設 計 自 動 化 46－5
（１９８９．２．２０）

ゲート敷き詰め型ゲートアレイの自動配線
― 手法と実現 ―

山田正昭　高野みどり　門脇春則
（株）東芝　ＵＬＳＩ研究所

ゲート敷き詰め型ゲートアレイ用の自動配線プログラムを開発したので、報告する。ゲートアレイのレイアウト方式の多様化に対応して、配線層数や配線領域の形状に依存しない柔軟なレイアウトモデルを想定した配線手法を提案している。

配線プログラムは、概略配線と詳細配線の２つのステップに分けられている。概略配線では、best-first searchの手法を使って、大規模なゲートアレイを高速に処理することを目指している。詳細配線では、線分探索法を使っているが、配線リソースのリザーブ機能、配線順序の動的変更、例外を許した縦横ルールなどを使って、配線率を高める工夫をしている。

A Router for Channel-Free Gate Arrays

Masaaki YAMADA, Midori TAKANO, Harunori KADOWAKI
ULSI Research Center, Toshiba Corporation
1, Komukai-Toshiba-cho, Saiwai-ku, Kawasaki 210, Japan

A newly developed routing program for channel-free gate arrays is described. To cope with various gate array layout methodologies, a flexible routing method, independent of the number of layers or the shape of routing area, was developed.

The router is divided into a global and a detailed router. The global router aims at high-speed routing for large scale gate arrays, using best-first search algorithm. The detailed router is basically a line-search router with features such as reserving wiring resources, dynamic net ordering and X-Y rule allowing exceptions.

1．はじめに

ゲート敷き詰め型ゲートアレイ用の配線プログラムを試作したので、そこで使われた手法および実際の回路に適用した結果などを報告する。

ゲート敷き詰め型ゲートアレイを利用すると、RAM、ROM、PLAなど規則的なブロックや既設計のブロックが容易に取り込める。また、セルとしても必ずしも従来のカラムーチャネル方式のように高さ（幅）を揃える必要はなく、かなり自由なセル設計ができる。このようなゲート敷き詰め型ゲートアレイの特徴は、ゲートアレイの可能性を広げるものであるが、反面、自動配置配線は多彩なレイアウトモデルに対して柔軟に対応することが必要になる。また、多様な回路をゲートアレイに取り込める結果、システムの大きな部分を1チップ上に載せることができるようになり、プロセスの微細化と相まって、ますます大規模化が進むことになる。配線多層化の方向も見逃せない。また、設計のターンアラウンドタイムは妥当な限度内に抑えることも必要である。

このような要求を考慮して、本プログラムでは多層配線を含む多様なレイアウトモデルに柔軟に対応し、かつ、高速な配線が可能であることをめざしている。

カラムーチャネル方式ではないため、チャネル配線は使わず、迷路法、線分探索法をベースにした配線を行っている。これらの配線手法は一般に処理時間がかかることが短所であるので、特に高速化には工夫をしている。

本プログラムは概略配線と詳細配線に分かれており、本稿では、両方の報告を順次行う。まず、第2節では今回用いた配線のレイアウトモデルを記述する。第3節、第4節では概略配線、詳細配線についてそれぞれ述べる。両節では、各々の問題設定、手法、および実際の回路に適用した結果について述べている。

2．配線のレイアウトモデルとプログラムの機能

本プログラムでは、次のような一般性のあるレイアウトモデルをサポートしている。

(i)使用する金属配線層数には制限がなく、何層の配線でも可能である。ただし、主として3～5層程度を想定している。

(ii)端子は、任意の配線層またはポリシリコン層上にあり、任意の直角多角形の形状を持つ。一つの端子が複数の層にまたがっていても良い。端子の領域は、いくつかに分離していても良い。端子の位置は配線にはいる前に固定されている。

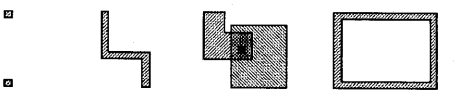


図1　端子の例

(iii)配線禁止領域は、任意の配線層上にあり、任意の直角多角形の形状を持つ。ビア禁止点も定義できる。

(iv)配線は、原則として、一様なグリッド上で行われる。特別な場合を除いて、配線セグメントはグリッドの線上、ビアはグリッドの交点上に置かれる。（グリッド系の詳細については後述。）

(v)デザインルールに違反する配線はしない。与えられるデザインルールは、各層の配線幅・ビア部分の幅・スペーシングなどである。これによって、例えば、隣接ビアの禁止、グリッド1本おきの配線などが行われる。

(vi)第1層は主に縦、第2層は主に横、第3層は主に縦.....方向に使われる。短い区間でこの原則に反することもある。

(vii)100%配線を第1目標とし、配線長最小化、ビア数最小化を第2目標とする。

(viii)事前の配線（電源線、クロック等の特殊配線やユーザが特に指定した配線など）を避けて配線する。

以上述べたことからわかるように、本プログラムには、チャネルやセル列といった概念がないのはもちろん、セルや機能ブロックというような概念もない。セルや機能ブロック等の外枠を考えることは、配置の段階までは必要であるが、一旦配置されてしまった後は必要ない。セルの機能を実現するためにほとんどの場合セル上の金属第1層は使われてしまっているので、そのことを考慮する必要はあるが、それについては配線禁止領域として定義しておけば良いだけである。また、端子に関してもある端子がどのセルに属しているかと言うことは重要ではなく、ただどのネットに接続しているかが配線に対する要求になる。従って、配線プログラムが認識する対象は端子と配線禁止領域だけであり、それらがチップ上に直接くくりつけられていると考えて良い。なお、配線禁止領域には既配線なども含まれる。

本プログラムでは、グリッド配線とグリッドレス配線を比較した結果、グリッド配線を採用した。ひとつには、大規模データを扱う場合にグリッドレス配線には相当時間がかかると予想されたためであり、他の理由としては、グリッド配線の方がプロセス上問題が少ないためである。ただし、グリッド配線と言っても各層でデザインルールが違う

ことやネット毎に線幅を変えることが可能なように、考え方には次のように柔軟性を持たせた。

＊グリッド系の考え方

(i)グリッドは、「その上以外は通れない場所（必要条件）」であって、「その上を通ればデザインルール違反をしない場所（十分条件）」ではない。

(ii)縦方向のグリッドと横方向のグリッドは、互いに独立のピッチを持つ。

3．概略配線

3．1　概略配線問題

前述したように、本プログラムにはチャネルやカラムなどの概念はないので、概略配線も配線をチャネルに割り当てるような問題にはならない。概略配線は、「チップ上を適当な格子に区切り、各配線がどの格子境界を通るかを決定する」問題になる。格子の区切り方は、ゲートアレイの下地とは全く独立に決めることが可能であるが、ある程度下地トランジスタの配列に整合させた切り方が望ましいと思われる。また、事実上チャネルやカラムがあるときは、それに整合させるのが良いと思われる。

本プログラムは多層配線をターゲットとしているので、配線層を考慮する必要もある。すなわち、格子境界を横切るときの配線層も決定する。前述のように、縦横ルールを採用しているので、格子境界を縦方向に通過するときは第１層、第３層、第５層．．．が可能であり、横方向に通過するときは第２層、第４層．．．が可能である。

端子は一般に任意の大きさ・形状を持っているので、一つの端子が複数の格子にまたがって存在することもあり得る。その場合も、その端子がどの格子に存在するかを前処理的に決めてしまうのではなく、そのままの形で扱う。すなわち、概略配線を進めていく過程で、最も適当な位置に配線が接続される。

3．2　概略配線の評価関数

概略配線の目的は、後続の詳細配線のフェイズに於て、１００％配線が可能であるように、配線混雑の集中を避けて配線径路を決定することである。そのために、混雑度の均一化を第１の目的関数とする。その他配線長、ビアの最小化も目的関数となる。本プログラムで採用している目的関数ｆは、次のような形である。

$$f = \alpha \sum_{\text{格子境界}} (\text{配線混雑度}) + \beta \sum_{\text{ネット}} (\text{配線長}) \quad (3.1)$$

配線長は、広義のものであり、ビアの個数も適当な係数によって長さに変換されており、また、配線層による配線長のパフォーマンスへの影響も調整されたものとする。α、βは重み付けの係数である。

配線混雑度は、各格子境界上に次のように定義される。その格子を通過できる最大配線本数（配線リソース）をＲ、その格子の実際の通過配線本数をＷとすると、配線混雑度Ｄは、

$$D = g(W - R) \quad (3.2)$$

と表わされる。ｇ()は、ゼロ付近で急速に立ち上がる関数で、例えば図２のようなものである。すなわち、通過配線本数が配線リソースより５本以上少ないときは、混雑度はゼロとみなされるが、通過配線本数が配線リソースに近づくに従って、少しずつ混雑度が上昇し通過配線本数が配線リソースを超えると非常に大きな値になるようになっている。

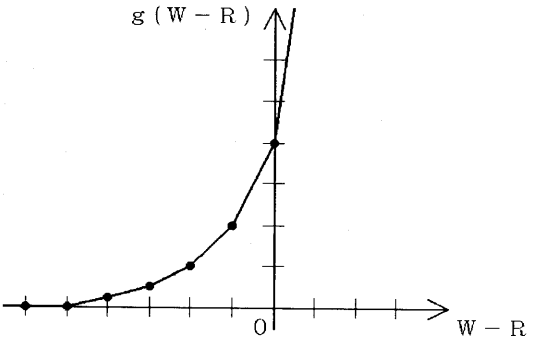


図２　混雑度関数の例

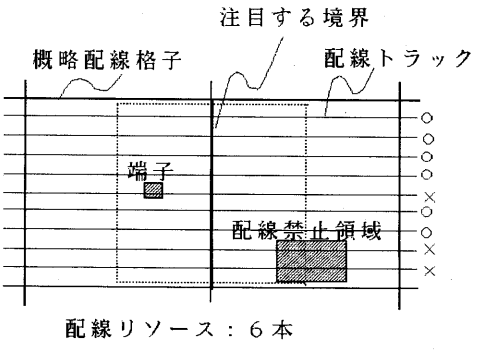


図３　配線リソースの見積り方

配線リソースは、実際に詳細配線をしてみない限り正確に決定することができないので、次のような近似的な方法で見積っている。図3のように、境界辺の両側に一定の幅を取り、その中に障害物があれば、その障害物のあるトラックは通れないものとして、障害物の全くないトラックの本数を配線リソースとする。

3．3　概略配線手法

本プログラムでは、逐次配線手法を採用し、格子グラフ上の最短径路探索をベースとしたアルゴリズムを開発した。処理手順を形式的に記述すると、図4のようになる。

```
(1) for(全てのネットについて)
    {
(2)     ・ネットの端子を一つ選び、それの存在する格子
          をDに入れる。
(3)     ・残りの端子の存在する格子を、Uに入れる。
(4)     while(U≠φ)
        {
(5)         ・DまたはUの一方を探索のターゲットに他
              方をスタートにし、最短径路探索を行う。
(6)         ・探索がターゲットに到達したら、それによ
              って得られる径路をPとする。
(7)         D←D＋P
(8)         U←U－P
        }
(9)     ・Dをそのネットの径路として登録する。
    }
```

D：径路に含まれている格子の集合
U：径路に含まれていない端子の存在する格子の集合

図4

一般には、端子は2つ以上の格子にまたがって存在することもあるので、図4(2),(3)において、1つの端子に対して2つ以上の格子が選ばれることもある。また、(6)における径路Pとしては、径路によって接続されたスタート格子、ターゲット格子も含む。

3．3．1　リソースの考慮

本プログラムでは、最短径路探索で径路を決めているが、その際考えているコストは単なる距離ではなく、混雑度を加味した仮想的な距離である。すなわち、格子境界を通過するためのコストcは、縦方向であるか横方向であるか上下方向（層を変える方向）であるかによって決まる一定値Lと、その境界の混雑度によって決まる値との和になる。

$$c = L + g'(W - R) \qquad (3.3)$$

ここに、Wはその境界を通過している配線の本数であり、Rはその境界を通過できる配線の本数（配線リソース）である。g'()は前述の混雑度関数g()の差分関数であって、この境界に1本配線が追加されるとどれだけ混雑度が増加するかを示す関数である。図2のg()の例に対しては、g'()は図5のようになる。Lとしては、縦方向、横方向、上下方向とも1を採用している。

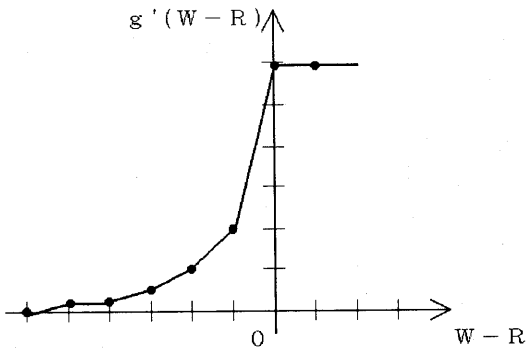


図5　混雑度関数の差分

3．3．2　Best-First Search

実際に最短径路探索をする部分では、格子点を節点、格子境界を枝とする格子グラフ上で例えばDijkstra法[3]を用いればよいのであるが、本プログラムでは高速化のためにbest-first search法[1]を採用している。なお、混雑度を考慮しなければ、Dijkstra法は迷路法と同じになる。また、混雑度を考慮しないbest-first search法は文献[4]の方法と同じになる。

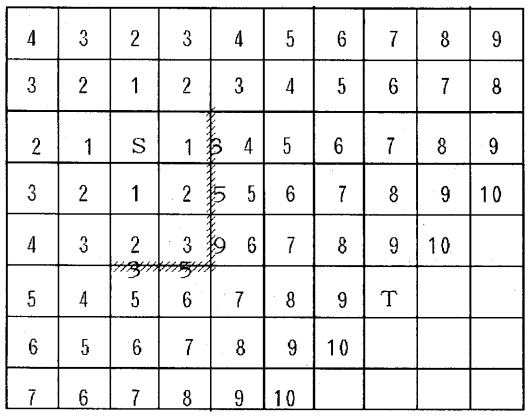
通常のDijkstra法では、次に探索を進める節点（格子点）としてスタートからの累積コスト

$$f(s, x) \qquad (3.4)$$

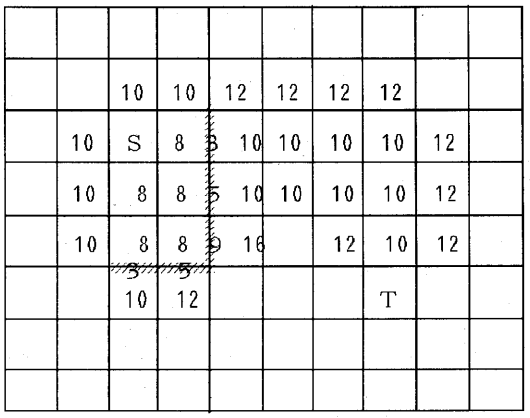
が最も小さい節点xを選ぶが、best-first search法では、スタートからの累積コストとターゲットまでの見積最小コストの和

$$f(s, x) + LB(x, t) \qquad (3.5)$$

が最も小さい節点xを選ぶ。ターゲットまでの見積最小コストとしては、マンハッタン距離を採用した。その際、ターゲットと層が違うときは、見積に層差のコストも加えた。この方法によって如何に探索範囲が狭くなるかを図6に示す。



（ａ）Dijkstra法の場合



（ｂ）Best-First Searchの場合

図６　手法による探索範囲の違い

なお、図６において格子境界上に付記された数字はその境界のコストを示す。ただし、付記のない境界のコストは１である。また、格子内に記入されている数字は累積コストの値である。

Best-first searchでは、見積最小コストが実際のコストに近いほど効率よく処理ができる。我々の問題の場合、ほとんどの径路がマンハッタン距離で配線できるという事実がこの方法を採用できる根拠となっている。しかし、このままでは、わずかに迂回を要する場合でも、大幅に探索範囲が拡大する場合があるので[4]、実際のインプリメンテーションでは、見積最小コストに１より大きな数ovplを掛けたものを使っている。

$$f(s,x)+LB(x,t)\cdot ovpl \qquad (3.6)$$

すなわち、見積コストをやや大きくして、多少の迂回を見込んだものにしている。

前述のように、一般には、スタートもターゲットも１点ではなく、多くの格子からなっている。特に、ファンアウトの大きなネットでは、既に発見された径路をスタートまたはターゲットとして探索を行うので、この傾向が甚だしい。

スタートが多点になっている場合は難しくない。すべてのスタート点から同時に探索を開始し、評価値の小さい格子から順次探索を前に進めればよい。ほとんどの場合、一つのスタート点だけから、探索が優先的に進み、スタートが多点になったためのオーバヘッドは少ない。

ターゲットが多点になっている場合が問題である。ある点からターゲットまでの見積最小コストを求める際に、正確に求めようとすると、ターゲット各点までのマンハッタン距離を求めてその最小をとらねばならず、探索を進めるたびに繰り返していては処理時間がかかりすぎる。そこで、あらかじめターゲットを適当なデータ構造に格納しておき、探索時には、最も近いターゲット点を高速に求める方法が考えられる（例えば、[6]）。しかし、ターゲットまでのマンハッタン距離を求めることはそもそも単に高速化の手段に過ぎないので、処理のオーバヘッドを考えても引き合うとは限らない。本プログラムでは、正確なマンハッタン距離を求めることはあきらめ、ターゲットの外接矩形までの距離で代用している。なお、以上の理由で、ターゲットが大きくなることは不利なので、図４の処理手順の(5)において、スタートとターゲットの決め方に自由度があるところでは、ターゲットの方が小さくなるように選ぶようにしている。

３．４　実験結果

ここで提案した手法の効果を調べるため、実際のゲートアレイ回路に適用してみた。単純な最短径路探索と、best-first searchの比較を表１に示す。Best-first searchについては、さらに、パラメタovplを変化させて調べた。パラメタovpl=1のとき、基本的なbest-first search(3.5)に一致する。

表１　手法による配線結果と処理時間の違い

		Best-First Search				Dijkstra法
ovpl		2.0	1.5	1.2	1.0	
総配線長		10929	11009	11055	10854	10895
混雑度 (W-R)	0	7	4	1	3	4
	-1	44	37	35	24	26
	-2	193	105	101	91	78
探索処理時間 (CPU・sec)		22	29	36	41	166

表１によれば、best-first searchを用いることによって、４～８倍の高速化がなされることがわかる。また、ovplを大きくすると、処理速度は期待通り速くなるが、局所混雑度は若干増加する。これは、次のような理由による。パラメタovplが大きくなると、(3.5)の第２項、すなわち、ターゲットまでの予想距離が強調される。そのため、ターゲットまでのマンハッタン距離が近いところから優先的に探索が進められ、その付近に混雑度が大きい境界があっても、ある程度無視してしまうためである。

以上からわかるように、ovplは必ずしも大きければ良いというものではなく、処理時間と配線結果の良さのトレードオフを考慮して決めるべきパラメタである。

４．詳細配線

４．１　詳細配線問題

概略配線の結果を参考にして、各ネットがデザインルール違反なく接続されるように、個々の配線セグメント、ビアの座標値を決定することが、詳細配線の役割である。詳細配線問題は比較的自明であって、100%配線が第一目標で、それが達成されれば、配線長、ビア数の最小化が第二目標となる。

４．２　詳細配線手法

チップ上を適宜な大きさの配線領域に区切り、各配線領域内の詳細配線を順次行うことによって、全体の配線を完成させる。各配線領域内配線の配線要求は、その領域内の端子と概略配線によってその領域の境界に割り当てられた配線との情報によって得られる。配線領域内配線において概略配線の結果を不整合なく利用するためには、配線領域と概略配線格子が一致するか、配線領域がいくつかの概略配線格子を包含するようになっていなければならない。両者の関係を図７に示す。

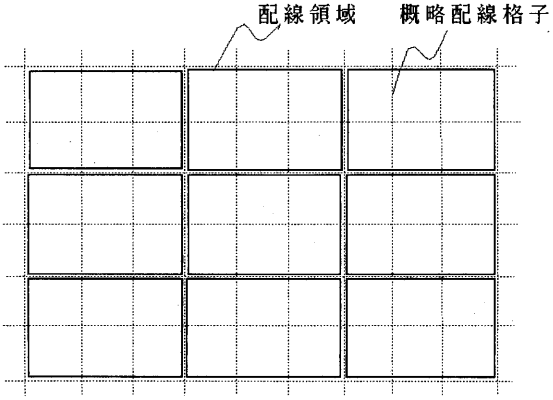
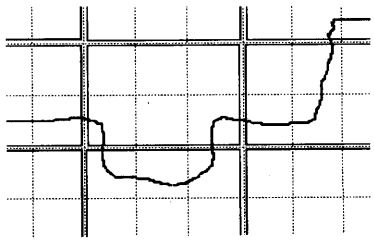


図７　配線領域と概略配線格子の関係

各配線領域内の配線は、もともとその領域内にある端子と概略配線によってその領域の境界辺上に割り当てられた配線を仮想端子として、同一ネットに属する端子同士を接続することによって行う。領域内部の端子は、前述のように、１点ではなく広がりをもっている。境界上の仮想端子は、隣接する領域が未配線のときは境界上のある範囲の中の任意の位置として定義され、隣接する領域が既配線のときは境界上の１点として定義される。概略配線でその領域を２度通るようにネットが割り付けられた場合、同一ネットであってもその配線領域内部で接続させるべきでない場合も存在するので、注意が必要である（図８参照）。



図ｘｘｘ８　配線領域をネットが２度通る場合

領域内の配線は、原則的に、線分探索を各端子対に逐次適用することによって行う。多端子ネットの場合は、２端子対に分解し、優先度をつけて順次処理する。端子対が接続されたらその間をつなぐ配線を含めて一つの端子として扱い他の端子と接続する目標とする。端子数が比較的少ないネットについては全２端子対の総当たりとするが、端子数が多いネットについては全端子を網羅するような適当な２端子対の集合を選んで処理する。一つのネットの端子対は引き続いて処理される必要はなく、あるネットの一つの２端子対の処理をした後、他のネットに移ることも可能である。

逐次配線の場合、先に配線されたネットによって後の配線が妨害され配線が不成功に終わるということが、常に問題となる。これを回避する方法として、(i)後の配線を妨害しないように考えて慎重に配線する方法と、(ii)後処理として引きはがし再配線をする方法とが考えられる。本プログラムでは、前者の方針をおもに採用し、補助的に後者を使うという方法をとった。ただし、引きはがし再配線はまだインプリメントされていない。後の配線を妨害しないようにするために、(i)配線リソースのリザーブと、(ii)配線順序の動的変更が行われるようになっている。

(i)配線リソースのリザーブ機能

既に行われた配線によって後の配線が妨害されるのは、多くの場合、未配線の端子の周りに他の配線が引かれて、配線の引出しが不能になるためである。そこで、各端子から配線が引き出されると予想される方向にある程度の長さ

の“勢力範囲”を設定しておき、その勢力範囲内は、他のネットの配線を制限するという方法をとった[5]。図9に勢力範囲設定の例を示す。

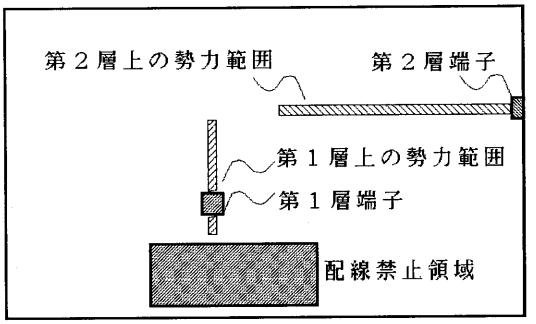


図9　勢力範囲設定の例

勢力範囲は、一般に端子から、その端子と同じ層でその層の主配線方向にのばした線分である。線分の長さは、配線領域の一辺の長さの、例えば、50%程度である。

(ii)配線順序の動的変更機能

配線に自由度があるものは、なるべく後に回して、周囲の状況が固まってから配線しようとする機能である。すなわち、曲がりなしで配線できるものや、１回曲がりで配線できるものは、最短距離で配線しようとすると、自由度が少ないが、曲がり回数が多いものは、最短距離で配線するにしても幾通りもの径路の取り方があるので、曲がり回数の少ないものから配線するようにしている。予め曲がり回数を予測して配線順序を決めているが、実際には、障害物によって予測した曲がり回数では配線できないこともある。そのため、ある端子対を配線しようとするときは、曲がり回数に制限をつけた線分探索を行い、制限内の曲がり回数で配線できないときは、とりあえず失敗として、その端子対の配線は延期するようにしている。また、当初は他のネットの勢力範囲は避けて配線を試行するが、それによって配線に失敗したネットが残った場合、後のフェーズでは勢力範囲を無視した配線を行なう。

各端子対の配線には線分探索法を使う。上記のように折れ曲がり回数に制限を設けて配線を行うことは、線分探索法の性質上、容易なことである。線分探索法を多層配線に応用することもそれほど難しいことではない。縦横ルールを使う場合は、層毎に線分の発生方向を制限すれば良い[7]。ただし、ここで採用したレイアウトルールのように、短い区間でのみ縦横ルールに違反しても良いという場合は多少の工夫がいる。本プログラムでは、次のような方法でこの問題を解決した。探索線を発生させる際に、縦横ルールに従う方向には、通常の線分探索法のように、障害物などに達するまで線分を延長するが、縦横ルールに反する方向には、たとえ障害物に達しなくとも、ある一定長さ（典型的には、１トラックピッチの長さ）だけしか線分を延長しない。

４．３　実験結果

ゲートアレイ回路の一部領域に詳細配線を施した例を図１０、図１１に示す。図１０は２層配線、図１１は４層配線である。これらの配線に、約１０MIPSの計算機で、５～１０秒要した。

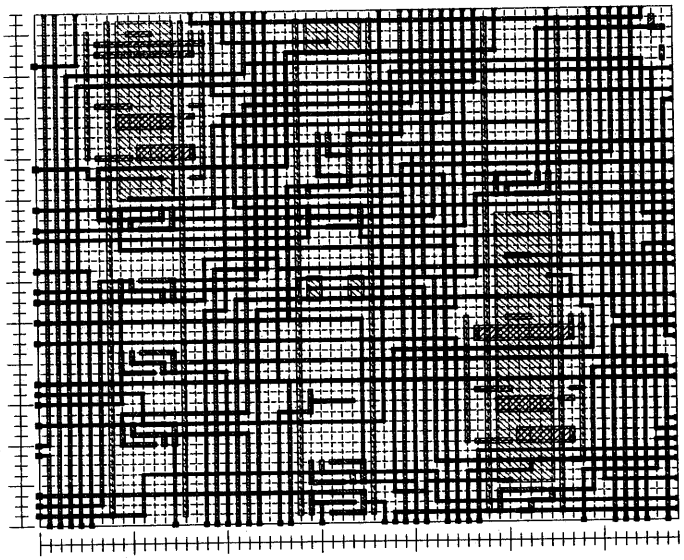


図１０　２層配線の例

5．まとめ

ゲート敷き詰め型ゲートアレイの配線プログラムとして、最も一般的なレイアウトモデルでの配線手法を提案し、実用的に利用可能であることを検証した。概略配線でのbest-first search、詳細配線での配線リソースのリザーブ、配線順序の動的変更などの方法は、処理時間の短縮や配線率の向上に効果があることが確認できた。

今後は、引きはがし再配線もつけ加え、全体としての性能を確認して行きたい。

（参考文献）

[1] N. J. Nilson, “Problem-Solving Methods in Artificial Intelligence”, Mcgraw-Hill, Ch.3, pp.43-78 (1971).

[2] G. W. Clow, “A Global Routing Algorithm for General Cells”, 21th D. A. Conf., pp.45-51 (1984).

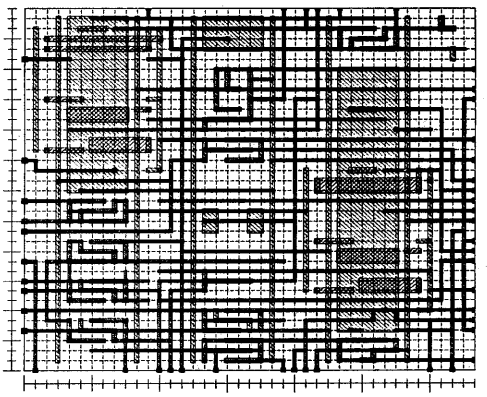
[3] E. W. Dijkstra, “A Note on Two Problems in Connection with Graphs”, Numerische Math., vol.1, pp.269-271 (1959).

[4] R. K. Korn, “An Efficient Variable-Cost Maze Router”, 19th D. A. Conf., pp.425-431 (1982).

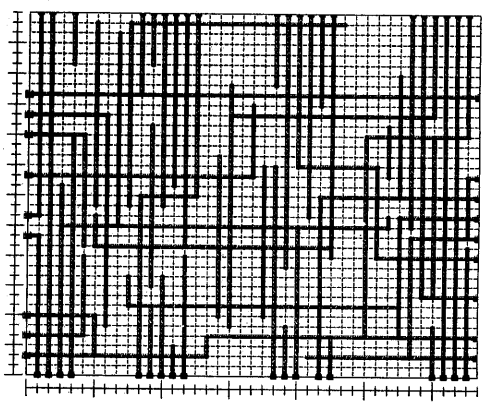
[5] J. P. Cohoon, P. L. Heck, “BEAVER: A Computational-Geometry-Based Tool for Switchbox Routing”, IEEE Trans. on CAD, vol.7, no.6, pp.684-697 (1988).

[6] I. Kalantari, G. McDonald, “A Data Structure and an Algorithm for the Nearest Point Problem”, IEEE Trans. on Software Eng., vol.SE-9, no.5, pp.631-634 (1983).

[7] 石塚，野田，西口，“大規模３層ゲートアレイの配線手法”，信学技報，VLD87-97 (1987).



（ａ）１，２層



（ｂ）３，４層

図１１　４層配線の例