT 未来ねっと研究所
〒239-0847 神奈川県横須賀市光の丘 1-1
minato@ieee.org

あらまし システム LSI 設計技術研究会（旧設計自動化研究会）は今年で設立 30 周年を迎えた．この間，ＬＳＩの設計自動化・設計支援技術は急速に進展し，中でも論理合成技術はこの 10 年間で最も目覚ましい進歩を見せた分野であると言える．今後，ＬＳＩのさらなる大規模化，高速化，超微細化に対応して行くためには，論理合成ツールは単なるブラックボックスとして用いることはできず，その動作原理を理解した上で，新しい設計方法を工夫して行く必要があると思われる．本稿では，論理合成ツールの内部で用いられているデータ構造やアルゴリズムについて解説する．手設計と共通する論理設計技法については簡単な紹介にとどめ，自動合成ツール特有の技術を中心に述べることとする．

キーワード 論理合成，論理設計，設計自動化，LSI, CAD, EDA

Logic Synthesis Techniques

Shin-ichi Minato

NTT Network Innovation Laboratories
1-1, Hikarinooka, Yokosuka-shi, 239-0847 Japan.
minato@ieee.org

Abstract SIG-SLDM (formerly SIG-DA) of IPSJ is celebrated its 30th anniversary this year. In this period, LSI CAD/DA technologies have extremely been developed, and among them, logic synthesis techniques have especially progressed in the last decade. For the future, we will have to deal with a larger scale, a faster clock cycle, and a smaller design rule of the LSI design, so it will be hard to use a logic synthesis tools as just a black box. We should develop new design methodologies based on the knowledge of the current logic synthesis methods. This tutorial paper mainly focuses on the data structures and the algorithms used in the contemporary logic synthesis systems.

key words logic synthesis, logic design, design automation, LSI, CAD, EDA

1　はじめに

システム LSI 設計技術研究会（旧設計自動化研究会）は今年で設立 30 周年を迎えた．この間, LSI の設計自動化・設計支援技術は急速に進展し，中でも論理合成技術はこの 10 年間で最も目覚ましい進歩を見せた分野であると言える．今日では論理自動合成システムは LSI 開発のための必要不可欠なツールとなっている．　論理合成ツールを開発・提供する EDA ベンダは主に米国西海岸において発達し，大きいところでは数千人もの従業員を抱える一大ハイテク産業を形成している．

論理合成技術は計算機科学の応用例としては最もビジネス的に成功した分野の 1 つではないかと思われる．特に新しい物質や物性を発見したわけでなく，スイッチング回路に関する数学的理論と，計算アルゴリズムの工夫，および大規模ソフトウエアの実装技術という，いわば論理的な約束事だけで現在のような産業を作り上げ，大きな経済効果を生み出しているわけである．

一方で，日本の多くの半導体メーカは，かつては自社内に独自の CAD 部門を抱え，論理合成技術についても盛んに研究開発を行っていたが，設計自動化技術の成熟に伴い，EDA ベンダが提供する市販ツールを用いるようになった．現在の CAD 技術者の主な役割は，自社特有のテクノロジ用の設計ツールを提供することや，ある特定の LSI 開発を設計者と一体となってサポートするといったことが多くなっている．

この間，論理合成技術に関して，日本からも多くの重要な貢献があり，決して米国の技術にただ乗りして来たわけではない．これらの技術は SLDM 研究会を核とした日本の研究者・技術者の人材の中に，有形無形の技術として蓄積されている．

今後，プロセスの超微細化が進むに連れて，従来の汎用的な論理合成ツールでは立ち行かなくなる場面が増えてくると思われる．その壁を破るためには，論理合成ツールはブラックボックスとして用いることはできず，その動作原理を理解した上で，新しい設計方法論を編み出して行かねばならない．そのためにも論理合成ツールの中で何が起こっているかを一通り理解しておくことは意味がある．

論理設計とスイッチング理論に関してはいくつかの教科書 [18, 53, 51, 26, 14] が出されているが，近年発達した設計自動化技術を主として書かれたものはまだ多くないようである．本稿が論理合成ツール利用者の参考になれば幸いである．

2　論理合成とは

昨今の典型的な LSI 設計工程をおおまかに書くと図 1のようになる．通常，まず自然言語や図表からなる仕様書が作られる．次にこれを RTL(Register Transfer Level) と呼ばれる，外部入出力ポートやレジスタ間の論理の因果関係とタイミングを表した記述形式に直していく．（この過程で仕様書の曖昧だった部分も徐々に明確になる．）RTL 記述は論理

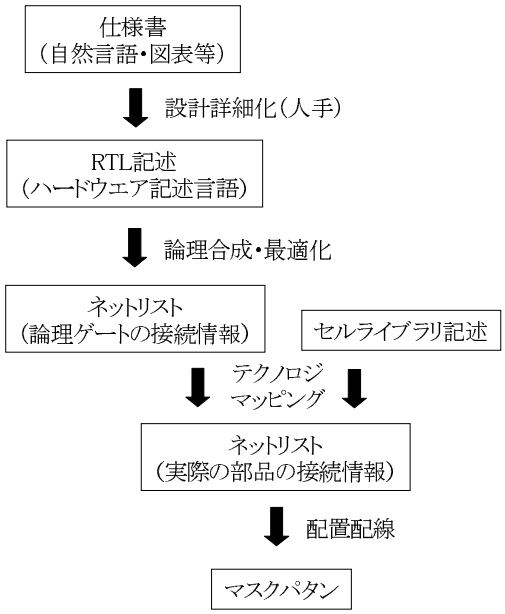


図 1: 典型的な LSI 設計フロー

合成処理によって，単純な論理ゲートの組合せに変換される．さらに，テクノロジマッピングと呼ばれる処理により，現実の LSI 製造プロセスで利用可能な部品（複合ゲート，セレクタ，レジスタ等）の組合せに変換される．変換後の記述（部品の接続関係を表したもの）はネットリストと呼ばれる．これを 2 次元の領域に配置配線すると，LSI チップ製造に必要なマスクパタンとなる．

かつては論理設計は人手で行うのが一般的だった．すなわち，仕様書を見ながら，論理ゲートやセレクタを組み合わせて演算器のような論理ブロックを設計し，それらをさらに階層的に組み合わせることで，所望の機能を構成していく方法がとられていた．またそれを支援するものとして，計算機の画面上に回路図を描画入力して行くと，自動的にネットリストファイルを出力する論理回路図エディタが使われていた．

1990 年代に入って，論理回路図を入力せずに，C 言語のようなプログラミング言語に類似したハードウエア記述言語を用いて RTL 設計を行い，そこから論理合成ツールで自動的にネットリストに変換するという設計スタイルが普及していった．この背景には，論理最適化技術が進歩して，熟練設計者の技に近い品質の論理回路が合成できるようになってきたことと，回路規模が増大して，そもそも人手では手に負えなくなってしまったということがあげられる．

論理合成システムの入力（ソース）となるのがハードウエア記述言語である．現在の市販ツールでは，VerilogHDL と VHDL の 2 つの言語が最も多く用いられている．他に著名な言語としては，UDL/I や SFL がある．UDL/I は 1990 年代前半に日本の学界・産業界が提案した標準化言語で，米国 EDA ベンダのデファクト標準である VerilogHDL や，米国国防省規格の流れをくむ VHDL と並んで比較検討され，互いの規格に影響を与えた．また SFL は NTT 研究所で 1980 年代後半に作られた論理合成システム Parthenon[62] の入力言語で，簡潔で記述しやすい特長があり，現在も研究教育分

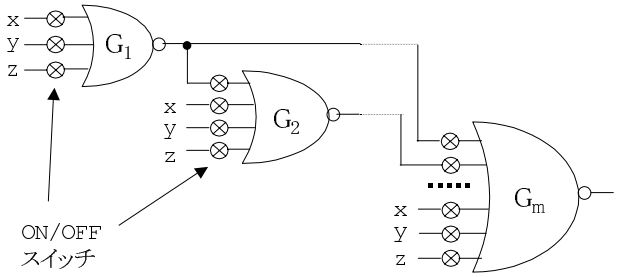


図 2: 任意の NOR ゲート回路を作るモデル

野で使われている．

一方, RTL よりも抽象度の高い動作記述言語として, C 言語やその拡張版が用いられる場合もある．また, 設計者がより詳細な構造を指定したい場合には, RTL より抽象度の低い論理式やネットリスト記述が併用される場合もある． 結局, (1) 有力な論理合成ツールが使えるか, (2) 設計者が記述しやすいか, (3) 必要なライブラリ（部品）が提供されているか, といった観点から記述言語やその抽象度レベルが選択される．

論理合成の目的は, 実際に使える部品の組合せで所望の機能を構成することであるから, どのような部品が使えるかによって処理の内容や品質の評価尺度は当然異なってくる．例えば, PLA(Programmable Logic Array) を使って実現する場合は, AND-OR の積和形論理回路を合成する必要があり, AND ゲートの数が少ないほどコンパクトな回路となる（このとき各ゲートの入力数は多くなっても構わない）． 一方, セルライブラリを用いた ASIC では, 複数のセルを多段に組み合わせた論理回路を合成する．このときは総配線本数（面積）や段数（遅延）等が重要なファクタとなる．また, FPGA を使う場合には, 各論理ブロックの入力数は固定されているので, その範囲内であれば複雑な論理をマッピングしても実現コストは変わらない．

RTL 仕様記述においてレジスタの個数や機能があらかじめ決まっている場合は, レジスタ間の論理を実現する組合せ回路だけを合成すれば良いが, レジスタの割り当てにも自由度が与えられている場合には, 順序回路として合成を行う必要がある．順序回路にもクロック同期式, 非同期式等の種々の方式があり, それぞれ設計方法が異なる．パストランジスタ回路やドミノ論理のような特定の回路方式を使用するならば, それなりの設計規則が必要となる．このように, 論理合成システムでは, 最終的な実現形式をある程度想定して論理設計アルゴリズムが決まり, それに応じて面積や遅延, 消費電力等の評価尺度を定めることになる．図 1ではテクノロジに依存しない論理合成処理とテクノロジマッピングとを分けて書いているが, 設計対象によってはこれを分けずに渾然一体として合成処理が行われる場合もある．

さて, 論理合成処理とは, 抽象度の高い仕様記述言語を, より抽象度の低いネットリストに変換する作業であるため, 同じ仕様記述から得られる合成結果は 1 通りとは限らず, 冗長な回路も含めて無数の候補が存在しうる．このとき, 先に述べた評価尺度が最適になるものを当然選びたい． すなわち,

表 1: NOR ゲート組合せ回路の数

入力数	ゲート数	スイッチ数	組合せ回路の数
3	7	42	4,398,046,511,104
3	8	52	4,503,599,627,370,496
4	7	49	562,949,953,421,312
4	8	60	1,152,921,504,606,846,976

論理合成技術とは, 仕様を満たす膨大な数の候補の中から最適な解を探索する組合せ最適化問題であると言える．

どのくらいの可能性があるかを直観的に示す例として, NOR ゲートだけを組み合わせて構成できる多段組合せ回路を考えてみよう．図 2に示すようなモデルを考えると, ゲートの各入力のスイッチを ON/OFF することで（冗長な解も含めて）任意の NOR ゲート回路が構成できる．外部入力数: n, ゲート数: m とすると, スイッチの総数は $(2n+m-1)m/2$ となる．具体的な n, m の値に対する回路の候補数は表 1のようになる．論理合成処理とは, この膨大な数の候補の中から, 所望の機能を正しく実現する回路を選び出し, さらにその中から, 現実に動作可能な NOR ゲートの最大の入力数（Fan-in）・出力数 (Fan-out) の条件を満たしつつ, 総ゲート数, 接続線数, 遅延段数等の評価尺度が最適となるものを探し出すということである．数個の NOR ゲートだけでもこれだけの組合せがあるわけであるから, より複雑なゲートを数百, 数千と使う回路の組合せ数は想像を絶する．記憶素子を含む順序回路であればなおさらである．このようにすべての可能性を網羅して厳密に最適な回路を合成する試みは 1960 年代に行われているが（数え上げ法 [15], 0-1 整数計画法 [34]）, 探索空間があまりに広いため, 極めて小規模な例題を除けば厳密解を求めることはまず不可能である．そこで, 現実的な時間で最適に近い回路をいかに合成するかという技術がこれまで研究されてきた． 探索空間の一部しか調べない発見的手法であるため, 運が悪ければ最適値にほど遠い回路しかみつけられない場合もある．（ただし本当の最適値もわからないことが多いので, 遠いか近いかを知ることさえも難しい．）それでも他に方法はないのであって, 実用的な規模の論理設計を現実的な時間で行うには, 今や論理合成ツールは不可欠である．せめて, どの手法はどのような問題が不得意なのか, という傾向（癖）がわかれば, 論理合成ツールを使いこなすための参考になると考えられる．

以下の章では論理合成システムで用いられている技術について解説する．

3 論理合成システムのデータ構造

前にも述べたように, 論理合成処理は外部入出力ポートやレジスタ間の論理の因果関係およびタイミングを表した記述から, これを実際に使用するセル（部品）の接続関係に変換することが主な作業である．この中で重要と考えられるデー

z w \ x y	00	01	11	10
00	1	1	0	0
01	0	1	1	1
11	1	1	1	1
10	1	0	0	0

図 3: カルノー図

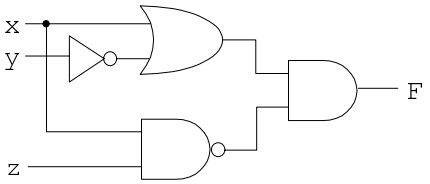


図 4: 論理回路図

タ構造として以下の 2 つがあげられる.

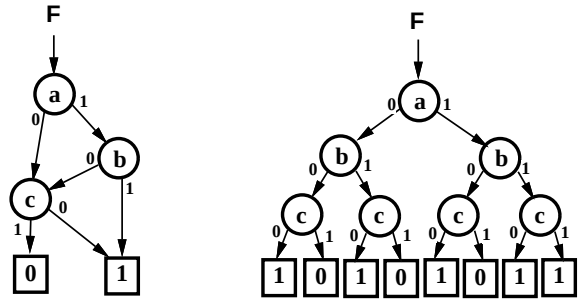
- 論理の因果関係を表す論理関数データの計算機内部での表現方法
- 合成の結果得られる大規模なネットリストを計算機内部に表現するデータ構造

かつて人手で設計していた時代には, 人間の目で見通しが良い図的な表現, 例えばカルノー図による表現（図 3）や論理回路図による表現（図 4）等が用いられてきた. しかし計算機で大規模な問題を処理する場合には, 見やすいことよりも, 表現がコンパクトであることや, 演算処理が高速に行えることが重要となる. 一般に n 入力の論理関数は 2^n 通りの入力組合せを持ち, それぞれの組合せに対し 0,1 の 2 値出力値を選ぶことができるので, 論理関数の総数は 2^{2^n} 個存在することになる. これらを固定長のデータ構造で識別するためには少なくとも 2^n ビットの記憶量は必要になる. 真理値表によるデータ表現（図 5）がそれに相当する. しかし実際の論理設計では必ずしも全ての論理関数が均等に出現するわけではないので, よく現れる論理関数がコンパクトに表現できる可変長のデータ構造が使われることが多い. また, 2 つの論理関数の間での論理演算や等価性判定等の処理が高速に実行できることも重要である. これまでに以下のようなデータ表現が提案され用いられている.

真理値表: カルノー図と本質的には同じであるが, 計算機で処理するので人間に見やすく並んでいる必要はなく, 単純な 1 次元配列で表現される. ベクトル計算機等の並列処理に向くが, どんな簡単な論理関数にも 2^n ビットの記憶量を必要とし, 論理演算や等価性判定にも指数時間を要するため, 入力数が多くなると実用的でない. （30 入力程度が限界である）

$x_1x_2x_3\ldots x_n$	F_1	F_2	F_3
000… 0	1	0	1
100… 0	0	1	0
010… 0	1	1	0
110… 0	0	0	0
001… 0	1	1	0
101… 0	0	1	0
011… 0	1	1	0
⋮	⋮	⋮	⋮
111… 1	1	0	0

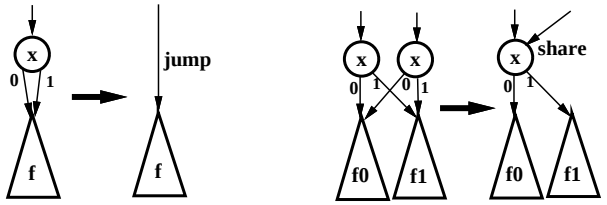
図 5: 真理値表表現



(a) BDD. (b) 場合分け二分木.

図 6: $F = (a \wedge b) \vee \overline{c}$ を表す BDD と場合分け二分木.

積和形論理式: 肯定・否定の 2 種類の入力変数に対する積和 2 段の論理式で論理関数を表すデータ構造である. Cube Set（積項集合）あるいは PLA 表現と呼ぶこともある. 記憶量は論理式に出現する文字数にほぼ比例する. 簡単な式で書けるような論理関数に対しては真理値表よりも少ない記憶量で表せることが多く, 実用的なデータ構造の 1 つである. しかし演算処理中に発生する冗長な積項を取り除くための簡単化処理が必要で, 等価性判定に時間がかかること, XOR 関数が効率よく表せないこと, および否定演算に手間がかかること, 等の欠点がある. 近年, 次に述べる BDD 技術を応用して, 積項の集合を非明示的にコンパクトに表現するデータ構造が提案されており [60], 論理式の文字数に必ずしも比例せずに, より効率よく処理できるようになった.



(a) 冗長節点の削除. (b) 等価な節点の共有.

図 7: BDD の簡約化規則.

BDD（二分決定グラフ）: 1980 年代後半から急速に研究が進み実用化された新しい論理関数のデータ構造で，図 6(a) に示すような論理関数のグラフによる表現である．これは，論理関数の値を全ての変数について場合分けした結果を図 6(b) のように二分木グラフで表し，これを縮約することにより得られる．このとき，場合分けする変数の順序を固定し，

- 冗長な節点を削除する (図 7(a)).
- 等価な部分グラフを共有する (図 7(b)).

という処理を可能な限り行うことにより「既約」な形が得られる．この BDD 表現は，多くの実用的な論理関数をコンパクトに表すことができ，性質のよい例題では数百入力までも現実的な記憶量で扱うことができる．そして，グラフの大きさにほぼ比例する計算時間で各種の論理演算を行うことができる．また論理関数に対して表現が一意に定まるという優れた性質を持っており，等価性判定も容易に行うことができる．BDD の処理法に関しては，米国 CMU の Bryant らのグループにより初めて実用的なアルゴリズムが示された [9]．その後これを改良する研究が，富士通研究所，京大，NTT，他の日本の研究機関からも数多く提案され [11, 32, 29]，国際会議等で活発に議論された．現在では BDD は多くの論理合成ツールやその他の EDA ツールで広く用いられている．また BDD パッケージとして公開されているものもある [44, 39, 59]．BDD の処理技法については文献 [58, 61] 等を参照されたい．

さて，上記で述べた論理関数のデータ構造とその処理技術は，計算機科学の基本的な問題でもあり，学会等でもよく取り上げられている．一方，論理合成の結果得られるネットリストのような大規模設計データを計算機で扱う実装技術も論理合成システムを実現する上で極めて重要である．

論理合成システムにおけるネットリストのデータ構造については，以下のように，かなり高度な要求条件を満たす必要がある．（論理合成に限らず EDA ツール全般に当てはまるものもある．）

- データは単純な配列では表現できず，ポインタ同士が複雑に参照し合う構造体のリストとなる．
- 一般的な大規模データベースのトランザクション処理のように定型作業を大量に繰り返すのではなく，ポインタのつなぎ換え・追加・削除，あるいは再帰的な呼び出しといった複雑な処理が多い．
- 論理最適化の過程でデータ量が激しく増減するため，最初に記憶場所を画定することが困難である．そのため動的なメモリ管理が必要である．またメモリがあふれたときの動作も常に意識する必要がある．
- 全ての部品に順次アクセスする作業と，部品同士の参照関係をたどってランダムにアクセスする作業の両方を高速に実行できることが要求される．データ量が大きいので，計算時間のオーダが $O(n \log n)$ なら我慢できるが $O(n^2)$ になるともう使い物にならない，というケースもある．
- 実設計に使われるためには安定した動作が要求される．ゴミが残る等の副作用は許されない．
- 複数の開発・保守担当者が共同で作業することが多いので，プログラムの読みやすさと曖昧性の排除は必須である．
- 使用可能な部品の変更やユーザの要求により，新しいデータ構造や処理手順が追加または修正されることがあるので，データ構造にも柔軟性・拡張性が必要である．

上記のようなデータ構造の実装技術は，コーディングが美しい・汚いと俗に言われるような話で，学会等ではあまり発表されることのない部分である．しかし現実にはこの部分のノウハウは非常に重要であり，EDA ツールが実用に耐えるかどうかを左右すると言っても過言ではない．

もちろん核となる論理合成アルゴリズムが劣っていては仕方がない．そのような核心の部分は多くの場合，論理設計に精通した 1 人あるいは極めて少人数の研究者・技術者によって考案・実装される．しかし市販ツールとして用いられるためには，大規模ソフトウエアの開発・保守管理に組織的なノウハウが必要であり，それができないツールは研究教育用に留まる．米国西海岸の EDA ベンチャ企業は，主力のツールが普及し始めると大手ベンダのいずれかに吸収合併されることがしばしばあるが，これは米国のビジネスモデルからだけでなく，このような技術的背景からも理解できることである．

今のところ，論理合成ツールに要求される高度な実装ノウハウを身につけるための組織的な方法はないように思われる．基本的なプログラミングテクニックを学んだ後は，過去の著名な論理合成プログラム（例えば [46] で公開されている ESPRESSO や SIS 等）のソースプログラムを読んだり試用したりして参考にすることと，プログラム開発プロジェクトに何度も参画して経験値を高めること，等が有効な方法であろう．

4 論理合成アルゴリズム

この章では，論理合成アルゴリズムについて解説する．まず基本となる組合せ回路の合成技術について，二段論理回路の最適化と多段論理合成について述べ，次にレジスタを含む順序回路の合成法について述べる．さらに論理合成対象となる半導体テクノロジに依存する技術についても解説する．なお，以下では，手設計と共通する一般的な論理設計技法については簡単な紹介にとどめ，自動合成ツール特有の技術を中心に述べることとする．

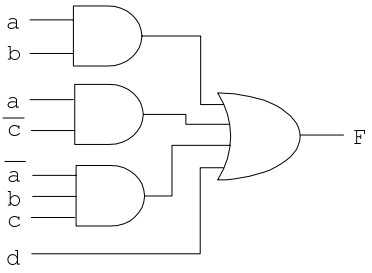


図 8: 二段論理回路

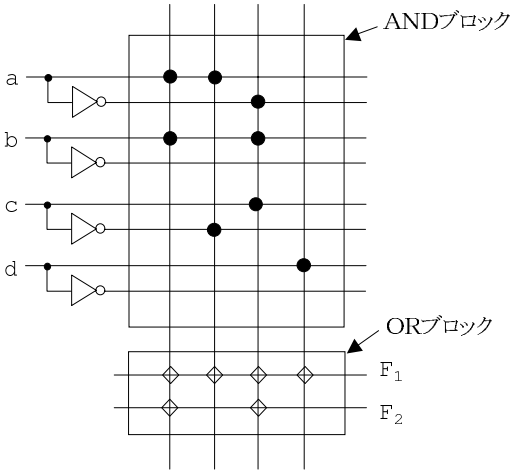


図 9: PLA による二段論理回路の実現

4.1 二段論理回路の最適化技術

AND-OR の積和形論理により実現される回路（図 8）を一般的に二段論理回路と呼ぶ．多くの場合，二段論理よりも多段論理の方がコンパクトな回路が得られるのだが，多段論理最適化は探索空間が広くて難しいため，まず二段論理最適化の研究が古くから行われてきた． 比較的小規模な例題であれば PLA（図 9）を使って，合成結果の二段論理回路をそのまま実装できる．積和形論理式は人工知能の分野で命題論理を表現するために用いられる CNF（Conjunctive Normal Form）とも等価なモデルであることから，その最適化技術は理論的にも重要性が高い．

従来の手設計では，積和論理式を図 10のようなカルノー図で表して最適化を行うことが一般的であった．この表現形式

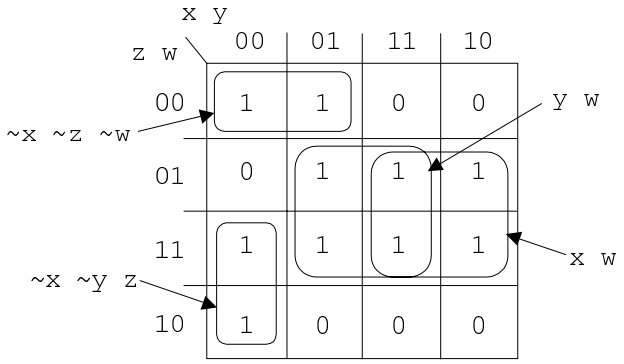


図 10: カルノー図による二段論理回路の設計

では，論理式の各積項はカルノー図の一部を囲む方形領域となる．積項の入力変数が少ないほど（= AND ゲートの入力数が少ないほど）大きな方形となる． 積和論理式の最適化とは，なるべく大きな方形領域をなるべく少数個使って，カルノー図の「1」の領域だけを全て囲むにはどうすればよいか，という問題となる．実際のプログラムでは，カルノー図ではなく，計算機処理に適した積項リスト等のデータ表現の上でこれを行うわけである．二段論理の最小化に関する最も古典的なアルゴリズムとして 1950 年代に考案された Quine-McClusky（QM）法 [38] がある． この方法は次のステップからなる．

1. 最小の領域（カルノー図の「1」のコマ 1 つだけ）を囲む積項を全てリストアップする
2. 隣接する積項同士を併合して，より大きな領域を囲む積項を作る． これを繰り返して，これ以上拡大できない積項（prime implicant: 素項または主項と呼ぶ．）を全てリストアップする．
3. リストアップした中から，最小コスト（積項数最小または文字数最小）で論理関数をカバーする素項の組合せを探索する．

QM 法は比較的単純なアルゴリズムで厳密に最小の積和論理式を求められる．しかしリストアップする素項の数が入力変数の数に対して指数関数的に大きくなる場合があり，また最小コストの素項の組合せを探索するために素項数に対して指数的な時間がかかる場合がある．このため計算機を用いても，現実的に処理できるのはたかだか 10 変数程度までとされている．

QM 法は大規模な問題には時間がかかり過ぎるため，ヒューリスティックな方法で比較的短時間に良い解を求める方法が研究開発されてきた．代表的なプログラムとして，MINI[16], ESPRESSO[6] 等がある．これらの方法では，

- 隣接する積項を併合したり，逆に縮小したりする操作を繰り返しながら，不要な積項をみつけて取り除いていく．
- 論理関数をカバーするために必須となる素項をみつけ，探索空間を絞り込む．

等の処理をヒューリスティックに適用し，限られた時間で，最小に近い積和論理式を求めている（詳細は [51] 等の解説書を参照されたい）．これらのヒューリスティックな方法により，入力変数が数十程度，積項数が数百程度までであれば，数分から数時間で比較的良い二段論理回路が合成できるようになった．

さらに大規模な二段論理回路の合成法として，BDD で与えられた論理関数から非冗長積和形 (irredundant sum-of-products form) に直接変換するアルゴリズム (Minato- Morreale(MM) 法 [28]) が提案されている．非冗長積和形は，どの 1 つの積項または変数を削除しても（= どの 1 つの信号線

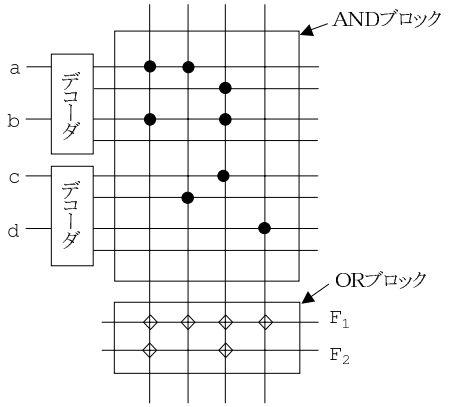


図 11: 4 値論理を用いた PLA

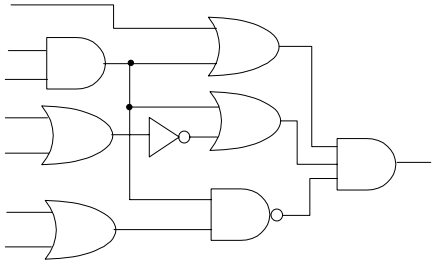
を切断しても）出力論理に影響があるという形で，必ずしも最小ではないが比較的良い局所解と言える．MM 法は，BDD でコンパクトに表せる関数であれば，ESPRESSO や MINI よりはるかに高速であり，3 章でのべた非明示的なデータ表現法 [29] を用いて，積項数が百万を超える非冗長積和論理式を現実的な時間で生成することができる．二段論理回路の合成法としてだけでなく，多段論理合成の前処理や，人工知能分野の処理にも用いられている．

その他，二段論理最適化に関するいくつかの参考情報を以下に述べる．

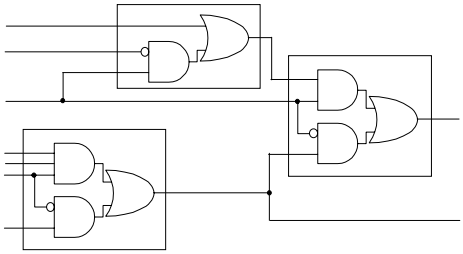
- 積和形と和積形は双対の関係にあり，出力を反転した論理の積和形は，和積形を求めたのと等価な形となる．統計的には，真理値表濃度（真理値表の中の「1」の割合）が大きいほど，積和形に含まれる素項の個数が多くなる傾向があるので，真理値表濃度が 50% を超えたら，出力を反転させて合成を行う方が良い結果が得られることが多い．（合成ツールによっては，否定論理も自動的に考慮しているものもある．）

- 図 11のように，2 値の入力変数を 2 つずつグループ化して，2 値変数にデコードすると，（デコーダのコストを差し引いても）よりコンパクトな二段論理回路が得られる場合がある．このような多値論理の最適化を行うツールとしては ESPRESSO-MV[41] 等がある．（詳細については，文献 [51, 1] 等を参照されたい．）

- 出力値として 0,1 以外に don't care を含む論理関数，すなわち入力組合せに対する出力応答が部分的にしか定義されていない不完全指定論理関数が，現実の設計ではしばしば扱われる．論理合成ツールでは，don't care を 0,1 どちらか都合の良い方に解釈することができるので，より最適な回路を合成するチャンスが増える．しかし don't care が多いとヒューリスティックな合成法で利用できる「手掛かり」が減って最適解をみつけにくくなる場合もあるので，注意が必要である．



(a) 基本ゲートの多段回路



(b) 論理ブロックの多段回路

図 12: 多段論理回路

4.2 多段論理回路の合成技術

多段論理回路は図 12(a)(b) に示すように，論理ゲートまたは論理ブロックを多段に組み合わせた論理回路である．ここでは信号のループやレジスタは含まれないものとする．(b) の場合，各論理ブロックの機能は真理値表・積和論理式・BDD 等の論理関数表現でそれぞれ保持している．現実世界では，論理ゲート・論理ブロックの入力数 (Fan-in) や分岐出力数 (Fan-out) および最大段数には制約がある．

2 章で例示したように，多段論理合成法は古くから研究されてきたが，探索空間が非常に大きいため，実用的なツールはなかなか現れなかった．1980 年代になって，二段論理最適化の研究が進み，その結果を利用して，積和論理式を代数的に因数分解して多段論理回路を得る代数的除算法 [7](algebraic division または weak division 法) が提案され広く使われるようになった．例えば，積和形論理式

$$f = a\ b\ d + a\ b\ \overline{e} + a\ b\ \overline{g} + c\ d + c\ \overline{e} + c\ h$$

は，$(a\ b + c)$ で割ることができる．中間変数 p を導入すると，

$$f = p\ d + p\ \overline{e} + a\ b\ \overline{g} + c\ h, \qquad p = a\ b + c$$

と分解できる．同様にして，次のステップで f は $(d + \overline{e})$ で割ることができる．このようにして，積和論理式を多段化すると，二段論理回路に比べて部品数を大幅に削減できる場合がある．

代数的除算は，除数と被除数が与えられれば答が一意に決まるため，1 回の除算は式の長さに比例する時間で高速に実行できる．しかし問題は何で割るかである．一般に，除数となる論理式は多数存在する．有効な除数候補をいかに選ぶかによって，合成結果の品質が左右される．代表的な多段化プログラム MIS（およびそれを改良した SIS）[7] では，カーネル抽出法と呼ぶ方法で，除数候補を見つけている．これは直観的には，論理式の中で複数回出現する変数をくくり出して，その変数にかかる多項式を取り出して除数候補とする方

法である．カーネル抽出法は比較的良い候補集合を高速に見つけだせるが，それでも除数候補は 1 つには決まらないので，どれかを選ぶ必要がある．高速性を優先して適当なヒューリスティック（例えばその時点で一番文字数を削減できる除数を選ぶ等）で済ませることが多い．

代数的除算にもとづく多段論理合成法は，論理式の長さにほぼ比例する処理時間で実行できるので，論理式の文字数が数百個程度であれば現実的な時間で論理合成が行える．ただし，XOR 回路や算術演算回路のように二段論理にすると指数爆発的に規模が増大する例題に対しては適用することができなかった．この問題については前に述べた BDD 技術による非明示的なデータ表現 [60, 30] により，論理式の長さではなく BDD のサイズに依存する計算時間で処理できるようになり，適用規模が格段に広がっている．

代数的除算は論理式の性質をすべて利用しているわけではなく，算術的な変形のみを用いているため高速に実行できるが，計算結果の品質（論理式の長さ）の面では，改善の余地がある．例えば $x+x=x$ であるとか，$x \cdot \overline{x}=0$ といった論理式特有の性質を用いて代数的除算法を改善するアルゴリズムがいくつか提案されている [7, 22]．このような技法を総称して論理除算法（Boolean Division または Strong Division）と呼ぶ．

論理除算は代数的除算と異なり，答が一意に決まらない．論理除算 $f = p\ q + r$ において，f と p が与えられたとき，なるべく小さな q, r を求める必要がある． 代数的除算では割れるか割れないかがはっきりしており，割れる場合は p, q が定数でない限り総文字数は必ず減少するが，論理除算ではどんな p でも強引に割ることができ，その結果，総文字数が増加する場合もある． MIS では，f と p が与えられたときに，q と r を don't care を含む不完全指定論理関数とし，これを ESPRESSO を用いてヒューリスティックに簡単化して，代数的除算よりも良い結果が得られれば採用する，という手法を用いている．一方，BDD を用いて don't care の集合を効率よく表し，論理除算を高速に行う方法 [55] も提案されている．いずれにしても，Boolean Division で合成品質を追求し過ぎると多大な計算時間や記憶量が必要となり現実的ではないため，代数的除算の合成結果を改善するために補助的に用いることが多い．

ここまで述べた二段論理を経由する多段論理合成手法とは異なり，多段論理回路を直接操作することで最適化を行うアプローチも古くから研究されている．2 章で例示した厳密最小化手法では記憶量や計算時間がかかり過ぎるので，より計算量の少ないヒューリスティックな方法が主に用いられている． このアプローチの代表的な方法として，トランスダクション法 [35] とその改良アルゴリズムがある．

トランスダクション法は図 13のような論理ゲートの多段回路において，配線を追加したり削除したりしながら回路を変形させる操作を繰り返し，ゲート数や配線数を削減させて行く手法である． ただし，やみくもに回路の変更を行うと出力の論理に影響を与えてしまうので，それを防ぐために，各ゲートの入力線について外部出力に影響を与えないで変更で

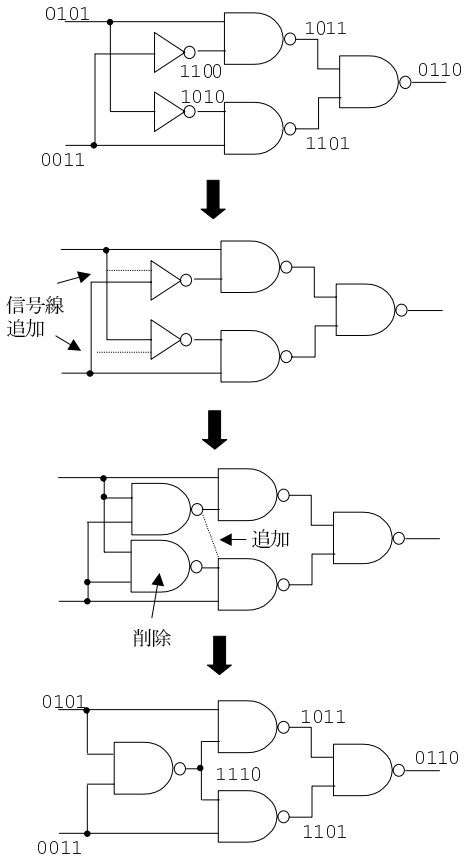


図 13: トランスダクション法

きる論理の条件（これを許容関数と呼ぶ）を計算し，それを Don't care を含む論理関数データとして計算機内部に記憶させておく．そして回路の変更を行う際には，許容関数の条件を満たす変更のみを許すことで，出力の正しさを保証している．許容関数の計算方法については [35, 57] 等を参照されたい．

トランスダクション法は先に述べた厳密最小解を求めるアルゴリズムよりは計算コストは小さいが，当初は許容関数を真理値表で表現していたため，外部入力変数の個数に対して常に指数の記憶量と計算時間が必要で，10 〜 20 ゲート程度の回路までしか適用できなかった．しかし，1990 年頃に許容関数を BDD で表現する方法 [25] が考案され，数百ゲート規模の回路の最適化が可能となり，実設計に用いられるようになった．

トランスダクション法以外にも，多段論理回路の部分的変更を繰り返す方法はいくつか存在する． 代表的なものとして，論理の等価性判定技術を応用した方法 [4]，および故障検出（テストパタン生成）の技術を応用する方法 [5, 13] がある．例えば，検出不可能な故障箇所が存在すれば，その線を削除しても出力には影響を与えることはない．そのような規則を利用して回路を変更し最適化を行う方法である．局所的な論理構造を解析し，検出不能故障を見つけるアルゴリズムはテストパタン生成の分野で長年研究されており，その成果を利用している．

トランスダクション法に代表される多段論理の逐次改善法では，許容関数や回路の局所的構造解析を用いて回路の変更

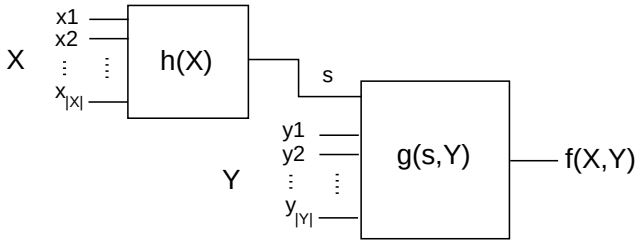


図 14: 論理関数分解

の可否を判定しているが，これは代数的除算になぞらえると，割り算を行ってよいかどうかを判定していることに相当する．しかし何で割るべきかということはまた別の問題である．すなわち，回路のどの部分に配線を追加・削除を繰り返すと今より小さな回路になるのか，という選択肢は，割り算の除数を選ぶのと同様に難しい問題である． 実際のプログラムでは，ヒューリスティックな変更ルール（例えば論理ゲートを1つ飛び越して接続してみる等）をいくつか用意して，試行錯誤を繰り返すようになっている．このあたりの戦略は理論的な評価が難しく，論文等には詳しく記されていないことが多い．

多段論理合成技術のもう1つ別のアプローチとして，仕様として与えられた論理関数をトップダウン的に複数のブロックに分解して多段論理を構成する「論理関数分解」の技術も古くから研究されている．例えば図 14のように，関数 $f(X,Y)$ を $g(s,Y)$ と $h(X)$ にきれいに分解することができれば，あとは g と h の中身を個別に合成すればよい．このとき，g と h の間の接続が1本だけになる分解を「単純分解」(simple decomposition) とよび，変数 X と Y に重なりがないものを「分離的（直交）分解」(disjunctive decomposition) と呼ぶ．単純かつ分離的な分解は任意の論理関数について必ず存在するとは限らないが，もし存在すれば是非採用したい有効な分解である． 論理関数分解アルゴリズムは，すでに 1950 年代から真理値表を用いた古典的な方法 [2, 40] が提案されていたが，計算量が大きく実用的ではなかった． 1990 年代後半になって，BDD を用いて実用的な規模の論理関数を高速に分解する方法がいくつか考案され [3, 24, 31]，現在では有効なアプローチの1つとなっている．単純かつ分離的な分解は，代数的除算で言えばきれいに割れる除数を見つけたことに相当するので，ほとんど試行錯誤なく実行できる．しかし，それ以外のより複雑な分解は論理除算と同様に選択肢が非常に多くなり，当たり外れも大きくなる．いくつかの手法 [42, 49] が提案されているが，決定版と言えるようなアルゴリズムはみつかっていない．

以上で述べた多段論理合成アルゴリズムは，一般的に，厳密解がわからない問題に対するヒューリスティック解法であるため，その性能を評価することは難しい．ことに大規模な多段論理回路では，初期回路が現実的なサイズで記述可能であるということ自体，すでに比較的コンパクトに表現されていると考えることもでき，初期回路の構造に最適化のための重要なヒントが含まれている可能性がある． 初期回路の構造情報を完全に消去して，入出力の論理関係のみを抽出して，そこから多段論理合成を行っても，初期回路より高品質の回路が得られないことがしばしばある．論理合成ツールを使う際には，高位レベルの設計仕様に含まれる構造情報を適宜残して合成ツールに与えることも，うまく使いこなすテクニックの1つであると言える．

4.3 順序回路の論理最適化

一般的なディジタル回路システムでは，共通のクロック信号でレジスタが動作する同期式順序回路が広く用いられている．同期式順序回路の設計法は次の手順に従う．

1. 冗長な状態を取り除き，状態数を最小化する．
2. 各状態を 2 値符号化し，レジスタを割り当てる． レジスタ間の論理がなるべく簡単になるような符号を選ぶ．
3. レジスタ間のタイミング条件を満たすように論理回路を合成する（組合せ回路合成）．

1，2 の部分が狭い意味で順序回路合成と呼ばれる部分である． 状態数の最小化や符号割り当ての最適化については，オートマトン理論として古くから多くの研究がなされており，KISS[27], JEDI[21], NOVA[47] のような自動最適化プログラムも作られている．しかし状態数最小化や符号割り当てのアルゴリズムは，計算量が大きい割には最適化の効果が顕著でない場合が多く，実設計で不可欠というほどには普及していない．その理由として以下のことがあげられる．

- 設計者が高位のシステム仕様を定める際に考えた状態集合を論理合成ツールにより変更されることはあまり好まれない．
- 設計者が状態符号割り当てまで考えている場合がある．
- 組合せ回路の最適化では何倍もコストが削減できるケースがしばしばあるが，状態符号割り当てでそのような劇的な効果が見られる例は経験的に少ない．もしあったとしたら，その符号ビットに重要な意味があってそれは設計者がすでに知っている可能性が高い．
- 人手で手に負えないような規模のレジスタ数だと，状態数はさらに天文学的な数になるため，計算機を用いても順序回路全体を最適化することは困難である．

以上のような理由で，古典的な順序回路合成法は成功しているとは言い難い．順序回路に関する合成ツールで，唯一実用的に使われているのはリタイミング [20, 23] と呼ばれる技術である．

リタイミングとは，図 15のように，レジスタ間の論理回路の遅延時間がクロック周期に間に合わない場合に，レジスタの位置を動かしたり，新しいレジスタを挿入したりすることによって，タイミングの仕様を満たすように回路を変更する技術である．配線に分岐等がある場合，単純にレジスタを移動すると論理的に正しい動作をしなくなることがあるので，レジスタを複製する等の処理が必要となる．リタイミング処

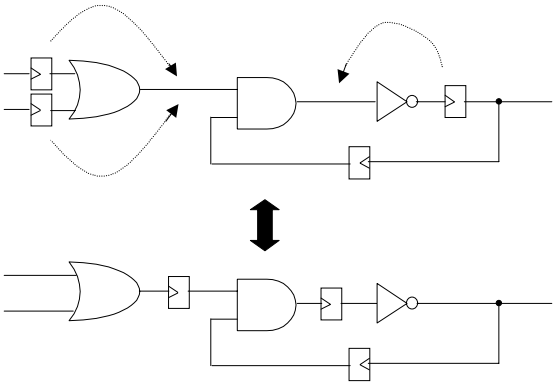


図 15: リタイミング

理で対応できる修正には限度があり，どんな遅延制約でも満たせるわけではない．リタイミングとは，状態割り当ての見直しを行うような大きな手戻りを発生させずに，局所的な変更だけでタイミング制約を満たすようにする（または性能をより向上させる）技術である．うまくいったときの実用的効果は大きい．

さて，ここまで述べた同期式順序回路と異なり，共通のクロック信号を用いない非同期式順序回路も最近使われるようになってきている [36]．（詳しくは，本研究会のもう 1 件のチュートリアル [54] で解説される予定である．）非同期式回路ではまず遅延モデルを定める必要があり，これにより論理設計法も影響を受ける．どれだけ大きな素子遅延のばらつきがあっても正しく動作する理想的な非同期回路から，特定の遅延時間を持つ遅延線を挿入して動作を保証する方式まで，種々の設計スタイルがある．またプロセッサのように繰り返し命令を実行するシステムであれば，非同期式といっても何らかのタイミングに同期して動作することにはかわりはない．それが回路全体に共通の同期クロックを使わないということであって，実際には 2 線式論理やハンドシェーク方式等により局所的な同期を取りながら計算を進めていくわけである．

非同期順序回路の自動合成ツールは最近研究が進みつつあるが [50]，まだ一般的に使われるまでにはなっていない．有望なアプローチとしては，非同期回路特有のタイミング処理を行う部分をなるべくモジュール化して隠蔽し，データフローのような形式で設計者が記述した仕様を，そのモジュールに自動的にマッピングして行くような方法が用いられるのではないかと思われる．

4.4 半導体テクノロジに依存する論理合成技術

これまで述べた論理最適化アルゴリズムは，ある程度汎用的に利用できるように，基本的な論理式または論理ゲートの組合せを合成するものであった．これを現実の半導体テクノロジに依存する部品セットに当てはめる処理が必要である．これをテクノロジマッピングと呼ぶ．

多段組合せ論理回路にセルライブラリを当てはめるテクノロジマッピングは 1980 年代後半に提案された tree-covering

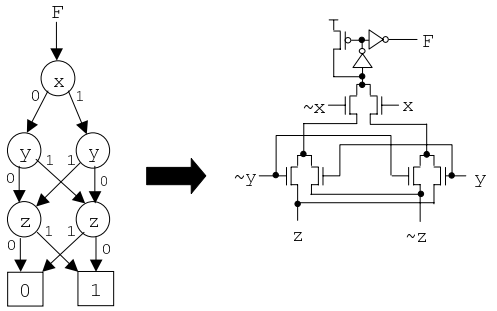


図 16: BDD から PTL へのマッピング

による手法 [17] が広く使われている．このアルゴリズムは動的計画法を用いて，部分回路に対する最適解を記録していくことにより，全体の最適なマッピングを求める方法である．計算量は基本的に回路規模に比例し，またセルライブラリの種類数や最大入力ピン数にも依存する．数百種類，最大 10 入力程度のライブラリであれば，実用的な規模の回路を現実的な計算時間で処理することが可能である．さらに，マッピングの際に遅延時間の評価も行い，遅延条件が厳しいところは段数が少なくなるようにするアルゴリズムも提案されている [10]．

論理合成ツールによっては，最初からテクノロジに依存する回路を対象とし，これを直接最適化するアルゴリズムも研究されている．特に 1990 年代以降，FPGA 回路の論理合成技術が盛んに研究されている [33, 42, 43]．FPGA では各論理ブロックの論理関数は自由に選べるので，通常の論理ゲートでは簡単に実現できない回路がコンパクトに実現できる場合がある．ただし，このような特定テクノロジ向けの最適化技術は局所的な改善効果に限られており，大域的な回路構造に関しては汎用的な合成技術が重要なことに変わりはない．なお，FPGA 合成に関しては，最近，SPFD (Sets of Pairs of Functions to be Distinguished) と呼ぶ新しい自由度の概念 [48, 52] が日本の研究者から提案され，これを利用した論理最適化アルゴリズムが海外でも研究されている [8]．

半導体テクノロジに依存する合成技術でもう 1 つ重要なのは，低消費電力設計に関する技術である．省電力化については，1 にデバイスの研究，2 に高位設計レベルでのバス・クロック・電源線の制御が重要と考えられるが，論理合成の分野でもいくらか貢献できることがある．CMOS 回路の消費電力は基本的に信号が変化する速度に比例するので，同じ論理を実現するならば，回路内部の信号の変化がなるべく少なくなるような回路を合成しようとするアルゴリズムが研究されている．代表的な方法としては 4.2 で解説した多段論理回路の局所的変更を繰り返す最適化の中で，複数の変更候補があった場合に信号変化が少なくなるように候補を選ぶという手法 [37] がある．またテクノロジマッピングを行う際に，信号変化が少なくなるように切り出すアルゴリズム [45] も提案されている．

低電力化や高速化を目的としてパストランジスタ回路を自動合成する研究も近年行われている．パストランジスタ回路は，1 線式 / 2 線式，静的 / 動的等，いくつかの方式があり，

接続の仕方に様々な電気的制約があるため，通常の論理ゲートとは異なる設計法が必要である．従来は人手の設計に頼っていたが，最近，論理関数をBDDで表現し，そのグラフ構造をパストランジスタ回路に自動的にマッピングする方法[19]がいくつか提案されている（図16に一例を示す）．ただし，大規模なBDDをそのままマッピングすると，段数や分岐数が大きくなり電気的な動作条件を満たさなくなるので，比較的小規模なブロックで，電力や速度がクリティカルな部分に用いるのが適当と考えられる．

ところで，LSIの大規模化に伴い，製造後のテストにかかる時間も増大する傾向にある．これをできるだけ容易にするような回路を合成することも必要とされている．論理合成と同時にテストパタンを自動生成する技術や，レジスタの内容を観測可能にするスキャン挿入技術等が研究されている．テストと関係する論理設計技術についての教科書としては，例えば[56, 12]が有名である．

5 おわりに

昨今の超高性能プロセッサの設計では，クロック周波数が極限近くまで高速化した結果，1クロック周期でせいぜい2,3段しかゲートを組めない場合もある．このような場合，人手でもある程度最適設計ができてしまうので，自動論理最適化技術の重要度は相対的に低下し，高位仕様記述からRTLに落とすところで大方の勝負はついてしまうことになる．あとはリタイミングで多少がんばれるか，というところである．またLSIプロセスの微細化に伴い，ゲートよりも配線の占める面積や遅延が相対的に増加している．このため，論理合成の工程だけを独立して処理しても最終結果との誤差が大きくなってしまう傾向にあり，今後は配置配線ツールと密接に関わりながら論理合成を行う技術が必要になると考えられている．

しかし，元来，論理設計と配置配線をなぜ切り分けたかと言えば，同時に考えると自由度が大き過ぎて計算量が膨大になって，限られた時間内には実用的な規模の最適化ができなかったからである．計算機の能力は昔に比べかなり向上してはいるが，問題の複雑化・大規模化の速度には追いついていない．このような背景からみて，今後の傾向としては，比較的小規模のブロックを丹念に最適化する技術と，大域的な構造を高速かつ安直な方法で最適化する技術に分かれていくだろうと思われる．

日本でSLDM研究会（および通信学会のVLD研究会）があるように，国際的にはIWLS Workshop (旧MCNC Workshop) が毎年開かれ，論理合成分野の研究者・技術者が主に米国西海岸に集まって最先端の議論を行っている．日本に比べれば米国はCADベンダの中心地とあって，依然として活発な研究活動が行われているが，やはりこの10年で成熟分野となってきた印象は強い．純粋な論理合成技術に関する論文はかなり減少しており，より上位レベルの合成（機能合成やHW-SW codesign）と，下流の物理レベル設計と密着した論理合成の2方向に研究はシフトしつつある．

設計自動化とは，職人芸であった論理設計を，あるいは職人ですら手に負えなかった複雑で大規模な設計を，誰にでも容易に行えるようにする技術である．しかし自動化技術は普及してしまえば，それまで持っていた技術を無力化してしまうという性質を持っている．例えば，鉛筆削り器が普及してナイフで鉛筆を研ぐ技能が重要でなくなったように，そしてシャープペンシルが普及して鉛筆削り器が売れなくなってしまったように，次々と技術の進歩をリードする側に立っていかなければ，現在の豊かな暮らしを維持していくことは難しくなる．

今後SoC時代に，どこでIT産業の競争力を維持して行くことになるのだろうか．画期的なアプリケーションのコンセプトで市場を押さえるのか，革新的な性質を持つデバイスを見いだすのか，複雑な仕様を短期間で最適に実現する設計製造技術なのか，またそれらの総合力で勝負するのか，SLDM研究会を始めとする学界，産業界が常に考えて行かねばならない課題と言えるだろう．

参考文献

[1] P. Ashar, S. Devadas, and A. R. Newton. *Sequential Logic Synthesis*. Kluwer Academic Publishers USA, 1992.

[2] R. Ashenhurst. The decomposition of switching functions. In *Proc. of an International Symposium on the Theory of Switching*, Apr. 1957.

[3] V. Bertacco and M. Damiani. The disjunctive decomposition of logic functions. In *Proc. of IEEE/ACM International Conference on Computer-Aided Design (ICCAD-97)*, pages 78–82, Nov. 1997.

[4] D. Bostick, G. D. Hachtel, R. Jacoby, M. R. Lightner, P. Moceyunas, C. R. Morrison, and D. Ravenscroft. BOLD: Boulder optimal logic design system. In *Proc. of IEEE/ACM International Conference on Computer-Aided Design (ICCAD-87)*, pages 62–65, Nov. 1987.

[5] D. Brand. Redundancy and don't cares in logic synthesis. *IEEE Trans. on Computers*, C-32(10):947–952, Oct. 1983.

[6] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers USA, 1984.

[7] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang. MIS: A multiple-level logic optimization system. *IEEE Trans. on Computer-Aided Design*, CAD-6(6):1062–1081, Nov. 1987.

[8] R. K. Brayton and S. Sinha. Implementation and use of SPFDs. In *Proc. of IEEE/ACM International Conference on Computer-Aided Design (ICCAD-98)*, pages 103–110, Nov. 1998.

[9] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Computers*, C-35(8):677–691, Aug. 1986.

[10] K. Chaudhary and M. Pedram. A near optimal algorithm for technology mapping minimizing area under delay constraints. In *Proc. of 29th ACM/IEEE Design Automation Conference (DAC'92)*, pages 492–498, June 1992.

[11] M. Fujita, Y. Matsunaga, and T. Kakuda. On variable ordering of binary decision diagrams for the application of multi-level logic synthesis. In *Proc. of*

IEEE The European Conference on Design Automation (EDAC'91), pages 50–54, Feb. 1991.

[12] H. Fujiwara. *Logic Testing and Design for Testability.* MIT Press, 1985.

[13] G. D. Hachtel, R. M. Jacoby, and K. K. C. R. Morrison. On the relationship between area optimization and multifault testability of multilevel logic. In *Proc. of IEEE/ACM International Conference on Computer-Aided Design (ICCAD-89)*, pages 422–425, Nov. 1989.

[14] G. D. Hachtel and F. Somenzi. *Logic Synthesis and Verification Algorithms.* Kluwer Academic Publishers USA, 1996.

[15] L. Hellerman. A catalog of three-variable or-invert and and-invert logical circuits. *IEEE Trans. on Electronic Computers*, EC-12:198–223, June 1963.

[16] S. J. Hong, R. G. Cain, and D. L. Ostapko. MINI: A heuristic approach for logic minimization. *IBM Journal of Research and Development*, 18(5):443–458, 1974.

[17] K. Keutzer. DAGON: Technology binding and local optimization by DAG mapping. In *Proc. of 24th ACM/IEEE Design Automation Conference (DAC'87)*, pages 341–347, June 1987.

[18] Z. Kohavi. *Switching and Finite Automata Theory (second edition).* TATA McGraw-Hill, 1978.

[19] K. Konishi, S. Kishimoto, B.-Y. Lee, H. Tanaka, and K. Taki. A logic synthesis system for the pass-transistor logic SPL. In *Proc. of Synthesis and System Integration of Mixed Technologies (SASIMI'96, Japan)*, pages 32–39, Nov. 1996.

[20] C. E. Leiserson, F. M. Rose, and J. B. Saxe. Optimizing synchronous circuitry by retiming. In *Proc. of 3rd CalTech Conference on VLSI*, pages 23–36, Mar. 1983.

[21] B. Lin and A. R. Newton. Minimization of symbolic relations. In *Proc. of IFIP International Conference on Very Large Scale Integration (VLSI'89)*, pages 187–196, Aug. 1989.

[22] A. Malik, R. K. Brayton, A. R. Newton, and A. L. Sangiovanni-Vincentelli. A modified apploach to two-level logic minimization. In *Proc. of IEEE/ACM International Conference on Computer-Aided Design (ICCAD-88)*, pages 106–109, Nov. 1988.

[23] S. Malik, E. M. Sentvich, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Retiming and resynthesis: Optimizing sequential networks with combinational techniques. *IEEE Trans. on Computer-Aided Design*, CAD-10(1):74–84, Jan. 1991.

[24] Y. Matsunaga. An exact and efficient algorithm for disjunctive decomposition. In *Proc. of Synthesis and System Integration of Mixed Technologies (SASIMI'98, Japan)*, pages 44–50, Oct. 1998.

[25] Y. Matsunaga and M. Fujita. Multi-level logic optimization using binary decision diagrams. In *Proc. of IEEE/ACM International Conference on Computer-Aided Design (ICCAD-89)*, pages 556–559, Nov. 1989.

[26] G. D. Micheli. *Synthesis and Optimization of Digital Circuits.* McGraw-Hill, 1994.

[27] G. D. Micheli, R. K. Brayton, and A. Sangiovanni-Vincentelli. Optimal state assignment of finite state machine. *IEEE Trans. on Computer-Aided Design*, CAD-4:269–285, July 1985.

[28] S. Minato. Fast generation of prime-irredundant covers from binary decision diagrams. *IEICE Trans. Fundamentals*, E76-A(6):967–973, June 1993.

[29] S. Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proc. of 30th ACM/IEEE Design Automation Conference (DAC'93)*, pages 272–277, June 1993.

[30] S. Minato. Fast factorization method for implicit cube set representation. *IEEE Trans. on Computer-Aided Design*, CAD-15(4):377–384, Apr. 1996.

[31] S. Minato. Finding all simple disjunctive decompositions using irredundant sum-of-products forms. In *Proc. of IEEE/ACM International Conference on Computer-Aided Design (ICCAD-98)*, pages 111–117, Nov. 1998.

[32] S. Minato, N. Ishiura, and S. Yajima. Shared binary decision diagram with attributed edges for efficient Boolean function manipulation. In *Proc. of 27th ACM/IEEE Design Automation Conference (DAC'90)*, pages 52–57, June 1990.

[33] R. Murgai, N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli. Improved logic synthesis algorithms for table look up architecture. In *Proc. of IEEE/ACM International Conference on Computer-Aided Design (ICCAD-91)*, pages 564–567, Nov. 1991.

[34] S. Muroga and T. Ibaraki. Design of optimal switching networks by integer programming. *IEEE Trans. on Computers*, C-21(6):573–582, June 1972.

[35] S. Muroga, Y. Kambayashi, H. C. Lai, and J. N. Culliney. The transduction method—design of logic networks based on permissible functions. *IEEE Trans. on Computers*, C-38(10):1404–1424, June 1987.

[36] T. Nanya, A. Takamura, M. Kuwano, M. Imai, M. Ozawa, I. Fukasaku, Y. Ueno, F. Okamoto, H. Fujimoto, O. Fujita, M. Yamashina, and M. Fukuma. TITAC-2: A 32-bit scalable-delay-insensitive microprocessor. In *Proc. of HOT CHIPS IX*, pages 19–32, Aug. 1997.

[37] R. Panda and F. N. Najm. Technology-dependent transformations for low-power synthesis. In *Proc. of 34th ACM/IEEE Design Automation Conference (DAC'97)*, pages 650–655, June 1997.

[38] W. V. Quine. A way to simplify truth functions. *Amer. Math. Mon.*, pages 627–631, Nov. 1955.

[39] R. K. Ranjan and J. Sanghavi. CAL-2.0: Breadth-first manipulation based BDD library. Public Software. `http://www-cad.eecs.berkeley.edu/Research/cal_bdd`.

[40] J. Roth and R. Karp. Minimization over boolean graphs. *IBM Journal*, pages 227–238, Apr. 1962.

[41] R. Rudell and A. Sangiovanni-Vincentelli. Multiple-valued minimization for pla optimization. *IEEE Trans. on Computer-Aided Design*, CAD-6:727–751, Sept. 1987.

[42] T. Sasao. Fpga design by generalized functional decomposition. In T. Sasao, editor, *Logic Synthesis and Optimization*, pages 233–258. Kluwer Academic Publishers, 1993.

[43] H. Sawada, T. Suyama, and A. Nagoya. Logic synthesis for look-up table based FPGAs using functional decomposition and boolean resubstitution. *IEICE Trans. Information and Systems*, E80-D(10):1017–1023, Oct. 1997.

[44] F. Somenzi et al. CUDD: CU decision diagram package. Public Software. `http://vlsi.colorad.edu/~fabio/CUDD`.

[45] V. Tiwari, P. Ashar, and S. Malik. Technology mapping for low power. In *Proc. of 30th ACM/IEEE Design Automation Conference (DAC'93)*, pages 74–79, June 1993.

[46] UC berkeley design technology warehouse. `http://www-cad.eecs.berkeley.edu/`.

[47] T. Villa and A. Sangiovanni-Vincentelli. NOVA: State assignment of finite state machines for optimal two-level logic implementations. *IEEE Trans. on Computer-Aided Design*, CAD-9:905–924, Sept. 1990.

[48] S. Yamashita, H. Sawada, and A. Nagoya. A new methods to express functional permissibilities for LUT based FPGAs and its applications. In *Proc. of IEEE/ACM International Conference on Computer-Aided Design (ICCAD-96)*, pages 254–261, Nov. 1996.

[49] S. Yamashita, H. Sawada, and A. Nagoya. New methods to find optimal non-disjoint bi-decompositions. In *Proc. of Asia and South Pacific Design Automation Conference (ASPDAC-98)*, pages 59–68, Feb. 1998.

[50] 宮沢, 中越, 今井, R. Morizawa, M. Ozcan, 中村, 南谷, 高橋, 若林. 非同期式 VLSI 設計用 CAD システムの提案. 情処研報, SLDM-101(42), May 2001.

[51] 笹尾勤. 論理設計 — スイッチング回路理論 —*(第 2 版)*. 近代科学社, 1998.

[52] 山下茂, 澤田宏, 名古屋彰. SPFD: 論理関数の自由度の新しい表現方法. 通信学会論文誌, J82-A(7):1047–1056, July 1999.

[53] 室賀三郎 (笹尾勤 訳). 論理設計とスイッチング理論 — *LSI, VLSI* の設計基礎. 共立出版, 1981.

[54] 小栗清. 非同期論理回路の設計技術. 情処研報, SLDM-102(本研究会), Nov. 2001.

[55] 松永裕介, 藤田昌宏. 順序つき 2 分決定グラフと許容関数を用いた多段論理回路簡単化 手法. 通信学会論文誌, J74-A(2):196–205, Feb. 1991.

[56] 藤原秀雄. コンピュータの設計とテスト. 工学図書, 1990.

[57] 藤田昌宏. トランスダクション法に基づく多段論理回路簡単化機能をもつ論理合成システムとその評価. 情報処理学会論文誌, 30(5):613–623, May 1989.

[58] 藤田昌宏, 佐藤政生 編. 特集「BDD (二分決定グラフ)」. 情報処理, 34(5):584–630, May 1993.

[59] 湊真一. 算術論理式計算プログラム BEM-II. Public Software. 人工知能学会 10 周年記念 CD-ROM(人工知能学会誌 1996 年 5 月号折込付録).

[60] 湊真一. ゼロサプレス型 BDD を用いた非明示的積和形表現の高速因数分解 法. 通信学会技術研報, VLD93-105(504):55–62, Mar. 1994.

[61] 湊真一. BDD(二分決定グラフ) とその応用. 応用数理, 9(3):2–14, Sept. 1999.

[62] 名古屋彰, 中村行宏, 小栗清, 野村亮. 高位記述からの大規模論理合成における多段論理最適化. 通信学会論文誌, J74-A(2):206–217, Feb. 1991.