Vol. 30 No. 4 情報処理学会論文誌 Apr. 1989

オンライン設計用タイル型データ構造の基礎検討†

鈴木五郎†† 浜田亘曼††

VLSI のマスク・パターンを編集中，設計規則チェックや自動コンパクションを即座に行うために必要なオンライン設計用データ構造を検討した．1984年に米国カリフォルニア大学がタイルの概念を導入したデータ構造を提案しているが，われわれは VLSI マスク・パターン設計の実情に合うように，(1)斜線を含む多角形と折れ線を扱う，(2)穴を持つ多角形を扱う，(3)大型パターンの入力や大域にわたるパターンの移動などの処理においてデータ構造更新のオーバヘッドを軽減する，などの点に留意してこのデータ構造を拡張した．1MIPS の計算機を用いて実験プログラムを作りデータ構造の性能を評価したところ，パターンの形状情報だけを持つ最も簡単なデータ構造に比べてデータ量が 5〜6 倍，またパターン入力と設計規則チェックあるいは自動コンパクションの組合せが0.1〜0.2秒程度の CPU 時間で処理できることが確認できた．

1. まえがき

VLSI マスク・パターン設計においては，設計されたパターンが設計規則を満足しているかどうかの検証を行う設計規則チェック（以下 DRC：Design Rule Check と呼ぶ）や設計規則を気にしないで入力したパターンを設計規則が満足されるように再配置する自動コンパクションなどの設計ツールが開発されてきた．これらの設計ツールは，パターン設計が一応完了した時点で全パターンを対象としてバッチ処理的に実行されており，パターンを入力する過程でインタラクティブに実行するという形態にはなっていないのが現状である．ところが VLSI の高集積化に伴い，設計工数を低減する目的から DRC や自動コンパクションをインタラクティブに実行しながらパターン設計を進めるオンライン設計のスタイルが必要となってきている．この設計スタイルを可能にする鍵は DRC や自動コンパクションの処理速度が握っていることは言うまでもない．ではどうすれば高速化が図れるのだろうか．われわれの経験でも処理速度を左右する大きな要因はパターン間の隣接関係を求める処理つまり近傍にある図形を求める処理にあることがわかっている[1]．そこで隣接関係が高速に認識できるように，データ構造面からの検討を行った．

オンライン設計用データ構造としては MOS ロジック LSI を対象としたパターン設計システム MAGIC[2)-4)]（米国カリフォルニア大学が提案）の中で採用しているタイル型データ構造[5]がある．編集している図面に含まれる全パターン数をNとするとき，隣接するパターンを認識する計算量が $O(N^2)$ となる画期的なデータ構造であるが，取り扱うパターン形状に制限があることや大型パターンの入力や大域にわたる移動などの処理においてデータ構造更新のオーバヘッドが大きい，などこのままではわれわれのパターン設計の実情に合わない．そこで MAGIC の持つデータ構造を拡張することにより，これらの問題点を解決した．

本論文ではまず MAGIC の持つデータ構造をそのままわれわれの設計に適用できない理由をあげ，次に新しく著者らの提案するデータ構造[6]について述べ，最後に新データ構造のプログラム評価結果を示す．

2. MAGIC の持つデータ構造

2.1 MAGIC のデータ構造

MAGIC ではタイルという概念を導入することによってパターン間の隣接関係をタイル間の隣接関係に置き換え，データ構造中にその情報を持ち込んで隣接パターンの認識を高速化している．この概念を**図1**に示す．パターンの各頂点から自分自身あるいは他のパターンの線分に行き当たるまで水平に線を延ばし，図1(a)のようにパターンの内外をタイルと呼ぶ矩形領域に分割している．図1(b)のようにすべてのタイルは上・下・左・右に直接隣接するタイルをさし示す4本の隣接ポインタ RT (Rightmost Top：上側に直接隣接するタイルの中で最も右側に位置するものをさし示す)・LB・BL・TR を持っている．これらのポインタを使って隣接するパターンを簡単に認識することができる．例えばパターン①の右側あるいは下側に存在するパターン②，③は T_{12}, T_{13} のようにポインタを次々とたどることによって認識できる．

またマスク・パターンには複数の層が存在している

† Tile Data Structure for Online VLSI Design by GORO SUZUKI and NOBUHIRO HAMADA (Hitachi Research Laboratory, Hitachi Ltd.).

†† (株)日立製作所日立研究所

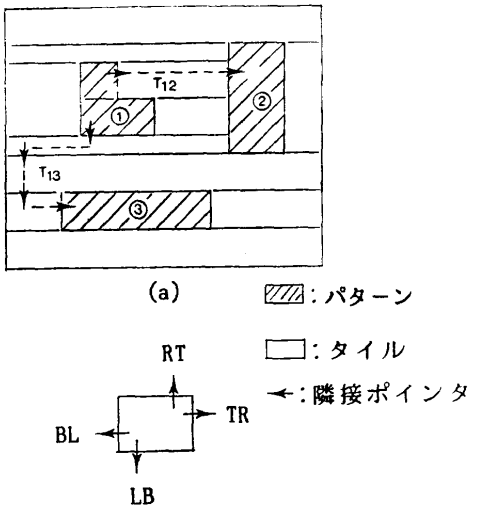


図 1 MAGIC のデータ構造
Fig. 1 MAGIC's data structure.

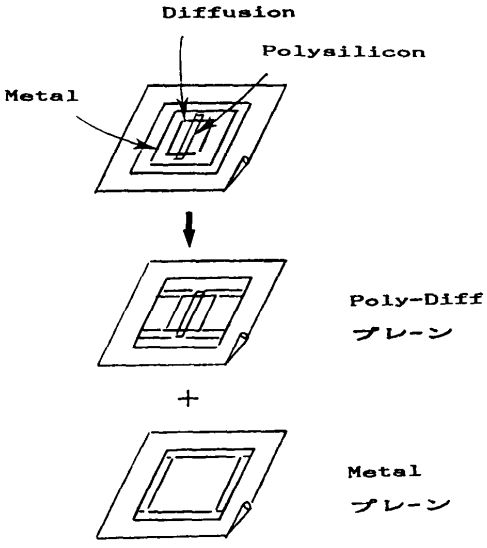


図 2 プレーンの概念
Fig. 2 Planes.

が，Metal および Poly-Diff と呼ぶ2つのプレーンを導入し，図2に示すように配線層のパターンを Metal プレーンに対応付け，拡散層など配線以外の層のパターンを Poly-Diff プレーンに対応付けして，各プレーンごとにタイル分割を行っている．

2.2 パターン形状と操作の相違点

MAGICとわれわれが扱うパターンの形状とパターンの操作に関しては次のような違いがある．

（1） パターン形状の違い

MAGICとわれわれが扱うパターンの比較を表1に示す．MAGICでは穴のない垂直・水平線分のみからなる多角形だけを扱い，トランジスタは拡散層とポリシリコン層の矩形パターンの組合せからなるシンボルとして扱うようになっている．一方われわれはパターンの種類として穴を持つ多角形だけではなく，データ量を減らすために一定幅のワイヤはその中心座標だけを持つ折線として扱っている．さらに，メモリ VLSI の周辺回路などのセルでは集積密度を上げるために斜45度のパターンも用い，拡散層とポリシリコン層のパターンを独立に入力してトランジスタを表現している．図3に典型例を示す．

表 1 扱うパターンの比較
Table 1 Pattern comparison.

	MAGIC が扱うパターン	われわれが扱うパターン
パターンの種類	シンボル 多角形	折 線 多角形(穴あり)
角 度	直 角	45度
トランジスタの入力	シンボル	多角形 + 折 線

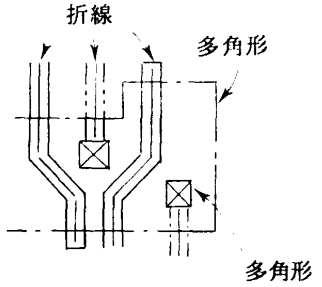


図 3 われわれが扱うパターン
Fig. 3 Our mask pattern.

（2） パターン操作の違い

われわれは，大きな面積を持つ WELL 層パターンを既に配置されている多数の他層パターンの上に配置するとか，既に配置されているものをかなり遠くの位置へ移動するなど大域にわたる変更操作を行う．またMAGICでは入力したパターンはその単位で自動コンパクションの処理を行うのに対して，われわれは入力単位はもちろんのこと辺単位だとかパターンの穴の部分だけを自動コンパクションの対象にする．

3. 新しく提案するデータ構造

3.1 新データ構造

第2.2節で述べたパターン形状と操作の相違点に対

処するために MAGIC のデータ構造を拡張した新しいデータ構造を提案する．

（1） タイルの拡張

斜45度のパターンも取り扱うことから，タイルの形状としては台形を考える．特殊な場合には三角形が必要となるが，台形の上(下)の辺の2つの頂点が1つに縮退したと考えることにする．

（2） フレームの導入

MAGICでは入力されたパターンがすべてタイルに分割されてしまい入力単位の情報が捨てられてしまうことから，辺単位の自動コンパクションを行うなどの処理では該当する辺の情報を作り直さなければならなく，処理上のオーバヘッドが大きくなる．またわれわれは多角形だけでなく，折れ線も扱うことから入力単位の情報が保存できるようにパターンおよび図面枠そのものを表現するフレームなる概念を導入する．**図4**にその一例を示す．パターンを入力するための図面枠，パターンの外枠および穴枠がそれぞれのフレームとなり，図面枠とパターン外枠のように直接包含関係にあるフレーム間にタイルが生成される．

（3） タイルとフレームの持つ隣接ポインタ

タイルとフレームには**図5**に示すような隣接ポインタを持たせる．タイルは MAGIC と同様に4本のポインタ RT・LB・BL・TR を持つ．RT と LB はそれぞれ上・下に直接隣接するタイルまたはフレームの中で最も右側あるいは左側に位置するものをさし示し，BL と TR は左・右に直接隣接するフレームをさし示すことにする．またフレームは大きさのないタイルが複数個連結されているものと考え（例えば(b)では大きさのない2つのタイルが連結されている），連結部分を除いてタイルと同一種類のポインタを持たせる．ポインタ RT・LB・BL・TR は，パターンどうしが接するような特殊な場合を除き，上・下・左・右と直接隣接するタイルをさし示している．例えば TR の場合，右側に隣接するタイルの中で最も上側に位置するものをさしている．図4では折れ線として入力したパターン①の持つフレームポインタから左に隣接するタイルを見つけ，そのタイルのポインタを使ってさらに左に隣接する穴明きパターン②を見つける例を表している．

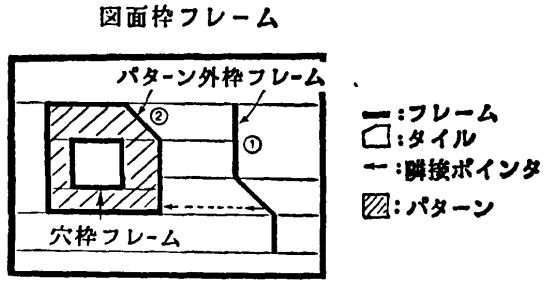


図 4 新しいデータ構造
Fig. 4 New data structure.

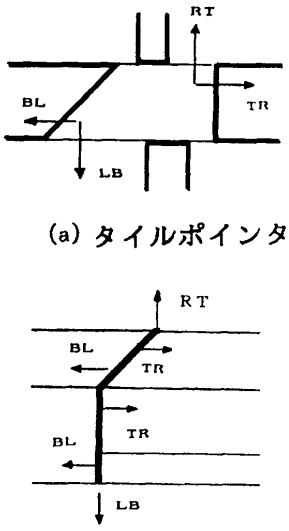


図 5 隣接ポインタ
Fig. 5 Neighbor pointers.

（4） プレーンの概念の拡張

MAGICでは2つのプレーンしか持たないために，大きな面積を持つパターンを入力したり，大域にわたる移動操作を行った場合，更新しなければならないタイルの数が非常に多くなる場合があり，タイル情報更新のオーバヘッドが大きくなる．そこでわれわれは**図6**に示すようにマスク・パターンに1対1対応するプレーンを持つことにする．こうすることによって，タイル情報の更新は入力・変更されたパターンの属するプレーンについてのみ行えばよくなり，パターン入力・変更時のオーバヘッドが軽減できる．ただし，異なる層に属するパターン間の DRC を行う場合には，注目しているパターンを他の層に対応するプレーンに写像（つまりどのタイルに存在することになるかを調べる）してから近傍パターンを見つけることになる．この写像処理は注目しているパターンの頂点ごとに行うが，より高速化を図るため**図7**に示すバケット分割[7),8)]を利用する．図面をあらかじめ同一サイズの矩形状のバケットに分割しておき，各バケットに存在するタイルの1つを記憶しておく．1つの点（図中×で示した指定点）を写像する場合，まずこの点がどのバケット

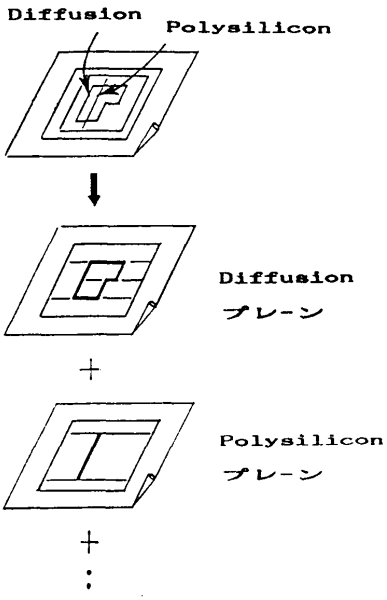


図 6 プレーンの拡張
Fig. 6 Extended planes.

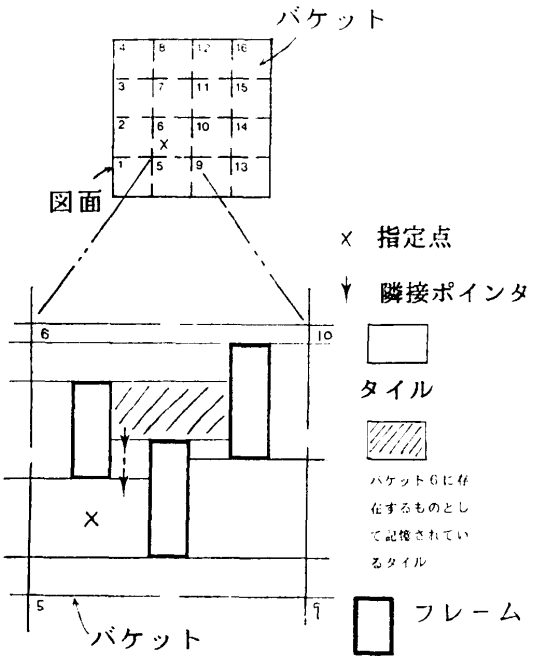


図 7 点の写像要領
Fig. 7 Point mapping.

に存在するかを求める．図の例では6番のバケット内に存在している．次に，該当バケット内に存在するタイルの1つ（▨のタイル）が記憶されていることから，そのタイルから出発して，タイルとフレームの隣接ポインタをたどって指定点が存在するタイルを見つける．特に大規模な図面を扱う場合には，バケット分割の利用が高速処理に効果を発揮することになる．

3.2 パターン入力・削除におけるデータ構造の更新手順

パターンを入力あるいは変更した場合にどのような手順でデータ構造を更新すればよいのだろうか．回転・ミラー反転・平行移動などの変更処理は削除と入力の2つの処理の組合せで対処できることから，ここでは入力と削除だけを取りあげて考え方を説明する．

（1） パターン入力時のデータ構造更新手順

パターンを入力すると，それと重なるタイルが変化し，入力パターンの内部に新しいタイルが発生する．この2つの処理は全く異なった処理が必要になると思われるが，タイル更新用フレームという概念を導入することによってほとんど同一の扱いができる．図8を使ってデータ構造更新の手順を説明する．

（S 1） 第3.1節で説明した写像処理を行って，入力パターンと重なるタイルを抽出する．

（S 2） （S 1）で抽出したタイル群の輪かくをとり，タイル更新用フレームを作る．この際，タイル更新用フレームと接するタイルあるいはフレームのポインタは一時的にこのフレームをさし示しておく．

（S 3） タイル更新用フレームの中にパターンを入力したと考え，このフレームとパターン外枠フレームの間をタイル分割し，さらに入力パターンの内部もタイル分割する．タイル分割の手順は第3.3節で説明する．

（S 4） タイル更新用フレームを取り除く．この際，

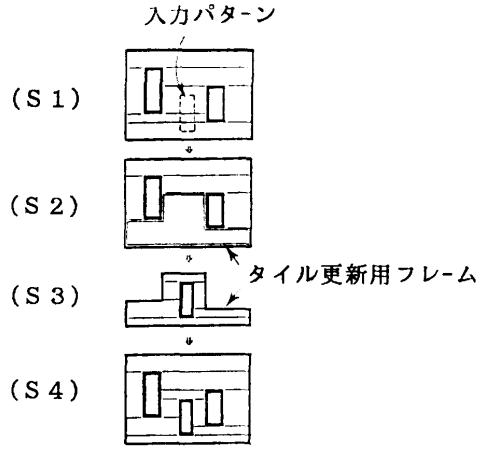


図 8 パターン入力におけるタイル更新手順
Fig. 8 Tile data maintenance process.

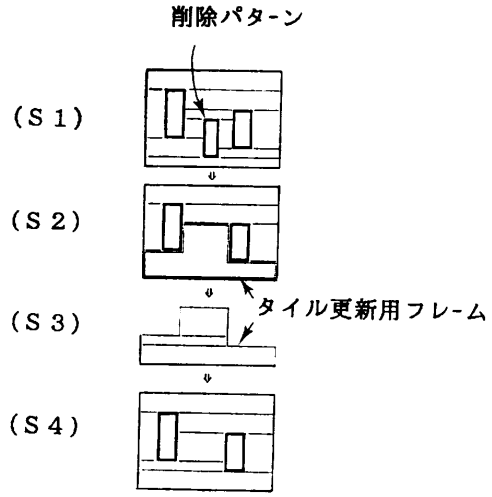


図 9 パターン削除におけるタイル更新手順
Fig. 9 Tile data maintenance process.

このフレームと接していたタイルあるいはフレームのポインタを実際に隣接するタイルあるいはフレームをさし示すように変更する.

(2) パターン削除時のデータ構造更新手順

パターンを削除する時には該当パターンの回りのタイルを**図 9** に示した手順で更新する.

(S 1) 削除パターンに隣接するすべてのタイルをポインタを用いて抽出する.

(S 2) (S 1)で求めたタイル群の輪かくをとり, タイル更新用フレームを作る.

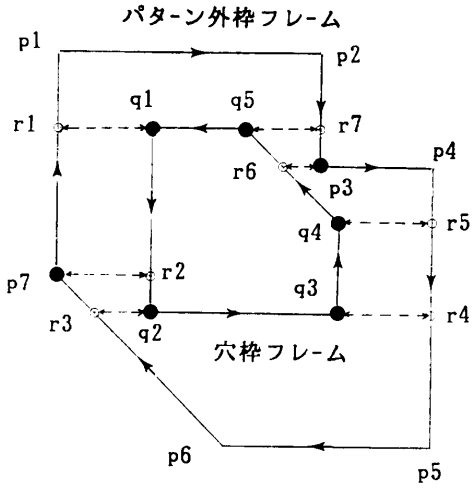
(S 3) タイル更新用フレームの内部をタイルに分割する. 分割の手順は第 3.3 節で説明する.

(S 4) タイル更新用フレームを取り除く.

3.3 タイルの生成手順

第 3.1 節および 3.2 節で述べたようにすべてのタイルはタイル更新用フレームを含めて直接包含関係にあるフレーム間, あるいは包含するフレームが存在しない場合にはフレームの内部に生成される. ここでは**図 10** のように包含関係にある 2 つのフレーム, 特にパターン外枠フレームと穴枠フレームを取りあげ, タイル生成手順を説明する. 2 つのフレームがタイル更新用フレームとパターン外枠フレームである場合も基本的には同一手順で処理することができる. また包含されるフレームが複数個存在する場合, あるいは穴無し多角形の内部のように包含するフレームが存在しない場合も統一的に扱うことができる.

以下で述べる生成手順では, フレームの各辺はパターンを右に見るように方向付けをした情報があらか



↔ 関連ポインタ ● 注目点

図 10 タイルの発生手順
Fig. 10 Tile generating process.

表 2 注目点と検索方向
Table 2 Noticed point and searching direction.

	注目点の条件 (////: パターン)	検索方向
A	or or	左側
B	or or	〃
C	or	〃
D	or	〃
E	or or	右側
F	or or	〃
G	or	〃
H	or	〃
I	or	左右

じめ与えられているものとする．

（S 1） 各フレームの頂点に注目し，その前後の辺の方向が**表2**（A)～(I)(内角を 90° 以上 270° 以下としている）のどれかを満足する場合にはその頂点を注目点とする．例えば頂点p3は(B)を満足していることから注目点とする．

（S 2） (S 1)で求めた1つの注目点から表2に記述されている検索方向（つまりパターンの存在する方向）を調べ，最初に出合った辺との交点を求める．例えば注目点p3では左側に注目し，最初に出合った辺との交点r6を求める．ここで，注目点とこの交点には互いの位置を示す関連ポインタを持たせておく．

（S 3） (S 2)をすべての注目点についてくり返す．

（S 4） 下向きの方向を持つ辺の1つに注目する．

（S41） 始点（最大のy座標を持つ点）から出発して辺の方向と関連ポインタを使って交点および頂点を次々と検索し，始点に戻ったところで1つのタイルを生成する．例えば，p2から出発した場合，パターン外枠フレームを下向きに進み，交点r7に出合ったところで関連ポインタを使ってq5へ移る．今度は穴枠フレームをq1まで進み，関連ポインタを使ってr1へ移る．パターン外枠フレームを上へ進みp1，さらにそこから右へ進んでp2へ戻る．以上の検索で，タイルp2，r7，q5，q1，r1，p1つまりタイルp2，r7，r1，p1が生成される．

（S42） 注目している辺上のすべての交点から出発し，(S41)と同様の処理を行う．

（S 5） (S 4)を下向きの方向を持つすべての辺についてくり返す．

以上述べたタイル生成処理では，フレームの各頂点を調べて注目点であるかどうかを判定し，注目点である場合には進行方向に向って右側に一番近い辺との交点を見つける処理が全処理時間のほとんどを占めていることから，フレーム頂点数の和をkとした場合，$O(k)$に比例した処理時間となる．

4. プログラム性能評価

今回提案したデータ構造をプログラム化し，その性能評価を行った．プログラム言語はFORTRAN 77，使用計算機はHITAC E 7300（約1MIPSのEWS）であり，評価に使用した図面は500トランジスタと40トランジスタからなる2種類のVLSIセル図面である．

4.1 データ量の評価

各図面に関して，拡散層とコンタクト層は多角形で入力し，ポリシリコン層とアルミ層は折れ線で入力した場合，**表3**に示すように500トランジスタと40トランジスタ（全パターン数はトランジスタ数の5～6倍）からなるセル図のデータ量はそれぞれ1.4Mバイトと0.11Mバイトである．両方ともパターンの形状情報だけを持つ最も簡単なデータ構造に比べて5～6倍のデータ量となっている．このことは理論的にも裏付けられる．平均8頂点を持つN個のパターンから構成される図面を考え，新しく提案するデータ構造と最も簡単なデータ構造のデータ量をD_NおよびD_*とすると，それぞれ次式で表される．

D_{Nt}：タイルのデータ量

D_{Nf}：フレームのデータ量

$D_N = D_{Nt} + D_{Nf}$

x，y座標およびポインタはそれぞれ4B（バイト）で表現する．

D_{Nt}＝全タイル数×{座標情報＋ポインタ情報}

$\leq 2\times 8\times N\times\{4\times 8\mathrm{B}+4\times 4\mathrm{B}\}$ （1）

ここで，全タイル数≦2×全頂点数となることの証明は付録参照．

D_{Nf}＝パターン数×{座標情報＋ポインタ情報}

表3 プログラム性能評価
Table 3 Program performance.

項目		図面	規模 550 Tr	規模 40 Tr
図形データ量(Mバイト)			1.4	0.11
処理時間*	エディタ	データ構造作成 (sec)	150	12
		1図形入力 (sec)	0.1	
		50図形コピー (sec)	5	
	逐次検証	1図形入力+DRC（1層） (sec)	0.12	
		1図形入力+コンパクション（3層） (sec)	0.15	
	全体検証	DRC（1層） (sec)	40	3.2
		コンパクション（3層） (sec)	240	19.2

* 1 MIPS 換算 CPU 時間

$=N\times\{8\times 8B+8\times 2\times 4B\}$ (2)

D_o=パターン数×{座標情報}

$=N\times\{8\times 8B\}$

よって，$D_N/D_o\leq 14$ となるが，実際には同一 y 座標となる頂点を持つパターンが多数存在することから，5～6倍となっている．

4.2 処理速度の評価

(1) エディタとしての処理速度

パターンの形状情報だけを持つ最も簡単なデータ構造から今回説明したタイル型データ構造を作成するのに必要となるCPU処理時間は，それぞれの図面で150秒と12秒である．両者ともこのうちの8割程度がタイル生成に費やされている．また各図面中に1個のパターンを追加入力した時，および既配置パターンの中から50パターンを任意に選択して他の場所へコピー配置した時の処理時間は，それぞれ0.1秒および5秒である．これらの処理では新しく配置した近傍のデータ構造だけを更新すればよく，図面に含まれるトランジスタ数には依存しない処理時間となっている．

(2) 逐次検証処理時間

2種類の図面の中に1個のパターンを入力すると同時にDRCを行い，設計規則違反個所があればエラー表示をするCPU処理時間はどちらも0.12秒である．また1個のパターンを入力する際に既配置パターンとの位置関係を認識し，設計規則がぎりぎり満足される場所を捜して（入力した場所から左・右・上・下どちらの方向に最終場所を決めるかはパラメータで指定）そこに配置するコンパクションのCPU処理時間はどちらも0.15秒である．

以上の処理時間も前述したのと同じ理由で取り扱う図面の規模には依存していない．

(3) 全体検証処理時間

2種類の図面全体に関してDRCを行う場合のCPU処理時間は，それぞれ40秒と3.2秒である．ここでは，指定したパターンについてのDRC処理を n 個分くり返す方式をとっている．また，タイル構造を用いてパターン間の隣接関係を求め，critical path 法[9]を使って各パターンの最終座標を求めて，最後にタイル情報を更新するコンパクションの処理時間はそれぞれ240秒と19.2秒である．ただし，x 方向へ1回だけコンパクションを行っている．

4.3 MAGIC が持つデータ構造との性能比較

MAGICの開発者は論文[5]の中で，斜めパターンを扱うデータ構造にすると2倍から10倍遅くなると予想しているが，文献2)に記載されているDRC処理速度，1,900タイル/秒に対してわれわれのデータ構造は1,200タイル/秒(共に1 IMIPS計算機のCPU時間)であり，1.6倍しか遅くなっていない．またMAGICが必要とするデータ量は，最も簡単なデータ構造の3倍と言われているが，われわれのデータ構造では5～6倍のデータ量であり，MAGICと比べて2～3倍のデータ量となっている．

5. むすび

オンライン設計用タイル型データ構造の検討を行った．米国カリフォルニア大学が提案したMAGICのデータ構造をマスク・パターン設計の実情に合うように，(1)パターンの種類として多角形だけでなく折れ線も扱える，(2)斜線も扱える，(3)パターンの入力・変更時におけるデータ構造更新のオーバヘッドが少ない，などの特徴を持たせて拡張した．実験プログラムを作成し，1 MIPS計算機を用いて性能評価を行ったところ，パターンの形状情報だけを持つ最も簡単なデータ構造と比較してデータ量が5～6倍，またパターン入力とDRCあるいは自動コンパクションの組合せが0.1～0.2秒のCPU時間で処理できることが確認できた．

謝辞 本研究を進めるに当たり，ご指導・ご助言をいただいた当研究所谷中主管研究員はじめご討論いただいた関係諸氏に深く感謝いたします．

参考文献

1) 鈴木五郎：インターラクティブDRCの一手法，信学技報，EC85-26，pp. 9-17 (1985).
2) Ousterhout, J. K. et al.: Magic: A VLSI Layout System, *Proc. of the 21st DAC*, pp. 152-159 (June 1984).
3) Taylor, G. S. et al.: Magic's Incremental Design Rule Checker, *Proc. of the 21st DAC*, pp. 160-165 (June 1984).
4) Scott, W. S. et al.: Plowing: Interactive Stretching and Compaction in Magic, *Proc. of the 21st* DAC, pp. 166-172 (June 1984).
5) Ousterhout, J. K.: Corner Stitching: A Data Structure for VLSI Layout Tools, *IEEE Trans. CAD*, Vol. CAD-3, No. 1, pp. 87-100 (Jan. 1984).
6) 鈴木五郎：オンライン設計用タイル型データ構造の検討，昭和62年信学部門全大，pp. 1-102 (1987. 11).
7) Wilmore, J. A.: The Design of an Efficient Data Base to Support an Interactive LSI

Layout System, *Proc. of 16 th DAC*, pp. 445-451 (June 1979).
8) Asano, T. et al.: Computational Geometry Algorithms, in *Layous Design and Verification*, ed. by T. Ohtsuki, pp. 334-335, North Holland (1985).
9) Hsueh, M. Y.: Computer-Aided Layout of LSI Circuit Building Blocks, *Proc. of ISCAS*, pp. 474-477 (June 1979).

付　　録

生成されるタイルの総数はパターン頂点数の総和のほぼ2倍になることを証明する．

(仮定)

① 各パターンの上・下の辺は水平である．
② 各パターンの左・右には同一数の頂点がある．
③ 各パターンは上・下に凹な部分を持たない．

$V(i)$：パターン i の持つ頂点数
N：全パターン数
$t_v(i)$：パターン i を入力することによって新たに生成されるパターン外のタイル数
$t_s(i)$：パターン i を入力することによって新たに生成されるパターン内のタイル数
T_v：パターン外に生成されるタイルの総数
T_s：パターン内に生成されるタイルの総数

(証明)

パターン外に生成されるタイルはパターンの入力順序には依存しないので，左から1個ずつ順に入力したとする．**図11**に示すように新しいパターン①を入力する前・後に注目すると，新しく入力したパターンの左・右にはそれぞれ $\{V(i)-4\}/2+1$ 個のタイルが，また上・下にはそれぞれ1個ずつのタイルが生成される．ただし，上・下に生成されるタイルの内1個は既に存在するタイルが変形したものであるから，パターン①を入力した時に新たに生成されるタイル数は

$$t_v(i) \leq 2\{\{V(i)-4\}/2+1\}+1=V(i)-1 \qquad (3)$$

で表現される．左側に存在するパターンに新たに入力したパターンと同一 y 座標の頂点があるとタイルの縮退が起こることから不等号となる．パターンが1個も存在しない状態でも1個のタイルが存在することを考慮すると，パターン外に生成されるタイルの総数は

$$T_v \leq \sum_{i=1}^{N} \{V(i)-1\}+1 \qquad (4)$$

となる．

一方新しく入力したパターン①の内部には $V(i)-3$ 個のタイルが生成されることからパターン内に生成されるタイルの総数は

$$T_s \leq \sum_{i=1}^{N} \{V(i)-3\} \qquad (5)$$

となる．入力パターンに同一 y 座標を持つ頂点が存在するとタイルの縮退が起こることから不等号となる．(2)，(3)より生成されるタイルの総数は

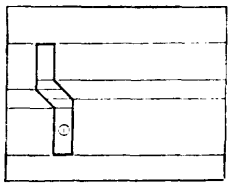
$$T_v+T_s \leq \sum_{i=1}^{N} \{2V(i)-4\}+1 \qquad (6)$$

となる．ここで $V(i) \gg 2$ とすると

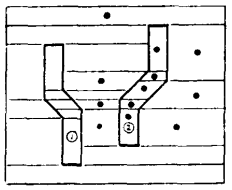
$$T_v+T_s \leq 2\sum_{i=1}^{N} V(i) \qquad (7)$$

となり，生成されるタイルの総数はパターン頂点数の総和のほぼ2倍となることが証明された．

(証明終)



(a) パターン② 入力前



新たに発生したタイル

(b) パターン② 入力後

図 11 新しく発生するタイル
Fig. 11 New generated tiles.

(昭和63年1月 9 日受付)
(平成 元 年2月 14 日採録)

鈴木　五郎

昭和50年慶応義塾大学理工学部電気工学科卒業，同年(株)日立製作所入社．以来 LSI・CAD システムの研究開発に従事，計算幾何学に興味を持ち，レイアウト・コンパクション，設計規則検証，接続検証，論理／回路図エディタ，マスク・パターン・エディタを開発，現在同社日立研究所第3部研究員．IEEE，電子情報通信学会各会員．

浜田　亘曼（正会員）

昭和41年，大阪大学工学部電気工学科卒業．同43年，同学科修士修了．同(株)日立製作所日立研究所入社．以来アナログ制御系，計算制御システム，高水準言語，分散型計装システム等の開発に従事．その間，昭和52年には米国カーネギーメロン大学，計算科学部にて客員研究生として分散型OSの研究に従事．現在，知識処理，EWSを応用したVLSI設計支援システムの研究開発に従事．工学博士．