6N−6

タイル方式により生成したモジュールのセル上配線通過可能領域抽出手法

塚本　美智子,　中尾　博臣,　岡崎　芳
三菱電機株式会社

1．はじめに

ROM,RAM 等規則構造モジュールを生成するモジュールジェネレータのレイアウト生成方式としてはタイル方式（リーフセルを隣接配置してレイアウトを生成する方式）が有効である。一般にこれらのモジュールは面積が大きく端子数が多いために、チップのレイアウトの際にモジュールの周囲は配線が混雑し易く、集積度の低下、配線長の増加という問題が生じる。これらを回避するために、モジュール上で配線の通過できる領域（配線通過可能領域）を配線領域として利用することが考えられる。本稿では、タイル方式により生成するモジュールを対象に、高速にしかも少ないメモリ使用量で配線通過可能領域を抽出する手法を提案する。

2．配線通過可能領域の形状

面積の大きいモジュールを用いてチップのレイアウトを行なう際にチップの集積度を上げるためには、モジュール上にできるだけ多くの配線を通すことが重要である。従って配線通過可能領域は、通過配線に使用するレイヤと同じレイヤのマスクパターンを除く、全ての領域であることが望ましい。しかし、任意の形状の配線領域は保持するデータ量が多く、配線プログラムの実行時間の増加を招き易いという問題がある。また、矩形の領域と任意の形状の配線領域では、その間を通る配線の本数には差がない場合が多い。したがって配線通過可能領域の形状は、実用上次に示す3通りの形状で十分である。

・モジュール上をX方向に貫通する矩形
・モジュール上をY方向に貫通する矩形
・モジュール上をXとYの両方向に貫通する十字形

このように限られた形状の領域を抽出するため、本稿で提案する手法では、配線通過可能領域抽出問題を下記の配線禁止領域（オブスタクル）を生成する問題として扱う。

①通過配線に使用するレイヤと同じレイヤのマスクパターンを含み、かつY方向にモジュールを貫通する矩形を、モジュールのオブスタクルとする。
②モジュール内にオブスタクルが複数存在する場合は、各オブスタクルの図形のORをとったものをモジュールのオブスタクルとする。（図1参照）
③②において、オブスタクルとオブスタクルの間隔が、配線の通過できる最小の領域幅（配線通過可能最小幅）より小さい場合はこれらのオブスタクルとオブスタクルをマージしたものをオブスタクルとする。

X方向に貫通するオブスタクルの形状は、上記①のYとXを入れ替えたものである。また、十字形はXとYそれぞれに貫通する矩形のオブスタクルのORをとればよい。

以後本稿ではY方向のオブスタクルに限って説明を行なう。

3．オブスタクルの生成手法

3．1　生成手法

オブスタクルの生成手法として、階層を全て展開（スマッシュ）したモジュールのマスクパターンから対象とするマスクパターンを抽出し、それらを合成しオブスタクルを生成する手法（ここではこの手法をスマッシュ法と呼ぶ）がある。

一般に、タイル方式により生成するモジュールのレイアウトには、①あらかじめ設計したリーフセルがタイル状に並ぶ、②モジュールが階層的に構成されている（ここでは最下層をリーフセルと呼び、リーフセルを隣接配置して構成した中間階層のセルをブロックと呼ぶ。ブロックを隣接配置して構成する最上層のセルをモジュールと言う。以後、リーフセル、ブロックをまとめてセルと呼ぶ）、③同一のセルが等間隔にアレイ状に配置されている部分が存在する、という特徴がある。したがって、①から③の特徴を利用した次の方法を用いることにより、スマッシュ法よりも高速にしかも少ないメモリ使用量でオブスタクルを生成することができる。

①モジュールのオブスタクルはリーフセルのマスクパターンからオブスタクルをあらかじめ抽出し、そのデータをリーフセルの配置座標にしたがって並べ、合成する。
②オブスタクルはモジュールの階層毎に生成する。
③同一セルがアレイ状に配置されている部分は、データ量削減のためにアレイの最下段のセル以外はオブスタクル生成の処理を省略する。

3．2　オブスタクル生成アルゴリズムと評価

図2を用いてY方向のオブスタクル生成のアルゴリズムについて説明する。各階層において、セルが回転や反転して配置されている場合には、各階層ごとに同様の方法でX方向にも貫通するオブスタクルを生成すればよい。

(1) オブスタクルの表現

オブスタクルは左辺と右辺のペアで表し、各辺はそれぞれそのX座標と、正方向、負方向の単位ベクトルで表現する。ブロック内のオブスタクルのひとつ、O_i はオブスタクルの左辺を v_{iL}、右辺を v_{iR} とした場合次のように表記する。

$O_i = v_{iL} v_{iR}$

$v_{iL} = (x_{iL}, e)$, $v_{iR} = (x_{iR}, -e)$

x_{iL}はオブスタクルの左辺のX座標、x_{iR}は右辺のX座標、eは単位ベクトルである。ここで、v_{iL}、v_{iR}は一般に次のように表記し、以後辺ベクトルと呼ぶ。

$v_k = (x_k, n_k\ e)$　（x_k ：オブスタクルの左辺、あるいは右辺のX座標、n_k ：整数）

(2) オブスタクルの合成

ブロック（あるいはモジュール）のオブスタクルは、その中に含まれるセルのオブスタクルを合成したものとなる。ブロックに含まれるセルの全てのオブスタクルは、各オブスタクルの辺ベクトルの群として表すことができ、これらの中で次の条件を満たす辺ベクトルがブロックのオブスタクルの辺ベクトルとなる。

ブロックに並ぶ左端の辺ベクトルから順に eの係数n_k を加算していく時、

A Method of Feedthrough Area Extraction of A Module Generated by Tiling Method

Michiko TSUKAMOTO, Hiroomi NAKAO, Kaoru OKAZAKI
Mitsubishi Electric Corporation

・係数の合計値が0から1になったときの辺ベクトル（オブスタクルの左辺。合計値の初期値は0とする）
・ eの係数の合計値が0になったときの辺ベクトル（オブスタクルの右辺）

(3) アルゴリズム

①ブロック内に存在する全ての辺ベクトルを、X座標に関してソーティングして生成したベクトル列を、$V=v_0\ v_1\ \cdots\ v_{max}$ とする。

②$W=\phi$

```
for k←0 until max do
    W=f(W, v_k)
```

関数fは辺ベクトルv_kと辺ベクトル列$W=v_i\ v_{i+1}\ \cdots\ v_j$に対して、(2) で説明した辺ベクトルを選択するための、次式で定義される関数である。

$f(W, v_k)=v_i\ v_{i+1}\cdots(v_j \oplus v_k)$
$(\phi \oplus v_k) = v_k$
$(v_j \oplus v_k)=(x_j, n_j\ e)\oplus(x_k, n_k\ e)$
$=(x_j,(n_j+n_k)\ e)$: if $(n_j+n_k \neq 0)$
$=v_j\ v_k$: if $(n_j=1$ and $n_k=-1)$
or$\{(n_j=-1$ and $n_k=1)$
and$(x_k-x_j)>$limit$\}$
$=\phi$: if $(n_j=-1$ and $n_k=1)$
and$(x_k-x_j)\leq$ limit

n_j, n_k：整数、limit:配線通過可能最小幅、ϕ：空列

③②の処理が終了した後、Wの辺ベクトル列 $W=v_0'\ v_1'\cdots v_t'$ の先頭から順に2つずつ、次のようにペアリングし、ブロックのオブスタクルとする。

$O_0=v_0'\ v_1'$, $O_1=v_2'\ v_3'$, ・・・,
$O_{t-1/2}=v_{t-1}'\ v_t'$

図2では、(X1, e) (X4, $-e$) 、(X5, e) (X8, $-e$) のペアが、生成されたオブスタクルである。

④以上の処理を一番上の階層のモジュールまで階層的に行なう。

(4) アルゴリズムの評価

辺ベクトルのソーティングの実行時間のオーダは、平均O（n logn）（nは辺ベクトルの総数）であり、関数fの実行時間はO（n）であることから、本アルゴリズムの実行時間はO（n logn）である。一般的にモジュールは階層的に構成されており、以下のことから処理が高速になる。

①一回のソーティングで扱うデータ量は一階層に存在する辺ベクトルの数だけでよい。

②既に生成したオブスタクルを上の階層で使う頻度が高いモジュールは、ソーティングするデータ量が減少する。

また、オブスタクルのデータを格納するメモリ使用量は一階層に存在するオブスタクル数だけでよく、メモリ使用量が節約できる。

4．適用結果

本手法を用いて、セルの配置座標とリーフセルのオブスタクルを入力とし、モジュールのオブスタクルを生成するツールを作成し、SOG 設計方式（トランジスタ敷き詰め方式）で用いるRAM ジェネレータの開発に適用した。ここでは、モジュールのポリシリコンのゲートの方向に対して、平行に貫通するオブスタクルを生成している。

RAM のオブスタクル生成時間を10MIPSワークステーション上で測定した結果を図3に示す。オブスタクルの総数は、RAM のメモリ容量（ビット数）に比例するものと考え、縦軸はオブスタクル生成時間、横軸はメモリ容量とした。グラフから、オブスタクル生成時間はメモリ容量にほぼ比例しており、O（n logn）よりnに対する処理時間の増加率は小さい。

図4.1 に、RAM ジェネレータが生成したメモリ容量1kビットのRAM のマスクパターンを、図4.2 に、RAM のオブスタクルの拡大図を示す。図4.2 は第2層アルミのマスクパターンとオブスタクルのみを示す。メモリ容量1kビットのRAM の内部のオブスタクルは43箇所であり、生成時間はCPU で6.2秒であった。

5．まとめ

タイル方式で階層的に構成されているモジュールを対象に、高速でしかもメモリ使用量の少ない、配線通過可能領域を抽出する手法を提案し、実験によりその有効性を確認した。

本手法に基づくプログラムは、SOG 用各種 RAMジェネレータで実用中である。

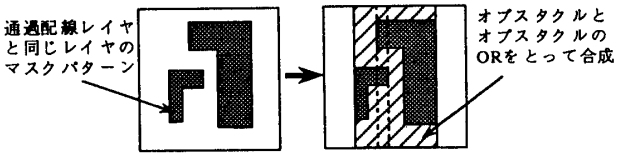

図1 オブスタクルのORの合成

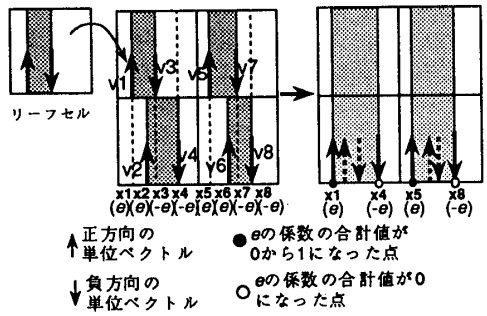

図2 オブスタクルのベクトル合成

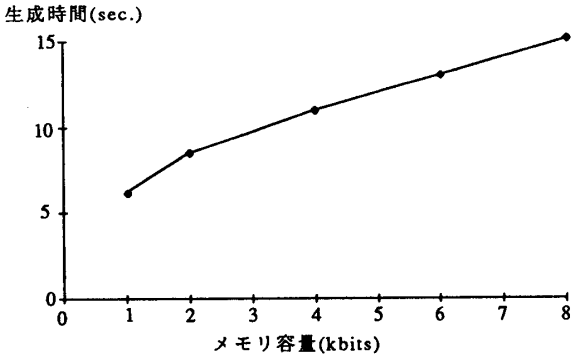

図3 RAMのオブスタクル生成時間

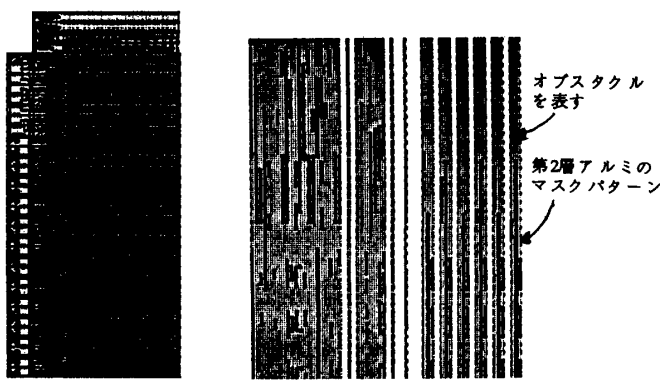

図4.1 RAMのマスクパターン　図4.2 配線通過可能領域の拡大図