7S-2

会話型ＶＬＳＩレイアウト検証のためのアルゴリズムとデザインルール記述法

枝廣　正人

日本電気株式会社　Ｃ＆Ｃシステム研究所　応用システム研究部

1．はじめに

近年ＬＳＩを記述するアートワーク図形数の増大は著しく、レイアウト検証においても設計手法に様々な変化が生じてきた。従来は修正の度に大型計算機上でチップ全体の検証を行なっていたが、最近ではレイアウトエディタ中の会話型検証ツールが用いられるようになり、最終的な検証にのみチップ全体の検証が行なわれるようになりつつある。そのような設計手法において設計者の様々な要求を満足するためには、会話型レイアウト検証はいくつかの条件を満足する必要がある。まず第一に、インクリメンタルな処理（１つのデータのまわりのチェック）が行なえる必要があり、さらにバッチ的に画面全体、モジュール全体についてチェックが行なえる必要がある。また、レイアウトエディタと同じデータ構造であることが望ましい。

現在ワークステーション上には多くの会話型検証システムが開発されているが、これらの条件をすべて満足するようなシステムは提案されていない。本論文では、バケット法を応用した会話型検証のためのＤＲＣアルゴリズムを提案する。また、検証内容（デザインルール）の記述について、チェックの変更を容易にするような新しい記述法を提案する。

2．ＤＲＣアルゴリズム

（１）従来手法

現在会話型検証のアルゴリズムとして、平面走査法とバケット法が知られている。平面走査法[5]とは、図形の頂点または交点を通過する水平線で領域を分割し、下から上に領域をたどりながら処理を行なう方法である（図１）。この方法はデータ全体をソートされたリストとして持つので、そのデータ構造のままでは部分的なデータのみをとりだすことは難しく、インクリメンタルな処理に用いることは難しい。また、大きいレイアウトに対しては領域を分割する水平線が長くなるので幅広い範囲のデータを扱わなければならなくなり、効率が悪くなる場合がある（図２）。そのため、図形の数をｎとするとき、処理時間は$O(n^{1.2-1.5})$程度であった。

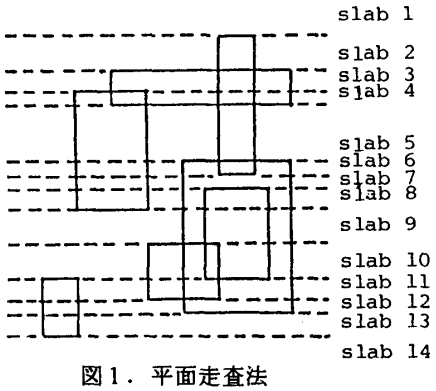

図１．平面走査法

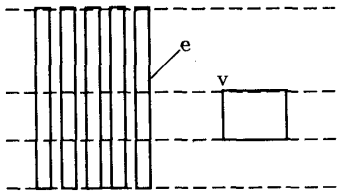

図２．平面走査法において効率が悪くなる場合

これに対してバケット法[1]は図形をメッシュに分割する手法である（図３）。インクリメンタルなレイアウト検証に非常に有効であり、エディタのデータ構造としても優れている[3]。しかしながら[3]の手法はバッチ的に用いると、同じ図形が何回も探索される場合があり、処理時間が線形時間にならない。

（２）提案する手法

本論文で提案する手法では、バケット法を基本とするデータ構造を用い、平面走査法のアルゴリズムを用いる。指定された領域内でのバッチ的な処理アルゴリズムは次の通りである。

Step 1 領域をバケットに分割し、図形をバケットにふりわける。

Step 2 指定された領域と重なるバケットを調べることにより、領域と重なりを持つすべての図形をとりだす。

Step 3 とりだされたすべての図形の頂点をｙ方向にソートする。

Step 4 すべての頂点を下から上にたどりながら演算を行なう。

Step 4における演算には平面走査法を基本にしたアルゴリズム[5]を用いる。従来の平面走査法において最も時間のかかっていた部分は、Step 3のソートおよびStep 4においての頂点のすぐ左側にある辺（図２の頂点ｖに対する辺ｅ）の探索であるが、これらはバケット手法を基本としたデータ構造を持つことにより平均的に線形時間で処理することができる。従って全体として平均的に$O(n)$で処理を行なうことができる。

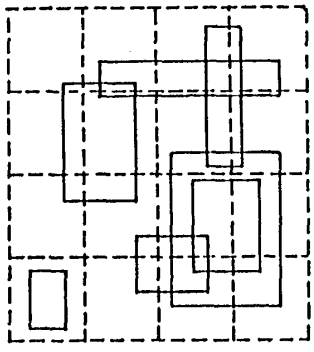

図３．バケット法

An Interactive DRC Algorithm and a Design-Rule Representation Method for VLSI Layout Verification

Masato EDAHIRO
NEC Corporation

指定領域を、与えられた図形のまわりとすることにより、インクリメンタルな検証も同様に処理できる。提案するアルゴリズムは、指定領域と重なりを持つすべての図形の頂点に対しStep 4の処理を行うため、従来のインクリメンタルアルゴリズム[3]よりも処理量は多くなるが、計算複雑度としては同じである。

このように提案するＤＲＣアルゴリズムは、バッチ、インクリメンタルの両方の処理を効率よく行なうことができる。

3．デザインルール記述法

（1）従来手法

デザインルールはＤＲＣを行なう際の処理内容を記述したものであるが、その記述法については従来あまり議論されることがなかった。図4は従来の記述法の代表例である。現実には記述は１００行を超えることもある。

このデザインルールには①使用するプロセスに特有のチェック項目、および演算内容と順序、②チェック寸法、③チェック種類、④入出力層番号、の変更がおこるが、これらの4種類は変更の頻度により2種類に分類できる。①に関してはプロセス（例：nウェル方式のＣＭＯＳ）に変更のない限り変更されることはない。それに対し②、③、④は頻繁に変更される。特に会話型ＤＲＣでは起動されるごとに変更されることも考えられる。

このような観点から従来の記述法を考えると、どの変更に対してもデザインルールの記述全体に変更の範囲が及ぶことがわかる。従って従来の記述法は、設計者にとって変更を行ないにくい方法であるといえる。

（2）提案する記述法

提案する記述の例を図5に示す。記述の種類は3通りである。

(a) 層名 = 層番号 [ﾒｯｾｰｼﾞ];
(b) 層番号 = 層名 (ﾁｪｯｸ寸法) [ﾒｯｾｰｼﾞ];
(C) 層名 = 層名 演算名 層名;

(a)は入力層定義、(b)はチェック種類、寸法を含む出力層定義、(c)は演算内容と順序の定義である。

この記述法にはいくつかの特長がある。まず①については(c)の中に記述され、②、③、④については(a)と(b)の中に記述されている。このように完全に切り分けることにより②、③、④の変更は記述全体を変更する必要はなく、(a)、(b)の変更のみですむ。

さらに、(b)に記述された出力層名から逆にたどることにより、必要な演算を(c)からとりだすことができる。例えば、図6の(a)、(b)の記述より(c)に記述された演算のうち下線の演算のみを行なえばよいことがわかる。

従って、使用するプロセスで考えられるすべての演算内容と順序をあらかじめ(c)の中に記述した場合、プロセスに変更のない限り(c)の記述には変更の必要がなく、頻度の高い②、③、④の変更は(a)と(b)の記述の変更のみでよい。しかも無駄な演算を行なうこともない。

（3）評価結果

この記述法は現在レイアウトエディタ[4]に組込まれて使用されている。機能として次の3種類が用意されている。

- デザインルールをコンパイルする（文法チェックおよびバイナリファイルへの書き出し）
- ＤＲＣ時にバイナリファイルを読み込む
- (c)の記述を基に、演算内容を会話的に選択する。

１００行の記述に対し、コンパイルは2～3秒程度、読み込みは1秒以内の時間で実行できる(EWS4800/10)。

4．まとめ

バケット法を応用した会話型ＤＲＣ向きのアルゴリズムを提案した。この方法はバッチ、インクリメンタルの両方の処理に適用可能である。また、デザインルールのための新しい記述法を提案した。この記述法によれば、プロセスが変更されない限りチェック内容の変更を容易に行なうことができる。本アルゴリズムおよびデザインルール記述法は、EWS4800上に開発されたレイアウトエディタシステムにて評価中である。

バケット法は階層処理にも適した方法であり、今後はレイアウト階層を利用し、より高速な会話型レイアウト検証システムを開発する予定である。

謝辞

日頃御指導頂いている日本電気㈱後藤部長、吉村課長、藤田主任、藤岡主任、石川主任に深く感謝します。

参考文献

[1]枝廣：ＬＳＩパターン設計におけるバケット分割手法とアルゴリズム．第33回情処全大 5R-3, 1986.
[2]枝廣：自動／会話型パターン設計手法．第34回情処全大 6F-7, 1987.
[3]枝廣：会話型ＶＬＳＩレイアウト検証のための一手法．第36回情処全大 4Y-2, 1988.
[4]藤岡、藤田、石川、枝廣：PROCEED-LED ＬＳＩレイアウトエディタ．第37回情処全大 5U-1, 1988.
[5]川西、西出、増田：ＬＳＩマスクパターンの図形処理の一算法．信学論，J63-A, pp.611-618, 1970.

```
poly = 2
diff = 4
al = 6
gate = poly AND diff
INT gate 1.2 OUTPUT 51
INT al 1.0 OUTPUT 52
EXT al 1.0 OUTPUT 53
cap = poly SUB diff
INT cap 1.2 OUTPUT 54
```

図4．従来の記述法

```
*(a)
poly = 2 [polysilicon];
diff = 4 [diffusion];
al = 6 [metal];
*(b)
51 = gate_width (1.2) [gate width];
52 = al_width (1.2) [metal width];
53 = al_space (1.0) [metal space];
54 = cap_mar (1.2) [cap_mar];
*(c)
gate = poly AND diff;
gate_width = INT gate;
al_width = INT al;
al_space = EXT al;
cap = poly SUB diff;
cap_mar = INT cap;
```

図5．提案する記述法（＊はコメント行）

```
*(a)
al = 6 [metal];
*(b)
52 = al_width (1.2) [metal width];
53 = al_space (1.0) [metal space];
*(c)
gate = poly AND diff;
gate_width = INT gate;
al_width = INT al;
al_space = EXT al;
cap = poly SUB diff;
cap_mar = INT cap;
```

図6．チェック種類の選択