第4章　機能・論理設計3

―論理合成―

若林　一敏 （NEC）

本章では、システムLSIの上流設計である「機能設計工程」と「論理設計工程」の自動化手法について述べる。機能設計は、システムLSIの動作の記述から、それを実現するハードウェアの概略構造（ブロック図、マイクロアーキテクチャ）を決定する工程である。それに続く論理設計工程は、上記概略構造を詳細なゲート回路図に変換する工程である。

（注意：上記2つの上流の設計工程の両方を総称して、論理設計工程と呼ぶ場合もある。その場合は、ゲート回路設計を行う工程を、「詳細論理設計工程」と呼んで区別することもある。）

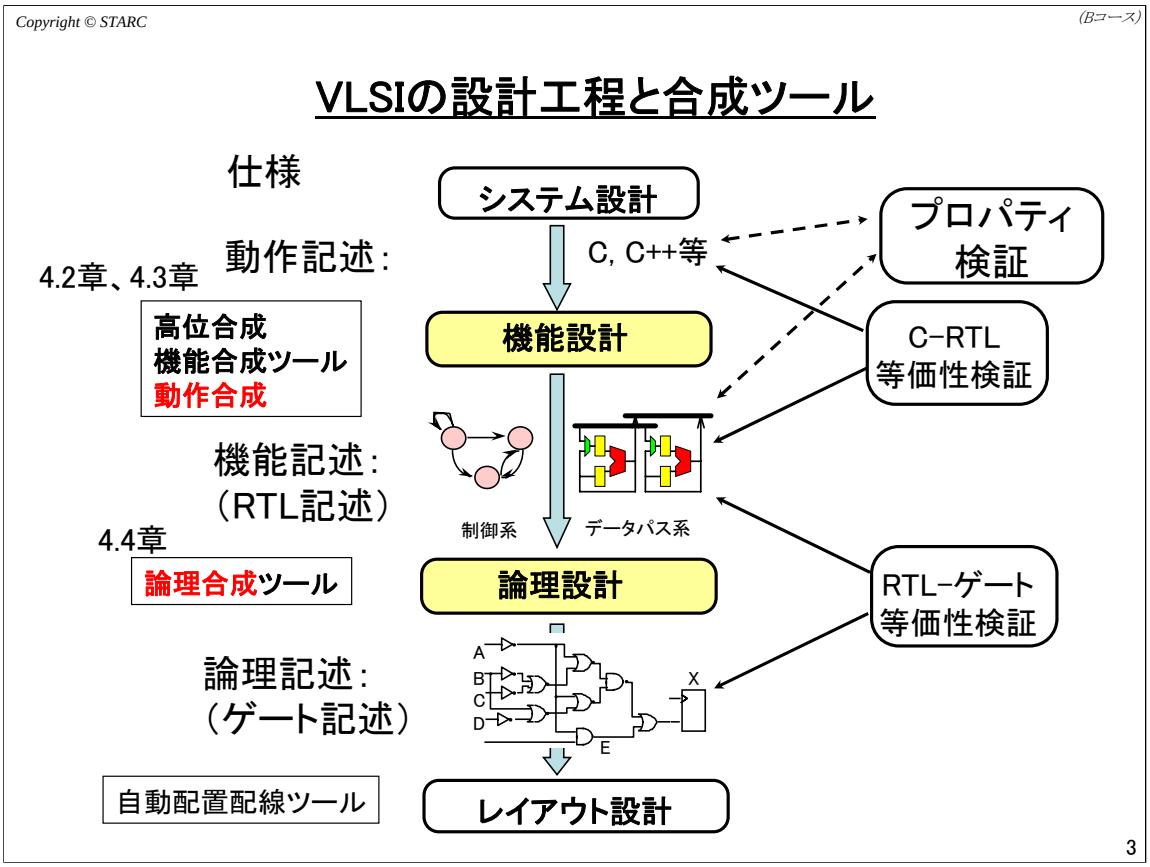
4.4 論理合成

4.4.1 順序回路生成・最適化

4.4.2 組み合わせ回路最適化

4.4.3 論理合成ツール

本節では、RTL記述からゲート回路を合成する論理合成手法について述べる。論理合成手法は、順序回路の生成、最適化と、組み合わせ回路の最適化の2つに分けられ、それぞれについて4.1節、4.2節で説明する。最後に4.3節で、RTL記述からゲート回路を合成する論理合成ツールの構成について述べる。



4.1章で述べたシステムLSIの設計工程とツールの図を再掲する。論理合成ツールは、人手のRTL、または、動作合成が生成したRTLから、ゲート回路を生成するツールである。レジスターレジスタ間の遅延制約を満たす範囲で、最小面積のゲート回路を生成することを主な目的とする。

論理合成技術の分類

順序回路合成 （状態遷移から制御回路の生成）	・状態数最小化 ・状態符号化 ・状態遷移関数作成
組み合わせ回路合成 （データパスや制御回路のゲート回路の最小化）	ブール式の簡単化 （テクノロジ独立最適化） ・2段論理合成（PLA向き） ・多段論理合成（ASIC向き）
	ライブラリゲートによる回路生成 （テクノロジ依存最適化） ・テクノロジマッピング

論理合成技術は、順序回路合成と組み合わせ回路合成の二つに分けることができる。順序回路合成は、制御回路を実現するステートマシンをワイアードロジックで実現する手法である。状態数の最小化や、状態符号化、状態遷移関数の作成からなる。組み合わせ回路合成は、ブール式の最小化であるテクノロジ独立な最適化と、ブール式をライブラリゲートの組み合わせで実現するテクノロジ依存最適化がある。「テクノロジ」は、半導体の0.13ミクロン等のプロセスルールを示し、この場合は、あるテクノロジのライブラリセルを使った回路か、ブール式かという区別である。ブール式の簡単化には、2段論理式の最小化、多段論理式の最小化の二つがある。

順序回路の合成と、組み合わせ回路の合成のどちらがより効果が大きいというようなことは一般には言えない。但し、現在の市販の論理合成ツールは組み合わせ回路の最適化を中心に行っており、順序回路の最適化はあまり行っていない場合が多い。これは、順序回路の最適化自体が難しいという側面もあるが、順序回路の最適化を行うと、レジスタの数が変更してしまい、次の章で述べる、RTLとゲートの等価性証明や、RTLでのシミュレーション、デバッグが行いにくくなるという検証面の都合にも大きくかかわっている。

4.4 論理合成

4.4.1 順序回路生成・最適化

状態数最小化、状態符号化

リタイミング

4.4.2 組み合わせ回路最適化

論理式の内部表現形式

2段論理最適化

多段論理最適化

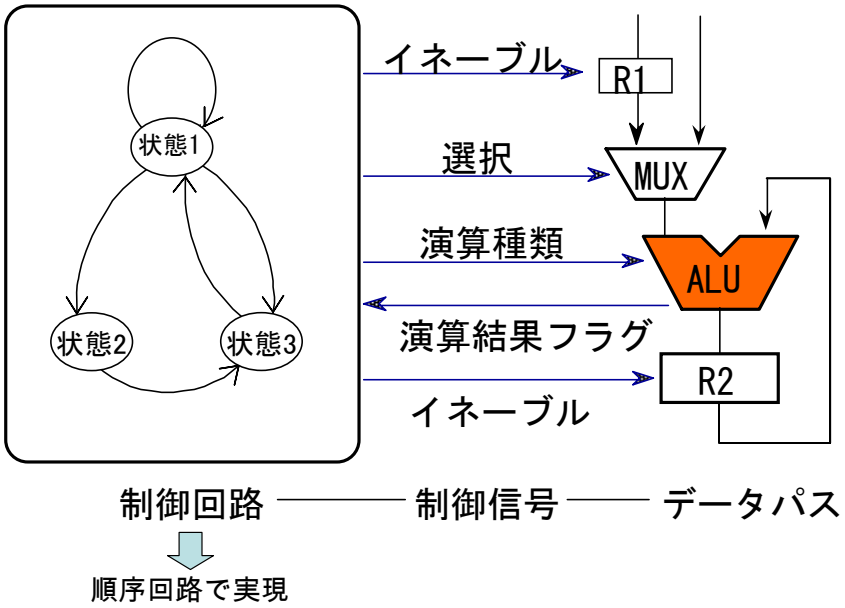
4.4.3 論理合成ツール

順序回路の合成技術。順序回路は、制御回路等に利用される。

カウンターを改造して、多少込み入った状態遷移を実現する方法がかつて利用されていたが(シーケンサの実現方法などとして現在でも紹介している本はある。)、本章で紹介するのは、任意の形状の状態遷移を実現する組織的な手法である。論理合成ツールが手軽に利用できるようになった現在では、カウンタを改造して状態遷移を作り出すような手法をとる必要はない。(ただし、注意深く行わないと、タイミング問題を起こしやすく、デバッグや修正も行い難い。)

制御回路とデータパス

（一般的な回路のRTL構造）

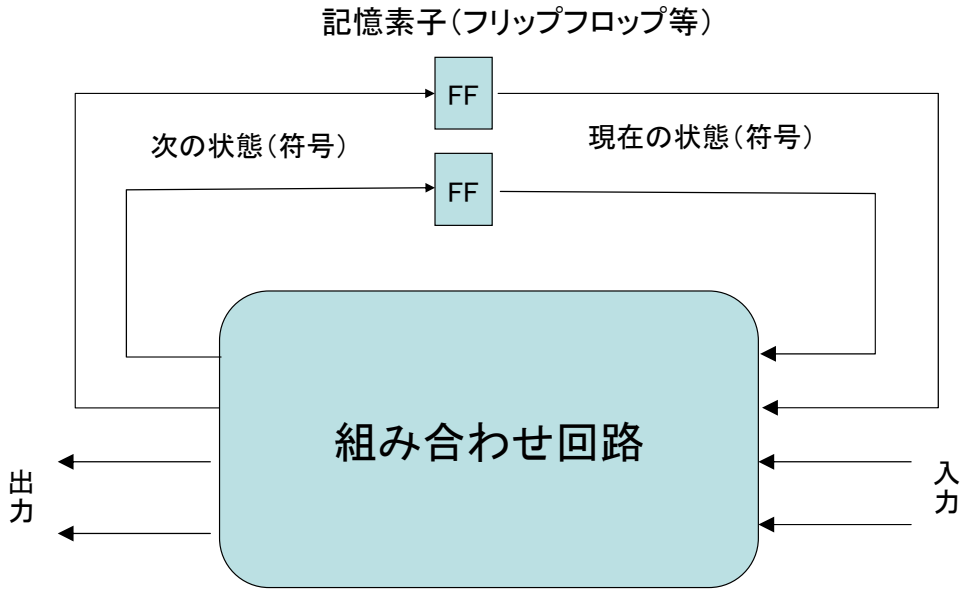


制御回路 ——— 制御信号 —— データパス

順序回路で実現

図は、制御回路とデータパスからなる、一般的な回路のRTL構造を示している。この図の左側の状態遷移は、順序回路で構成できる。順序回路は、フリップフロップ（レジスタ）と組み合わせ回路で実現できる（ワイアードロジック実現）。また、データパスも、データを保持するレジスタと、演算器、セレクタ、専用の組み合わせ回路からなる。つまり、RTL構造をゲート回路にするためには、組み合わせ回路を設計すればよい。（制御回路は、プログラムカウンタを用いた構成でも実現が可能である。この方法では、2分岐程度が普通で、多分岐を実現するのは難しい。現在、論理合成では、ワイアードロジック実現による方法だけを扱っており、汎用プロセッサでも、現在はワイアードロジックによる方法が主流である。）

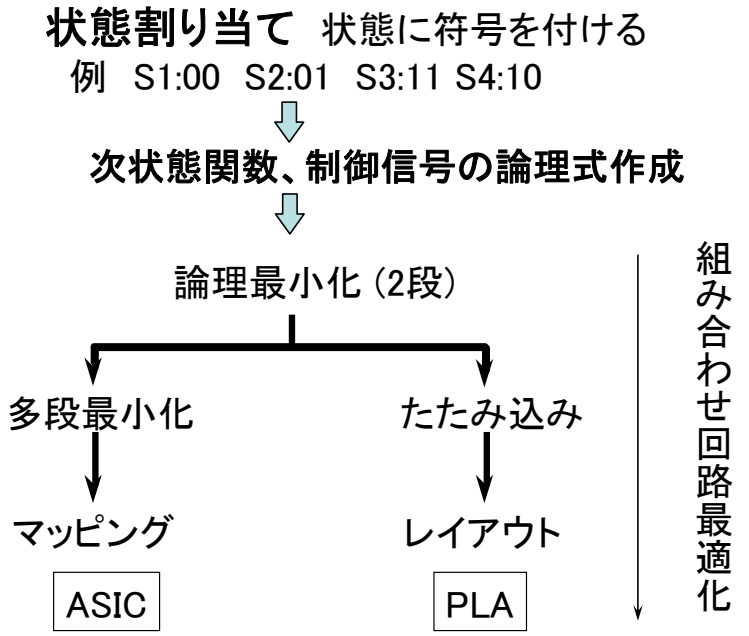
順序回路の構成：記憶素子と組み合わせ回路



順序回路をワイアードロジックで実現する場合、フリップフロップ等の記憶素子で状態を表し、組み合わせ回路に入力される信号と現在の状態を論理演算することで、次の状態と出力信号を計算する。論理合成では、記憶素子としてエッジトリガタイプのD-FF（クロックの立ち上がりエッジで、D入力を保持し、出力する）を用いる。一つのFFで、0と1の二つの状態を表すことができる。よって、N個の状態がある場合、log N個以上の整数のFFがあれば表現可能。また、多くてもN個のFFで、表現可能である。たとえば、状態数が4ならば、2個（00, 01, 10, 11)から4個(0001, 0010, 0100, 1000)のFFで、状態が表現できる。一般に、多くのFFを利用した方が、FFの面積が大きくなるが、状態をデコードする遅延時間が短くなる。

（省）

順序回路の合成

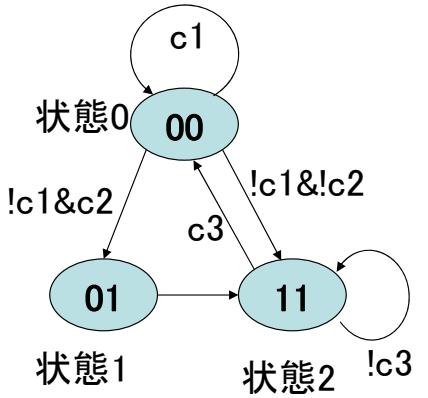


順序回路の合成では、まず、フリップフロップの集合（レジスタ）で表される状態に符号をつける(状態のエンコード)。たとえば、状態S1を00、状態S2を01等である。状態の符号はユニークである必要がある。符号化には、2進数やグレイコード、one hot encodingなど様々なエンコード手法がある。この状態のエンコードにより、制御回路部分の面積や遅延、電力が異なる。これらの中で、大規模な順序回路の時に重要なのは、遅延である。one hot encoding(各状態の中で1が一つだけしかない状態割り当て）は、多くの状態レジスタを必要とするが、高速に動作することが知られている。実用回路で利用されることも多い。次状態関数、制御信号等の論理式は、組み合わせ回路合成で実現される。この論理式は、まずAND-ORの2段論理として最適化されることが多い。PLAで回路を実現する場合は、この2段論理のまま実現される。ASICやFPGAで実現される場合は、2段論理をより面積の小さな多段論理に変換し、ライブラリセルにマッピングしてゲート回路とする。

なお、設計者が記述するRTLでは、状態はシンボリックな状態名でも良いし、人手で符号化したものを記述しても良い。

シンボリックな状態名がつけられた場合のみ、「状態割り当て」が実施される。動作合成ツールは、状態割り当て済みのRTLも、割り当て前のシンボリックな状態名のRTLも出力することが可能である。

順序回路(制御回路)生成



制御回路の状態遷移図

1)状態割り当て
状態0:00, 状態1:01, 状態2:11
2)状態遷移の論理を生成
状態レジスタSTの論理を生成

```
switch(ST){
      case 00: if (c1)  ST= 00 ;  (次状態)
                  else  {
                     if (c2) ST= 01; (次状態)
                   else   ST= 11}  (次状態)
               st0d=1;  (状態デコード信号)
      case 01:            ST= 11;  (次状態)
                  st1d=1;  (状態デコード信号)
      case  11:  if (c3) ST= 00;  (次状態)
                   else     ST= 11;  (次状態)
                  st2d=1;  (状態デコード信号)
}
```


制御回路を順序回路(順序機械)で構成する方法を例を用いて説明する。

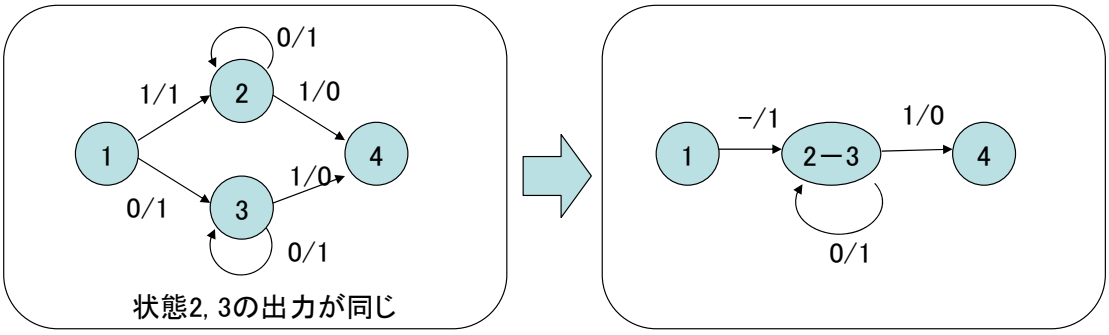
まず、状態を表すレジスタを用意する。図の例では、状態が3つあるから、2ビットのレジスタを用意し、00, 01, 11と状態割り当てを行う(2ビットであるので4状態表せるから、10は利用されない状態となる。正常回路では遷移しない状態であるため、In-valid状態などと呼ばれる)。次に、図の状態遷移を行うような論理を作成する(順序回路理論の次状態関数の作成に相当)。図では状態レジスタSTが次にどの値をとるか(どの状態に遷移するか)をプログラム形式で示した。

たとえば、現在STが00(状態0)にいる時、c1が真なら、次も00。c1が偽でc2が真なら01(状態1)に変化、c1、c2共に偽なら11(状態2)へ変化する。(この記述はRTLである。)他の状態も同様である。この制御回路からは、現在の状態を示す状態デコード信号が出力される。たとえば、st0dは、現在制御が状態0にあることを示す。(これは順序回路の出力関数に相当する。FSMDでは、順序回路の出力関数は通常状態デコード信号のみである。)

（省）

状態数の最小化

**仕様として与えられた状態遷移図（FSM：有限状態機械）が、
冗長な場合等に、等価な状態数の少ないFSMに置き換えること。**



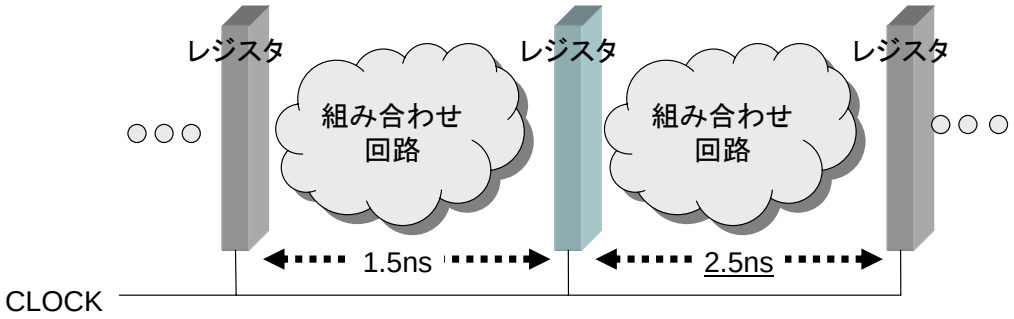
**1）現在の、論理合成ツールでは、順序回路の最適化はあまり利用されていない。
（状態割付機能は持っているが、デバッグ等の都合で、人手で行うことが多い。）**
2）動作合成を利用した場合、その出力RTLの制御回路には、冗長な状態は（通常）無い。

状態数の最適化の例を示す。FSM1では、状態1から、入力が1の時、状態2へ遷移、入力が0の時、状態3へ遷移する。状態2, 3は、同じ入力に対し、状態遷移と出力が同一である。（入力が0の時は、出力が1で次状態が自分自身。入力が1の時は、出力が0で、次状態が状態4。）結局、状態2と3は区別する必要がなく、ひとつの状態にして良い。この変換を行ったものが、FSM2である。状態数が4から3とひとつ小さくなっている。

（実際の状態数最小化は、この例より複雑な最適化も行う。たとえば、状態が6つ（S1からS6）まであるものを、4つの状態T1(S1, S2)、T2(S3)、T3(S2, S4, S5)、T4（S5, S6）に最小化する等。これらの詳細は順序機械論（参考文献「6」）等を参照のこと。）

状態数の削減により、ゲート数の削減を期待している。ただし、現在の数百万ゲートクラスの設計では、このような最適化の効果はあまり期待できない。現実に、ほとんどの論理合成ツールは、このような最適化を行っていない。これは、最適化のための計算量（計算時間）が膨大な割りに、得られるゲート削減や遅延削減効果が小さいことも一因である。

リタイミング(1)



最大遅延は2.5ns
→周波数は 1 / 2.5 ns = 400 MHz が限界

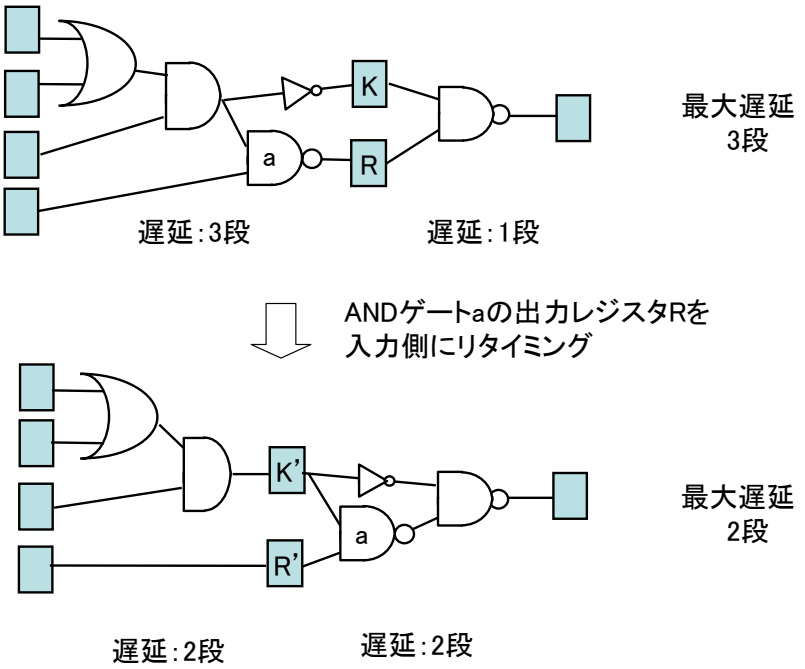
⇩

真ん中のレジスタ(FF)の位置を変えて、
それぞれの遅延を2nsにそろえられれば、1/2ns=500MHzに

同期式回路のクロック周波数を決定するのは、レジスターレジスタ間の組み合わせ回路部分の最大遅延時間である。

上記の図では、1.5nsと2.5nsの遅い方の2.5nsでクロック周波数が決定される(1/2.5ns=400MHz)。真ん中のレジスタを右側に移動して、遅延を0.5ns、右の組み合わせ回路から左へ移せれば、1/2ns=500MHzとすることができる。このように、レジスタの位置を変えることで、論理を変えることなくクロック周波数を増加する手法をリタイミングと呼ぶ。動作合成のスケジューラでは、演算器やマルチプレクサといったRTL部品レベルを単位として考えていたが、リタイミングでは、ゲート回路レベルでレジスタの位置を考える。(ゲートレベルのスケジューリングとも考えられる。)

リタイミング(2)



リタイミングの具体例を示す。上の図では、レジスタ間の組み合わせ回路の最大遅延は、ゲート3段分である。よって、周波数はこの3段で決定される。そこで、NANDゲートaの出力につながるレジスタRとインバータの出力レジスタKを、それぞれ、NANDゲートとインバータの前に移動(リタイミング)する。RはNANDゲートの入力に移動されるから二つに増加するが、この場合はちょうど、レジスタK'と同じ論理であるから、共用できる。よって、この例では、レジスタ増加による面積増加がなく、クロック周波数を向上できた。リタイミングは、プロセッサのパイプライン演算器等では利用可能であるが、一般の論理合成ではそれほど多用されない。RTL記述を入力とする場合、リタイミングをしてしまうと、レジスタの数などが変わってしまい、ゲート回路とRTLのシミュレーション結果の突合せなどが難しくなり、デバッグも困難になることなどが理由である。但し、リタイミングの前後は、形式的検証によって、比較的容易にその等価性を証明可能であるため、リタイミング自体の正しさの証明は容易である。

4.4 論理合成

4.4.1 順序回路生成・最適化

状態最小化、状態符号化

リタイミング

4.4.2 組み合わせ回路最適化

論理式の内部表現形式

2段論理最適化

多段論理最適化

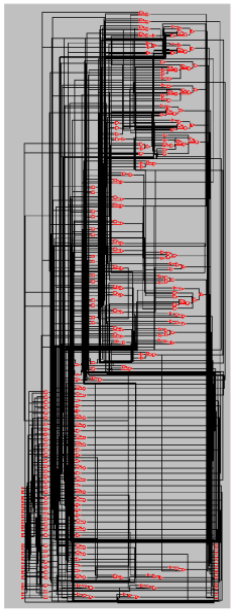
テクノロジマッピング

4.4.3 論理合成ツール

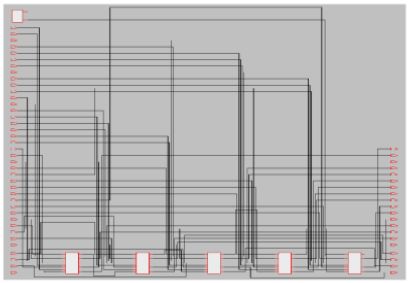
演算器やセレクタ等は組み合わせ回路で実現される。定数配列も組み合わせ回路で実現できる。(アドレスを入力とし、値を出力とする真理値表と考えればよい。)このほか、順序回路の次状態関数やデコーダ等も組み合わせ回路で実現される。この組み合わせ回路を最適化する手法を紹介する。

演算器をゲート回路へ

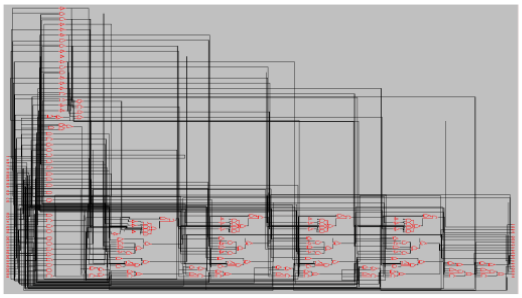
～19bit　加算器の合成例～



BLC
322cells
8ns
高速加算器



4bit H/M * 5
161cells
19ns
リップルキャリー方式



CLA
262cells
14ns
キャリールックアヘッド方式

加算、乗算等の算術演算などは、あらかじめ任意ビットの構成方法を登録したライブラリを参照して、ゲート回路を出力する。どのような演算器構成を選択するかであるが、基本的には演算器のゲート構成が遅延制約を満たすように決定する。通常は遅延制約を満たす中で最小面積の回路構成を、あらかじめ登録された回路の中から求める。図の例は、19ビットの加算器を、高速加算器(BLC，左)、4ビットのハードマクロ(H/M)を5つ利用したリップルキャリー加算器(右上)、キャリールックアヘッド方式(CLA，右下)でそれぞれ合成したゲート回路図である。図の数字は、加算器の実行時間と面積を示す。BLCはもっとも高速だが面積も最大である。H/Mを利用したリップルキャリー加算器は最小面積だが遅延が最大である。CLAは遅延、面積とも中間的である。このように、同じビット数の加算器でも、さまざまな速度や面積のものを合成することができる。

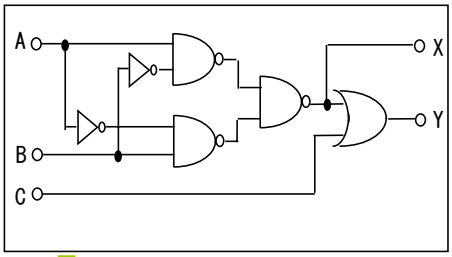
BLC = Binary Look ahead Carry

CLA = Carry Look Ahead

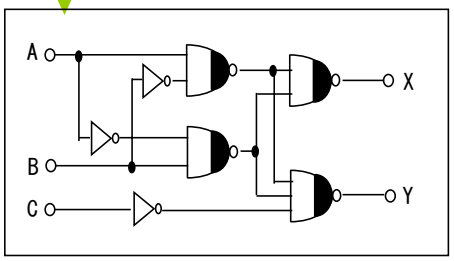
組み合わせ回路合成＝変換と最適化

```
architecture RT of MAC1 is
  signal A,B,C,X,Y : BIT ;
  begin
   X <= (A and (not B)) or
    ((not A) and B ) ;
   Y <= (A and (not B)) or
    ((not A) and B) or C ;
end RT ;
```

変換



最適化＆
マッピング

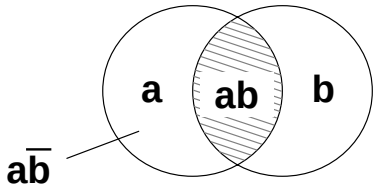


組み合わせ回路合成の概略の流れを示す。

最初に、左上に記述されたRTL記述（VHDL）が、その右側にある論理式（ゲート回路図）へ「変換」される。次に、論理式（ゲート回路図）を「最適化」（論理構造の変化）し、さらにライブラリセルへ「マッピング」（割付）して、下のような最終的なゲート回路を得る。このような工程を経て、RTL記述がVLSIの部品からなる回路に変換される。CMOS回路では、AND，OR回路より、NAND，NOR回路が面積・遅延の点で有利であるため、多用される傾向にある。以下、最適化とマッピングの手法を述べる。

ブール論理式の簡単化

$ab + a\bar{b} = a$



2段論理の簡単化の例

$f = a\bar{b}cd + \bar{a}\bar{b}cd + \bar{a}bcd + \bar{a}\bar{b}cd + \bar{a}\bar{b}\bar{c}d$

$f = b\bar{c}d + \bar{a}cd + \bar{a}\bar{b}d$

（注）+は、論理ORを示す

組み合わせ回路の最適化は、古典的なブール式の最小化問題の拡張である。最も、単純なものは、AND-ORの積和形式（SOP：SUM-OF-PRODUCT）の最小化問題である。論理積abと論理積ab’の論理和は、ベン図から簡単にaとわかる。つまり、ab+ab’=aとなる。これは、同じブール式（この場合はa)に、ある変数（この場合はb)をそのままANDしたものと、その変数の否定をとってANDしたものをORした場合、その変数は削除できると法則化できる。この原理を下の積和論理式に当てはめると、次の様に簡単化が図れる。fの最初の2つの項は、bcd’というブール式に、aとa’をANDしたものがORされているので変数aを削除でき、bcd’と簡単化できる。以下、同様である。なお、ブール代数では同じブール式を何度ORしても論理が変換しないことに注意されたい。

ブール式の簡単化は一般の教科書（参考文献[8]等）に詳しくでているため、本章では簡単に概要に触れるにとどめる。

4.4 論理合成

4.4.1 順序回路生成・最適化

状態最小化、状態符号化

リタイミング

4.4.2 組み合わせ回路最適化

⇨ 論理式の内部表現形式

2段論理最適化

多段論理最適化

テクノロジマッピング

4.4.3 論理合成ツール

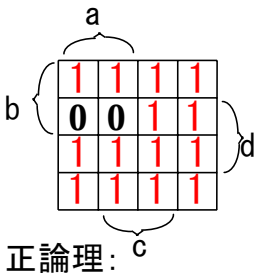
次に組み合わせ回路の最適化手法について説明する。最初に、論理式を計算機内部で表現する各種方式を紹介する。

（省）

正論理と負論理

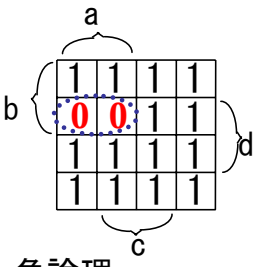
・負論理で記述した方が簡潔に記述できる場合がある

→ 合成ツールも最適化しやすい



正論理：

f=a*b*!c*!d+a*b*c*!d+
!a*b*c*!d+!a*b*!c*!d+
!a*b*c*d+ …
=？



負論理：

f=!(a*b*!c*d+a*b*c*d)
=!(a*b*d)

論理を記述する際に、1になる論理を中心に記述する方法と、0になる論理を中心に記述する方法がある。

前者を正論理、後者を負論理と呼ぶ。どちらで表現するかによって、論理合成結果が異なることもある。

（値が0の場合に条件が成立したと考えることを負論理と呼ぶ。たとえば、RESET信号が0になるとリセット状態になる場合に、この信号は負論理と呼ぶ。）

左の正論理では1を成立と考え、1になる最小項をORしている。右の負論理では0を成立と考え、0になる最小項（この場合は2つ）をORしている。（最後に全体の否定をとって、正論理と同じ論理にしている。）この場合、右の記述の方が簡単に簡単化でき、先に述べたブール式の簡単化によって変数cが削除でき、簡単に!(a*b*d)という解が求まる。（論理合成ツールを使わずに手で最適化していた時代にはこのような工夫は重要であった。現在では重要度は減っては来ているが、論理合成ツールの性能によって合成結果が異なるので、できるだけ簡素なブール式を与える方が良いことには代わりがない。）

論理式の内部表現形式

論理式　　真理値表　　カルノー図　　BDD

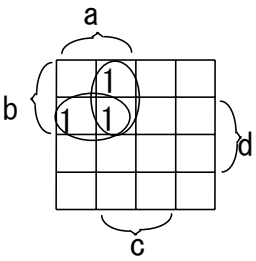
(Binary Decision Diagram)

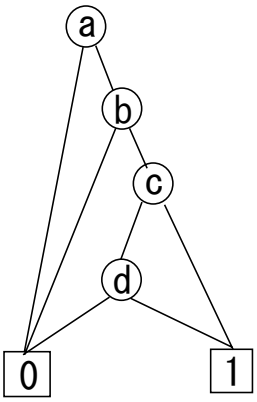
積和形式
(2段論理)
f = abc + abd

ファクター形式
(多段論理)
f = ab(c+d)

上記式で
ANDは省略
+ はORを示す

abcd	f
0000	0
0001	0
0010	0
0011	0
0100	0
0101	0
0110	0
0111	0
1000	0
1001	0
1010	0
1011	0
1100	0
1101	1
1110	1
1111	1





これらはすべて同じ論理を表現

論理関数を表現するのには図のように様々な方法がある。真理値表やカルノー図は設計者が考えるときにはわかりやすいが、大規模な論理を表現するのに不向きであり、論理合成ツールで利用されることはまず無い。BDD(検証の章参照)も、合成の段階の一部で利用されることもまれにあるが、最近では形式的検証系以外ではあまり利用されない。論理合成ツール内部の基本的なデータ構造は、論理式をベースにしたブーリアンネットワーク(P.26参照)と呼ばれるデータ構造が利用されることが多い。動作合成ツールのCDFG同様、そのデータ構造は各ツールごとにそれぞれ異なっている。

4.4 論理合成

4.4.1 順序回路生成・最適化

状態最小化、状態符号化

リタイミング

4.4.2 組み合わせ回路最適化

論理式の内部表現形式

⇨ 2段論理最適化

多段論理最適化

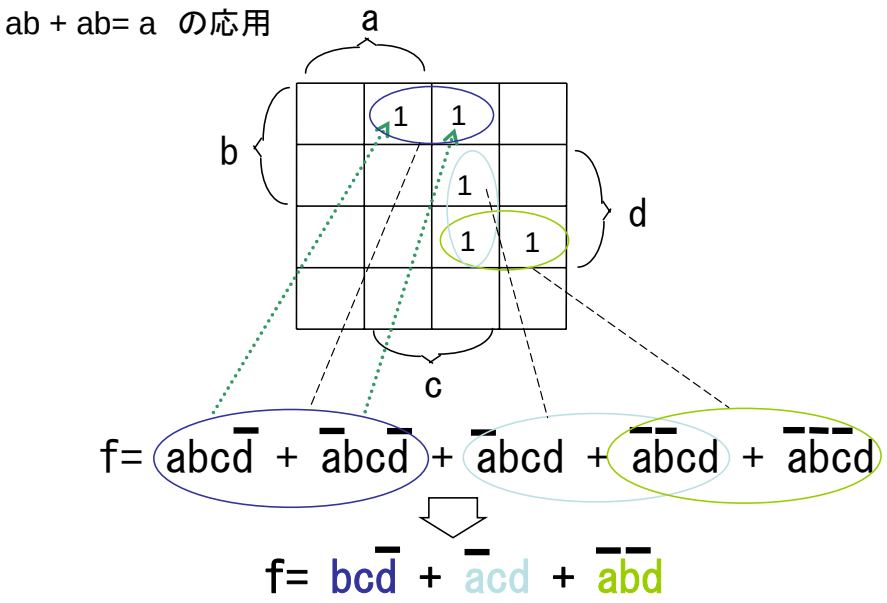
テクノロジマッピング

4.4.3 論理合成ツール

組み合わせ回路の最適化のうち、論理式の最適化(テクノロジセルを考えない、論理変数だけでの最適化)には、2段論理最適化と多段論理最適化の2つの手法がある。2段論理最適化は、扱いが易しいことから、多くの最適手法、ヒューリスティック手法が提案されている。(但し、実用的な回路を2段論理で表現することは、規模の点から不可能である。)

本節では、2段論理最適化の基礎を示す。2段論理最適化については、多くの教科書がでているので、そちらを参考にしてほしい。

論理最適化(2段論理最適化)

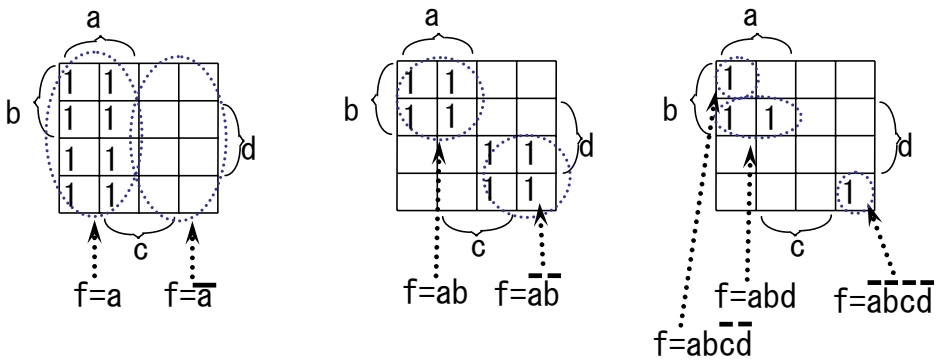


論理最適化の最も古典的な課題は、2段論理(積和形、SOP:SUM-OF-PRODUCT)の最適化である。計算機で実行可能な手続きとして、クワインマクラスキー法等様々な方法が提案された。しかし、ある程度以上大規模な回路では、2段論理で回路を表現することは不可能なため、論理の一部を最適化するのに使われる。たとえば、後述するブーリアンネットワーク上の2段論理をそれぞれ最小化する場合等に使われる。ブーリアンネットワークのいくつかのノードを一つにまとめ、2段論理に展開した後に、2段論理最適化をかけるといった際にも利用される。

カルノー図

ab + a$\overline{b}$= a の応用

1の個数が2のべき乗個になる最大の矩形でカバーする



・4変数までの論理を簡単に机上で最適化できる。

上記式で
ANDは省略
+ はORを示す

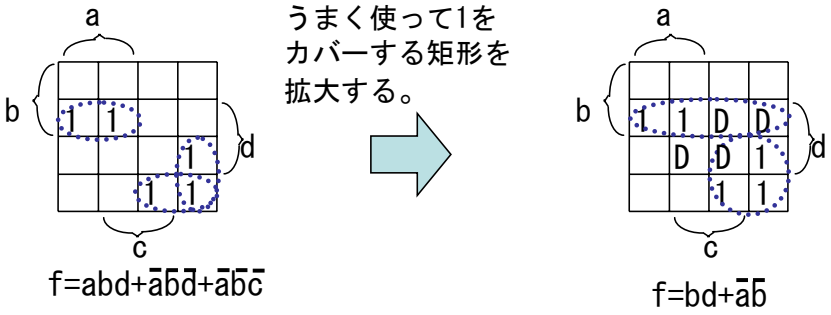
カルノー図は組み合わせ回路の最小化を表を用いて簡易に行える手法である。ab+ab'=aの簡単化を視覚的に行えるように工夫された図である。図で空白は0を示し、変数a, b, c, d付きの括弧が示されているが、括弧で示された部分はその変数が1であることを示している。たとえば、左端のカルノー図で一番左上の部分は、a, bが1で、c, dが0であることを示している。左端の図は、aの括弧で囲われた部分がすべて「1」となっているので、この論理式は直ちにf=aとわかる。真ん中の図はaとbの括弧で囲われた左上の4つの箱が「1」となっており論理はab、右下の4つの箱が「1」になっている部分はaで囲われてない部分とbで囲われていない部分のANDとなっているので、a'b'となる。

図に示す通り、4変数程度までの回路の簡単化に利用可能であるが、それ以上の変数を含む式にはあまり適さない。(もちろん、論理式を簡単化するだけであり、遅延を最適化する、即ちゲート段数を最適化することは難しい。)

論理合成ツールでは、この考え方を一般化させ、多変数にも対応可能なヒューリスティック算法であるESPRESSO(参考文献[9])等が利用されている。

ドントケア

·回路の動作を規定する必要の無い条件。つまり、どのように動作しても構わない条件。
·ドントケアをうまく用いると回路を効率よく実現できる。



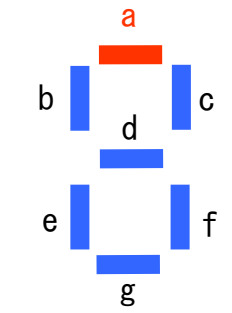
真理値表等であたえられる論理空間で、出力が0でも1でもよい入力の組み合わせをドントケア(冗長)と呼ぶ。

ドントケアを利用すると回路をより小さくできる。左の図は1になる条件だけで論理を形成した。右図では、ドントケア部分をうまく0と1に割り当て、論理表現を簡易化することができている。

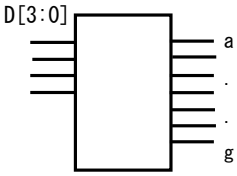
補足:

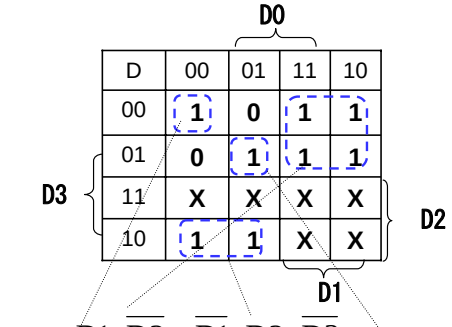
ここで説明したドントケアは外部入力や外部出力のドントケア条件を示しているため、より詳細に外部ドントケアと呼ぶ場合がある。多段論理式表現をした場合の中間変数に関係するドントケアを内部ドントケアと表現するようになり、通常のドントケアを外部ドントケアと呼ぶこともある。詳しくは後述する。

ドントケアの例



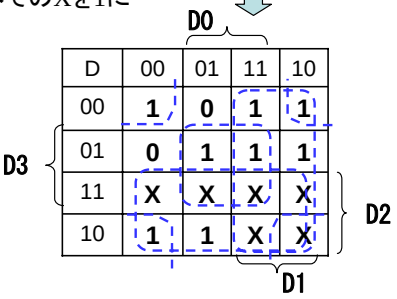
	D	abcdefg
0	0000	1110111
1	0001	0010010
2	0010	1011101
3	0011	1011011
4	0100	0111010
5	0101	1101011
6	0110	1101111
7	0111	1010010
8	1000	1111111
9	1001	1111010
10	1010	XXXXXXX
	....	
15	1111	XXXXXXX





$$a = \overline{D1} \cdot \overline{D2} + \overline{D1} \cdot D2 \cdot \overline{D3} + \overline{D0} \cdot \overline{D1} \cdot \overline{D2} \cdot \overline{D3} + D0 \cdot \overline{D1} \cdot \overline{D2} \cdot D3$$

すべてのXを1に



$$a = D1 + D2 + D0 \cdot D3 + \overline{D0} \cdot \overline{D3}$$

上記の7セグメントの例では、0から9までの値しか表示されないため、入力D（4ビット）が10以上になる組み合わせはすべてドントケア（冗長）である。このドントケアの値を適切に決定すると論理式のリテラル数を最小化することができる。真理値表で色付けした列はaを表示する回路を示している。この回路のカルノー図を右上に示す。ドントケア部分をXで示している。Xを0と考えて、1の部分だけで論理式を作成した場合（右上）と、ドントケア部分Xをすべて1とした場合（右下）の論理式の大きさを比較されたい。右下の方が小さな論理式で表現できており、実現する論理回路も小さくなると思われる。本例では、すべてのドントケア(X)を1にした場合が、最も小さな論理式を得られる。（Xは0か1のどちらかにする必要があるわけではない。X一つ一つに対して、0か1を決定していけばよい。上記の例は、説明の簡便さのために、すべて0と1とした。）

4.4 論理合成

4.4.1 順序回路生成・ 最適化

状態最小化、状態符号化

リタイミング

4.4.2 組み合わせ回路最適化

論理式の内部表現形式

2段論理最適化

⇨ 多段論理最適化

テクノロジマッピング

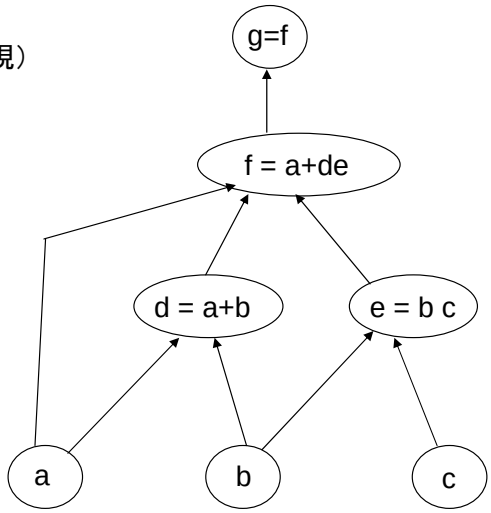
4.4.3 論理合成ツール

一般の回路は多段論理で表現される。よって、多段論理は2段論理より一般的な回路を表現しているといえる。多段の論理の最適化は非常に難しい問題であり、90年代まで精力的に研究されて、多くの方法が提案されている。本節では、ごく基本的な手法として、多段の論理式の表現方法、内部ドントケア、2段論理を多段にする方法等を示す。

論理式の内部表現形式
ブーリアンネットワーク

多段論理式をSoP（積和形で表現）

g=a+(a+b)bc;
↓
d＝a+b
e＝bc
f＝a+de
g＝ f



各ノードは2段最適化手法が
適用可能

多段論理式の内部表現として有名なものに、ブーリアンネットワークがある。各ノードに積和式をもち、これらのノードのネットワークからなる。一見、多段論理回路と全く同等の情報を持っているように見えるが、各ノード毎に処理を行う場合、中間変数（d, e, f）に対するある条件を考慮しなければならなくなる。これが、内部ドントケアであり、充足性に基づくドントケア（SDC）と観測性に基づくドントケア（ODC）の二つがある。

（詳）

多段論理のドントケア条件(1)

0）**External don't care (ExDC):外部条件に基づくDC**

禁止項、冗長項（前出）

例：3状態のFSMを2ビットのFFで→一つは起こりえない

1）**Satisfiability don't care (SDC):充足性に基づくDC**

ブーリアンネットワーク

x= a+b;
y = ax;（出力）

⇨ a=1かつx=0は起こりえない
（中間変数xのDC条件）

SDCx = x ⊕ (a+b)　(xとa+bが同じ値をとらない条件）
= a'b'x+ax'+bx';

このドントケア条件を用いて、y=ax;は、y=a;

☆もともとの多段論理を展開すると　y=a(a+b)=a+ab=a;

（注）⊕は排他的論理和を示す

先に説明したドントケアは、入力端子のありえない組み合わせ（禁止項、冗長項）を示しており、Externalドントケア(DC) と呼ばれる。（外部入力（プライマリインプット）からセット不可能な組であるCDC (Controllability DC)と、外部出力（プライマリアウトプット）から観測不能な組であるODC (Observability DC)がある。）

ブーリアンネットワークで多段論理を表現すると、このExternalドントケア以外にInternal(内部)ドントケア条件が必要となる。これは、多段回路を中間変数を導入して表現することに起因し、その中間変数に関して必要になるもので、SDCとODCと呼ばれる2つの内部ドントケアがある。

中間変数xについてのSDCxは、中間変数xとその値a+bが常に値が同一でない条件である。左辺と右辺が違うのは、明らかに起こりえない。よって、この左辺xと右辺a+bの排他的論理和(EXOR)が絶対に起こりえないドントケア条件となる。y=ax;という中間変数xを使った式を簡単化する時は、このドントケア条件が利用でき、y=aとなる。もともとの多段の論理式であれば、展開して簡単にy=aを得ることができる。つまり、ブーリアンネットワークで各ノードのみに注目して、中間変数を入力変数のように扱った場合の最適化の際に、このSDCを考えることが必要となる。（逆にいえば、小さい論理式の場合は、単純に展開したほうが高速である。最後の行参照。）ネットワーク全体に対するSDCは、各中間変数のSDCの論理和で得られる。

（詳）

多段論理のドントケア条件（2）

2）**Obsevabirity don't care (ODC):観測性に基づくDC**

x = a'bc + ab'c + abc'; 中間変数 x

y= ab+a'x+c'x'; 出力 y

◆ xの値が外部に影響を与えない条件＝ODC

a=b=1 ⇒ y = 1

a=c=0 ⇒ y = 1

a=c=1 ⇒ y = b

変数 x を0にした場合と1にした場合の出力Y（ $Y_{(x=0)}$、$Y_{(x=1)}$）が等しい場合は、x は y に影響しない（外部から観測できない）

y の x に関するODC ＝ $\overline{Y_{(x=0)} \oplus Y_{(x=1)}}$

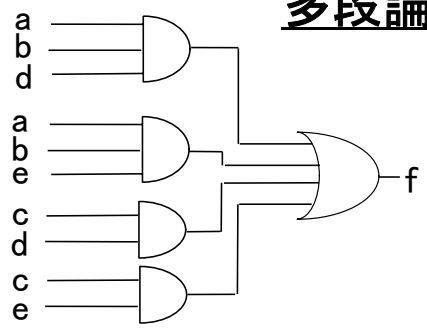
＝ $\overline{(ab+c') \oplus (ab+a')}$

＝ ab + a'c' + ac;

中間変数を用いて表現することによって生じる、ドントケアのもうひとつは、ODCである。これは、多段表現のために導入された中間変数が、0でも1でも、外部に影響を与えない（外部から観測できない）条件である。スライドの例では、変数(a, b) が（1, 1）、変数(a, c) が（0, 0）、（1, 1）の場合に、中間変数 xの値に関係なく、出力yの値がそれぞれ1, 1, b と決まるため、ab+a'c'+ac が ODCである。

ODCは、以下のようにして求めることができる。出力yに対して、入力xの値を0と1に固定した場合の値（Co-factorと呼ぶ）をY(x=0)、Y(x=1)と表記すると、その二つの値が同一である条件（排他的論理和の否定）では、中間変数xの値に関係なく、出力yの値が変化しないことになる。この条件がODCである。このODCを利用して、y=ab+a'x+cx'というブール式を最適化できる。 中間変数を利用した出力が複数ある場合は、それぞれの出力のODCの論理積が全体のODCとなる。但し、このODCは計算量も多く必要であり、大規模回路ですべて利用することは計算量的に難しい。現実には、これらのDCから、いくつかを選択して利用する。たとえば、直感的な選択方法として、ある出力関数f1と共通の中間変数をもつ出力関数が複数とある場合、入力変数がf1のサブセットとなっている出力関数のDCだけを利用する方法がある。（サブセットサポートフィルター）

多段論理最適化(多段化)



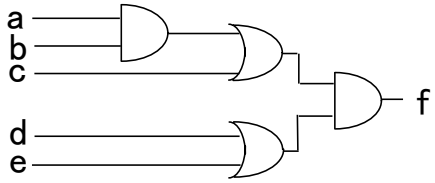
2段論理

f= abd + abe + cd + ce

リテラル数:10

多段化 2段化

(d+e)で除算



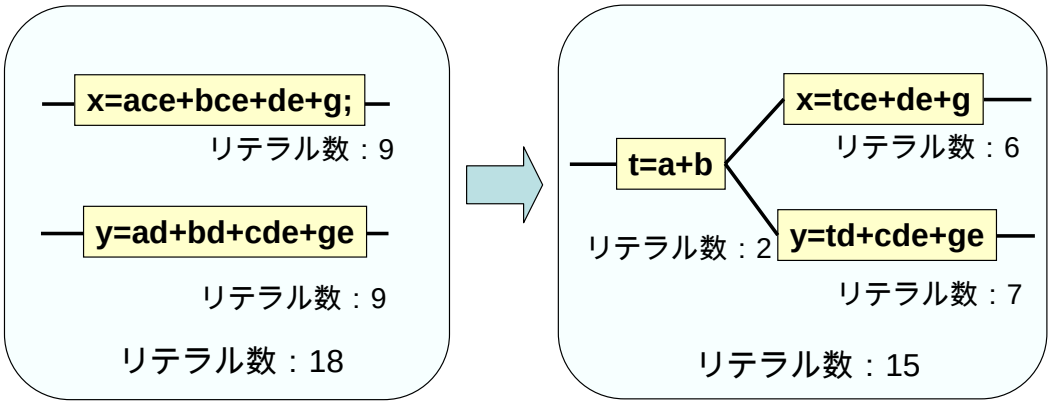
3段論理

f= (ab + c) (d + e)

リテラル数:7

組み合わせ論理はすべて2段で実現可能であるが、3段以上にすることを多段化と呼ぶ。通常、多段論理は2段論理より、小さな面積(少ないゲート数)で実現可能である。多段にする際、部分論理を共有できることもあり、さらに面積削減が可能である。たとえば、図で、2段論理ではリテラル数が10であるが、3段論理ではリテラル数が7と削減している。 (図で、＋は、論理和、abはa,bの論理積を示す。以下同様)

共通因子による分解

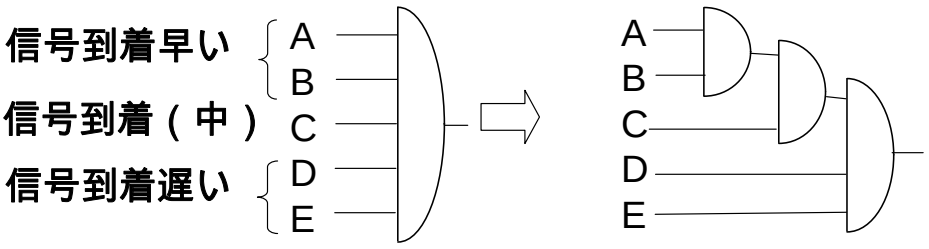


複数の論理式の間でも、共通の論理を共有することで面積を削減することができる。これは、複数のブール式から共通の論理式（共通因子）を見つける＝ブール式の因数分解を行うことで、面積が削減できることを示している。

図では、左側の二つの2段論理式x、yから、(a+b)という共通因子をくくりだすと、右の図のように中間端子tを使って表現できる。リテラル数の総数も、18から、15に削減している。

但し、この共通因子の発見は計算量が大きい割りに効果が限定的なこと、DSM（Deep Submicron）時代にはゲート数より遅延がより重要であり、因数分解は必ずしも遅延に有利でないことなどから、その重要性は低下している。

遅延を考慮した多段化



遅延の余裕（Slack)
= 遅延到着が必要な時間 - 信号到着時間
（Required Time) (Arrival Time)

Slackの大きい入力変数から因数分解していけば、遅延バランスがとれる。

因数分解による多段化は、面積削減だけでなく、遅延削減にも有効である。

左図で、A, Bの信号は早期に得られ、D, Eの信号到着は遅いとする。その場合、右の図のようにA, Bを多数段とし、D, Eを少数段とすれば、全体として遅延をバランスすることができる。

より詳細には、各信号ごとに、遅延の到着時間（Arrival Time)と、要求時間（Required Time）を設定し、その差（余裕：Slack）の大きい信号（変数）を含む因数から分解していく。但し、テクノロジマッピング前には、詳細な遅延は得られないので、遅延を予測（たとえば、ゲート段数から遅延を予測）して行う必要がある。簡単な方法としては、入力から出力のすべてのゲート段数ができるだけそろうようにするという方法が考えられる。

（詳）

論理関数の除算

ブール式の除算は色々な答え（一意でない）
F = a’c’+a’b+ac;
D = a’+c;
F= (a+c’)D+a’b; (= ac+a’c’+a’b)
= (b+c’)D+ac; (= a’b+a’c’+cb+ac)

☆除算の考え方
A)代数的分割（Algebraic Division)
同じ変数を持たない積和論理式で分割 (但し、a･a’は0とする。）
例（ab+d)(c+e)=abc+abe+dc+de
B)ブール代数的分割（Boolean Division)
同じ変数を持つ積和論理式でも分割
ブール式として扱う

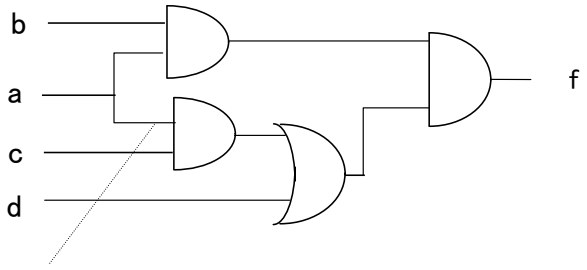
多段化は、2段回路を多段回路に変更することであるが、これは、積和論理式を除算により因数分解することにより得られる。この除算には、代数的な分割手法、ブール代数的な分割手法等が提案されている。イメージ的には、代数的分割は、論理式をブール式と考えずに因数分解する方法、ブール代数的分割は、論理式のブール代数の性質（a+ab=a等）を利用した方法である。一般に、ブール代数的分割の方が、よい解を得られる可能性が高いが、解の発見に必要な計算量が多い。実際の論理合成では、設計者の与えた多段論理をそのまま利用して最小化することが多く、与えられた組み合わせ回路全体をいったん2段論理に展開した後で、多段にするということはまれである。（部分的に2段化し、多段化するということは行われる。）

（省）

ATPGを利用した冗長回路除去

組み合わせ回路の製造テスト用のテストパターン生成(ATPG)を利用

ATPGにより、組み合わせ回路のテストできない故障（s-a-1、s-a-0）を発見する。→その故障を定数1または0で置き換える。



s-a-1を発見するテストパターンは生成不能→定数1とできる。

f = ab(ac+d)
=abc+abd=ab(c+d)

※s-a-1 = stack at 1

回路の冗長部分を除去する方法として、非常に有望な方法として、ATPGを利用する方法がある。組み合わせ回路のATPG（Automatic Test Pattern Generation:製造時のスタックアット故障（例：s-a-1：スタックアット1、故障で常に1となってしまう）を発見する手法）は、大規模な回路に効率的に適用可能になってきているからである。図で、s-a-1と示した故障は、どのような入力の組でも検出できない。s-a-1を発見するためには、その値を0としなければならないが、aを0とすると、fが必ず0となってしまい、s-a-1が検出できない。つまり、この部分は値をいつも1にしておいても、出力には影響を与えない。よって、s-a-1を定数1とすると、ab(c+d)という冗長部分を除去した回路が得られる。故障がなくなるまで、ATPGをかければ、すべての冗長性を除去することができる。本手法は、もとの回路を2段化することなく、多段のまま適用できる。

4.4 論理合成

4.4.1 順序回路生成・最適化

状態最小化、状態符号化

リタイミング

4.4.2 組み合わせ回路最適化

論理式の内部表現形式

2段論理最適化

多段論理最適化

⇨ テクノロジマッピング

4.4.3 論理合成ツール

LSIは、メーカーやテクノロジルール（90ナノメータ、0.13ミクロン等）ごとに、利用可能なライブラリセルが定義されている。

論理回路をLSIで実現する時は、論理式を、これらライブラリセルを利用して構成する必要がある。この論理式を、ライブラリセルに割り付ける工程をテクノロジマッピングと呼ぶ。本節では、この基本概念を示す。

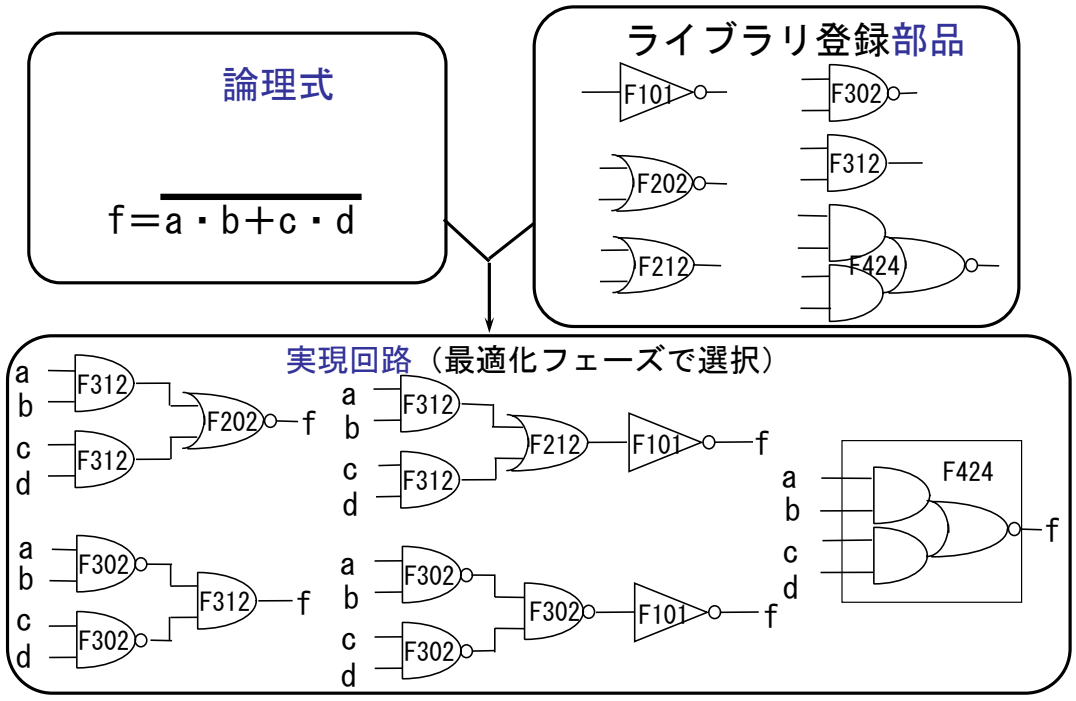
テクノロジ依存の最適化

テクノロジセルを用いた設計手法

・あらかじめ用意されたAND, NAND等のセルを組み合わせて、
回路を設計する手法。
・セルは、カウンタやアドレスデコーダのようなMSIクラスのものは
最近はあまりない。(ゲート回路やAND-AND-OR等の複合ゲート、
小ビットの加算器程度。)
但し、最近のFPGAでは、専用の乗算器を持つことが多い。
・トランジスタレベルの設計は行わない。
・基本的に、エッジトリガタイプのD-FFを用いた一相同期回路

前節までの組み合わせ回路最適化は、ブール式を簡単化するもので、テクノロジに依存しない最適化であった。この方法では、ANDでもORでもNANDでも同じように扱うことになるが、実際にはそれぞれ、面積も、遅延も大きく異なることが多い。よって、テクノロジ非依存の最適化で大まかに最適化されたブール式に対して、本節で説明するテクノロジに依存した最適化を行うことで、可能な範囲で所望の面積や遅延を実現することができる。

テクノロジマッピング



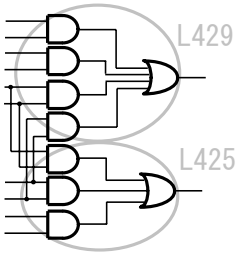
テクノロジマッピングは最適化された論理式を、ライブラリに登録された部品によって実現する工程である。テクノロジとは、使用可能なセルの種類を示す。（あるテクノロジに存在するライブラリセルへのマッピングというような意味である。）遅延計算式や駆動能力等の情報も持っている。

左上の論理式を右上の部品を使って実現すると、たとえば、下図の5つの実現方法がある。右端の実現では一つの部品で構成されているのに対し、左や真ん中の実現では6個、8個のセルから実現されている。それぞれ面積でいうと、5～9倍程度の差である。このように、テクノロジマッピングは、回路性能に大きな差となって表れ、DSM時代の大規模回路向け論理合成で非常に重要な工程である。逆に、ライブラリ部品の品揃えの豊富さも回路の全体性能に大きく影響する。セルへの割り当てであるため、セルバインディングとも呼ぶ。

（詳）

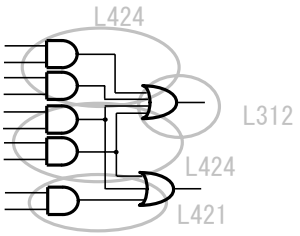
面積評価指標(テクノロジ独立レベル)

面積　小
リテラル数　大



面積 7 リテラル数 21

面積　大
リテラル数　小



面積 9 リテラル数 17

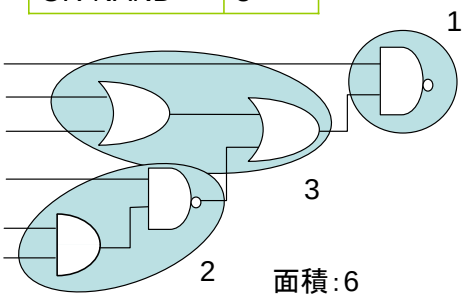
正確な面積評価指標の研究開発

1) マクロ(ライブラリのファンクションブロック)使用予測
2) ゲート入力数の制限

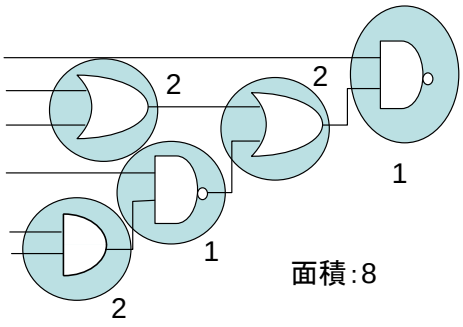
テクノロジ非依存の最適化で面積最適化の指標としてきたリテラル数は、実はセルで設計した場合の面積の正しい指標とは言い切れない。図で、Fan-inはそれぞれ、21と17であるが、逆に面積は7と9と逆転している。つまり、リテラル数の最小化は、面積の最小化とはいえない。また、現在では、面積の最小化とともに、遅延の最小化がより重要な目標となっているため、論理合成におけるリテラル数の最小化は、その重要度を下げている。

テクノロジマッピングの例

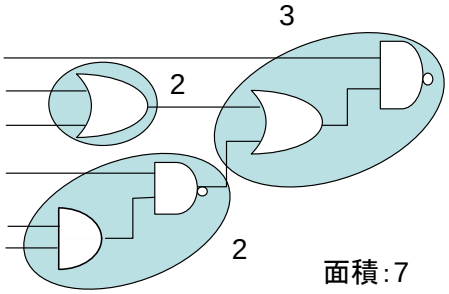
セル	面積
OR2	2
AND2	2
NAND	1
NAND-OR	3
NAND3	2
OR3	3
OR-NAND	3



面積:6



面積:8



面積:7

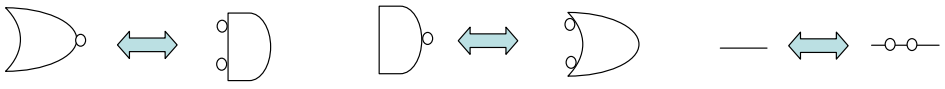
テクノロジマッピングのイメージ図である。左上のようなライブラリセルがあった場合、同じ論理回路を図のような様々な部品によって実現できる。この場合、左下の実現方法が、最も面積が小さい。実際には、ANDをNANDに変換する等も含めてマッピングするので、図の例よりも、優れたマッピングを探し出す。

トピック：手で簡易にマッピング（ドモルガンの定理）

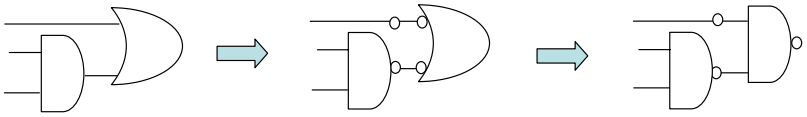
（省）

$$\overline{a \cdot b} = \bar{a} + \bar{b} \qquad \overline{a + b} = \bar{a} \cdot \bar{b}$$

式の否定　= >　ANDとORを変換して、各項の否定を取ると同じ論理（一般化可能）



この変換を利用すると、手で簡易にNAND系回路に変換したりできる

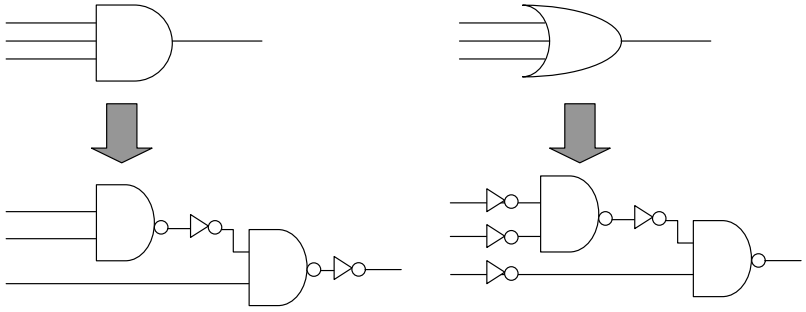


トピック:ドモルガンの定理を利用した人手のマッピング。CMOS回路では、ANDやORより、NAND，NORといった否定形論理のゲートの方が遅延や面積が有利なことが多い。

そこで、かつては人手で回路図をNAND系に置き換えることがあった。そのときに、ドモルガンの定理を利用した図形変換を行うと容易にできた。現在はツールが行ってくれるので現実には必要ないが、ちょっとした回路を机上で考えるときに便利である。

多入力ノードの分解とNAND標準形

・ 任意の論理関数は2入力NANDの木に分解可能
→すべて2入力NANDからなる標準形にして
テクノロジマッピングを行う



(a)ANDゲートの分解　　(b)ORゲートの分解

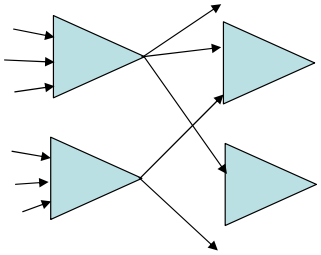
テクノロジマッピングの内部形式として、NAND標準形がある。(他にも色々な形がある。)これは、論理式を2段の論理式だけで表現するものである。(NANDは完全性があるので、任意の論理式を表現可能である。)たとえば、多入力のANDやORは図のように、2入力のNANDで表現できる。二入力のNAND(とインバータ)のみから表現できるため、パターンマッチをするためのパターン数を節約できるなどが、よい点である。

テクノロジマッピングの手法

（詳）

1）ルールベース
　設計者のまねをする
　すべてのタイプのセルを扱える
　→段階的にセルに置き換えていく

2）ヒューリスティック算法
　・ツリーカバリング（Tree Covering）
　　NAND標準形のネットワークを木の集合に分ける
　　ライブラリセルで、木をカバーしていく
　　（Dynamic　Programming法などの利用）

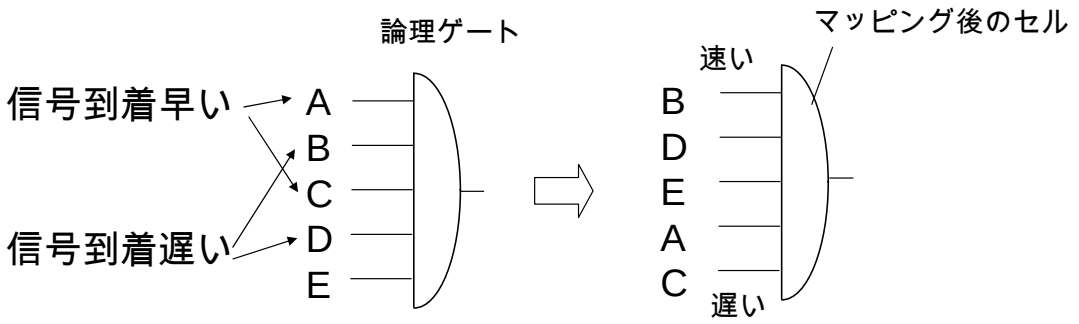


一般の有向グラフでは最小被覆は困難
（DAGのカバーとなる。）
それぞれの木のカバーは、効率のよい厳密解法
（ツリーのカバーとなる。）

テクノロジマッピングには、大きく分けて、ルールベースに基づく手法、ヒューリスティック算法、その両方を利用するものがある。（厳密解法は、現時点では、大規模回路には適用できない。）

ヒューリスティック算法の代表例は、ツリーカバリング手法である。これは、AHOのコンパイラのコード生成アルゴリズムを基本アイディアとしたものといわれている。実用システムの多くで利用されている。回路は一般にDAG（Directed Acyclic Graph 非循環有向グラフ）であるが、これを部品で被覆するのは、難しい問題である。そこで、図で三角で示すような木の集合とし、それぞれの木をカバーする問題として解く方法である。

遅延を考慮したマッピング
(Fanin Ordering)

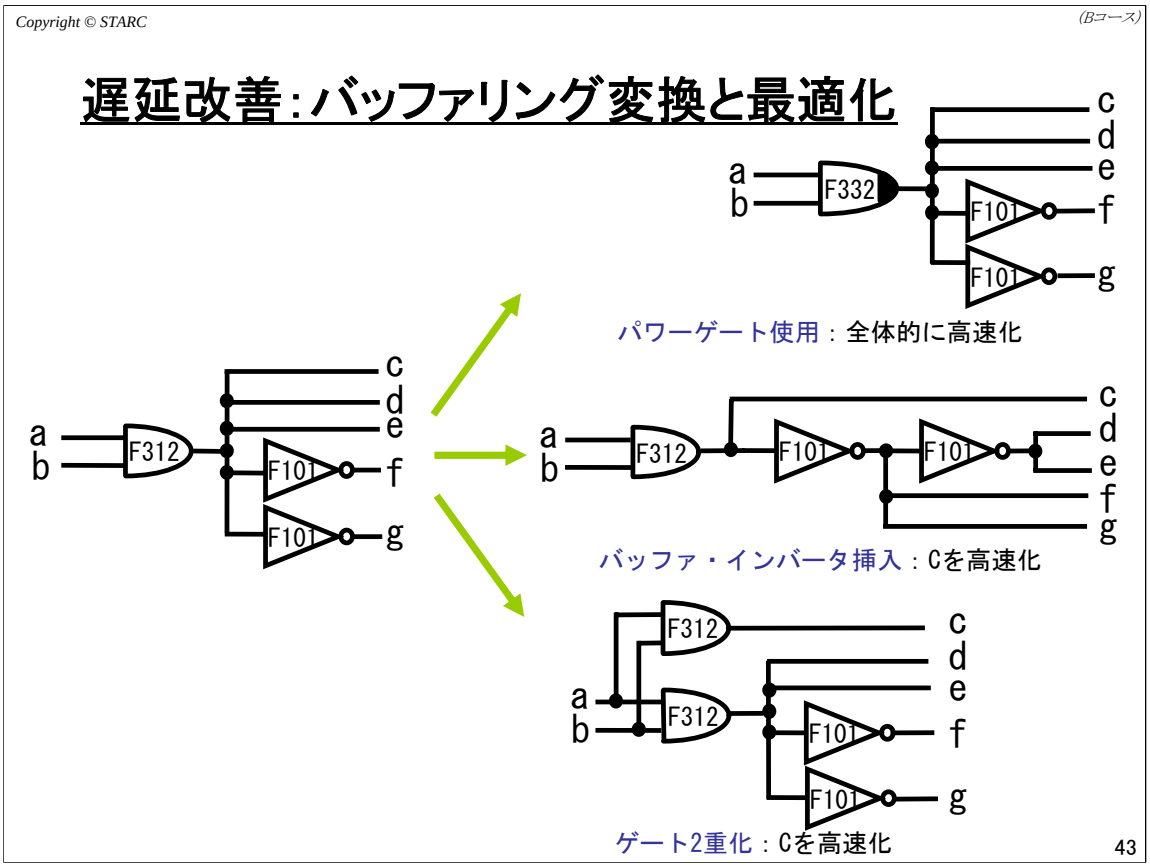


遅延の余裕 (Slack)
= 遅延到着が必要な時間 - 信号到着時間
(Required Time) (Arrival Time)

多入力のブロックは各入力端子ごとに遅延値が異なる場合がある。
→ Slackの大きい変数からより高速な端子へ接続

テクノロジマッピングの遅延改善手段として、ファンインオーダリングと呼ばれる手法がある。多入力ゲートには、可換な端子間でも、出力までの遅延値が異なる場合がある。この場合、遅延のSlackをみて、その値の小さいものから、より高速な端子に割り当てることで、回路全体の高速化が図れる。

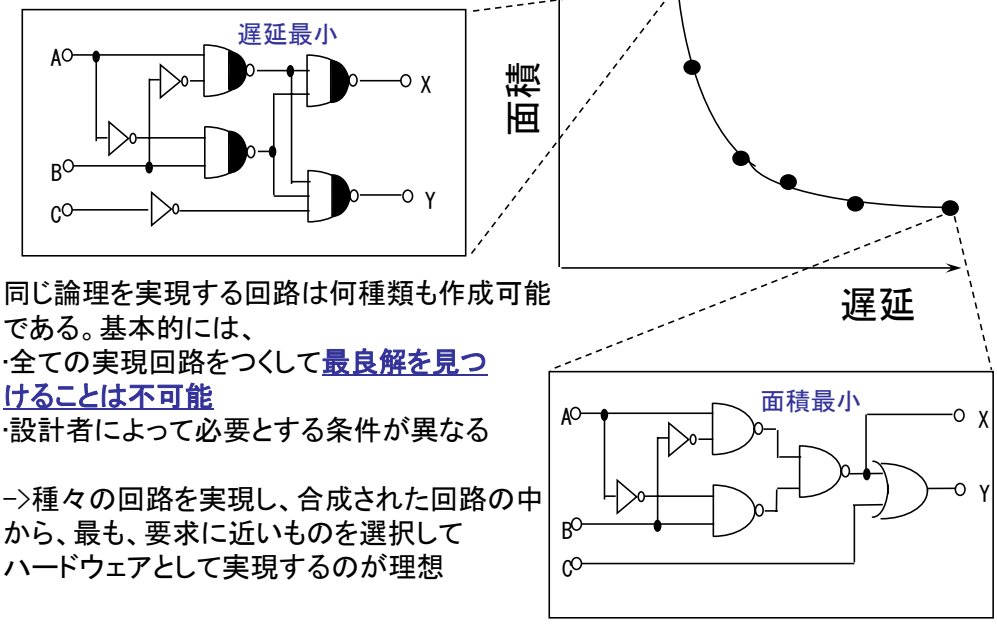
また、より一般的には、TREEカバーリングなどのテクノロジマッピングの際の評価関数(コスト関数)にSlackをいれて計算することでも、最大遅延の小さなマッピングを得ることができる。



テクノロジマッピングにおける遅延改善の代表的手法として、パワーゲート（後段のゲートの駆動能力の大きいゲート）がある。

ゲート回路の遅延は、論理段数（ゲート段数）だけで決まるわけではない。出力につながるすべてのゲート群の入力を1や0にする時間で決まる。よって、高速化のために、様々な手法が用いられる。パワーゲートは駆動能力の高いゲート（出力トランジスタが大きい）のことで、これを使用することで高速化できる。駆動能力の高いバッファ・インバータを挿入することで、パワーゲートと同等の効果が得られるが、さらに、バッファ・インバータを工夫して多段に挿入することで、特定の信号線だけを高速化することができる。また、後続するゲートが多い（ファンアウトが大きい）場合、もとのゲートを複数にすることで、直接つながる配線（後続ゲート）を減らすことができ、これにより高速化が可能となる。（ブール式の因数分解による論理最適化は、ファンアウトを大きくし、論理段数を大きくする手法であるため、高速回路ではあまり有効に機能しないと説明したのは、このゲート多重化からも分かる。）

面積−遅延トレードオフ

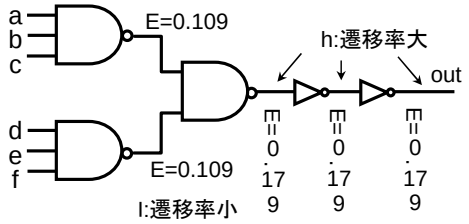


同じ論理を実現する回路は何種類も作成可能である。基本的には、
·全ての実現回路をつくして**最良解を見つけることは不可能**
·設計者によって必要とする条件が異なる

−>種々の回路を実現し、合成された回路の中から、最も、要求に近いものを選択して
ハードウェアとして実現するのが理想

動作合成で、面積(演算器等の数)と性能(サイクル数、周波数)とがトレードオフであったのと同様、論理合成でも、面積(ゲート数)と性能(レジスタ間の組み合わせ回路遅延)はトレードオフの関係にある。(もちろん、動作合成に比べると、非常に局所的な範囲でのトレードオフである。)現時点では、論理合成は非常に時間のかかる工程である(大規模回路では数日)ため、何度も論理合成をかけ直すことは難しく、できるだけ早期に遅延制約を満たす解を見つけるのが重要である。そのためには、いかに「良い」RTLを記述できるか、適切な遅延制約を与えられるかが重要である。(但し、動作合成ツールを利用するようになると、ツールが遅延制約を満たすRTLを出力するようになるため、現在論理合成ツールを利用する際のテクニックの大半は不要になってくると考えられる。)

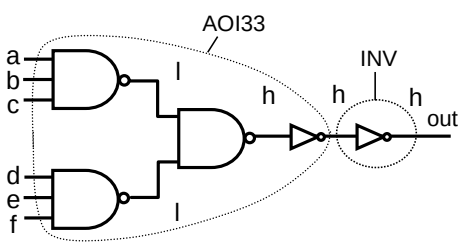
トピック：低電力を目指したマッピング （省）



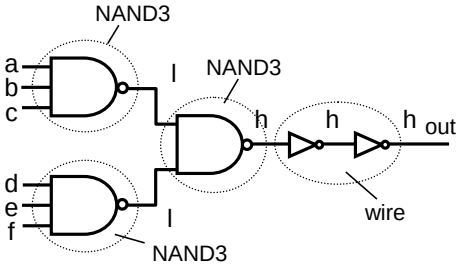
(a) 元回路と信号遷移確率

Gate type	Area	Intrisic cap.	Input Load cap.
INV	928	.1029	.0514
NAND2	1392	.1421	.0747
NAND3	1856	.1768	.0868
AOI33	3248	.3526	.1033

(b)ライブラリセルの諸元



(c)面積最小マッピング



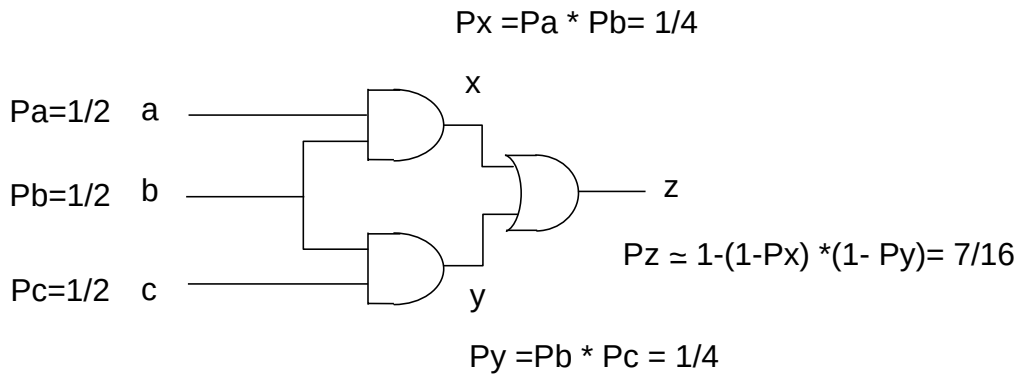
(d)電力最小マッピング

低電力を目指したマッピング手法も提案されている。基本的には、面積を目的関数とせず、信号の遷移確率（信号が1→0、0→1と変化する確率）が下がることを目的関数にすることで実現できる。例えば、上図（a）でlの配線は遷移確率が低く、hの配線は遷移確率が高いとする。（c）はAOI33という複合セルと否定論理INVの二つのセルで実現でき、面積は3248＋928 = 4176, で、（d)の面積（NAND3×3 = 1956×3 = 5868)よりも小さく、面積の小さいマッピングといえる。一方、電力を考えると、（c)は遷移確率の高い部分hが2カ所実際の配線になっているのに対し、（d)は一カ所のみである。（複合ゲートはトランジスタ論理で設計され、内部配線は無いか、あっても、外部のAL配線に比べ容量が格段に小さい。） 従って、遷移確率の高いAL配線の数が少ない（d）の方が、動的な電力（充放電電力）が小さいと考えられ、電力消費の少ない低電力マッピングとなる。

信号の遷移確率の計算法については、次ページに述べる。本手法で得られる電力削減効果は大きくはないが、数mWの電力削減を行いたい場合などに役立つこともある。

（省）

トピック：信号遷移確率の静的な求め方



Zの信号遷移確率： Ez = 2*Pz*(1-Pz)=2*(7/16)*(9/16)=0.49

信号の遷移確率を静的に求める原理を示す。まず、入力が1である確率を求める。ここでは、すべて0.5とする。次に、論理をおって、信号が1になる確率を求める。たとえば、xが1になる確率Pxは、信号aとbが1になる確率の積で求まる。Zが1になる確率は、1から、xとyが0になる確率を引くことで求まる。最後に、出力Zの信号遷移確率は、Zが今1で次に0になる確率と、今0で次に1になる確率の和であるから、図に示すように求めることができる。消費電力は、信号遷移確率に比例するから、この確率を下げるような回路を合成すれば、低電力な回路とすることができる。

4.4 論理合成

4.4.1 順序回路生成・最適化

状態最小化、状態符号化

リタイミング

4.4.2 組み合わせ回路最適化

論理式の内部表現形式

2段論理最適化

多段論理最適化

テクノロジマッピング

⇨ 4.4.3 論理合成ツール

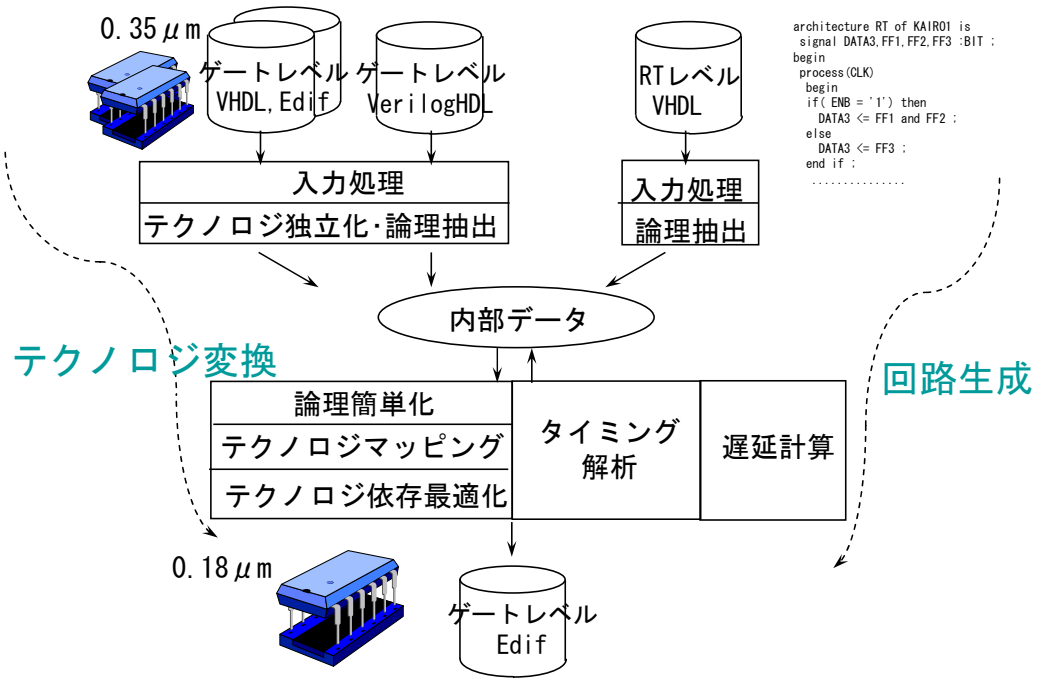
これまで述べた、順序回路最適化、組み合わせ回路最適化、テクノロジマッピングをツールの形にしたのが、論理合成ツールである。入力言語として、RTLの言語、VerilogHDL, VHDLを用いることが多い。VerilogHDL, VHDLは、イベントドリブンシミュレータ(次の、機能論理検証の章参照)を意識して設計されており、論理合成にもちいるには必ずしも適していない。たとえば、レジスタやラッチ等の基本的なRTL部品すら直接的に表現することができない。(VerilogHDLが出現する以前に、電機メーカーなどで独自に設計されたRTL言語は、レジスタ等を直接表現できるものが多かった。)この論理合成には必ずしも適していないVerilogHDL, VHDLを入力とするツールが現在の世界のデファクトスタンダードツールとなっている。

論理合成とは

- 論理合成とは
  - RTLで表現された論理を、論理回路に変換し、
  - ある目的関数に従って最適化し、
  - 所望のテクノロジのライブラリセルにマッピングすること
- 目的関数
  - 面積、遅延時間、消費電力、等

論理合成とは、RTL記述をゲート回路に変換するもので、広義には順序回路の状態割付等も含む。(動作合成を利用する場合は、動作合成にも状態割り当て機能がある場合が多い。RTLを人手で記述する場合も、状態符号化は人手で行うことが多い。これは、デバッグ時に状態番号を把握しやすいため等の理由による。但し、論理合成の工程の中で、真に生成と呼べるのは状態符号化だけであり、この部分がないと、論理合成でなく、論理最適化であると定義する研究者もいる。)論理式への変換、最適化、テクノロジマッピングの3つの工程からなる。古典的な組み合わせ回路最適化は、変数(リテラル数)の最小化を目指していたが、ディープサブミクロン(DSM)時代ではあまり意味がなくなってきている。面積より遅延が重要になってきており、DSMではゲート遅延より配線遅延が支配的になりつつある。遅延を考慮すると、リテラル数の最小化と面積の最小化はあまり相関がなくなってきている上に、大規模回路では局所的な最適化は合成時間の割に効果が限られるからである。最後の、テクノロジマッピングはDSM時代でも非常に重要な技術である。

論理合成の処理概要



論理合成ツールの処理概要を示す。通常はRTL記述を所望のテクノロジ(0.18ミクロン等の設計ルールや製造メーカごとにライブラリセルが異なる)のライブラリセルのネットリスト(ゲートの接続関係を示す情報)に変更する(右側の回路生成)。

また、変わった使い方としてテクノロジ変換(図の左側)と呼ばれる使い方がある。これは、古い設計データ(たとえば0.35μmでのネットリスト)を、0.18μm の半導体テクノロジのネットリストに変換する(左端)ような場合に利用する。古い設計データのマッピング済みのゲート回路データを読み込み、それを論理式に変換する(テクノロジ独立化)。その後、再度その論理式を新しいテクノロジのライブラリを使ってマッピングしなおし、新しいテクノロジのゲート回路を生成するのである。

論理合成ツールは、先に説明した面積を最適化する論理簡単化やテクノロジマッピングの他に、遅延最適化のためのタイミング解析、遅延計算の機能も備えている。遅延計算には時間がかかるため、回路を最適化する際に遅延計算を全体にかけて行うと合成時間が増大してしまう。現在の論理合成ツールでは一般に、遅延最適化は面積最適化に比べ論理合成時間がたくさんかかる。

論理合成の処理(1)

（詳）

- HDL記述入力・解釈
  - HDLを解釈し、ゲート回路に変換する。このとき、記憶素子(レジスタ)と組み合わせ回路部を分離する

  （この工程を有力ベンダーツールではインファレンス（推定）と呼ぶ。）
- （テクノロジ独立化）
  - ネットリストを入力した場合は、論理式等のテクノロジに依存しない表現にする

  例　F312 (NEC CMOS8) → AND, F202 (NEC CMOS8) →NOR
- 論理簡単化（テクノロジ独立最適化）
  - 2段論理最適化、多段化等論理式のレベルで最適化
- テクノロジマッピング
  - 論理式レベルの表現を、テクノロジに存在するライブラリセルで実現
- テクノロジ依存最適化
  - 論理を保存したまま、構成法を変更し、テクノロジ毎のデータに基づき、遅延、面積を計算し、最適化してゆく処理

論理合成の処理を詳細に分けると、上記5つの段階に分けられる。それぞれの動きは今までに説明した通りである。

2番目のテクノロジ独立化は、テクノロジ変換の場合だけに利用され、RTLからの回路合成のフローには不要である。

（詳）

論理合成の処理(2)

- 遅延計算
  - 素子や配線の遅延を計算する。
- タイミング解析
  - パス解析により静的にタイミングチェックを行う。
- デザインルールチェック
  - デザインルールに違反した箇所をチェックし修正する。

論理合成では、通常、与えられた遅延制約を満たす回路を作ることが重要である。即ち、レジスタとレジスタの間の組み合わせ回路遅延の総和が、クロック周期より短いことを保証する必要がある。そのため、論理合成ツールには、ゲート回路の遅延解析や遅延違反をチェックするサブシステムが搭載されている。

RTL記述からネットリストへ
～RTLの論理合成サブセット～

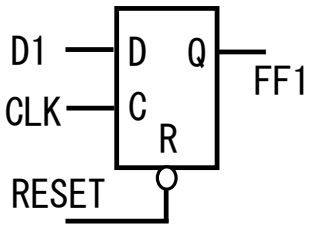
（詳）

- 論理合成可能なHDL記述の規則
  - 合成不可能なHDL記述がある
- 論理合成システムにより相違
  - 基本的な部分は一致
- 禁止される範囲
  - フリップフロップか3stateゲートかが確実には認識（「推定」）不可能な表現
  - 実数型や物理型等、直に論理変換で実現できないもの
  - 不定値（X）との比較などハードウェアで実現できない記述

最初に、RTLから、ゲート回路構成に変換する工程では合成可能なRTL記述かのチェックを行う。どのようなRTL記述もRTLシミュレーション可能であるが、論理合成可能なのはRTLのサブセットだけである。これを論理合成サブセットと呼ぶ。たとえば、設計対象を検査するためのテストベンチ（テストパターンを発生するRTL記述）はシミュレーションは可能であるが、もちろん合成することはできない。このように本質的にハードウェアにならないような記述以外にも、図に示すようないくつかの合成可能な規則があり、これは論理合成ツール毎に多少異なっている。少しの記述の相違で全く違う回路になることがあるので、論理合成サブセットとそれから合成される回路を理解するのは非常に重要である。

RTL記述から、RTL部品を「推定」(1)

- クロックのエッジ記述の条件で値が変化する信号
  - エッジトリガタイプのFFと認識
- ある条件下で値を保持する信号
  - レベルセンシティブタイプのラッチと認識
- ある条件下で'Z'を生成する信号
  - 3stateゲートと認識
- その他
  - 組み合わせ回路と認識




```
if(RESET = '0')then
    FF1 <= '0' ;
elsif(CLK'event and CLK='1')then
    FF1 <= D1;
end if ;
```

```
always@(posedge CLK or negedge RESET)
begin
 if (~RESET) FF1 <= 1'b0;
 else        FF1 <= D1;
end
```

VHDL　D-FFと推定される記述例　Verilog-HDL

RTL記述では、レジスタやラッチ、3ステートゲートなどを表現するために、ある規則に従ったコーディングを行う必要がある。(動作合成では、動作記述の変数が自動的にレジスタ等に割り当てられたが、論理合成では設計者がRTL記述に明示的に、レジスタ等を記述する。)

このコーディング規則は厳密であり、少しの書き間違えでまったく意図しない回路が合成されてしまう。

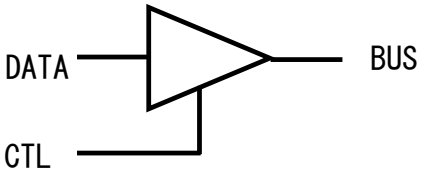
注:

図の下方にD-FFと推定される記述例を示す。元々はD-FFをシミュレーションするための記述でありD-FFの動きを正確に記述されている。但し、D-FFという非常によく使われる汎用的な部品を記述するためにこのように5行程度の記述が必要になるのはあまり生産的ではない。VerilogHDLやVHDLがもともとイベントドリブンシミュレータ用に開発され、後に論理合成に流用されたことがこの原因である。(VerilogHDLやVHDLが出現する前のRTL記述言語ではレジスタは単純にレジスタ変数として宣言できるものが大半であり、VHDL等よりわかりやすく、コンパクトに記述できたが、これらの言語は各社で個別に利用されただけで、デファクトスタンダードにはならず、現在ではほとんど使われていない。)

RTL記述から、RTL部品を「推定」(2)

- 3-state認識
  - ある条件下でハイインピーダンス'Z'を生成

```
if( CTL = '1')then
  BUS <= DATA ;
else
  BUS <= 'Z' ;
end if
```



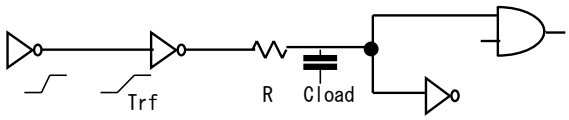
- マルチプレクサ(セレクタ)認識
  - 一定の形式のif文、case文をマルチプレクサと認識
    (その後の処理で通常のゲート化も可能)
- 合成ツール固有の指定による認識
  - ディレクティブなどにより演算器やマルチプレクサを明示的に指定

3ステート回路やマルチプレクサ等の認識ルール。RTL部品を直接表現する構文は持っていないため、一定の文型で、それらを表現しなければならない。if文は書き方によって、DFFやラッチ、マルチプレクサなどに認識されるので、複雑な場合は十分に注意して記述すべきである。(RTLシミュレーションで正しく動作していても、合成された回路が意図した通りでない可能性があり得る。)

（詳）

遅延計算(1)

- 素子や配線の遅延を計算
  - パラメータ：入力波形なまり(Trf)、負荷容量(Cload)、配線抵抗(R)
- レイアウト前（予測配線）
  - 予測配線長によりCload計算（ワイヤーロードモデル）。Rは無視。TrfはCloadより計算。
- レイアウト後（実配線）
  - 実配置、配線結果からCload、Rを計算。TrfはCloadより計算。



論理合成では、レジスターレジスタ間の遅延制約を守ることが必須である。そこで、テクノロジマッピング等の際に、回路の遅延を見積もることが重要である。図は、素子や配線の遅延の計算方法の一例である。容量や抵抗を正確に見積もり計算していると、遅延計算だけで多大な時間がかかるため、如何に高速に遅延を計算するかが重要である。また、回路を一部だけ変更した場合に全体を計算し直すことなく高速に遅延を計算するなどの工夫が、論理合成ツールでは重要である。近年は、ゲート遅延より配線遅延の方が大きい場合もあり、レイアウト前の配線長の見積もりが難しいことによる、遅延予測のずれが大きな問題になってきている。実際にチップを作成する場合は、レイアウト後の実配線長を得た後での遅延計算が必須となっている。

遅延計算(2)

（詳）

- ワイヤーロードモデル
  - 素子のファンアウト数によりCloadを計算するモデル
  - テクノロジやレイアウト対象となるチップ・マクロのサイズに応じて、複数のモデルを切り替える
- 合成ツールでは予測配線レベルの遅延計算を行うのが一般的
- レイアウト後の遅延計算は、外部から計算結果をSDFでアノテートする
- 予測配線レベルでは精度が不十分

論理合成時には、配線情報が得られないため、ゲート素子のファンアウト数から、後段の容量を想定する方法を用いる。ワイアーロードモデルと呼ぶ。配線長は予測配線長を用いる。しかし、予測配線では、精度が不十分であり、レイアウトをした後で、更に論理合成をかけ直すことなどが必要になる。

合成とレイアウトとの連携

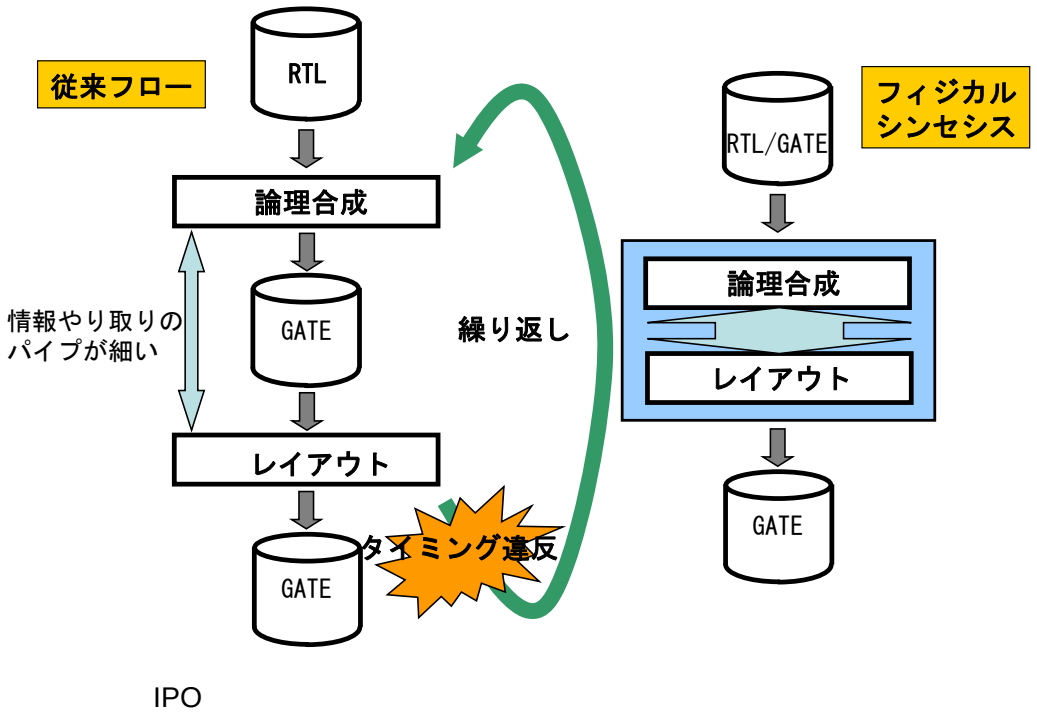
(詳)

- IPO(In Place Optimization)
  - 配置、配線後のタイミング制約違反を、既存の配置配線結果を保存しつつ、局所的な修正により回避する。
- IPO手順
  - レイアウト後の正確な遅延計算結果を合成ツールにアノテート。
  - タイミング制約違反を検出。
  - バッファ挿入、素子の置き換えにより最適化。
  - 遅延を再計算(既存の配置、配線結果をベースに簡易計算)。
- TDL(Timing Driven Layout)＝Physical Synthesis
  - IPO相当の処理をレイアウトツール側で行う。
  - 遅延の再計算時に、配置配線情報を利用することが出来るので正確。

論理合成とレイアウトを繰り返す手法の一つとして、IPOがある。レイアウト後の遅延違反を、論理合成により局所的に修正(バッファ挿入、ハイパワーゲートへの置き換え)する。その結果を再度レイアウトするが、先の配置配線結果とできる限り近い形でレイアウトを行う。これを繰り返すことで、遅延制約を満たす回路を得る。

タイミングドリブンレイアウトは、このような処理をレイアウトツールの中で行うことをいう。「タイミング」と呼んでいるが、レジスタ間の「遅延」を守った上でのレイアウトという意味である。

フィジカルシンセシス



IPO

最近、使われているのは、フィジカルシンセシスと呼ばれるレイアウトツールである。従来、左の図のように、論理合成とレイアウトを繰り返すことで遅延違反をなくすように図ってきたが、あまりうまくいかなかったため、レイアウトツールに、バッファー挿入（論理合成と異なり、幾何学的な位置も考えたバッファ挿入が行える）、ハイパワーゲートへの置き換え、多入力ゲートのファンインの並べ替え等の従来論理合成が遅延最小化のために行っていた工程を導入したものである。このようなツールの出現により、従来に比べ、大規模な回路でも高速な回路が合成できるようになってきた。

（省）

コラム：論理合成の歴史と今後

- 論理式最適化の研究は非常に古い。主に、2段論理の最小化等に重点があった。適用可能デバイスがPLAだったため。
- 80年代前半：一部企業で実用化始まる。
- 80年代後半：商用ツールが一般に実用に使われ始める。
  －初期の論理合成は主にマッピング機能が使われた。つまり、入力はほぼゲートレベルであった。
- 90年代：純粋に機能を書くようなRTL記述が使われ始めた。
  －機能的なRTL普及と、高周波数化とともに、クロック周期を満たせないRTL記述が増え、遅延最適化の要求が強まった。論理段数の最小化研究が盛んになった。
- ゲート遅延より配線遅延が大きくなるようになり、論理段数最小化があまり意味をもたなくなってきた。（例：共通因子化は、ファンアウトが増え、却って遅延が増大してしまう等。）
- 論理合成とレイアウトを同時に行うPhysical Synthesisが重要度を増した。配置配線時に、バッファーツリーの形状やバッファーサイズを計算する。

論理合成の研究とツールの歴史を簡単に振りかえり、今後の展望を、動作合成＋論理合成というフローで考える。

論理合成のまとめ

- 論理合成はRTL記述をゲート回路に変換し、面積や遅延を最適化し、特定の半導体ライブラリのセルにマッピングするツールである。
- ASIC用とFPGA用では最適化、マッピングが大きく異なる。
（本講座は、主にASIC向けの技術）
- レイアウトと論理合成を同時に考慮するツールが広く利用されはじめている。
- 研究の流れ

1) 組み合わせ回路最適化、順序回路最適化とも、古来はリテラル数（面積）最小化を目指して研究されてきたが、現在では遅延最小化が最大の課題である。（リテラル数の最小化はASICやFPGAの最適化のよい指標ではなくなった。）
情報理論の最適化問題という側面より、電気特性をもつ回路の最適化という側面がより強くなってきている。
2) RTLの検証は時間的に不可能になりつつあり、動作合成の普及とともにRTLは設計者が直接扱うことは少なくなると思われる。

筆者の予想：
RTLと論理合成は、コンパイラのアセンブラとコード生成系のような存在になる。

論理合成工程のまとめ。FPGAはLUT（Look－Up-Table）とよぶ数ビット入力のメモリで任意の論理を表すデバイスであり、論理合成のなかでテクノロジマッピングに相当する部分が大きく異なる。また、すべての論理をMUX(マルチプレクサ）で実現するデバイスもあり、これも、異なった手法が必要である。いずれにせよ、ゲート遅延より、配線遅延が重要になるに従い、合成結果の遅延などの制御をRTLの記述方法で行うことが非常に難しくなっている。論理合成とレイアウトを融合させたツールで、できる限り自動で、遅延制約を達成できる手法がより重要になっている。

用語・略語(テクニカルターム)

用語・略語	フルスペル	解説
SOP 積和標準形	Sum Of Product	重要:ブール式の表現形式のひとつ。複数のAND式をORでつなぐ形式。2段論理式とも呼ぶ。最小のANDの項数が最小になるものを最小積和標準形と呼ぶ
多段論理式		重要:2段より段数の多い論理式のこと。
代数的分割	Algebraic Division	ブール代数を因数分解して多段論理式に変換する最適化手法の一つ。ブール式を代数的にみて(a*!a=0等のブール式の性質を利用しない)因数分解する手法。
論理的分割	Boolean Division	ブール代数をブール式の性質を利用して因数分解する手法。代数的方法に比べて、高い性能が期待されるが、計算量が大きい。
テクノロジマッピング		重要:ブール式で表された論理回路を、所望のテクノロジライブラリのセル(NANDやNOR等)で実現すること。
NAND標準系		論理式をすべてNANDゲートで表現すること。かつては、テクノロジマッピングを行う前にNAND標準系にしておくシステムが多かった。(現在は、RTL部品を直接マッピングするものが増えている。)
遅延最適化		レジスタとレジスタの間の論理を実行する遅延時間を、クロック周期以内に最適化すること。

用語・略語	解　　説
ファンアウト数	あるゲートの出力に直接つながるゲートの入力数の合計数（厳密には、入力ゲートのファンイン数の合計）。ファンアウトが大きいと、すべての後続ゲートへつながる配線を0か1にするのに大きな電流が必要になり、大きな駆動能力が必要になる上、遅延が大きくなる。
バッファードゲート	ファンアウトが大きなゲートは大きな駆動力が必要になるため、大きなトランジスタをもつゲートを利用する必要がある。これをバッファードゲートと呼ぶ。
バッファーツリー	バッファードゲートの変わりに、バッファーをツリー状にすることで、ファンアウトの大きなゲートの遅延を高速化することができる。このバッファーツリーの形状を最適化することが高速化には重要であり、Physical Synthesisでも主要なタスクである。
トランスファーゲート	スイッチとして動作するゲート。スイッチオンで導通、オフで絶縁される。バスへの接続やMUXセルの内部構成で利用される。但し、現在のASIC設計においては、ユーザの利用は極力さけるように推奨されることが多い。
リテラル	ブール式の変数のこと。但し、変数とその変数の否定は別のものとされる。PLAの時代は、リテラル数の最小化が、回路の最適化の良い指標であった。
クワイン・マクラスキー法	2段論理最適化の厳密解法のひとつ
ESPRESSO（-Ⅱ）	2段論理最適化のヒューリスティック解法を実装したプログラム。米国カリフォルニア大学バークレー校（UCB)で基本形開発。
MIS, SIS	UCBで基本開発された多段論理合成システム。80年代後半から90年代前半のブール式の最適化アルゴリズムの研究プラットフォームの役割を果たした。

用語・略語	フルスペル	解　　説
論理積 キューブ	Product Cube	リテラルの論理積
最小項	Minterm	すべての変数を使った1となる論理積
ブーリアンネットワーク	Boolean Network	2段論理式をノードとするネットワーク。多段論理式をあらわす内部表現形式の一つ。論理合成ツールMISで採用された。
BDD 二分決定木	Binary Decision Diagram	ブール式の内部表現形式の一つ。変数の0と1で二分岐するTREEである。分岐する変数の順序を固定して、最小化した場合、同じ論理は必ず同型のグラフになるという性質を持つ。よって、表現できた場合は、二つの論理変数の等価性が、即座に検証できる。但し、乗算を表現すると組み合わせ爆発を起こす等、すべての論理式をコンパクトに表せるわけではない。
ファクタードフォーム	Factored form	回路図の通り、ブール式にしたような因数分解した形式。 たとえば、e(b+c)(a+bd)+g'のように、2段論理の積と論理和であらわされるような形式である。
SDF	standard delay format	標準遅延フォーマット（standard delay format)　遅延情報を格納

参考文献

ー論理合成の教科書
【1】 G. De Micheli,
“Synthesis and Optimization of Digital Circuits”, McGrawhill, 1994
【2】J.Bhasker、
「Verilog HDL論理合成入門ーRTL記述＆ネットリストのリファレンス」DesignWave Book

ーRTL言語入門
【3】Z.ナバビ、
「VHDLの基礎」、日経BP出版センター、1994
【4】小林優、
「入門Verilogー HDL記述」、CQ出版、1996
【5】 長谷川裕恭、
「VHDLによるハードウェア設計入門」、CQ出版、1995

ー順序回路
【6】当麻喜弘、コンピュータ基礎講座5 「順序回路論」、昭晃堂、1976
ー組み合わせ回路設計
【7】三上廉治
「ASIC時代の論理設計」、電波新聞社 1988
【8】笹尾勤、「論理設計 スイッチング回路理論」、近代科学社、1998
【9】ESPRESSO等の論理合成手法の論文集
R.Brayton, G.Hachtel, C.McMullen and Sangiovanni-Vincenteli, “Logic Minimization Algorithms for VLSI Synthesis,” Kluwer Academic Publisher, 1984

B4章
機能・論理設計3
理解度テスト

B4章　理解度テスト　(B.4.4)

(1) 論理合成について誤っている文の番号を答えよ。

① 論理合成ツールはRTL記述からゲート回路を合成するツールである。

② RTL記述は順序回路部分と組み合わせ回路部分に分けて考えることができるが、現在の論理合成ツールは組み合わせ回路部分の最適化を中心に扱っている。

③ ゲート回路記述のことを(詳細)論理記述とも呼び、論理記述を合成するという意味で論理合成と呼ぶ。

④ 論理合成で合成したゲート回路はバグを含まないため、検証の必要はない。

B4章　理解度テスト （B.4.4）

(2) 論理合成に関する説明で誤っている文の番号を答えよ。

① 順序回路の最適化は、状態遷移図の最適化や、レジスターレジスタ間の遅延削減を狙ったリタイミング等がある。

② 組み合わせ回路の最適化には、テクノロジ独立のものと、テクノロジ依存のものがある。

③ 論理合成の最初のステップは、RTL記述をブール式やゲート回路に変換することである。

④ 動作合成、論理合成、レイアウトの3つの工程で比較すると、レジスターレジスタ間の遅延の削減は主に論理合成で行われる。

B4章　理解度テスト （B.4.4）

(3) 順序回路最適化に関して誤った文の番号を答えよ。

① 与えられた状態遷移の状態数を削減できれば、面積や遅延を削減できる可能性がある。

② 各状態に与える符号を最適化すると、状態のデコード回路や、次状態を計算する回路を小さくしたり、高速にしたりできる可能性がある。

③ リタイミングはFFの位置を移動することで、レジスターレジスタ間の遅延をバランスさせることで、レジスターレジスタ間の最大遅延を小さくすることを狙っている。

④ 順序回路最適化は、組み合わせ回路最適化より効果が大きく、現在の論理合成ツール内でもより重要な技術といえる。

B4章　理解度テスト　(B.4.4)

(4) 組み合わせ回路最適化に関して、誤った文の番号を答えよ。

① 組み合わせ回路のテクノロジ独立の最適化は、ブール式の最小化を基礎にしており、ab'+ab=aというような最小化を行う。

② 2段回路は、論理積の和(Sum Of Product:SOP形式)で表わされる(例abc+bd)。

③ リテラル数を最小化することは、確実にゲート数の最小化につながる。

④ 一般の論理式を2段の論理式で表そうとすると、非常に大規模になることが多い。

B4章　理解度テスト （B.4.4）

(5) ドントケアについて誤っている文の番号を答えよ。

① ドントケアは実際には起き得ない入力の組を示す。

② ドントケアは動作を規定する必要のない入力の組、すなわちどのように動作してもよい場合の入力の組を表す。

③ ドントケアの入力に対する出力の値は0でも1でもよいが、どちらかに統一しなければならい。

④ ドントケアの部分の値を適切に設定すると、回路の最小化に効果がある。

B4章　理解度テスト （B.4.4）

(6) 多段回路の論理合成について誤っている文の番号を答えよ。

① 多段回路の表現方法の表現形式のひとつにブーリアンネットワークがある。

② 多段回路の内部ドントケアとはブーリアンネットワーク表現の各ノードを最小化する時に利用可能なドントケアであり、SDCやODC等がある。

③ 多段回路は2段回路より、一般に論理をよりコンパクトに表現することができる。

④ 多段回路は必ず2段回路にしてから最小化される。

B4章　理解度テスト （B.4.4）

(7) 論理最適化に関して誤っている文の番号を答えよ。

① 多段論理式を2段論理式にしたり、2段論理式を多段論理式にしたりする変換は可逆変換である。（但し、全く同一の式に戻すのは必ずしも簡単ではない。）

② 複数の論理式から共通の因子を見つけ、その因数で両方の論理式を分解することで、リテラル数を削減できる。

③ 論理式の除算には代数的除算とブール代数的除算がある。代数的除算の方が最適化力が弱いが高速である。

④ ブール代数の割り算は、算術割り算と同様に割り算の結果は一意に決まる。

B4章　理解度テスト （B.4.4）

(8) テクノロジマッピングに関して誤っている文の番号を答えよ。

① テクノロジライブラリの部品にマッピングする工程をテクノロジマッピングと呼ぶ。

② テクノロジマッピングはブール式最適化に先立って行われるのが普通である。

③ テクノロジマッピングは遅延を最適化することができる。

④ テクノロジマッピングは同一のブール式を様々な面積の回路で実現する方法を示すことができる。

B4章　理解度テスト （B.4.4）

(9) テクノロジ依存の最適化に関して正しい文の番号を答えよ。

① テクノロジマッピングでは面積の削減が主な目的であり、遅延の最小化はほとんどできない。

② ファンアウトの多い配線には、バッファーゲートを挿入することで回路の高速化を図ることができる。

③ テクノロジマッピングで消費電力を削減することは原理的に不可能である。

④ 消費電力はおおむねゲート数の多少によってきまり、信号の遷移する頻度はほとんど関係ない。

B4章　理解度テスト　(B.4.4)

(10)論理合成ツールに関して誤っている文の番号を答えよ。

① 論理合成ツールを利用して、あるテクノロジの回路を別のテクノロジに回路に変換することができる。

② 論理合成ツールはRTL記述からゲート回路を合成するツールであり、合成可能なRTL記述に特に制限はない。

③ RTL記述はレジスタ等の部品を直接記述できないため、論理合成ツールはRTL記述のある決まったパターンからレジスタ等の部品を推定する必要がある。

④ 論理合成ツールは与えられた遅延制約を満たす範囲で最小の面積の回路を出すことが大きな目的である。

B4章　理解度テスト　(B.4.4)

(11) 論理合成ツールに関して誤っている文の番号を答えよ。

① 微細化により配線遅延がゲート遅延より大きくなってきており、論理合成時での遅延見積もりが難しくなってきている。

② レイアウト後の配線遅延をフィードバックし、その遅延を元に論理合成するということを繰り返す方法(IPO)が一時期とられたがうまく収束することが難しかった。

③ 配線遅延を考慮したうえでの論理合成を行うために、レイアウトツールに論理合成機能を持たせるフィジカルシンセシスが重要になっている。

④ チップの遅延は論理段数で決まるため、論理合成ツールの性能によるところがほとんどであり、レイアウトツールのよしあしの影響は小さい。

演習問題1:論理式の簡単化

以下の論理式を簡単化せよ。

1) S = ab’ + a’b + ab
　　式変換でリテラル数を最小化
　（ab+a=a等を利用）

2) t = ac + ad + bc + bd + e;
　p = ce + de;
　代数的分割で共通因子を見つけ、最小化せよ。

演習問題2: 真理値表とカルノー図

abcd	出力 f
0000	0
0001	d
0010	d
0011	d
0100	0
0101	d
0110	0
0111	1
1000	0
1001	1
1010	0
1011	1
1100	1
1101	1
1110	d
1111	1

左図の真理値表であらわされる論理式を
カルノー図を用いて最小化せよ。

dはドントケアとする。

演習問題3：内部ドントケア

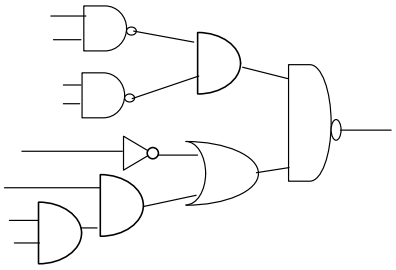
設問1)
回路全体のSDCを求めよ。(2段論理式形式で)

x = a’ + b; y = abx + a’cx;

設問2)
yのxに関するODCを求めよ。

x = a’b + ab’;
y = ab + xc’ + x’b’;

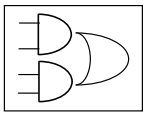
演習問題4 テクノロジマッピング

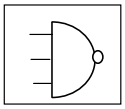


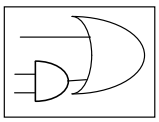
上図の回路を、右の部品を用いて、
できるだけ小面積の回路で実現せよ。

ヒント：ドモルガンの定理を使い、部品群の論理にうまく変換する。

部品群


3セル


2セル


2セル

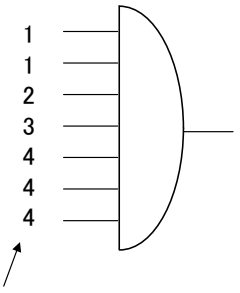

2セル


3セル

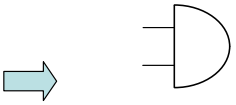

1セル

演習問題5 遅延最適化

以下の回路を、回路周波数が向上するように、最適化せよ。
但し、回路部品として、 2入力ANDのみが利用可能とする。
2入力ANDのゲート遅延は1である。
各入力に書いた値は、入力レジスタからの遅延値である。



レジスタからの遅延



2入力ANDだけで回路を
構成する