設計自動化46－7
（1989. 2. 21）

ＡＳＩＣ用対話型セル設計システム ＳＰＡＣＥ

鈴木　五郎[1] 山本　哲也[1] 浜田　亘曼[1]　岡村　芳雄[2] 加藤　真司[2] 佐藤　多加志[2]

[1]日立製作所　日立研究所　　[2]同デバイス開発センター

大規模セルから構成されるＡＳＩＣ用に、高密度で高性能なセルを短期間で設計するための対話型設計システムを開発した。エンジニアリング・ワークステーションを使用しており、階層設計機能やユーザ独自の命令体系が簡単に構築できる基本的なパターン編集機能の他に、パターン編集過程で幾何学的な設計規則違反箇所がないか否かを即刻チェックする機能を持っている。本システムは高性能論理ＡＳＩＣの設計に適用し有効性を確認している。

Interactive Cell Design System for ASIC

Goro Suzuki Tetsuya Yamamoto Nobuhiro Hamada [1]
Yoshio Okamura Shinji Kato Takashi Sato [2]

[1] Hitachi Res. Lab. [2] Hitachi Device Development Center

Interactive large cell design system for ASIC is described. This system can handle not only hierarchical design and macro-command definition but also geometrical design rule incremental checking. This system has been applied to the high performance logic ASIC and justified its effectiveness.

1．まえがき

大規模セル（最大５０００個程度のトランジスタから構成される）から構成されるＡＳＩＣでは、高密度で高性能なセルを短期間で設計しなければならない。この分野ではモジュール・ジェネレータ等の自動設計システムが種々報告されているが、上記した条件を全て満足するにはいたっていないのが現状である。このため、人手による設計を基本とせざるをえない。そこで我々は対話型パターン編集機能と高速な検証機能とを密結合したシステムＳＰＡＣＥ（ Sophisticated Pattern Editor for VLSI Cell Design ） を開発した。[1)2)3)]エンジニアリング・ワークステーションを使用しており、 パターン編集機能としては、効率よく大規模セルの設計ができるよう階層設計機能を用意し、ユーザ独自の命令体系が簡単に構築できるようにマクロ・コマンド編集機能を持たせている。検証としてはパターン間の間隔等の幾何学的な設計規則をオンラインでチェックする機能（ 以下ＤＲＣ： Design Rule Checkと呼ぶ ）を持たせている。本報告ではＳＰＡＣＥの最も大きな特徴であるオンラインＤＲＣを取り上げ、 機能の詳細とその実現方法を中心に述べる。

2 オンラインＤＲＣの機能

基本的にはＤＲＣをかけながらパターンの編集を行い、設計規則違反箇所がある場合にはその場で修正をしてエラーを後の工程に残さない設計を考えている。ただし、複数のパターンが揃わないとＤＲＣをしても意味が無い場合もあるため、パターン編集中常時チェックを行うか否かを自由に制御できるようにしている。又既に配置されているパターンに関して指定した矩形領域内に存在するもの、あるいは個別に指定したものをコマンド投入時にチェックする機能も用意した。

設計規則に違反する該当箇所にはエラー図形を表示するが、パターンを修正することによって設計規則違反箇所が消滅した場合には、エラー図形の消去を行って修正作業が正しいか否かが即刻分かるようにした。

設計規則は４０種程の演算子（寸法演算子、論理演算子、位相演算子からなり、表１にその一部を示す）を組合せて表現する。図１に示した３つの例に関して設計規則を記述したものを図２に示す。ＧＬＯＢＡＬとＧＬＯＢＡＬ－ＥＮＤ文で挟まれた部分でＮ層領域に存在するＬ層図形とＮ層領域以外、つまりＰ層領域に存在するＬ層図形とを区別し、Ｎ層領域に存在するゲートとＰ層領域に存在するゲートとを区別している。ＧＬＯＢＡＬ－ＥＮＤ文以降では、上記した演算結果を用いてゲート間の間隔、ゲートとコンタクト間の間隔、ＦＧＰ層パターンとＬ層パターン間の余裕と侵入度合いに関する設計規則を記述している。

3 フィールド・ブロック・データ構造

前章で述べたように、ＤＲＣをかけながらパターンの編集を行うことを基本としていることから、パターンの編集処理時間に対してＤＲＣ処理時間は無視できるものでなければならない。一個のパターンに関してＤＲＣをかけた場合、ＣＰＵ処理時間が平均０．１秒以下になるように目標を設定した。

ＤＲＣ処理では、近傍図形を検索する時間が処理時間全体のほとんどを占めることから、近傍図形検索が高速処理できるデータ構造が必要である。我々は、実用性に重点を置いて、次の４つの条件 １）高速編集操作 ２）高速検証処理 ３）少ないデータ量 ４）容易なデイスク・スワッピングを全て満足できるデータ構造（以下、フィールド・ブロック・データ構造と呼ぶ）を検討した。[4)5)]

3．１フィールド・ブロック・データ構造の基本概念

図３に示すように、図形情報を格納するメモリ領域として、フィールド・ブロック・メモリと呼ぶ同一サイズのメモリ・ブロックをあらかじめ用意しておく。入力した図形は入力順に次々とこのメモリ・ブロックに格納し、一つのメモリ・ブロックが満杯になったら次のメモリ・ブロックに格納するという処理を行なう。各図形は、それを完全に包含する面積最小な矩形（以下図形包含矩形と呼ぶ）の左下点及び右上点の座標（P_{min}，P_{max}）をその管理情報として持ち各メモリ・ブロックもその中に格納されている全ての図形を完全に包含する図形包含矩形（P_{min}，P_{max}）をその管理情報として持つ。図３の右の図は、２個のメモリ・ブロックにｇ１～ｇ６（ｇｉのｉは図形入力順を表す）の図形情報が格納されている様子をまた左の図は、図面上でそれぞれのメモリ・ブロックに対応した図形包含矩形（以下フィールド・ブロックと呼ぶ）がどのような広がりになるかを表している。

既配置図形を移動した場合にデータ構造をどのように更新するかを図４を用いて説明する。例えばｇ３とｇ４を移動した場合、該当図形の管理情報と該当図形がそれぞれ含まれるメモリ・ブロックの管理情報とを更新する。両方とも図形包含矩形情報の更新となるため、高速に処理できる。

ここで、このフィールド・ブロック・データ構造を用いた近傍図形検索手順について説明する。以下では図５のように、図面中で指定した矩形状ウインドウ（図中▨で示した図形）内に存在する図形を検索するウインドウ検索を考える。

（Ｓ１）ウインドウにかかるフィールド・ブロックを抽出する。

（Ｓ２）（Ｓ１）で抽出したフィールド・ブロックの

中に存在する全ての図形に注目し、ウインドウにかかる図形包含矩形を持つ図形を抽出する。

（Ｓ３）（Ｓ２）で抽出した図形の中から、詳細形状がウインドウにかかるものを抽出する。

フィールド・ブロック・データ構造は主記憶と補助記憶との間のデータ・スワッピング処理も容易に行なうことができる。例えば図６のように全メモリ・ブロックの管理情報は主記憶上の制御テーブルに常駐させておく。このテーブルには管理情報の他に該当するメモリ・ブロックが主記憶と補助記憶のどちらに存在するのか、さらに存在する位置に関する情報も持たせる。図６の例では補助記憶上に全メモリ・ブロックを常駐させ、主記憶上のメモリ・ブロック・テーブルにその一部分を常駐させており、必要なメモリ・ブロックが主記憶上に存在しない場合には補助記憶上からローデングする。例えば、メモリ・ブロック・テーブルｋ番地に存在する不要なメモリ・ブロックＦＢＩを補助記憶上Ｉ番地へもどし、補助記憶上Ｊ番地にあるＦＢＪの情報をメモリ・ブロック・テーブルｋ番地へローデングする。フィールド・ブロック・データ構造は全てのメモリ・ブロックが同一サイズであることから、このデータ・スワッピング処理が簡単に行なえる。

以上、フィールド・ブロック・データ構造の概念を説明したが、このデータ構造はフィールド・ブロックの広がりが大きくなると近傍図形検索速度が遅くなるという弱点がある。どのような大きさの図形を、どのような順序で入力したとしてもフィールド・ブロックどうしがなるべく重ならないように制御する次の方式を考案した。

３．２　バケット分割法との組み合わせ

図面をあらかじめ図７（ａ）のように複数個の同一サイズの矩形状バケットに分割しておき、各バケットに対応してメモリ・ブロックを用意しておく。一つのメモリ・ブロックで不足した場合は、複数個を対応させる。各図形情報をその図形包含矩形の左下点が所属するバケットに対応するメモリ・ブロックに格納する。今６個の図形を入力したとする。例えばｇ５のばあい、その図形包含矩形の左下点（図中・を付けた点）はバケット２に所属していることから、バケット２に対応するメモリ・ブロックに格納することになる。４個のフィールド・ブロックの広がりを示すと図７（ｃ）のようになる。

バケット分割法を利用することによって、もう一つの利点が得られる。図８に示すようなウインドウにかかるフィールド・ブロックを抽出する場合、指定したウインドウの右上点（図中・を付けた点）が所属するバケットよりも上側あるいは右側に存在するバケットに対応するフィールド・ブロックはウインドウにかかるはずがないことから、これらのフィールド・ブロックに関するチェックは初めから不要となる。特にウインドウが図面の左下付近に設定された場合にはチェックしなければならないフィールド・ブロックを相当絞りこむことができる。

３．３　大型図形の封じこめ

大型図形があちこちのメモリ・ブロックに分散してしまうと、大型図形が存在するフィールド・ブロックの広がりが大きくなってしまう。そこで、大きさがほぼ同じ図形を同一メモリ・ブロックに格納することによって、この現象が回避できる。図９（ｂ）の例では、大型と小型の図形に対応して２つのメモリ・ブロックを用意している。図９（ａ）のように１２個の図形を入力した場合、大型図形は全て一つのメモリ・ブロックに格納され、結果として図９（ｃ）に示すようにフィールド・ブロックの重なりがうまく制御できることになる。

大型と小型の図形を区別する手段としては面積を実際に計算し、２つのグループに分ける方法がある。しかし、ＶＬＳＩマスク・パターン図形の場合は、それがどのマスク層に所属するかによって、大きさがほぼ一定になる。例えば、コンタクト層とアルミ層に属する図形は大型と小型の図形にはっきりとグループ化できる。つまり、図形の大きさに対応してメモリ・ブロックを用意する代わりに、マスク層に対応してメモリ・ブロックを用意してもかまわないことになる。

マスク層に対応してメモリ・ブロックを用意することによって、次に示す効果も現われる。

（１）図面編集中にあるマスク層に所属する図形だけを選択するという操作をよく行なうが、この処理が高速に行なえる。

（２）設計規則として考慮しなければならないのは、３～４個のマスク層である場合が多く、ＤＲＣを行なう場合にもマスク層毎に図形が区別できたほうが、効率が良い。

３．４　多段階層管理化

フィールド・ブロックの集合をマクロ・フィールド・ブロックとすることによって、管理の階層を容易に増やすことができる。図１０にその例を示す。

第３．１節で説明したウインドウ検索は次の手順に従う。

（Ｓ０）ウインドウにかかるマクロ・フィール・ブロックを抽出する。

（Ｓ１）
〃　第３．１節の（Ｓ１）～（Ｓ３）と同じ。
（Ｓ３）
以上の考え方を拡張することによって、ｎ段階層化が図れる。

４．　オンラインＤＲＣの処理手順

注目している図形の図形包含矩形を設計規則の最大値Ｄだけ拡大し、それをウインドウとして第３．４節で述べた手順に従ってチェックの対象となるパターンを抽出した後、次ぎの手順でパターン間の距離の計算を行う。

（Ｓ１）注目しているパターンの一つの辺に注目する
（Ｓ２）その辺を基準にして、その前後にＤだけ余裕をもたせた状態で図１１のようにパターンの外側あるいは内側に、幅Ｄの矩形Ｒを発生させる。ここで、矩形をどちら側へ発生させるかは表２に示すようにチェックしたい設計規則の種類によって決められる。
（Ｓ３）チェックの対象となるパターンを構成している辺の中から、距離計算を行う対象となるものを抽出する。
　（Ｓ３１）各パターンの辺は、時計回りの方向情報を持っているとすると、注目している辺の持つ方向とチェックしたい設計規則のの種類によって、距離計算を行う辺の持つべき方向が表３のように一意的に決められる。そこで、まず方向情報だけを使って距離計算の対象となる辺の候補を上げる。
　（Ｓ３２）（Ｓ３１）で抽出した辺の中で、矩形Ｒ内に存在するものを抽出する。
（Ｓ４）（Ｓ３）で抽出した辺と注目している辺との距離を求める。[6]
　注目辺をＬ、距離計算対象辺をｌ、各辺の左右の頂点をＰ１，Ｐ２，ｐ１，ｐ２とすると、４個の頂点から相手の辺までの距離をもって２辺間の距離とする。頂点から相手の辺に垂線を下せる場合にはその垂線の長さ、下せない場合には相手の辺の頂点のうちで一番近いものまでの長さ、をもって頂点と辺との距離と定義する。図１２に４個の頂点から相手の辺への距離①～④を表した例を示す。
（Ｓ５）（Ｓ４）で求めた辺間の距離のうちで設計規則ｄよりも短いものが一個でも存在する場合にはエラー図形を表示する。４個の頂点から相手の辺への距離と設計規則ｄとの関係により、例えば図１３に示したエラー図形を表示する。
（Ｓ６）（Ｓ１）～（Ｓ５）を注目しているパターンを構成している辺の数だけ繰り返す。

５．　プログラム性能評価結果

ＤＲＣをかけながらパターンを入力する処理時間を測定した。結果を図１４に示す。横軸は編集している図面の規模つまり図面中に含まれるパターンの数を表している。また縦軸は３ＭＩＰＳ計算機を使用した場合のＣＰＵ時間を表している。図１の（ｂ）に示したように、ＦＧＮ層パターンを入力し、ＣＯＮＴ層パターンとの間の間隔チェックを行う場合を例題としているが、２０秒～３０秒の処理時間である。

図１５は既に配置されているパターンを指定し、ＤＲＣする時間を表している。図１（ｂ）（ｃ）に示したように、ＦＧＰ層パターンを指定し、ＣＯＮＴ層パターンとの間の間隔チェックとＬ層パターンとの間の余裕と侵入度合いのチェックを行う場合を例題としているが、１０秒～２０秒の処理時間である。

扱っている図面の規模が大きくなると、チェックの処理時間がどんどん長くなると懸念されるが、実際には図面の規模にほとんど依存しない処理時間となっていることが分かる。また、最初我々が設定したＣＰＵ処理時間０．１秒の目標を達成している。

６．むすび

ＡＳＩＣ用対話型セル設計システムＳＰＡＣＥを開発した。ＳＰＡＣＥの持つ機能のうちで最も大きな特徴であるオンラインＤＲＣを高速処理するためにデータ構造から検討を行った。プログラムの性能評価をした結果、取り扱う図面の規模にはほとんど依存せずに一個のパターンに関するＤＲＣでは０．１秒以下のＣＰＵ処理時間（３ＭＩＰＳ計算機）であり、十分実用化できることが分かった。

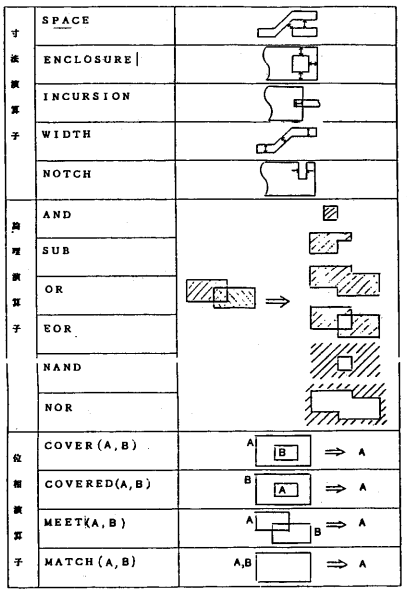
ＳＰＡＣＥは現在メモリや高性能論理ＬＳＩのセル設計に適用されている。

参　考　文　献

１）加藤　他：ＳＰＡＣＥ：対話型マスク・パターン設計システムーシステム概要ー、第３７回情報処理全国大会、ｐｐ．1800-1801 、昭和６３年９月
２）山本　他：ＳＰＡＣＥ：対話型マスク・パターン設計システムーオンライン・マクロ定義ー、第３７回情報処理全国大会、ｐｐ．1800-1801 、昭和６３年９月
３）鈴木　他：ＳＰＡＣＥ：対話型マスク・パターン設計システムーオンライン検証ー、第３７回情報処理全国大会、ｐｐ．1800-1801 、

昭和６３年９月
4）鈴木五郎：大規模図面情報管理における高速処理方式の検討、情処論、Vol.27,No.4,pp.454-461 昭和６１年４月
5）G.Suzuki:Practical Data structure for Incremental DRC and Compaction,Proc. of ICCD'87 pp.442-447(Oct.1987)
6）T.Asano et al.:Computational Geometry Algorithm (in Layout Design and Verification), North Holland,pp.295-347,1986

表１ 演算子の種類



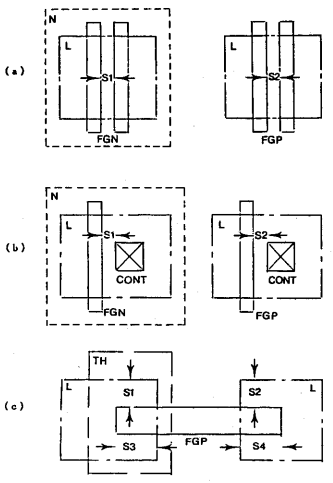


図１ 条件付き設計規則例

```
GLOBAL
        LN=AND (L, N)
        LP=SUB (L, LN)
    LNGATE=AND (LN, FGN)
    LPGATE=AND (LP, FGP)
GLOBAL-END

M GATE-GATE SPACE
        ER1=SPACE (LNGATE)          0<S1<1.5
        ER2=SPACE (LPGATE)          0<S2<1.0

M GATE-CONT SPACE
    LNCONT=AND (LN, CONT)
    LPCONT=AND (LP, CONT)
        ER1=SPACE (LNCONT, LNGATE)  0<S1<1.5
        ER2=SPACE (LPCONT, LPGATE)  0<S2<1.0

M DIFFUSION-POLYSILICON SPACE
         W1=AND (LP, TH)
         W2=SUB (LP, W1)
        ER1=ENCLOSURE (FGP, W1)     0<S1<1.5
        ER2=ENCLOSURE (FGP, W2)     0<S2<1.0
        ER3=INCURSION (FGP, W1)     0<S3<1.5
        ER4=INCURSION (FGP, W2)     0<S4<1.0
```

図2 'SPACEにおける設計規則記述例

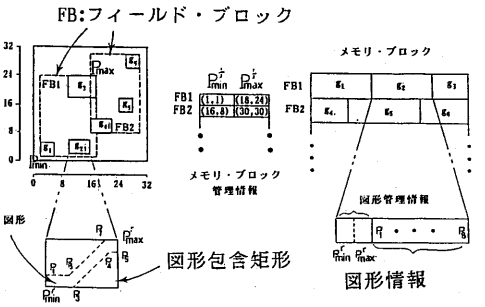


図３ フィールド・ブロック・データ構造の基本概念

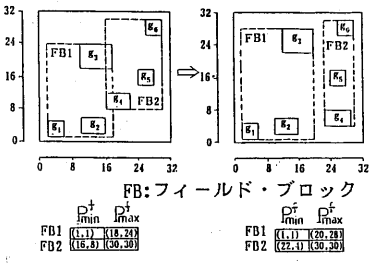


図４ データ構造の更新

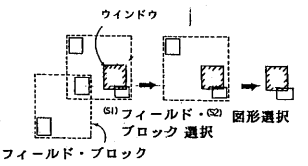


図５ ウインドウ検索処理手順

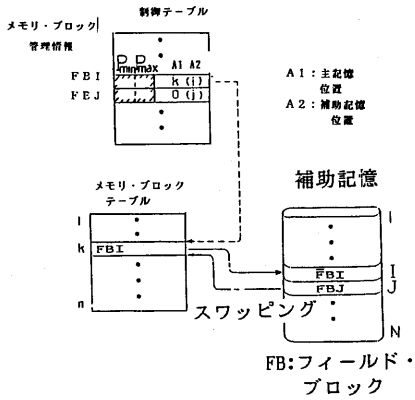
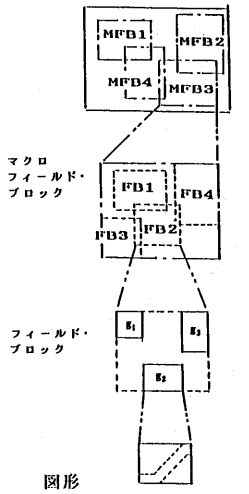
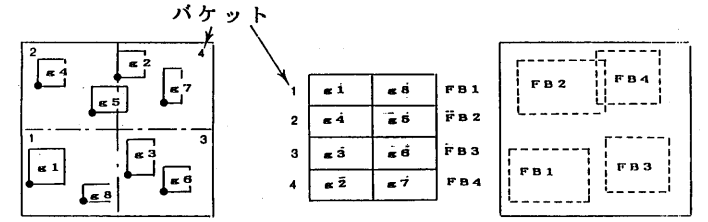


図6　データ・スワッピング・メカニズム



図形

図１０　マクロ・フィールド・
ブロックの概念



(a) バケット分割　(b) 各バケットに対応したメモリ・ブロック　(c) フィールド・ブロックの広がり

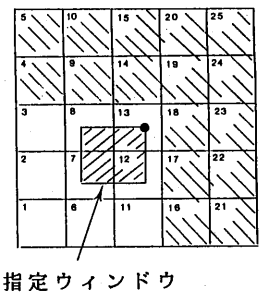
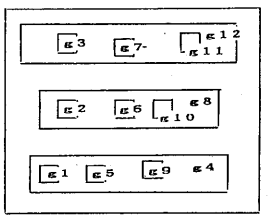
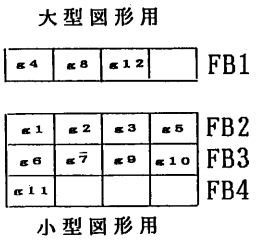


図8　ウインドウ検索のバケット利用

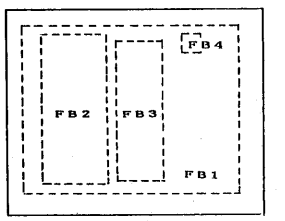
図７　バケット分割法との組み合せ



(a) 図形入力状況



(b) メモリ・ブロックのグループ化



(c) フィールド・ブロックの広がり

図９　大型図形の封じ込め

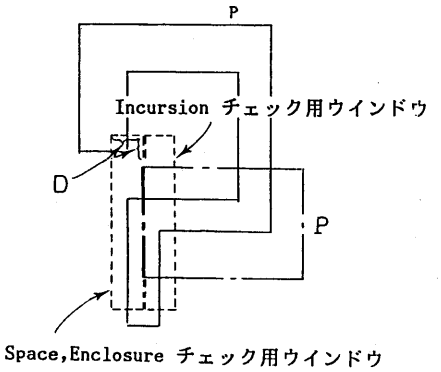


図11　ウインドウの設定

表 2　ウインドウ R を設定する方向

1	Space	外側
2	Enclosure	外側
3	Incursion	内側
4	Width	外側
5	Notch	外側

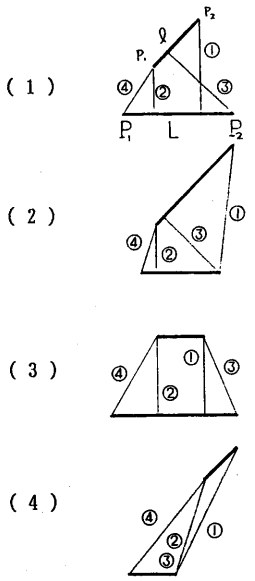


図12　2 辺間の距離

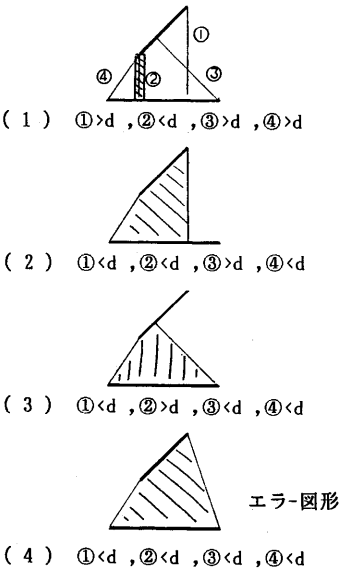


図13　エラー箇所表示図形

表 3　注目辺と距離計算辺の持つ方向

注目辺の持つ方向	Space Incursion Width Notch	Enclosure
→		
↗		
↑		
↖		
←		
↙		
↓		
↘		

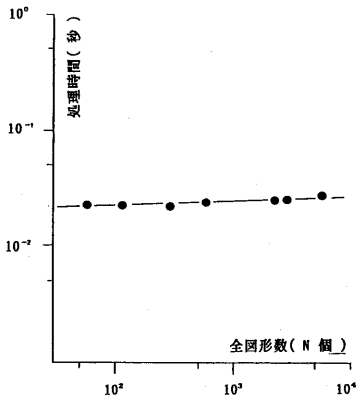


図14　DRC 付きパターン入力時間

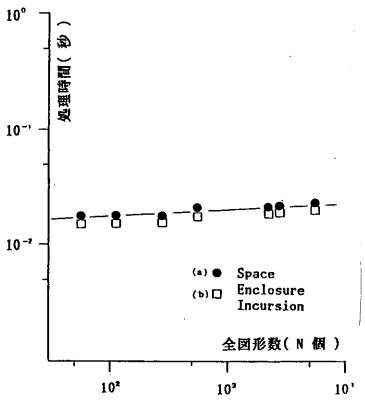


図15　Space,Enclosure,Incursion チェック処理時間