マイクロコンピュータによる LSI パターン図形演算法†

田丸啓吉†† 山田泰生††

LSI 技術が進歩し，チップ上に集積される回路が複雑になるに伴い，レイアウト設計の誤りを検出することが重要になる．従来，このようなデザインルールチェックは，大型計算機のソフトウェアで実現されてきたが，処理の高速化と低コスト化のために，複数台のマイクロコンピュータを使用した専用計算機が有効と考えられる．本論文は，このような専用計算機の中の1台のマイクロコンピュータで実行する，LSI パターンの図形演算法について考察している．使用している方法は，スリット法とワークリスト法の考えにもとづき，入力多角形からスリット幅で切断したスライスベクトルをつくり，図形重なりや輪郭を求める図形演算を行い，結果のスライスベクトルから出力多角形を復元する．これらの処理は，すべて現在のスリットに関連する図形部分のみについて行い，スリットを移動しながら，順次，処理をすすめていく．とくに，スライスベクトルから出力図形を順番に復元して成長させる方法は，新しく考案した手法で，簡単なポインタ操作で実現している．このためメモリ量も少なく，マイクロコンピュータで十分処理できる．簡単な図形について本方法の正しさを確認する実験を行い，実際の場合には16ビットマイクロプロセッサに3～4Mバイトのメモリをつければよいことを示した．

1. まえがき

LSI 技術が進歩し，チップ上に集積される回路の規模が大きくなるにつれて，レイアウトパターン（マスクパターン）の複雑さが急激に増大する．これに伴い，レイアウト設計における設計誤りの含まれる確率も大きくなる．設計誤りによる試作のやり直しにかかる損失は膨大なものになるため，試作に先立つ検査がますます重要になっている．レイアウトパターン設計を検査するデザインルールチェック（以下 DRC と略記する）は，従来から大型計算機の利用がすすめられてきた[1)~4)]が，回路の大規模化に伴い，その処理時間の増加が問題になってきた．そこで DRC を専用計算機で処理し高速化と低コスト化を実現できれば非常に有効である．このDRC 専用計算機の一つの実現方法として，複数台のマイクロコンピュータの並列処理方式を使用し，各マイクロコンピュータは分担した領域内の DRC を行う方法が考えられる．本論文では，この DRC 専用計算機の1台のマイクロコンピュータで使用するパターン図形処理の手法について考察する．

LSI のパターン図形を扱う DRC では，図形演算は必須のものであり，ビットマップ法[5)]や領域区分法[6)]，コーナーベース法[3)]などの手法が発表されている．とくに処理速度の点で有利であると考えられているのが，1次元の領域分割（スリット）を使うスリット法[1)]である．この方法では，y 軸に平行なスリット内の水平または斜ベクトルの方向によって，図形演算をするもので，ワークリスト法[7)]を使用して，演算をしながらスリットを移動させる手法が使用されている[8)]．この方法による大型計算機のプログラムが実用になっている[4)]．本論文で述べる図形演算は，原理的には同じスリット法とワークリスト法の考えを使用し，入力図形のベクトル化から出力図形の復元まで，すべて1台のマイクロコンピュータで実行するように考えられている．すなわち，図形演算は大別して，スリットにかかる図形からスリット幅のスライスベクトルをつくり，次に図形演算を行って演算結果の出力スライスベクトルを求め，この出力スライスベクトルから出力図形を復元する3段階からなっている．これらの処理は，すべて現在のスリットに関連する図形部分のみについて行い，出力図形の復元もスライスベクトルを順次接続しながら図形を成長させる手法を使用している．この処理については，従来明らかにされていなかった点であるが，本論文で述べる処理手順により，比較的簡単なポインタ操作で図形を復元することができることを示している．以下，2章ではベクトルスライス法による図形演算について述べ，3章ではスライスベクトルの接続による図形復元の方法について述べる．4章では簡単な実験による本手法の確認と使用するマイクロコンピュータについての考察について述べる．

2. ベクトルスライス法による図形演算

LSI のマスクパターンという多量の図形データを対象として図形演算を行う場合，演算対象となる図形を速く捜し出す方法と，それらの間の重なり関係を効率

† LSI Layout Pattern Operation Method Using a Microcomputer by KEIKICHI TAMARU and YASUO YAMADA (Electronic Division, Faculty of Engineering, Kyoto University).

†† 京都大学工学部電子工学科

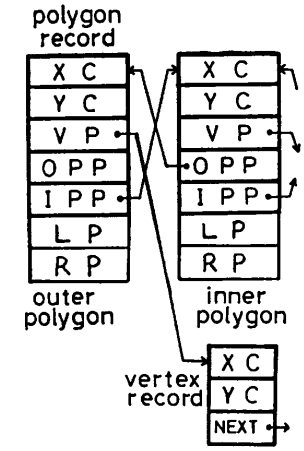


図 1 多角形の表現形式
Fig. 1 Expression form of polygons.

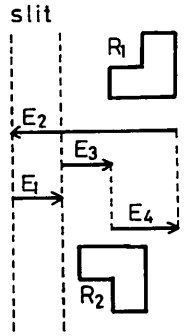
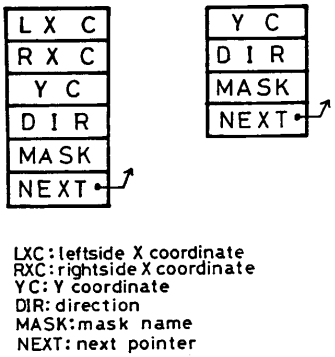


図 2 多角形のベクトル変換
Fig. 2 Transformation of polygons into vectors.



(a) Vector record. (b) Sliced vector record.

図 3 ベクトルレコードとスライスベクトルレコード
Fig. 3 Vector record and sliced vector record.

よく調べる方法が重要な点である．ベクトルスライス法では，対象図形の選択にはスリットを，重なり認識にはベクトルを使用したものである．

マスクパターンデータは，他の CAD システムとの融通性を考えて多角形表現されている．多角形は**図 1** に示すように，鍵となる x 座標と y 座標，頂点リストを指す頂点ポインタ，包含関係を示す外図形ポインタと内図形ポインタおよび左右の技を指すポインタからなる多角形レコードで表されている．多角形は，左端最下点の座標を用いて，(x 座標)×(y 方向最大座標)+(y 座標) の値を鍵とし，2 分木構造をもつ多角形リストにより管理する．次に多角形を x 軸に平行な水平ベクトルによる表現に変換する．ベクトルの向きは，多角形の内部を左手に見る方向を正と定める．多角形が斜線を含む場合は，斜ベクトルが発生するが，ここでは斜ベクトルは考えないことにする．ベクトルはベクトルリストに登録する．

次にスリットを定める．スリットは y 軸に平行な 2 本の境界線ではさまれた領域で，内部にはベクトルの端点や交点が存在しない領域である．スリットの境界はベクトルの端点や交点になり，この境界でベクトルを切断して，スリットごとに分割されたスライスベクトルにする．一般にスライスベクトルの数は非常に多く，その格納には大きなメモリエリアを必要とするため，一度にまとめて全図形のベクトルをスライスするのは望ましくない．そこでスリットの移動につれて，一部分でもスリットにかかっている多角形のみをベクトルに変換し，スライスベクトルにする方法を使用する．**図 2** で多角形 R_1, R_2 は破線で示してあるスリットに含まれないので，まだ変換されずベクトルリスト中には存在しない．スリットの移動に伴って，多角形は順次，ベクトルに変換されていくのであるが，一方，演算の終わったベクトルはリストより除去されるので，ベクトルリストが占有するメモリエリアは，多角形の占めるメモリエリアに比べて小さくてすむ．

ベクトルリストに登録されている各ベクトルは，**図 3** (a) に示す左端 x 座標，右端 x 座標，y 座標 (左端)，方向，ベクトルが属するマスク層名，ネクストポインタをもったレコードにより表現されており，これらのレコードは左端 x 座標 (同じなら y 座標) でソーティングされた線形リスト構造により管理されている．ベクトルをアクセスする場合には，線形リストよりも木構造のほうが有利であるが，スリットが移動するのに伴って左側 (x の小さいほう) からベクトルが削除され，右側から追加されるため，線形リストに近い木しか処理されない．そこでベクトルリストは，

レコードをリンクするポインタが1個ですむ線形リスト構造にする．

スリット幅は，そのスリット内でベクトルの状態が変化しない範囲で広く取ることが必要である．スリットの幅は次のようにして決定する．ベクトルリストのベクトルを，左端が現在のスリットの左側に一致するもの（図2で E_1, E_2, これを第1グループとする）と，スリット左端よりも右側にあるもの（同図で E_3, E_4, 第2グループとする）に分ける．第1グループは今回スライスされるベクトルであり，第2グループはまだスライスする必要のないベクトルである．スリットの右側は

1) 第1グループのベクトル右端 x 座標の最小値
2) 第2グループのベクトルの左端 x 座標の最小値
3) まだ変換されていない多角形の左端 x 座標の最小値

の三つのうち，最も小さい値からきめる．図2では 1) の E_1 の右端と 2) の E_3 の左端が一致して，スリット幅をきめている．このようにすれば，スリットの条件を満たしながら，スリット幅を最も広く取ることができる．

次に第1グループのベクトルから，スリット幅の部分を切り取り，スライスベクトルリストに登録する．残りの部分があれば，第2グループのリストに加える．第1グループのベクトルは，左端 x 座標は同じ値で y 座標の昇順になっているので，第2グループへの挿入はベクトル数に比例する手数で完了する．スライスベクトルは左端も右端も x 座標については共通の値であるから省略し，図3(b)に示すように，y 座標，方向，マスク層名，ネクストポインタをもつレコードで表される．スライスベクトルリストは，線形リスト構造を使用している．

スリット内でベクトルの向きを下から順番に走査すると，右向きベクトルならば多角形の内側に移り，左向きベクトルならば多角形の外側へ移ることになるから，二つの層の図形の重なり（AND）や輪郭取り（OR）などが求められる．求める状態になったとき，右向きスライスベクトルを出力し，求める状態でなくなったとき左向きスライスベクトルを出力すれば，求める図形に対応する出力スライスベクトルが得られる．得られた各スリットのスライスベクトルを接続して多角形を復元すれば，図形演算ができたことになる．

3. スライスベクトルの接続による多角形復元

スライスベクトルから多角形を復元するために，隣り合ったスリットのスライスベクトルを接続する．このとき，中ぬき図形や凹部分を正しく接続することが必要である．この場合にも，全部の出力スライスベクトルを作成してから接続するのは，メモリ量の点から望ましくない．また，このようにすると，中ぬき部分がどの多角形の内側にあるかを調べるのに手間がかかる．そこで，スリットの移動に伴って出力されるスライスベクトルを，復元中の多角形に順次接続して，多角形を成長させながら復元処理を進める方法を考察し，この方法を成長法と呼ぶことにする．成長法によれば作業領域が限定されるので，復元処理に必要なメモリエリアが小さくてすむ．

多角形の復元中には，中ぬき部分や凹部分を区別することは不可能であるが，これらの部分がどの多角形の内側に存在するかという，内側，外側多角形の包含関係を把握しておくことが必要である．そこで凹部分も中ぬき部分と同様に，一つの多角形として表現しておき，外側多角形との間で包含関係を示す双方向のリスト構造を構成する．双方向にするのは，内，外いずれの多角形からでも，他を参照できる必要があるからである．一つの外側多角形が二つ以上の内側多角形（中ぬき部分や凹部分を含む）をもつことがあるが，この場合は**図4**に示すように，内側多角形の先にリストを延ばすことにより表現する．一つの外側多角形が，複数の内側多角形を直接にポイントするようにはしない．

復元中の多角形は，現在のスリットの左側に接しているので，その接点を切口と呼ぶことにする．切口の座標は，x 座標がスリット左側の x 座標と同じであり，y 座標は前のスリットにあったスライスベクトル

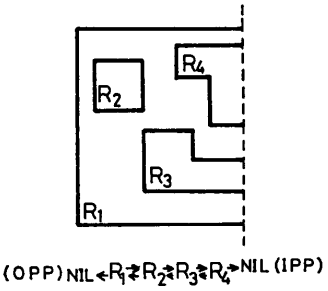


図4 多角形の包含関係の表現
Fig. 4 Expression of the relation between inner and outer polygons.

の y 座標に対応している．多角形は二つの切口をもっている．切口は切口レコードをもち，y 座標によりソーティングされた双方向リスト構造の切口リストをつくっている．双方向にするのは，追加や削除が頻繁に発生するから，それらをすばやく処理するためである．切口レコードは**図5**に示すように，リンク用ポインタの他に，切口がどの多角形のものであるかを示す，多角形ポインタをもっている．同じ多角形の2個の切口は，同一の多角形レコードを指している．スリットの移動に伴い，一方の切口が他の多角形と接続されることがある．このとき，他方の切口レコード中の多角形ポインタを修正しなければならないので，二つの切口が互いに他を指す切口相互ポインタを設ける．さらに，切口の頂点を指す頂点ポインタをもつ．これは復元中の多角形の頂点が連記されたリストへのポインタでもあり，新たな頂点が発生するたびにリストに加えていけば，復元終了時には頂点を連記した多角形表現が得られる．接続中の切口におけるポインタの状態を**図6**に示す．

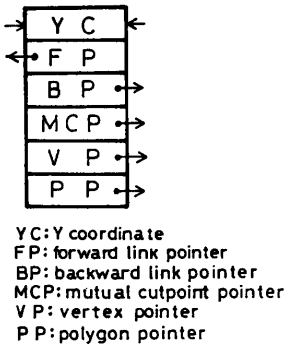


図5　切口レコード
Fig. 5　Cutpoint record.

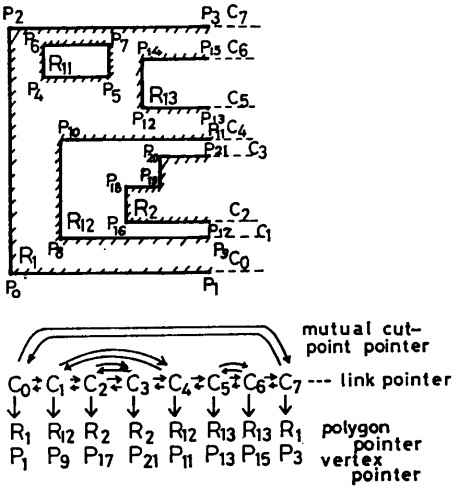


図6　切口におけるポインタの状態
Fig. 6　Pointers in the cutpoints.

切口にスライスベクトルを接続する方法は，切口とスライスベクトルを併合して y 座標でソートし，下側，すなわちリストの先頭から二つずつ取り出して接続する．実際には，二つのリストをマージソートしながら，二つずつ取り出して接続するペアをつくり出せばよく，併合されたリストは存在しない．このとき，多角形の内か，外かを憶えておく必要がある．二つずつ取り出して接続するとき，次のような場合がある．

（1）　切口とスライスベクトルが同じ y 座標の場合

これは最も単純な場合である．**図7**(a)のように，切口側を伸長すればよい．すなわち，切口の頂点 (P_1) の x 座標を書き替える ($P_1 \rightarrow P_1'$) ことである．このとき，多角形の内側と外側では向きが逆になる．

（2）　切口とスライスベクトルが異なる y 座標の場合

図7(b)のような場合である．段差が生じるので，新たな頂点 (P_1, P_2) を生成し，前の切口 (P_0) と接続する．この場合，図形の内部を左に見るように，多角形の外側ならば $P_0 \rightarrow P_1 \rightarrow P_2$ の順に，内側ならば $P_2 \rightarrow P_1 \rightarrow P_0$ の順に接続する．

（3）　切口と切口（内側）の場合

図8の C_2-C_3 や C_4-C_5 の場合である．C_2-C_3 のように，同じ多角形に属する切口の場合には，内側多角形が閉じることになる．これに対して C_4-C_5 のように，異なる多角形に属する切口の場合には，外側多角形が接続されることになる．このとき接続された多角形の他方の切口 (C_1 および C_{12}) が，接続後の多角形を指すように，切口レコード中の多角形ポインタを変更し，頂点ポインタを参照して頂点リストを一つにまとめる．さらに，内側多角形を引き連れている場合に

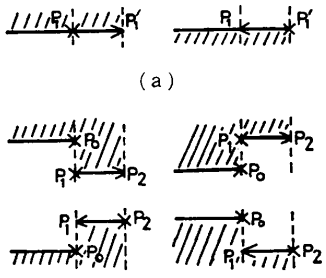
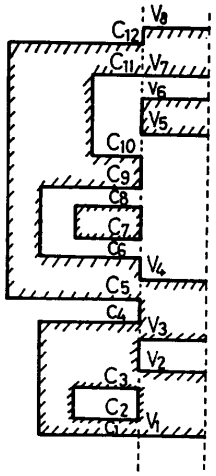


図7　切口とスライスベクトルの接続
Fig. 7　Connection between cutpoints and sliced vectors.



C_1-V_1-C_2-C_3-V_2-V_3-C_4-C_5-V_4-C_6-
-C_7-C_8-C_9-C_{10}-V_5-V_6-C_{11}-V_7-C_{12}-V_8

図 8 切口とスライスベクトルの接続による図形の復元
Fig. 8 Regeneration of polygons by the connection of sliced vectors to cutpoints.

は，それらもまとめなければならない．

（4） 切口と切口（外側）の場合

図8の C_7-C_8 や C_9-C_{10} の場合である．C_7-C_8 のように，同じ多角形に属する切口の場合には，外側多角形が閉じることになる．この場合，その外側多角形に属する内側多角形があっても，これはすでに閉じているから，外側多角形が閉じた時点で，これを出力多角形ファイルへ移す．C_9-C_{10} のように，異なる多角形に属する切口の場合は，内側多角形の接続である．この場合，（3）の外側多角形の接続と同様に，他方の切口（C_6 および C_{11}）の多角形ポインタを変更し，頂点リストをまとめるが，さらに内側多角形を引き連れている場合でも，それは変更せず，接続される多角形を処理するだけである．

（5） スライスベクトルとスライスベクトル（内側）の場合

図8の V_2-V_3 の場合であり，新たに内側多角形が発生したことになる．これに対応する多角形レコードを作成するが，この多角形を包含している外側多角形をさがして，親子（外・内）関係を結ばねばならない．この方法は，直前に現れた切口（図では C_3）の属する多角形の子（内側多角形）としてリストに加えればよい．この場合，図の C_3 のようにその多角形が内側多角形であっても，同じ親（外側多角形）に属することになるからである．

（6） スライスベクトルとスライスベクトル（外側）の場合

図8の V_5-V_6 の場合であり，新たに外側多角形が発生したことになる．この多角形に対応する多角形レコードを作成する．

以上に述べた各処理を，場合に応じてきちんと実行すれば，現在のスリットのスライスベクトルは切口のほうへ吸収され，多角形が成長する．複雑なように見えるが，処理内容はポインタの変更が中心であり，大きな処理時間は必要としない．

4. 考　　察

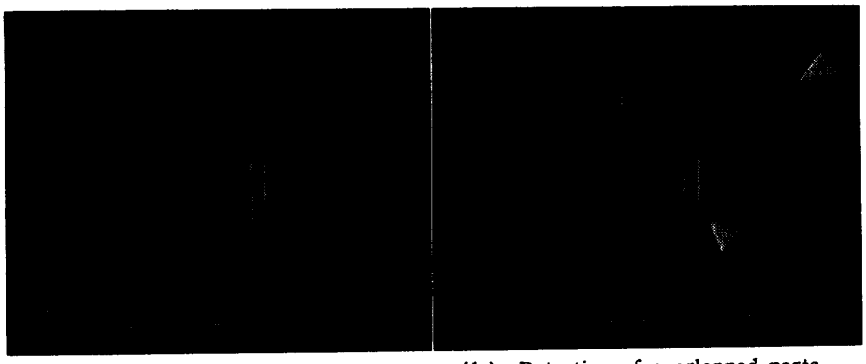
4.1 実 験 例

これまでに述べた方法を確認するため，簡単な実験をした．マイクロコンピュータには，64kバイトのメモリをもつZ80を使用し，プログラムは 1) 拡大・縮小，2) 単一層検査，3) 単一層輪郭抽出，4) 2層間の重なり部分抽出，5) 1層間の輪郭抽出，6) 2層間の差抽出（SUB）の6種の図形演算機能をもっている．ここで単一層検査は，内部を右に見る部分と重なりを誤りとして検出するものである．プログラミング言語はPASCAL を使用した．

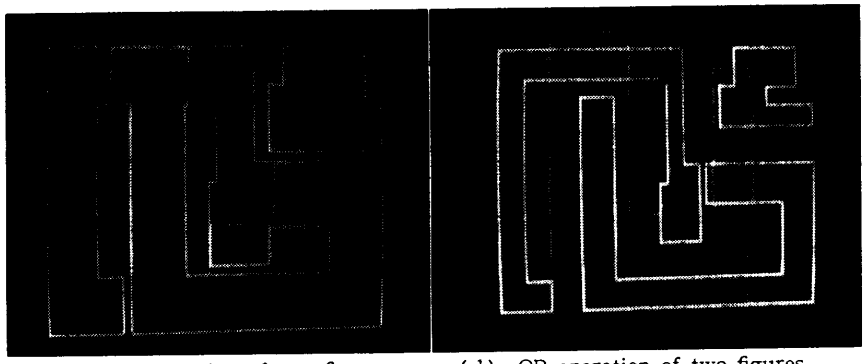
このシステムによる簡単な図形演算の例の写真を**図9** に示す．図(a)は内部の塗られた図形を拡大して白線の図をつくり，重なり部分が図(b)の3個所（矢印の所）に検出表示された例である．実際のディスプレイ上ではカラー表示される．図(c)は2個の図形の重なり（AND）部分，図(d)は2個の図形の輪郭（OR）部分が塗られて表示されている例である．凹凸や中ぬき部分のある図形でも，正確に多角形が復元されている．

4.2 マイクロコンピュータの検討

本システムに使用するマイクロコンピュータについて考えてみる．1台のマイクロコンピュータは，全体の中の担当する領域の図形演算を行う．領域はスリットの方向に沿って y 軸方向には一定長とし，x 軸方向に幅をもつ矩形にとる．領域の大きさは，本来は各プロセッサの仕事量を平均化するために，スリット数が等しくなるように選定するのが望ましい．しかし現実にはスリット幅が前もってわからないので，スリット数を一定にすることはできず，結局一定幅にとることになる．領域幅はデータ量と境界をまたぐパターンの量で定まる．データ量についてみると，1個の矩形に対して多角形レコードは48バイト，2種類のベクト



(a) Expansion. (b) Detection of overlapped parts (indicated by arrows).



(c) AND operation of two figures. (d) OR operation of two figures.

図 9 簡単な図形演算例
Fig. 9 Operation examples of simple patterns.

ルレコードは合わせて20バイト，切口レコードが22バイト必要である．出力側でもスライスベクトルレコードと多角形レコードがある．いま，簡単のため入力多角形はすべて矩形であるとし，入力矩形が一層当り24k個あり，スリットに平均4k個かかるとする．また，同様に24k個の矩形に相当する出力図形が得られるものと仮定すると，データの総量は約3.9Mバイトになる．このなかで入力と出力の多角形レコード(この場合矩形)の占めるデータは3.5Mバイトで約90%になり，あとの10%分がスリット内のベクトルに関するデータである．したがって，計算機のメモリが3～4Mバイト程度と仮定すると，16～24k個の矩形に相当する図形を含む領域が扱えることがわかる．領域の境界をまたぐパターンの数は，別に行う境界処理の量が増えることになるので，少ないほうがよい．パターンにはまとまった機能ごとに同じパターンがくり返し現れる性質があるから，最もパターン数の少ない機能回路の境界点で分割するのが望ましい．たとえば，レジスタはフリップフロップ単位でパターンがくり返されるから，フリップフロップの境界が最もパターンが少ない．このようなことを考慮して境界を定めればよい．

プロセッサのビット数は，扱う領域の座標数とアドレス空間の大きさによってきまる．いま y 方向に 64k 点の座標をとるとすると，16 ビットプロセッサを使用することが必要である．最近の16ビットマイクロプロセッサは，アドレス空間が 16M バイトのものがあるから，アドレス空間の点からも適当である．今回は小規模実験のため8ビットプロセッサを使用したが，実際の場合には16ビットプロセッサに主メモリを3～4Mバイト程度つけることを考えている．

5. むすび

LSI レイアウトパターンのデザインルールチェックを行う専用計算機を考えるにあたって，並列動作させるマイクロコンピュータで実行する図形演算の方法について考察した．この方法は，(1)ベクトルスライス法を使用して多角形からスライスベクトルをつくる，(2)図形の重なりや輪郭取りなどの図形演算を行う，(3)演算結果のスライスされたベクトルから出力多角形を復元する，の3段階からなっている．とくに図形復元法については，簡単なポインタ操作による方

法を考案した．これらの図形演算は全多角形を一度に処理する方法ではなく，スリットを移動しながら関係する図形のみをベクトル化し，また，演算を終了した部分から順次出力多角形を復元する方法をとっているので，メモリ容量も少なく，マイクロコンピュータで十分処理できる．簡単な図形ではあるが，この方法を8ビットマイクロコンピュータで実行して正しい結果を確認した．また，実際には16ビットプロセッサを使用し，3～4Mバイトのメモリをつければよいことを示した．この結果，マイクロコンピュータの並列動作を使用した専用計算機が可能であることが明らかになった．

参考文献

1) Lindsay, B. W. and Preas, B. T.: Design Rule Checking and Analysis of IC Mask Designs, Proc. of 13th DA Conf., pp. 301-308 (1976).
2) Yoshida, K. et al.: A Layout Checking System for Large Scale Integrated Circuits, Proc. of 14th DA Conf., pp. 322-330 (1977).
3) Arnold, M. H. and Ousterhout, J. K.: Lyra, A New Approach to Geometric Layout Rule Checking, Proc. of 19th DA Conf., pp. 530-536 (1982).
4) Tsukizoe, A. et al.: MACH: A High-Hitting Pattern Checker for VLSI Mask Data, Proc. of 20th DA Conf., pp. 726-731 (1983).
5) Seiler, L.: A Hardware Assisted Design Rule Check Architecture, Proc. of 19th DA Conf., pp. 232-238 (1982).
6) McCaw, C. R.: Unified Shapes Checker—A Checking Tool for LSI, Proc. of 16th DA Conf., pp. 81-87 (1979).
7) Baird, H. S. and Cho, Y. E.: An Artwork Design Verification System, Proc. 12th DA Conf., pp. 414-420 (1975).
8) Kozawa, T. et al.: A Concurrent Pattern Operation Algorithm for VLSI Mask Data, Proc. of 18th DA Conf., pp. 563-570 (1981).

(昭和59年11月19日受付)
(昭和60年1月17日採録)