# 論理生成システムＡＮＧＥＬの最適化手法

遠藤 真　星野 民夫　唐津 修
（日本電信電話公社 厚木電気通信研究所）

## １．まえがき

論理ＬＳＩの大規模化、複雑化の傾向に対して、現状の設計自動化技術の限界が問題になりつつある。殊に、自動配置配線プログラムの実用化によって、レイアウト工数が大幅に削減された結果、論理設計工数がＬＳＩ設計工数の大半を占めるようになった。

我々は、論理設計の自動化をめざして、レジスタ・トランスファ・レベルの機能仕様から論理ゲートレベルのネットワークを自動生成するシステムＡＮＧＥＬ[1]を開発した。図１に、ＤＡシステムの全体をしめす。自動配置配線プログラムが論理とレイアウトを結びつけているように、ＡＮＧＥＬは機能から論理への橋渡しをする。ＡＮＧＥＬの開発によって、機能からレイアウトに至るＤＡシステムが完結したといえる。

ＡＮＧＥＬは、図２に示すようなレジスタ・トランスファ・レベルの機能記述言語ＦＤＬ[2]で記述された機能仕様を、単純なオペレーションに分割して論理を生成した後、論理の最適化を行う。

論理生成システム実用化の鍵は、生成した論理を所定のテクノロジの制約(基本ゲート構成、ファンイン、ファンアウト等)の下で、いかに最適化するかにある。

本報告ではＡＮＧＥＬの論理最適化手法と、生成した論理をレイアウト・パターン面積上で人手設計と比較した結果、そして今後の課題について述べる。

## ２．論理最適化手法

ＡＮＧＥＬの論理最適化手法は、次の２つに大きく分けられる。1)局所的最適化、2)大域的最適化である。1)は、ゲート近傍の接続関係を調べて、冗長な部分を除く。2)は、多段の論理を２段の論理(キューブ形式)に変換し、多段の論理に再構成する過程で、論理全体をグローバルに見て最適化する。いずれも最適化の対象は組み合わせ論理に限られる。

### ２．１　局所的最適化

次の５つの手法で、冗長部分を除いている。

（１）　定数ファシリティ削除

AND(OR,XOR)に入力される'1'('0')固定入力、及び空入力端子の削減によるファンイン数の削減。

AND(OR)に'0'('1')固定入力があるなど、出力が'0'あるいは'1'に固定されたゲートの削除。

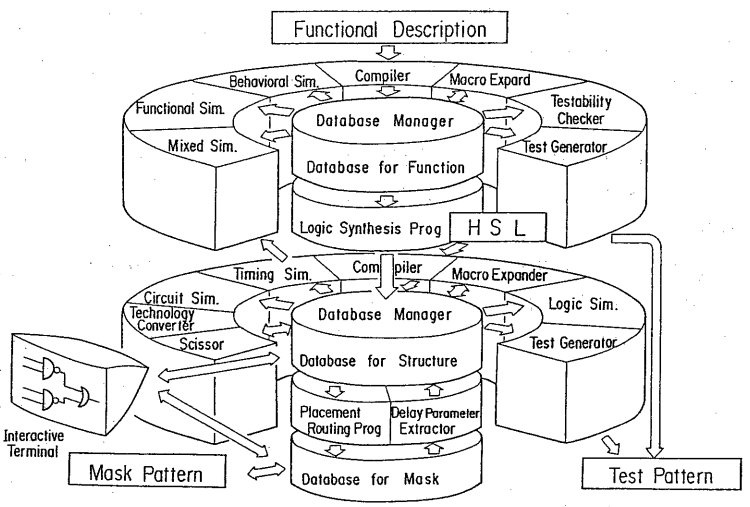



図1．ＤＡシステム

```
NAME    : SAMPLE ;
PURPOSE:ATG ;
LEVEL   :CHIP ;
<EXT>    RUN,CLR,READ,DBUS(1:8),IRIN(1:4),XBUS(1:8);
<INP>    .RUN,.CLR,.READ,.DBUS,.IRIN;
<OUT>    .XBUS ;
<CLK>    P;
<TER>    A(1:8),B(1:8),C(1:2),X(1:8)   ;
<REG>    ACC(1:8),BUF(1:8),IR(1:4) ;
<BOO>    A:=ACC,B:=BUF,C:=IR(3:4) ;
<BOO>    ?C #0  X:=B
            #1  X:=¬B
            #2  X:=A&B
            #3  X:=A!B
             ;
         ;
<AUT>    CNTL:P;
         <STA>;
         IF1:  .RUN!.CLR : ?.CLR? ->IF? % -> IF3 ;;
         IF2:  ACC<-8D0,BUF<-8D0,IR<-4D0,->IF4 ;
         IF3:  .READ : IR <-.IRIN ,->IF4 ;
         IF4:.RUN   : ?IR(1:2) #1 ->EXLD
                               #2 ->EXAR
                               #3 ->EXST
                               %  ->IF1 ; ;
         EXLD:.READ: BUF<-.DBUS,->IF1 ;
         EXAR: ACC<-X , ->IF1 ;
         EXST: .XBUS:=ACC,->IF1 ;
         <END> ;
<END>    CNTL ;
END : SAMPLE ;
CEND ;
```

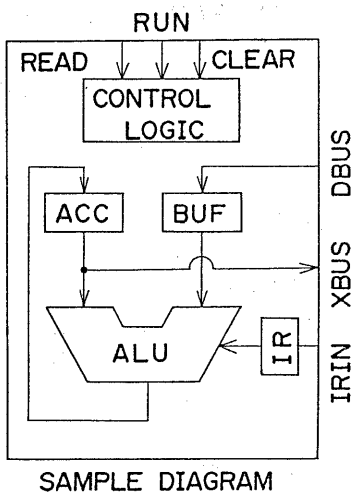



図2. ＦＤＬ記述例

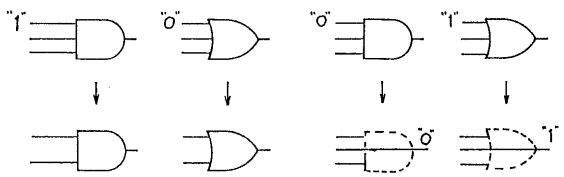



図3. 定数ファシリティ削除適用例

(2) 未利用ファシリティ削減

その出力が他で用いられていないゲートの削除。

(3) スルー・ファシリティ削除

１入力AND,OR,XOR等、入力と出力が常に等しいゲートの削除。(1)等の処理の結果生じる。

(4) ファシリティ共用化

入力信号及びその機能が同一のファシリティの共用化。

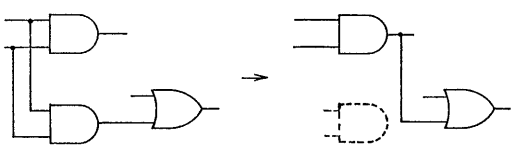


図4. ファシリティ共用化適用例

(5) データ・パス・スイッチ共用化

入力信号が同一で、そのスイッチの閉じる条件が異なるデータ・パス・スイッチの共用化。ビット幅展開前に適用。

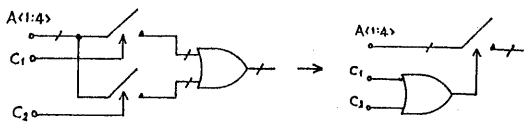



図5. データ・パス・スイッチ共用化適用例

以上のうち(1)~(3)は適用をうけるゲートがなくなるまで繰返される。

２．２ 大域的最適化

ここでは、次の手順で処理を行う。

(1) 論理接続の切り出し

(2) 切り出した論理の積和形式変換

(3) 積項数の縮小化

(4) 積項の共用化

(5) 論理接続への変換

論理を積和２段に変換して最適化を行い、

再び多段論理を組立てているが、本手法の利点は、生成論理に依存せず、論理段数を考慮した最適化が行えることである。以下、各処理の詳細を述べる。

2.2.1　論理接続の切り出し

大域的最適化の対象は組合せ回路なので、レジスタ、メモリ等の入出力端子を境界として回路を切り出し、そこをブロックに置き換える(図6)。回路がレジスタ等で完全に分離される場合、ブロックは複数生じる。

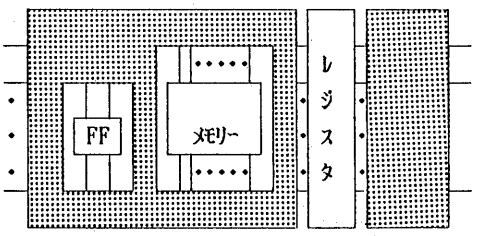



図6．論理接続の切り出し

最適化に要する演算時間、記憶域等の増大を押えるため、回路を適当な単位に分割して処理する必要が生じる。そこで、以下のように、切り出す回路の境界を設定出来るようにしている。

（1）　指定されたファシリティ、あるいは外部端子を入力(出力)側の境界とする。

（2）　指定されたファシリティ、あるいは外部端子を入力(出力)側の境界として、そこから指定された段数のファシリティを出力(入力)側の境界とする。

（3）　全ての入出力外部端子を境界とし、出力側から指定された段数ごとに境界を設定する。

ここで、境界は入力側あるいは出力側のみを指定することも双方を同時に指定することもできる。入力側のみ指定した場合、出力側境界は外部出力端子となる。

なお、論理段数は、AND,OR,XORのうち同一のファシリティが続く場合、それらを１段と数える。なぜならば、それらは、図7のように１つのゲートにおきかえられるからである。

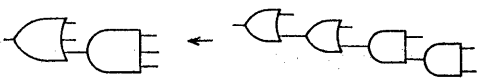


図7．論理段数を１段と数える場合

2.2.2　論理の積和形式変換

切り出した論理接続は、入力端子側から、１ビット・デコード形式のキューブに変換していく。変換された論理は最適化処理の後、論理接続に戻すことも、ＰＬＡとして残しておくことも可能である。

2.2.3　積項数の縮小化

キューブ上で、補元則(AB+$\bar{A}$B=B)、簡化則(A+$\bar{A}$C=A+C)、吸収則(A+AC=A)等、基本的な最適化をする他、MINI[3]で用いている演算によって積項数を縮小する。

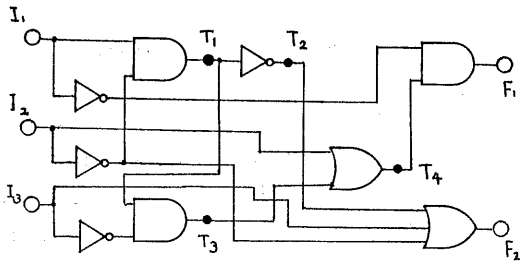
2.2.4　積項の共用化

本処理において、２段の論理を多段論理に再構成する。多段論理は、図8のように中間変数を入力パート、出力パートにとった拡張キューブ形式で表わすことにする。こうすると、２段から多段まで同一のデータ形式なので、データが、統一的に扱え、操作しやすい。

ここで、最適化にあたっての評価関数を考える。ＬＳＩのコストを考えるとき、面積を評価関数にとればよさそうである。局所的最適化ではゲート数とファンイン数、大域的最適化(3)では、積項数を減少する方向で処理を進めた。ＰＬＡの面積は積項数に比例するが、一般の回路では、回路を構成する論理ゲートの面積がトランジスタ数にほぼ比例し、そのトランジスタ数が論理ゲートのファンイン数にほぼ比例する。したがって、面積にかわり、論理ゲートのファンイン数の総和(総ファンイン数)を評価関数として用いることにする。この妥当性につ

いては後に、レイアウト・パターン面積の評価で述べる。局所的最適化処理も、総ファンイン数を減少する方向に進めていると見ることができる。

(面積)∝(トランジスタ数)∝(ゲートの総ファンイン数)




| | 入力部 $I_1I_2I_3T_1T_2T_3T_4$ | 出力部 $F_1F_2T_1T_2T_3T_4$ |
|---|---|---|
| 1 | 1 0 x x x x x | 0 0 1 0 0 0 |
| 2 | x x x 0 x x x | 0 0 0 1 0 0 |
| 3 | x x 0 1 x x x | 0 0 0 0 1 0 |
| 4 | x 1 x x x x x | 0 0 0 0 0 1 |
| 5 | x x x x x 1 x | 0 0 0 0 0 1 |
| 6 | 0 x x x x x 1 | 1 0 0 0 0 0 |
| 7 | x x x x 1 x x | 0 1 0 0 0 0 |
| 8 | x 0 x x x x x | 0 1 0 0 0 0 |
| 9 | x x 1 x x x x | 0 1 0 0 0 0 |

図8. 拡張キューブ形式

2.2.4.1 AND共用化処理

本処理は基本的には因数分解(factorization)である。この処理をキューブ形式のデータに適用するため、マスク積(mask product)[4]とサブトラクト(subtract)演算を定義する。

[定義1] キューブ間のマスク積

2つのキューブ

$c=c(1)\,c(2)\cdots c(r)\,|\,d(1)\,d(2)\cdots d(t)$

と

$e=e(1)\,e(2)\cdots e(r)\,|\,f(1)\,f(2)\cdots f(t)$

のマスク積を

$c \;m\; e$

で表わし、次のように定義する。但し、c(i),e(i)はキューブc,eの入力パート、d(j),f(j)は出力パートのjビット目である。

入力部

$c(i)\;m\;e(i) = c(i)$ if $c(i)=e(i)$

$= X$ otherwise

出力部

$d(j)\;m\;f(j) = d(j)$ if $d(j)=f(j)=1$

$= 0$ otherwise

[定義2] キューブ間のサブトラクト

2つのキューブc, eのサブトラクトを

$c \;s\; e$

で表わし、次のように定義する。

入力部

$c(i)\;s\;e(i) = X$ if $c(i)=e(i)$

$= c(i)$ otherwise

出力部

$d(j)\;s\;f(j) = d(j)$

因数分解による積項の共用化アルゴリズムを以下に示し、例を図9に示す。

ここで、

$F \ni c_i$

$c_i = c_i(1)\,c_i(2)\cdots c_i(r)\,|\,d_i(1)\,d_i(2)\cdots\cdots d_i(t)$

$1 \le i \le n$

$M = M(1)M(2)\cdots M(r)\,|\,N(1)N(2)\cdots N(t)$

[積項の共用化アルゴリズム]

(ステップ1) マスク演算により、キューブの共通項を求める。

$M(i) = c_1(i)\;m\;c_2(i)\;m\;\cdots\;m\;c_n(i)$

$N(j) = 0 \quad 1 \le j \le t$

(ステップ2) 各キューブから、共通部分をサブトラクト演算によって除く。

$c_i = c_i\;s\;M \qquad 1 \le i \le n$

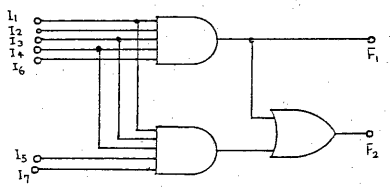
(ステップ3) 新しい中間変数のパートを入力部(r+1)、出力部(t+1)に設ける。

(ステップ4) $c_i$,Mに中間変数をセットする。

$c_i(r+1) = 1$

$N(t+1) = 1$

| $I_1I_2I_3I_4I_5I_6I_7$ | $F_1F_2$ |
|---|---|
| □1□□x1x | 11 |
| □x□□1x1 | 01 |



| $I_1I_2I_3I_4I_5I_6I_7T_1$ | $F_1F_2T_1$ |
|---|---|
| x1xxx1x□ | 110 |
| xxxx1x1□ | 010 |
| □x□□xxxx | 00□ |

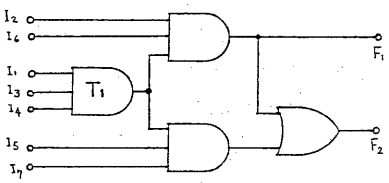


図9. 積項の共用化処理

この処理によって常に総ファンイン数が減少するわけではない。総ファンイン数が処理の前後で不変のときは、本処理によって論理段数が1つ増加するので、処理を行わない。減少するときのみ、行う。(図10参照)

積項を共用化する順序により、多段化したときの論理構成が大きく変わり、総ファンイン数も変わる。

順序づけの基本は共通項の数が最も多いものから行うことである。

ここで共通項の数をキューブの類似度で表し、これを要素とする類似度行列を定義する。

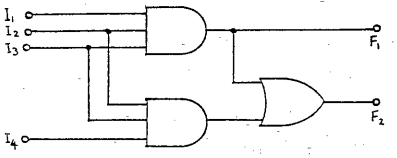
[定義3] キューブcとeの類似度sは入力部でX以外に一致しているパートの数である。

[定義4] アレイFの類似度行列 S

N個のキューブ$c_i$からなるアレイFの類似度行列Sは$c_i$, $c_j$の類似度を$s_{ij}$とすると、

$$S(i,j) = s_{ij} \quad (i \neq j)$$
$$= 0 \quad (i=j)$$

| $I_1I_2I_3I_4$ | $F_1F_2$ |
|---|---|
| 1□□□x | 11 |
| x□□□1 | 01 |



| $I_1I_2I_3I_4T_1$ | $F_1F_2T_1$ |
|---|---|
| 1xxx□ | 110 |
| xxxx□ | 010 |
| x□□□xx | 00□ |

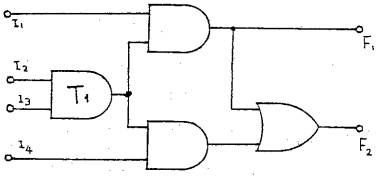


図10.積項の共用化でファンイン数が減少しない例

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 3 | 2 | 0 | 1 |
| 2 | 3 | 0 | 3 | 0 | 0 |
| 3 | 2 | 3 | 0 | 1 | 2 |
| 4 | 0 | 0 | 1 | 0 | 3 |
| 5 | 1 | 0 | 2 | 3 | 0 |

図11. 類似度行列の例

次に、類似度行列を用いた積項共用化の順序づけアルゴリズムを述べる。

[積項共用化の順序づけアルゴリズム]

(ステップ1) 類似度行列Sをもとめる。

(ステップ2) Sから最大値に等しい要素$s_1, s_2, \ldots, s_n$を取り出す。但し、類似度の定義より$s_{ij}=s_{ji}$なので要素はどちらかを選ぶ。

(ステップ3) 総ファンイン数が減少するようなキ

ューブの組（$c_i$, $c_j$）$\lfloor s_{ij} \rfloor$がなければ、終了。減少するなら、共用化処理をして(ｽﾃｯﾌﾟ1)へ。

処理が進むに従い、項数が１のキューブが生じてくる。このようなキューブは他のキューブと項を共用化することがないので、処理対象からはずし、類似度の計算を省くことによって、処理時間の短縮を図っている。

2.2.4.2 ＯＲ共用化

これまでの処理はANDの共用化と見られるが、同様にORの共用化が考えられる。即ち、複数の出力間でORをとられているキューブを共用化するのである。

ANDの共用化ではキューブの入力部に着目して、マスク積により共通項を求めたが、ORの共用化では出力部の各出力に対応するビット列をキューブにみたてて共用できるキューブを探す。処理の過程はANDの共用化と同様である。図12に例を示す。総ファンイン数は１８から１７に減少している。

2.2.5 論理接続への変換

本処理は多段化されたキューブ表現の論理を論理ゲートのネットワークに変換する。各キューブの入力部がANDゲートに対応し、出力部がORゲートに対応するのはＰＬＡと同様だが、中間変数の出力が入力にも現れている点が異なる(図8参照)。

2.2.5.1 ファンイン制限処理

論理接続への変換の際、ゲートのファンイン数を制限して変換できる。しかも、以下に述べるように、回路の論理段数を考慮して処理できるという特徴をもつ。これらの処理は、論理接続変換後に行うより、キューブ上で行う方がはるかに容易である。

ファンインを制限して展開する方法には２つある。

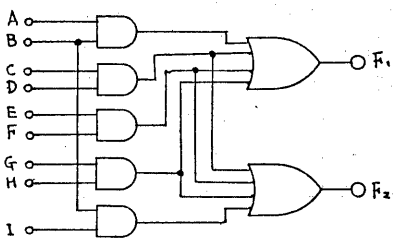
(1) 論理段数を等しくする方法(図13)

(2) 回路全体の平均化を考慮する方法(図14)

である。(1)は入力から出力までの段数が、皆等しくなるように、木状に展開する。(2)は端から制限値に等しい入力を収容していく方法で、回路の入力端子から各入力までの段数を調べ、段数の多い入力を、展開後に段数の少ない入力端子に割付る。こうすることによって、回路全体の段数のばらつきを緩和し、クリティカル・パスの伸びを押える。(1)は展開したゲートの段数を小さく押えられるが、(2)より総ファンイン数が多くなる。一方(2)は、段数がむやみに増えることがあり回路全体の段数のばらつきを、かえって増す場合がある。そこで、(1)と(2)を併用することも考えられる。

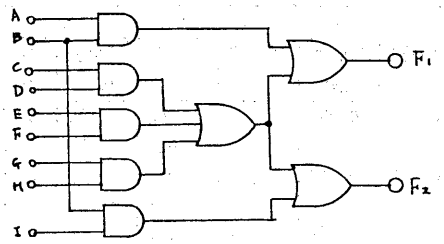
例えば、ファンイン数を３にしたいとき、(1)で７にし、さらに(2)で３にすると図15のようになり、段数の差はそれほど大きくならずにすむので、この方法は有用である。

なお、本処理はANDとOR独立にファンイン数を設定できる。




| | A | B | C | D | E | F | G | H | I | $F_1$ | $F_2$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | x | x | x | x | x | x | x | 1 | 0 |
| 2 | x | x | 1 | 1 | x | x | x | x | x | □ | □ |
| 3 | x | x | x | x | 1 | 1 | x | x | x | □ | □ |
| 4 | x | x | x | x | x | x | 1 | 1 | x | □ | □ |
| 5 | x | 1 | x | x | x | x | x | x | 1 | 0 | 1 |

図12-1　ＯＲの共用化




| | A | B | C | D | E | F | G | H | I | T | $F_1$ | $F_2$ | T |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | x | x | x | x | x | x | x | x | 1 | 0 | 0 |
| 2 | x | x | 1 | 1 | x | x | x | x | x | x | 0 | 0 | 1 |
| 3 | x | x | x | x | 1 | 1 | x | x | x | x | 0 | 0 | 1 |
| 4 | x | x | x | x | x | x | 1 | 1 | x | x | 0 | 0 | 1 |
| 5 | x | 1 | x | x | x | x | x | x | 1 | x | 0 | 1 | 0 |
| 6 | x | x | x | x | x | x | x | x | x | 1 | 1 | 1 | 0 |

図12-2. ＯＲの共用化

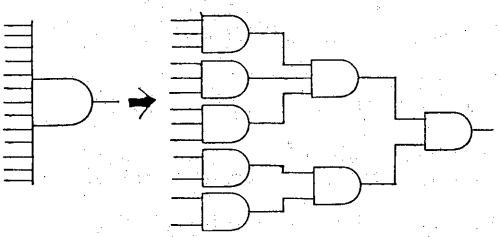


図13. ファンイン制限処理 (1)

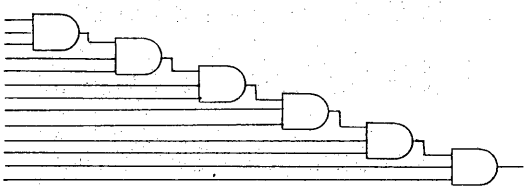


図14. ファンイン制限処理 (2)

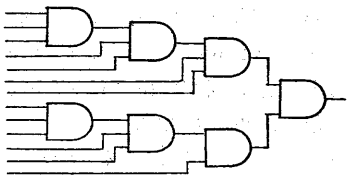


図15. ファンイン制限処理 (1),(2)併用例

## ３．自動設計と人手設計の比較

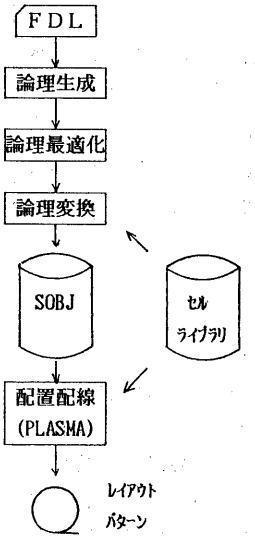



図16. 自動設計過程

図16に示すようにＦＤＬで記述した機能仕様から論理生成、２章で述べた最適化処理を経て、テクノロジ変換[5]により、実際のセルライブラリを用いた論理構成に変換した後、自動配置配線プログラム ＰＬＡＳＭＡ[6]によって、レイアウト・パターンを作成した。このように自動設計によって得られた論理回路及びレイアウト・パターンと人手設計したものを比較した結果を表１に示す。自動設計のためのＦＤＬ入力仕様は機能シミュレーション用に記述されたデータをそのまま用いた。セル・ライブラリはＣＭＯＳ ２．５μmルールのものである。

最適化処理を行わない場合には、人手設計の１．２～２．７倍であった生成論理が[1]、最適化処理により、セル、配線領域を合わせたレイアウト・パターンの全面積で、０．７～１．４の範囲に納めることができた。セル面積、ファンイン数とも全面積とほぼ同様の傾向である。

また、セル面積がファンイン数にほぼ比例することから、面積を評価するのにファンイン数を用いることが妥当であることを確認した。

回路Ｃで、ファンイン数の比に対し、セル面積、全面積の比が小さいのは、テクノロジ変換の結果、複合化により高機能化した、密度の高いセルをうまく利用することができたためである。

図17に、回路Ｃの自動設計レイアウト・パターンの例を示す。

## ４．考察・まとめ

ＡＮＧＥＬは比較的単純な局所的最適化と、論理全体をキューブ形式に変換して行う大域的最適化によって、生成論理を最適化している。機能仕様からレイアウトパターンまでの完全自動設計を試みた結果、論理生成を意識せずに記述された仕様から、人手設計に近い論理を自動生成できることが確認できた。

| | 人手設計 | | | | ANGEL | | | |
|---|---|---|---|---|---|---|---|---|
| 回路 | ゲート数 | ファンイン数 | セル面積 ($mm^2$) | 全面積 ($mm^2$) | ゲート数 | ファンイン数 | セル面積 ($mm^2$) | 全面積 ($mm^2$) |
| A | 49 | 87 | 0.057 | 0.090 | 35 | 89 | 0.059 | 0.097 |
| | | | | | | (1.02) | (1.03) | (1.08) |
| B | 90 | 177 | 0.119 | 0.237 | 107 | 202 | 0.138 | 0.265 |
| | | | | | | (1.14) | (1.16) | (1.12) |
| C | 464 | 732 | 0.553 | 1.381 | 371 | 736 | 0.406 | 0.953 |
| | | | | | | (1.01) | (0.73) | (0.69) |
| D | 743 | 1596 | 0.985 | 2.244 | 981 | 1955 | 1.275 | 3.101 |
| | | | | | | (1.22) | (1.29) | (1.38) |

注：（）内［ANGEL］/［人手設計］

表１．ANGELによるレイアウト結果比較

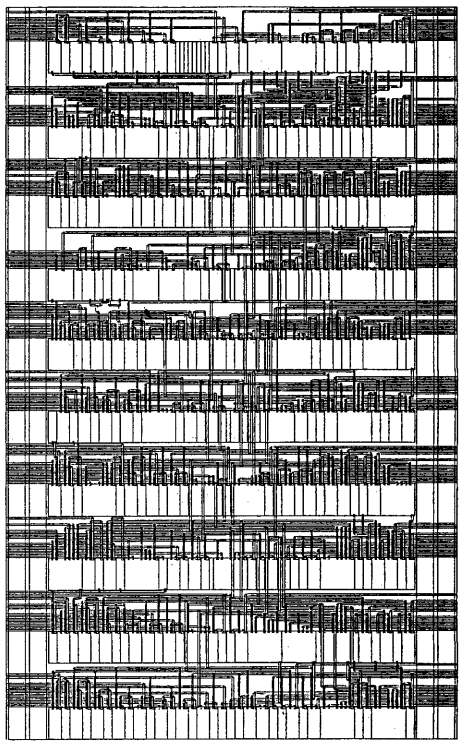


図17．自動設計レイアウト・パターン例

このようにＡＮＧＥＬは生成された論理の品質から見て、十分実用的なシステムとなっているが、まだ開発途上にあり、課題も残されている。以下に、これまでに得られた知見と、最適化に関するＡＮＧＥＬの今後の課題について述べる。

（１）　現在のＡＮＧＥＬは、論理の最適化のみを行っている。今後は、同時に使用されない演算器等のハードウェア資源の共用など、機能レベルでの最適化が必要である。

（２）　回路規模の増大に伴い、最適化処理に要する時間が大きな割合を占めてくる。最適化アルゴリズムの高速化、効率的な適用を図る必要がある。

（３）　処理時間を押えるため、回路を適当な規模に分割する方法がある。しかし、機械的に分割したのでは最適化の効果があがらないことがある。機能に着目した分割法などを検討する必要がある。

○参考文献


1) 遠藤、星野、永谷『ゲート自動生成プログラムＡＮＧＥＬの試作』
昭57後期情処全大3N-11

2) 樋浦、渡辺、菊地、遠藤、和田、杉浦『ＦＯＲＥＳＴ言語(ＦＤＬ)トランスレータ』
昭56信学全大457

3) S.J.Hong, R.G.Cain, and D.L.Ostapko, "MINI : a heuristic approach for logic minimization," IBM J.Res.Develop.,vol. 18, pp.443-458. Sept. 1974

4) P.Roth,"Computer logic, testing, and verification," Computer Science Press.

5) 松広『論理回路変換プログラム』
昭57信学総全大355

6) 永谷、杉山、堀口、須藤『機能セルによるレイアウトプログラム』　昭56信学全大431

7) 星野、唐津、中島『ＶＬＳＩ設計言語拡張ＨＳＬ』　昭59信学総全大386