設 計 自 動 化 67−12
（1993．6．25）

アナログレイアウトエディタに適した位相配線のデータ構造と修正アルゴリズム

村田 洋*‡　梶谷 洋司*†

*北陸先端科学技術大学院大学 情報科学研究科
†東京工業大学 電気電子工学科
‡株式会社 村田製作所

単層アナログレイアウトの設計ツールとして，設計者が指定するひとつの部品の連続移動に対応して周囲の配線を連続移動させるレイアウト連続移動エディタが求められているが，その有効性は処理の早さにかかっている．筆者らは，連続移動を位相的に処理する段階 (位相配線処理) とそれに続く配線の物理的実現の段階 (物理配線処理) とに分けることで高速化をはかるエディタを開発中である．本稿では，全体の概要と位相配線処理部分について記述する．部品の配置に応じて領域を三角形分割し，位相配線はそれが通過する三角形の系列で表現する，というデータ構造において部品の連続移動を多角形の変形の連続で捉え，それぞれはデータ構造の部分変更で対応するワームクリープ法を提案する．実験のために仮の物理配線処理を組み込んでエディタを試作したところ，実時間エディタとして設計者が満足できる早さで動作させることができた．

A Topological Wire Data Structure and its Modification Algorithm for Analog Layout Editor

Hiroshi Murata*‡ and Yoji KAJITANI*†

*School of Information Science, Japan Advanced Institute of Science and Technology, East
†Department of Electrical and Electronic Engineering, Tokyo Institute of Technology
‡Murata MFG. CO., LTD.

As an interactive design tool for analog circuit layout, a layout editor is useful if wires are transformed continuously with a continuous move of a module which is instructed by a designer. To realize this function with a quick response time , the editor we are developing divides the continuous move into topological move phase (topological wiring process) and physical realization phase (physical wiring process). In this paper, after the overview of the whole system, topological phase is studied. Using the triangulation data structure of the wiring space, topological wires are expressed as a sequence of passing triangles. The proposed algorithm named *Worm Creep Method* moves one polygonal object by a sequence of local data modifications which is a polygon deformation with surrounding space. With a test version of physical phase algorithm, a prototype editor is developed and demonstrated to response quick enough for interactive use.

1 はじめに

プリント配線基板やハイブリッドICなどのアナログレイアウトは，比較的回路規模が小さいので人手で設計することも可能である．しかし，短期間に多品種を設計するために，設計者の作業効率を高めることのできる対話型のCADツールが求められている．そこで，次のようなレイアウトエディタを開発することにした．

単層上に配置配線された平面回路のレイアウトを考える．レイアウトは部品と配線からなる．部品は半田づけのための場所を兼ねた凸多角形状の端子の組，配線は2つの端子を結ぶ幅を持つ連続曲線(帯)である．画面は常時現在のレイアウトを表示しており，設計者はこれに基づいてマウスなど適当な入力装置を使ってひとつの部品の位置座標の修正をエディタに指令する．エディタは部品の位置修正とともに周囲の配線を適切に(デザインルールを守って)修正して次の段階のレイアウトを表示する．設計ツールとして使うときには設計者は適当な初期レイアウトをまず入力し，前述の操作を繰り返して最終レイアウトを得る．

そのようなエディタにはひとつの部品の移動後の位置だけに注意してあらためて最適配線経路を求め直す「再配線エディタ」と，ひとつの部品の連続移動に伴って配線が連続移動するように見える「連続移動エディタ」とが考えられる．人は，現レイアウトに基づいてその改善をはかるものであり彼の思考を中断させないのが良いツールであろう．そこで後者を開発の目標とした．人手で扱うエディタとして待ち時間を感じさせない高速性を実現することが性能上の主な目標となる．ディジタル計算機でそのまま実現するには，十分小さい刻みで繰り返し計算しなおすことで疑似連続性を保ちながら移動中のレイアウトを求めることになり，大量の計算のためにとても実用的なエディタは得られない．

高速処理のためにはデータの抽象化が必要である．アナログレイアウトの処理では連続曲線を扱うために膨大な座標データを処理しなければならないように思われるが，曲線そのものはデザインルールを満たすものとして周辺状況から推測できるものであるから常に保持していなければならないものではない．このことに注目して，連続移動の問題を，

- レイアウトの位相的な情報(端子配置と位相配線)を表現したデータ構造上で物体を連続に移動する過程(位相配線処理)と，
- 移動が完了した時点で位相配線を物理配線に変換する過程(物理配線処理)

に分割する．

位相配線のデータ構造はLeiserson,Maley[1]以来，平面の三角形分割[2]が直接あるいは間接に用いられてきた．但し，端子を点とみなせるLSIのレイアウトを前提としたものが一般的で，アナログレイアウトで本質的に必要な面積を持つ端子形状に対応した位相配線のデータ構造は見当たらない．端子形状の限定に問題はあるが，[3]の2次元コンパクションは端子移動を扱っている点で興味深い．しかしこれは，処理速度よりも基板面積の最小化をはかるもので対話型ツールの実現を目指していない．

本文では，まず2章で連続移動の問題を定式化するとともにこれを位相配線処理と物理配線処理に分割する．ついで3章で両処理の基礎となるレイアウトの位相情報を表現するデータ構造を導入し，4章でワームクリープ法とよぶ位相配線処理のアルゴリズムを与える．このアルゴリズムは，単位移動として，位相配線を伴ったひとつの端子の連続移動を実現する．

物理配線処理のアルゴリズムは現在研究中であり本稿には含まれない．しかし，アナログレイアウトエディタの一部として位相配線処理の性能を試すには物理配線処理が必要である．そこでそのための簡便アルゴリズムを試作してエディタのプロトタイプを完成した．5章ではこのプロトタイプについて報告する．同様の機能を目指すエディタとして性能を比較する対象はないが，対話型ツールとして満足できる速度で動作した．

2 エディタの機能と構成

2.1 対象とするレイアウト

レイアウトの図形はアパーチャとよばれる光の円盤を，感光フィルム上で移動して製作される．従ってレイアウトは円盤の軌跡図形である．本稿では円盤の中心点の軌跡に注目して，レイアウトを次のようにモデル化する．

レイアウトは端子と配線という2種類の素子からなる．端子は頂点数1以上の凸多角形，配線は自己交差のない有限長連続曲線である．端子と配線を単層の基板上に配置配線した平面回路を考える．回路として，端子と配線には接続という関係が指定されている．端子Tと配線Wが接続しているとすれば，Wの一方の端点とTの外周上の1点が重なり，かつ，WとTが重なりを持つのはその点だけである．Tの外周上のどこで接続されても等価である．レイアウト$\mathcal{L}$は次のレイアウトの条件を満たす端子と配線の集合である．

- すべての配線は異なる2つの端子と接続している．
- 接続関係を除けば2つの素子が重なることはない．

以上のようにアパーチャ中心点の軌跡で定義したレイアウトをスケルトンモデルという．レイアウトの実図形はスケルトンモデルに適当なアパーチャ直径を与えて作成する．図1に例を示す．

2.2 エディタの機能

端子の連続移動の概念を，それを主体的に起動するエディタの観点から把握するには，時間と関連づけて説明するのが適当である．時刻τにおけるレイアウトを$\mathcal{L}(\tau)$とする．$\mathcal{L}(\tau_1)$におけるひとつの端子Tに着目する．移動指令Mは，

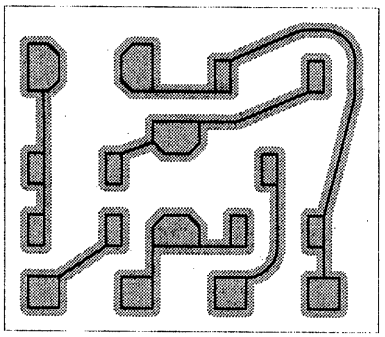

図 1: レイアウト

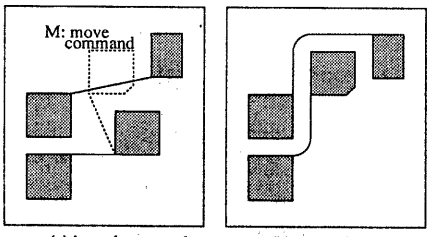

図 2: 自由領域単端子連続移動

$\tau \in [\tau_1, \tau_2]$ の間に T がその位置を連続的に移動する経路として与えられる．具体的には本開発エディタでは T の 1 頂点が $\tau \in [\tau_1, \tau_2]$ に対して描く軌跡を折れ線で指定したデータと，$\tau = \tau_2$ における T の凸多角形 $T(\tau_2)$ を指定したデータが与えられる．$T(\tau_1)$ と $T(\tau_2)$ は形が異なってもかまわない．但し，M は $\mathcal{L}(\tau_1)$ において位置が確定している他の端子と衝突しないものとする．これを自由領域移動という．配線は横断してもかまわない．図 2(a) に $\mathcal{L}(\tau_1)$ と M の例を示す．このような T の連続移動に対して $\mathcal{L}(\tau_1)$ にあったすべての配線は，任意の時刻 $\tau \in [\tau_1, \tau_2]$ において，前述のレイアウトの条件を満足していなければならない．そして，$\tau = \tau_2$ において $\mathcal{L}(\tau_2)$ が確定していなければならない．図 2(a) の入力に対して以上の条件を満たすように配線が連続移動するならば，τ_2 において例えば図 2(b) に示す $\mathcal{L}(\tau_2)$ が得られるであろう．

このような，

手続き

入力: $\mathcal{L}(\tau_1)$, T の移動指令 M
出力: $\mathcal{L}(\tau_2)$

を，**自由領域単端子連続移動** (free area single terminal move) という．これをエディタの機能として実現することが目標である．

自由領域単端子連続移動は，「端子の移動を与えれば，その端子は周囲の配線を押し退けるようにして移動する」という直観に合致するので設計者にとって極めて把握しやすい概念である．そこで，移動の経過は表示されないとしてもエディタが表示するレイアウト $\mathcal{L}(\tau_2)$ は連続に移動した自然な結果として設計者に受け入れられ，彼は思考を中断されることなく新しい指令をエディタに与えることができる．

自由領域移動に限ったのは移動経路が他の端子 T' と衝突すれば，T' の移動を T の移動指令に従属的に含ませなければならず，人手がもっとも不得手とする階層的指令 (2 手先読み) を必要とするからである．(逆にいえば，設計者はそのような必要性を感じないであろう．おそらく自由領域移動を繰り返すことによって対応するであろう．)

単端子移動に限ったのは複数の端子それぞれの移動を独立に考える必要性が認められないことと問題の簡単化のためである．しかし，一般に部品は複数の端子によって支えられており，それらは相対的に一定の位置を保って移動しなければならない．単端子移動の繰り返しではかなり高度技術を要求される．しかし，本稿には含まれないが筆者らは簡単なコマンドの追加でこれに対応できると考えており，開発プロジェクトに含めている．

2.3 エディタの構成

データの抽象化をはかるために位相配線の概念を導入する．レイアウト中の端子を固定して配線のみを連続移動した時実現が可能なレイアウトのすべてを同一視して抽象化したレイアウトの表現を位相レイアウト (あるいはレイアウトの位相) といい，位相が等しいレイアウトの要素として対応する配線のすべてを同一視して抽象化した配線の表現を位相配線 (あるいは配線の位相) という [1]．位相配線と対比的にひとつの曲線を明示した普通の配線の表現を物理配線といい，端子と物理配線を指定した普通のレイアウトの表現を物理レイアウトということがある．時刻 τ の物理レイアウトを $\mathcal{L}(\tau)$，位相レイアウトを $\dot{\mathcal{L}}(\tau)$ と表記する．

自由領域単端子連続移動を次のように位相的に取り扱う．まず，時刻 τ_1 において $\mathcal{L}(\tau_1), \dot{\mathcal{L}}(\tau_1)$ が得られているとする．位相レイアウトのデータ構造と作成方法が問題になるが，詳細は 3 章で述べる．そして $\mathcal{L}(\tau_2), \dot{\mathcal{L}}(\tau_2)$ を求める処理を 2 つの手続きに分けて順に実行する．

手続き 1: 位相配線処理

入力: $\dot{\mathcal{L}}(\tau_1)$, T の移動指令 M
出力: $\dot{\mathcal{L}}(\tau_2)$

位相配線処理では τ_1 から τ_2 にかけて物理配線が移動したときに得られる位相配線を，物理配線を移動しないで求める．これを位相配線を移動するという．

手続き 2: 物理配線処理

入力: $\dot{\mathcal{L}}(\tau_2)$
出力: $\mathcal{L}(\tau_2)$

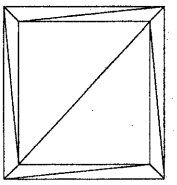

図 3: 空のレイアウトの TTILE

物理配線処理ではτ_2において位相配線を物理配線に変換する．そのアルゴリズムの開発は今後の課題である．

3 位相レイアウトのデータ構造

位相レイアウトのデータ構造は設計開始時に設計者が初期レイアウトとして入力した物理レイアウトから変換して1度だけ作成する．(以後は位相配線処理によって保守される．) 以下，位相レイアウトのデータ構造と作成方法を，端子凸多角形の配置を表す部分と位相配線を表す部分に分けて述べる．

3.1 端子配置のデータ構造と基本操作

端子配置は平面の三角形分割に基づいて表現される．平面の三角形分割とは，平面上の点集合に対してそれらの点で決まる凸包内の領域がすべて三角形になるようにすべての点を交差しない辺で結んだものである [2]．辺で囲まれた各三角形領域をタイルとしてデータ化する．各タイルの情報は，頂点データへの3つのポインタと，辺で隣接する3つのタイルへのポインタを含む．このデータ構造を**三角形タイルデータ構造** (Triangle TILE data structure,**TTILE**) と呼ぶ．

端子配置を TTILE で表現するには，まず空のレイアウトに対応した TTILE を作成する．すなわち，レイアウトを囲む十分大きな矩形と，それより大きな矩形の4角に合計8個のダミー点を配置して図3のような TTILE を準備する．

内側の4点はすべての配線を凸包の中に含める目的で導入するものである．外側の4点は探索手続きが注目する可能性のあるタイルが必ず3つのタイルに隣接することを保証して探索手続きを簡単にする目的で導入するものである．ダミー点は TTILE の要素としては点形状の端子と同等に扱われるが，レイアウトの要素として扱われることはない．

一旦 TTILE が得られれば，以下に定義する3つの基本修正手続きを利用できる．それぞれの動作例を図4に示す．

TTILE 点挿入手続き (点 p){
  /* 新しい点 p を TTILE に追加する．*/
  点 p をその上にもつ △ abc を探索する;
  △ abc を削除する;
  △ abp, △ bcp, △ cap を作成する;
}

TTILE 辺挿入手続き (点 p, 点 q){
  /* 挿入済みの2点 p,qの間に辺を実現する．*/

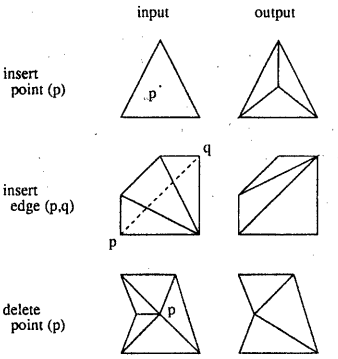

図 4: TTILE 基本修正手続き

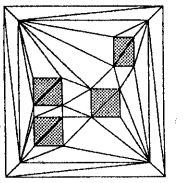

図 5: 三角形タイルデータ構造,TTILE

  線分$\overline{pq}$が内部を通過するタイルを連結した多角形の外周を $V = \{r_1, r_2, \ldots, r_n\}$ とする;
  Vに含まれるタイルを消去する;
  while(Vの頂点数 > 2){
    Vの頂点 r_k ($\neq p, q$) を，
      $\{r_{k-1}, r_k, r_{k+1}\}$ が左回りになる点とする;
    △ r_{k-1} r_k r_{k+1}を作成する;
    Vから頂点 r_kを削除する;
  }
}

TTILE 点削除手続き (点 p){
  /* 挿入済みの点 p を削除する */
  点 p を1頂点として持つタイルを連結した多角形の外周を $V = \{r_1, r_2, \ldots, r_n\}$ とする;
  while(Vの頂点数> 3){
    Vの頂点 r_kを四角形 $\{r_{k-1}, r_k, r_{k+1}, p\}$ が凸であるように選ぶ;
    △ r_{k-1} r_k p, △ r_k r_{k+1} p を削除する;
    △ p r_{k-1} r_{k+1}, △ r_{k-1} r_k r_{k+1}を作成する;
    Vから頂点 r_kを削除する;
  }
  △ r_1 r_2 p, △ r_2 r_3 p, △ r_3 r_1 p を削除する;
  △ r_1 r_2 r_3を作成する;
}

上の手続きは，例えば点挿入手続きにおいて新しい点が既存のタイルの辺上にある場合など，特殊な場合を繁雑になるので省いた．しかしこれに対処するのは容易である．

目的の TTILE はレイアウトのすべての端子を空の TTILE に挿入することによって得られる．端子をひとつ挿入するには，その端子凸多角形の各頂点と各辺に対して点挿入と辺

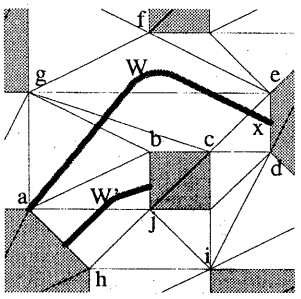

図 6: TTILE 上の物理配線

挿入を実行する．例を図5に示す．図のように，端子凸多角形の外周はかならずタイルの辺として実現される．

以後の参照のためにタイルを次のように分類する．タイルの辺を，配線がそれを通り抜けることができる自由辺と，それ以外の非自由辺に分類する．タイルを自由辺の数によって 0-free タイル，2-free タイル，3-free タイルの三種に分類する．0-free タイルは端子上にあり端子タイルとも呼ばれる．2-free タイルと 3-free タイルは自由領域にあり総称して自由タイルとも呼ばれる．これらの分類はタイルからポインタで指される頂点データにその頂点が所属する端子の識別子を含めることにより容易に判断できる．例えば，3 つの頂点がすべて異なる端子の点であるようなタイルは 3-free タイルである．タイルを三角形の頂点で参照するときは，頂点を左回りにならべて $\triangle abc$ のように書く．タイルに限らず多角形を参照するときにも頂点を左回りに並べて多角形 $\{abcd\}$ のように書く．

3.2 位相配線のデータ構造

領域は三角形のタイルに分割された．以下図6に示す例を参照しながら位相配線のデータ構造を導入する．

まず配線に沿ってそれが内部を通過する自由タイルを追跡し，それぞれの通過するタイルについて突入辺 (頂点) と脱出辺 (頂点) の組を記録する．例えば図 6の配線 W を接続点 a の側から追跡すると，タイル $\triangle abg$に頂点 a から突入し辺$\overline{bg}$から脱出するので $(\triangle abg, a, \overline{bg})$ なるデータが最初に得られる．このデータはタイルの内部領域を通過する際の詳細については述べず，タイルの外周と配線の交点がどの辺 (点) の上にあるかという位相的な情報のみを保持する．そこでこのデータを**単タイル位相配線** (Topological Wire on Single tile, **TWS**) とよぶ．配線 W 全体では，

$$\{(\triangle abg, a, \overline{bg}), (\triangle gbc, \overline{gb}, \overline{cg}), (\triangle gce, \overline{gc}, \overline{eg}),$$
$$(\triangle gef, \overline{ge}, \overline{ge}), (\triangle gce, \overline{eg}, \overline{ce}), (\triangle cde, \overline{ec}, \overline{de})\}$$

という TWS 列が得られる．自由辺に沿って 2 つの端子を直線的に結ぶ物理配線の場合，便宜的に，隣のタイルの内部を通過すると見なす．また，配線を追跡する方向によって列は逆順にもなり得るが，逆順の TWS 列は同一視する．それでもなお，位相が等しい別の物理配線では異なる TWS 列が得られるだろう．それでは配線の位相を一意に表すデータとして不都合である．そこで，既約化 (reduction) とよぶ以下の処理を行なう．

配線を仮想的に連続移動して，TWS 列の長さを最短化する．配線の途中部分を移動する場合と，端子との接続部分を移動する場合がある．例えば，図 6の Wはその途中部分を移動して $\triangle gef$を通らないようにすることができる．また，端子との接続部分 x を端子外周上でスライドさせて $\triangle cde$ を通らないようにすることができる．これら 2 種類の最短化をそれ以上適用できなくなるまで繰り返すと，最短な TWS 列が得られる．図 6の Wでは

$$\{(\triangle abg, a, \overline{bg}), (\triangle gbc, \overline{gb}, \overline{cg}), (\triangle gce, \overline{gc}, e)\}$$

という長さ 3 の TWS 列が得られる．Wと同位相の配線から出発すれば必ずこの TWS 列が得られる．

最短 TWS は，その長さが 2 以上の場合は，配線の位相に対して 1 : 1 に対応する．しかし，長さが 1 の場合にはこの限りでない．例えば図 6の配線 W'と同位相の配線から得られる最短 TWS 列は $\{(\triangle ahj, \overline{ah}, j)\}$，$\{(\triangle ajb, a, b)\}$ など多様である．最短化の結果得られる TWS 列の長さが 1 の場合は，TWS を 3-free タイル上に置くように配線を仮想移動すれば，最短 TWS 列は 2 つに限定される．例では $\{(\triangle abg, a, b)\}$ と $\{(\triangle hij, h, j)\}$ の 2 つである．この 2 つの TWS を，間に任意個の 2-free タイルを挟んで隣接した 2 つの 3-free タイルの対向する辺に沿った TWS として，同一視する．以上の既約化により，

- 3-free タイルの頂点から出発し，
- 自由タイルの 2 つの自由辺を結ぶ TWS を最少個数経由して，
- 3-free タイルの頂点で終了する，

という，**3-free タイル終端最短 TWS 列** (3-free tile terminated shortest TWS List, **TWSL**) で位相配線を一意に表す．TWSL を図示する時は，各 TWS をタイル上の矢印として記入して図 7のように表す．

面積を持つ端子形状 (凸多角形) に対応した位相配線のデータ構造が実現されたことは 2 つの意味で重要である．ひとつはアナログレイアウトの位相配線が表現できるようになったこと，もうひとつは次章で述べるワームクリープ法による端子移動が可能になったことである．(ワームクリープする端子は，たとえそれが点であっても，移動中線状に拡大するので，少なくとも直線分形状の端子に対応した位相配線のデータ構造を要求する．)

位相配線を効率よく参照するには TWSL を TTILE と共に記録することが必要である．この段階で，[3] では物体の隣接関係を表現する多くのポインタを導入している．一概に優劣をつけられることではないが，我々は「省略できるデー

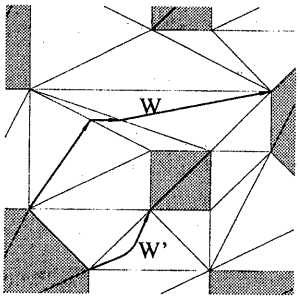

図 7: TTILE 上の位相配線

タは極力省略してその保守に必要な手続きを不要にする」という方針で次のように記録する．

TWSL 中の TWS をタイル内に記録する．但し，

1) TWSL をリストとして保持するポインタを導入しない．

2) タイル内で同居する他の TWS との隣接関係を保持しない．

3) 2-free タイル上の TWS は記録しない．

省略したデータは必要になった時点で周囲のデータから生成する．TWSL はリスト構造に依らずともタイルの隣接関係によって追跡でき，同居する配線との隣接関係が必要なときは TWSL を追跡すれば判明する．但し，1),2) の両方を実施したので，(大きく一回りして) 同じタイルの同じ辺を同じ方向から 2 度横断する位相配線は扱えない．3) については TWSL 上の前後の 3-free タイル上の TWS から復元できる．

4 位相配線処理

尺取り虫はまず線状の一端を支点としてその点に収縮し，次に支点を他方の端点にかえて線状に伸びるという変形を繰り返し，結果として移動している．このような物体の移動方法をワームクリープと呼ぶ．端子をワームクリープさせることで位相配線を伴って端子を移動する．

4.1 ワームクリープ法

次のような 2 種類の位相配線処理を考える．ひとつは時刻τ_1において凸多角形$T(\tau_1)$である端子Tが，ひとつの頂点の位置を保って収縮し，時刻τ_2においてその 1 点になる端子収縮処理である．もうひとつは逆に時刻τ_1において 1 点である端子Tが，その点を 1 頂点として持つ凸多角形であり続けながら拡大し，時刻τ_2において凸多角形$T(\tau_2)$になるという端子拡大処理である．**ワームクリープ法** (Worm Creep Method) とは，位相配線処理を端子収縮処理と端子拡大処理の繰り返しに帰着して実行する次に示すアルゴリズムである．

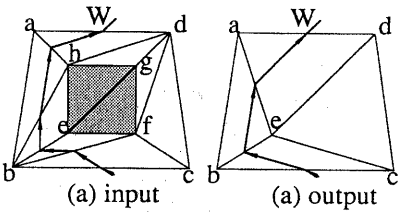

図 8: 端子収縮処理

ワームクリープ法 ($\dot{\mathcal{L}}(\tau_1)$, Tの移動指令 M){
/* M = (Tの 1 頂点の軌跡, $T(\tau_2)$ */
/* Tの 1 頂点の軌跡 = 折れ線 $\{p_1, p_2, \ldots, p_n\}$ */
/* $T(\tau_1) = p_1$を 1 頂点に持つ凸多角形 */
/* $T(\tau_2) = p_n$を 1 頂点に持つ凸多角形 */
Tの形状 $T(\tau_1)$ を点 p_1にする端子収縮処理;
for($i = 1; i < n; i$++){
Tの形状を線分$\overline{p_i\, p_{i+1}}$にする端子拡大処理;
Tの形状を点 p_{i+1}にする端子収縮処理;
}
Tの形状を凸多角形 $T(\tau_2)$ にする端子拡大処理;
}

4.2 端子の変形に伴う位相配線の移動

端子収縮処理と端子拡大処理は逆処理として良く似たアルゴリズムで実現できる．そこで収縮について詳細に述べ，拡大については要点だけを述べる．

端子収縮手続き ($\dot{\mathcal{L}}(\tau_1)$, Tの移動指令 M){
/* $T(\tau_1)$ = 凸多角形 $\{p_1, p_2, \ldots, p_n\}$ */
/* M = (Tの 1 頂点の軌跡, $T(\tau_2)$) */
/* Tの 1 頂点の軌跡 = $T(\tau_2)$ = $\{p_1\}$ */
1) $T(\tau_1)$ から点 p_1を除いた図形の包を Hとする;
2) $T(\tau_1)$ の外周を右回りに見た点列を Hの点列の p_1の位置に挿入して多角形外周 A を作る;
3) A 内の位相配線 (TWSL 部分列) を A との交点対に変換する;
4) A 内の TWS を消去する;
5) TTILE の点 $p_2, p_3, \ldots, p_n$を削除する．
6) A との交点対で表された位相配線を Hとの交点対に変換する;
7) Hとの交点対で表された位相配線を H内の TWSL 部分列に変換する;
}

図 8に端子収縮処理の例を示す．図 (a) は時刻τ_1の位相レイアウト$\dot{\mathcal{L}}(\tau_1)$である．$T(\tau_1)$ = 四角形 $\{efgh\}$ から $T(\tau_2)$ = 点 e への収縮に伴って，位相配線 Wが移動され，時刻τ_2の位相レイアウトが$\dot{\mathcal{L}}(\tau_2)$ のように求められる．以下，この例について各ステップの動作を調べる．

1) 図 (a) において，四角形 $\{efgh\}$ から点 e を除いた図形の包として H = 多角形外周 $\{ab\underline{e}bcd\}$ が求められる．ここで，図形の包とは，TTILE 上でその図形と一部でも重なりを持つタイルについて，その図形と一部でも重なる辺を取り去った時に現れる多角形外周であ

る．下線の点 e は収縮する点として手続き中の点 p_1 に対応する．

2) $H = \{ab\underline{e}bcd\}$ の下線の位置に $T(\tau_1)$ = 四角形 $\{efgh\}$ を右回りに見た点列 $\{hgfe\}$ を挿入して A = 多角形外周 $\{abehgfebcd\}$ が計算される．A は図 (a) において，自由タイルを連結した多角形領域の外周である．このような多角形外周が得られるのは，凸多角形 $T(\tau_1)$ が収縮する点を唯一の接点として包 H に内接するからである．

3) A の領域内の 2-free タイルについての TWS を復元してから，A の領域内の TWSL 部分列に沿って位相配線を追跡し，A との交点対を得る．交点は，それが頂点上にあればその頂点の外周番号を記録し，それが自由辺上にあればその辺の外周番号を記録する．ここで，外周番号とは A に現れる点と辺に対して，現れる順に振られる自然数である．但し，A 上で連続した同一端子の点と辺には同じ数が与えられる．図 (a) において $A = \{abe\underline{hgfe}bcd\}$ の下線部は同一端子上にあるから，各点と辺

$$a, \overline{ab}, b, \overline{be}, e, \overline{eh}, h, \overline{hg}, g, \overline{gf}, f, \overline{fe}, e, \overline{eb}, b, \overline{bc}, c, \overline{cd}, d, \overline{da}$$

のそれぞれに対して，

$$1, 2, 3, 4, 5, 5, 5, 5, 5, 5, 5, 5, 6, 7, 8, 9, 10, 11, 12$$

という外周番号が与えられる．配線 W の TWSL は A の内部に $\overline{bc}$ で突入し $\overline{eb}$ で脱出する部分と $\overline{be}$ で突入し $\overline{da}$ で脱出する部分に分けられて，$(8,6),(4,12)$ という外周番号で表された 2 つの交点対に変換される．

このように，A 内の位相配線を A との交点対で表すことができるのは，A がその内部に点を含まないからである

4) 図 (a) に記された 5 つの TWS がすべて消去される．

5) TTILE 基本修正手続きのひとつとして 3 章で定義した TTILE 点削除手続きにを利用して，点 f, g, h を削除する．各点を削除する時に，その点の包の内部が再分割されて図 (b) のような TTILE を得る．但し，この時点ではまだ図に記された TWS は記録されていない．

6) 図 (b) において，$H = \{abebcd\}$ は自由タイルを連結した多角形領域の外周になっており，かつ，その内部に点を含まない．この頂点と辺に 3) と同様に外周番号を振る．

$$a, \overline{ab}, b, \overline{be}, e, \overline{eb}, b, \overline{bc}, c, \overline{cd}, d, \overline{da}$$

のそれぞれに

$$1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12$$

という外周番号が与えられる．そして，3) で作成した W と A との交点対 $(8,6),(4,12)$ を W と H との交点対とみなす．この「みなし」処理が端子収縮にともなう位相配線移動の中枢である．その正当性は次のように証明できる，

[証明] A の $T(\tau_1)$ 上のすべての辺が $\tau \in [\tau_1, \tau_2]$ に対して収縮し，$\tau = \tau_2$ において H に一致するとき，端子 T は凸多角形から指定の点に収縮する．従って，A と H は端子収縮により変形する領域の変形前後の外周として位相的に対応する．A の外周番号は収縮して 1 点になる部分に同じ番号が振られているので，A と H のそれぞれ位相的に対応する部分は同じ外周番号を持つ．従って W と A の交点対を W と H の交点対と見なすことで，τ_1 に A 内にあった位相配線は端子収縮に伴って移動される．5) は T 上の端子タイルを除く A の外部のタイルを修正しないので，A の外部の位相配線は移動する必要がない．(証明終り)

7) H との交点 $(8,6),(4,12)$ で表された位相配線を図 (b) に記した TWS 部分列に変換する．これは，次のように簡単に行なえる．図 (b) の H 内の 3-free タイルのすべてについて，その 3 辺と 3 頂点の各々と 6) で得られた位相配線との交差判定を行ない，結果によってそのタイル上の TWS を作成する．タイル頂点と位相配線の交差判定は外周番号の一致を調べればよい．タイル辺と位相配線の交差判定は，外周番号の最大値を m として，正 m 角形の 2 つの対角線の交差判定の問題に帰着できる．例えば，図 (b) の $\triangle abe$ の辺 $\overline{be}$ について調べるならば，これを H の外周上の辺に対応づけて辺の両端の点の外周番号 $(3,5)$ を得る．そして，位相配線 $(4,12)$ との交差判定するときは，正 12 角形の 2 つの対角線 $(3,5)$ と $(4,12)$ の交差判定に帰着し，「交差する」と判断する．

次に端子拡大処理を考える．収縮の場合とは逆に，拡大後の凸多角形 $T(\tau_2)$ を基に包 H と多角形外周 A を作り，H と A との役割を逆にして次のように計算すれば良い．

端子拡大手続き ($\mathcal{L}(\tau_1), T$ の移動指令 M){
/* $T(\tau_1)$ = $\{p_1\}$ */
/* M = (Tの 1 頂点の軌跡, $T(\tau_2)$) */
/* Tの 1 頂点の軌跡 = $\{p_1\}$ */
/* $T(\tau_2)$ = $\{p_1, p_2, \ldots, p_n\}$ */
1) $T(\tau_2)$ から点 p_1 を除いた図形の包を H とする;
2) $T(\tau_2)$ の外周を右回りに見た点列を H の点列の p_1 の位置に挿入して多角形外周 A を作る;
3) H 内の位相配線 (TWSL 部分列) を H との交点対に変換する;
4) H 内の TWS を消去する;
5) TTILE に点 $p_2, p_3, \ldots, p_n$ と，辺 $\overline{p_1p_2}$, $\overline{p_2\ p_3}, \ldots, \overline{p_{n-1}p_n}, \overline{p_np_1}$ を挿入する;
6) H との交点対で表された位相配線を A との交点対に変換する;
7) A との交点対で表された位相配線を A 内の TWSL 部分列に変換する;
}

ワームクリープ法の全体では，$T(\tau_1)$ と移動指令 M を合わせた図形の包の外部のタイルは修正されないことになる．そこでタイル修正の手間はレイアウトのサイズ(例えばタイル数)によらず局所的な包のサイズだけに依存する．一方タイルの探索にはこのような局所性がないが，最後に参照されたタイルを記録するなどの方法によって，実用的には，ほとんどの場合局所的な探索で済むであろう．このような局所性は，レイアウトのサイズに依らず一定の時間で応答することが必要なエディタの内部処理としてワームクリープ法が適当であることを意味している．

5 レイアウトエディタの試作

実験のため，以上の結果を組み込んでエディタのプロトタイプ CALTETT(Continious Analog Layout Transformation Editor over Triangle Tiles) を開発した．(C 言語と Sun Sparcstation II(25MIPS) を使用．)

エディタとして動作させるには物理配線処理と部品機能(端子の組を一括して扱う機能)が必要であるのでそのための仮のアルゴリズムを試作して組み込んだ．操作者はマウスを用いて次のコマンドを発行できる．

1) 部品入力/消去コマンド

2) 配線入力/消去コマンド

3) 部品移動コマンド

4) 部品回転コマンド

1),2) は初期レイアウトの入力に利用される．3) 部品移動コマンドは部品を並行移動させるように各端子について移動指令を生成する．4) 部品回転コマンドは部品を(それを取り囲む長方形の中心を回転の中心として)時計回り/反時計回りに回転させるように各端子について移動指令を生成する．

このプログラムを用いて図 9 の初期レイアウトから図 10 の最終レイアウトを得るデモンストレーションを行なった[4]．このデモでは，部品移動コマンドを十数回，部品回転コマンドを 1 回発行する．応答時間はほとんど意識できぬ短さで完全に満足できるものであった．

6 むすび

単層アナログレイアウトの対話型設計ツールとして部品が配線を押し退けて移動するようにみえる「連続移動エディタ」を高速に実現することを目的とし，まず凸多角形の端子を扱える位相配線のデータ構造を導入した．そして単位移動のアルゴリズムとして，位相配線を伴ったひとつの端子の連続移動を実現するワームクリープ法を提案した．実験のため試作したエディタは対話型ツールとして満足できる速度で動作した．

今後はまず，物理配線処理を実現してエディタの完成を目指したい．次に配置配線済みのレイアウトの配置最適化，

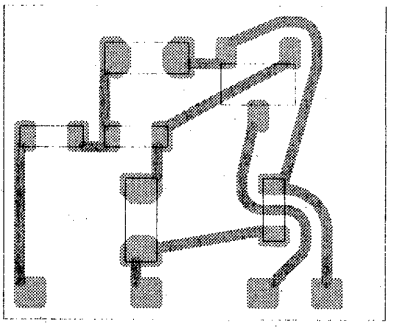

図 9: 初期レイアウト

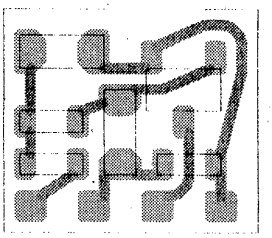

図 10: 最終レイアウト

さらには配置配線の一括設計に展開できればよいと考えている．

謝辞

本研究の機会を与えてくださった(株)村田製作所 ファンクショナルデバイス事業部 ハイブリッド商品統括部 枝 茂男 部長に感謝致します．

参考文献

[1] C.E.Leiserson and F.M.Maley. Algorithms for routing and testing routability of planer VLSI layouts. In *Proc. 17th Ann. ACM Symp. on Theory of Computing*, pages 69–78, 1985.

[2] F.P.Preparata and M.I.Shamos. 計算幾何学入門. 総研出版, 1992.

[3] J.Valainis, S.Kaptanoglu, E.Liu, and R.Suaya. Two-dimensional IC layout compaction based on topological design rule checking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, CAD-9(3):260–275, March 1990.

[4] 村田 梶谷. CALTETT: アナログレイアウトエディタ. 電子材料, Vol.32(1):142–145, January 1993.