設計自動化 44－4
（1988. 10. 28）

論理接続照合の一手法

田中 節子　　井上 雅夫　　羽山 繁
松下電器産業(株) 半導体研究センター

Connection Condition Flag (CCF) を利用した論理接続検証の発見的手法について報告する。グラフ同形判定を利用した論理接続照合手法においては、グラフの頂点及び枝を、対象とするグラフと1対1に対応づける際の効率が問題となる。**CCF**は固有化された頂点からの距離と頂点の固有名を表現し、これを頂点のラベルとして使用することにより、論理回路の接続関係をすべて簡潔に表現することができる。我々は、**CCF**を導入することにより対応付けとエラー箇所の限定が容易に行える発見的手法を開発した。さらに本手法では、相互関係の不明確で重複したエラー情報を分類統合することにより、的確なエラー出力を行うことも可能である。実験結果についても報告する。

AN ALGORITHM FOR LOGIC INTERCONNECTION VERIFICATION

Setsuko Tanaka,　　Masao Inoue　and　Shigeru Hayama

Semiconductor Research Center, Matsushita Electric Industrial Co., Ltd.
3-15 Yagumonakamachi, Moriguchi, Osaka, 570 JAPAN

A new heuristic algorithm for logic interconnection verification, which uses connection condition flag (**CCF**), is described. In logic interconnection verification using graph-isomorphism, it is important how efficiently vertices and edges can be binded. **CCF** expresses distances from binded vertices and properties of vertices. By using **CCF** as labels of vertices, all the interconnections can be expressed simply. We have developed the new algorithm which can bind and restricts error points easily. In addition, this algorithm can classify complicated error informations of which relations are not clear, and make them into a proper error report. Experimental results are also described.

1. はじめに

近年、多くのVLSI自動レイアウトツールが発表されているが、タイミングの問題を解決するため、あるいはチップ面積の最小化等のため、まだまだ人手によるレイアウト設計が必要とされている。

この様な人手によるレイアウト設計では、レイアウト検証が非常に重要であり、特に論理接続照合はレイアウト設計で論理設計通りの回路を実現しているかを検査するため、必要不可欠なものとされている。

論理接続照合の対象として、従来から、

(1) トランジスタレベル

(2) 論理の認識後のゲートレベル

(3) 階層的なセルレベル

の3つのレベルでの報告がされている。

また、その手法も様々なグラフ同形判定を用いた論理接続照合アルゴリズムが研究開発され、報告されている[1)~9)]。しかし、いずれの手法においても適用できない回路が存在したり、高速なエラー箇所の限定が難しいという課題があった。

我々の開発した手法は、グラフ同形判定を用い、トランジスタレベル、ゲートレベル、セルレベルのいずれのレベルにも適用でき、かつミックスレベルでの論理接続照合も可能とすることを特徴とする。

また、我々はグラフ同形判定の際に、判定対象のグラフの要素を一対一に対応付ける処理の効率が問題と考え、効率を向上するため、回路接続関係の表現をラベルによる対応付けを考慮した**CCF(Connection Condition Flag)**を用いた回路グラフ表現を提案する。本論理接続照合システムは**CCF**を用いて、高速で的確なグラフの対応付けと、エラー箇所の限定が容易なヒューリスティックアルゴリズムを開発し、実用化したものである。

本論理接続照合は、的確なグラフの対応付けにより、図1に示すような交換可能な端子(等論理端子)や、階層的に交換可能な端子を持つ素子も扱うことができる。

また、従来の手法では、レイアウト設計のエラー箇所を列挙するようなエラー情報であったが、本手法では、後処理としてエラー情報の統合を行うことで、効率的なレイアウト修正作業の行えるエラー情報を、設計者に提示することを可能とした。

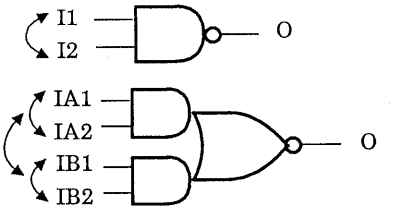


図1 等論理端子をもつ論理

以下、論理接続照合、エラー処理、適用結果について述べる。

2. システム概要

VLSIマスクパターンより抽出した回路接続情報(以後これをマスク側と呼ぶ)と論理設計時に作成した回路接続記述(以後これを論理側と呼ぶ)をそれぞれ回路グラフに変換し、この2つのグラフの同形判定を行った後、間違いが存在すれば、エラー処理部において的確なエラー箇所の限定を行い、エラーを出力する。

本システムの論理接続照合の処理の流れを図2に示す。

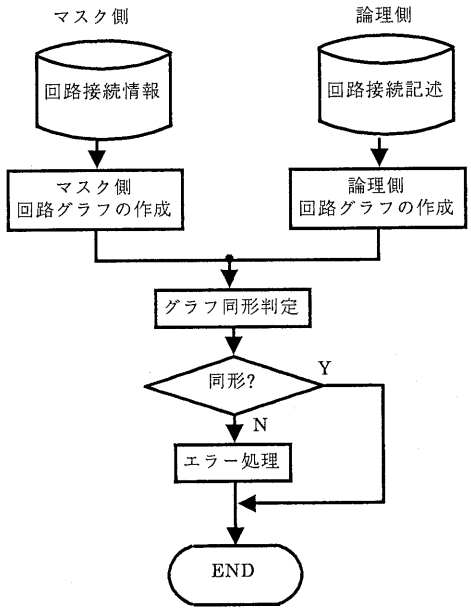


図2 論理接続照合の処理の流れ

3. 論理接続照合の手順

3.1 回路グラフの作成

トランジスタ、ゲート、セル各々の回路グラフ変換例を図3に示す。

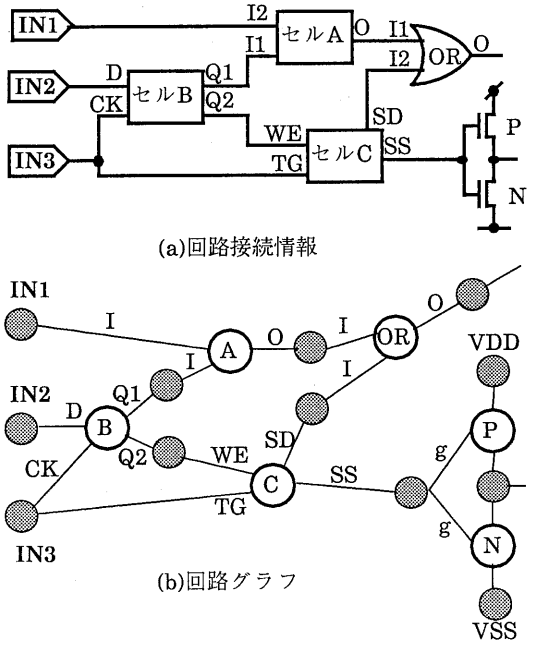


図3 回路グラフ変換例

本手法における回路グラフの各要素を表1に示す。

表1 回路グラフの要素

	素子頂点	ネット頂点	枝
変換対象	Tr ゲート セル	等電位ネット	頂点間の接続
ラベル	素子固有名	なし (外部端子は端子名)	各素子の端子名

各頂点間の接続は無向枝として示し、枝のラベルは各素子の端子名とする。ただし、等論理の端子には同じラベルを与えることにより、交換が可能な枝として取り扱う。

3.2 グラフ同形判定

同形判定の処理の流れを図4に示す。

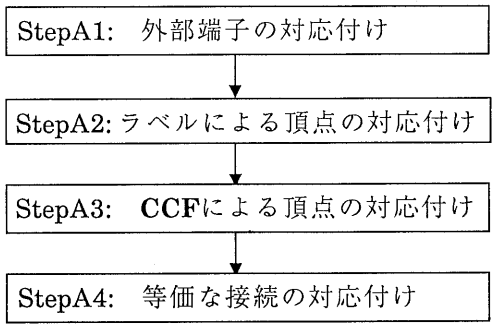


図4 グラフ同形判定の処理の流れ

3.2.1 StepA1: 外部端子の対応付け

3.1で述べたようにネット頂点にはラベルとして外部端子名のみが付加されている。従って、外部端子の対応付けはマスク側、論理側の両回路グラフにおいて、ラベルが等しいネット頂点に対し一対一に対応したことを示す固有な番号を回路グラフの各頂点に与える(以後固有化すると呼び、固$_n$と示す)ことによりなされる。

3.2.2 StepA2: ラベルによる頂点の対応付け

固有化された頂点固$_n$にラベルLeの枝で接続しているラベルLvの頂点集合$\{v_V\}$に着目し、ラベルLe,Lvを組み合せて両回路グラフを対応付ける。以下、対応付けの手順を示す。

① Le,Lvの組みが多対多に対応している頂点集合があればその頂点に対し同じグループ対応番号**GP**を与え、これを新しい頂点のラベルとして更新する。この新しい頂点のラベルにより図5に示す様な場合も簡単に固有化できる。

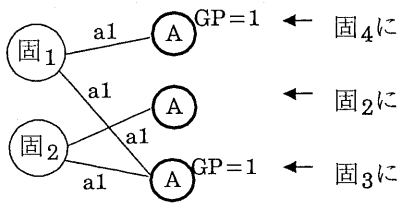


図5 グループ対応番号**GP**による固有化

② 一対一に対応する頂点があれば、固有化し固有化された頂点間の枝は削除する。図6にLe,Lvの組合せラベルによる頂点の固有化例を示す

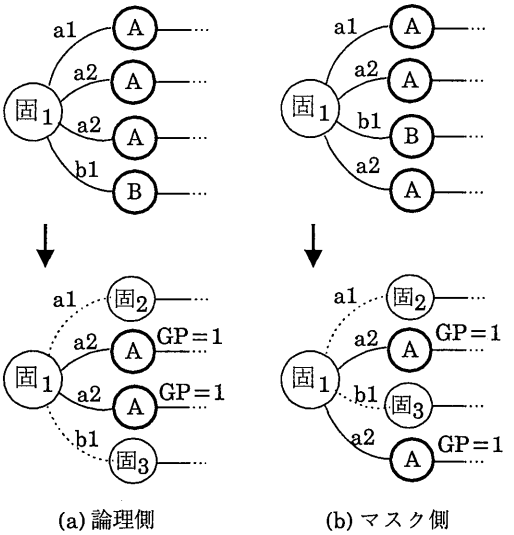


図6　ラベルによる固有化例

StepA2を繰り返した結果は、a)~c)の3通りに場合分けされる。

a) 全頂点が固有化され、全枝が削除されれば両回路グラフは同形である。

b) 両端の頂点が固有化されるが、削除されない枝はエラー枝を表す。また、対応する相手のない頂点はエラー頂点を表す。回路グラフ中に残っている頂点または枝が、全てエラーを表せばエラー処理に進む。

c) 図7に示すサブグラフは、固有化された頂点との接続箇所が全て多対多対応になっている。このようなサブグラフは、ラベルによる固有化をサブグラフ内部に進めることはできない。そこで、StepA3に移行して頂点の対応付けを行う。

3.2.3　StepA3: CCFによる頂点の対応付け

StepA3では、ある固有な頂点からの距離とその経路となる頂点･枝のラベルを同時に考慮することにより、接続関係のユニークさを表現するフラグである**CCF**を頂点に与え、ラベルによる固有化ができないようなサブグラフに対し、対応付けを行う。

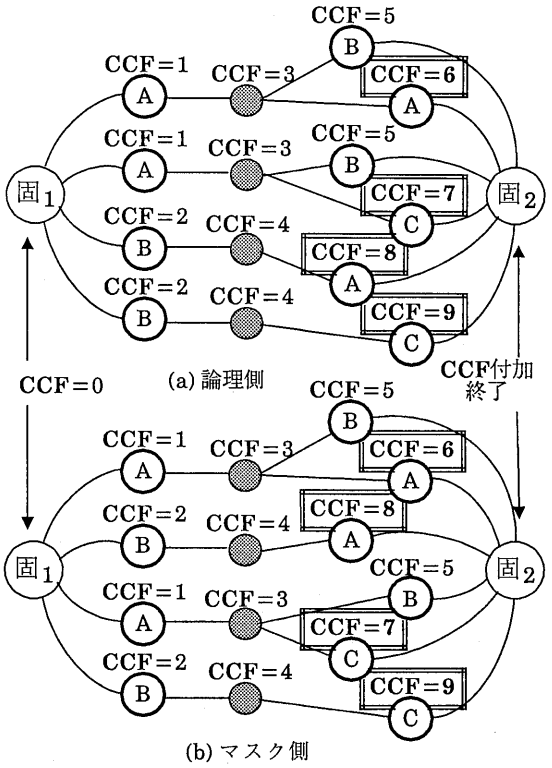


図7　ラベルによる固有化ができない例

①**CCF**を付加する。

a) ラベルによる固有化では固有化できない接続を持つ固$_a$に着目し、固$_a$に**CCF**=0をあたえる。

b) 同じ**CCF**を持つ頂点からの距離1の頂点集合を、枝と頂点のラベルが等しい部分集合に分類し、各部分集合に属する頂点に同じ**CCF**を与える。

c) 全ての経路が既に固有化されている頂点にたどりつけば、**CCF**付加は終了である。

但し、**CCF**を与える接続としての枝は1度しか通らないこととする。また、対応する相手のないエラー頂点が存在する場合は、このエラー頂点からの経路は切断し、エラーの影響のない部分のみを考える。

②CCF付加サブグラフの対応付け

a) 一対一に対応するCCFを持つ頂点があれば、これを固有化する(図7ではCCF=6,7,8,9の頂点を固有化することができる)。また、固有化された頂点に接続する頂点についてStepA2に戻りラベルによる固有化を行う。

b) 一対一に対応するCCFを持つ頂点がない図8に示すサブグラフはStepA4の等価な接続の対応付けに進む。

3.2.4 StepA4: 等価な接続の対応付け

図8において、連結サブグラフ1、連結サブグラフ2は論理的に全く等しく、連結サブグラフ中の矢印で示した枝は全て交換可能である。この様な接続を等価な接続と呼ぶ。

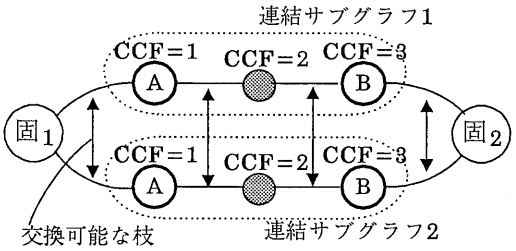


図8 等価な接続例

StepA4では、等価な接続を以下の手順で処理する。

① マスク側、論理側2つのサブグラフを、全ての端点が固有化された頂点への枝となる連結サブグラフに分解する(図8における連結サブグラフ1、連結サブグラフ2)。

② マスク側、論理側の連結サブグラフの数が一致するか確かめる。

a) 連結サブグラフの数が一致するときは、マスク側、論理側のCCFが等しい頂点どうしを任意に対応付け、枝を削除する。

b) 連結サブグラフの数が一致しないときは、余った連結サブグラフを構成する頂点にエラーフラグを付けエラー頂点として出力し、枝を削除する。

3.3 エラー処理

エラー処理の流れを図9に示す。

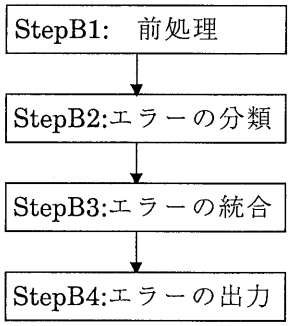


図9 エラー処理の流れ

3.3.1 StepB1: 前処理

図10に示すようにエラー頂点にとり囲まれた未処理のサブグラフが残る場合、エラー処理の前処理として未処理のサブグラフの対応付けを行う。

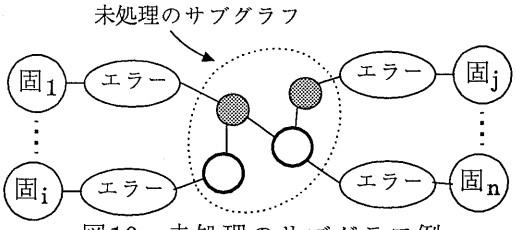


図10 未処理のサブグラフ例

① 未処理の頂点(固有番号もエラーフラグも付加されていない頂点)の中でユニークなラベルを持つ頂点が存在すれば、この頂点を固有化し、StepA2に戻り固有化された頂点に接続する頂点の対応付けを行う。

② 未処理のサブグラフの中から頂点数N=2の連結サブグラフを列挙する。

③ 構成要素がユニークな連結サブグラフが存在すればこれを構成する頂点を固有化し、StepA2に戻り固有化された頂点に接続する頂点の対応付けを行う。

④ 構成要素がユニークな連結サブグラフが存在しなければ、頂点数N=N+1として②に戻る。(ただし、N≦未処理のサブグラフの頂点数)

3.3.2 StepB2: エラーの分類

以上の処理を行うことにより、固有化されずに残っている頂点(以後エラー頂点と呼ぶ)、および削除されずに残っている枝(以後エラー枝と呼ぶ)は、マスク側、論理側2つの回路グラフの不一致箇所すなわちエラー箇所である。StepB2では、このエラー頂点とエラー枝の分類を行う。ただし、要素の余りが論理側、マスク側のどちらに発生するかで意味が大きく違ってくる。

以下エラー頂点及びエラー枝の分類の手順について述べる。

①エラー頂点の分類

a)素子頂点のエラー

エラー素子頂点V_{serr}に対して距離2の頂点集合を$V_{d=2}(V_{serr})$とする(素子頂点から距離2の頂点集合は素子頂点である)。頂点集合$V_{d=2}(V_{serr})$がすべて固有化されている場合のみを考える。論理側のエラー素子頂点の集合を$\{V_{serrlogc}\}$、マスク側のエラー素子頂点の集合を$\{V_{serrmask}\}$とする。Case-1、Case-2に関して図11、図12に示す。

(Case-1) $V_{d=2}(V_{serrlogc}) \subseteq V_{d=2}(V_{serrmask})$ 又は $V_{d=2}(V_{serrlogc}) \supseteq V_{d=2}(V_{serrmask})$であるエラー素子頂点$V_{serrlogc}$と$V_{serrmask}$が存在するとき、素子の割り付けミス

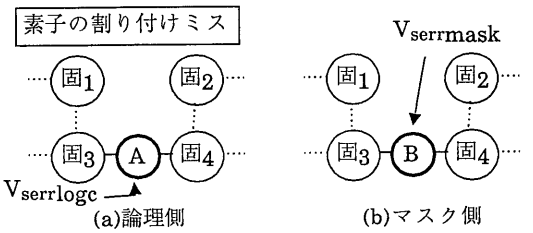


図11 素子の割り付けミス(Case-1)例

(Case-2) $V_{d=2}(V_{serrlogc}) \subseteq V_{d=2}(V_{serrmask})$ 又は $V_{d=2}(V_{serrlogc}) \supseteq V_{d=2}(V_{serrmask})$であるエラー素子頂点$V_{serrlogc}$と$V_{serrmask}$が存在しないとき。

Case-2に属する$V_{serrlogc}$は素子抜け

Case-2に属する$V_{serrmask}$は素子余り

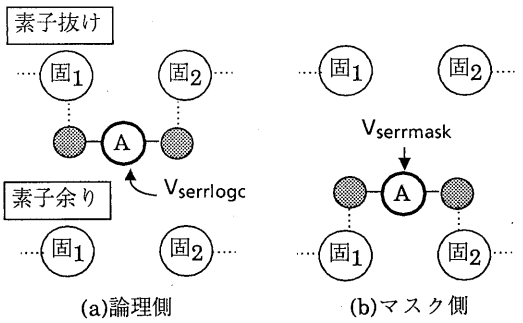


図12 素子抜け及び素子余り(Case-2)例

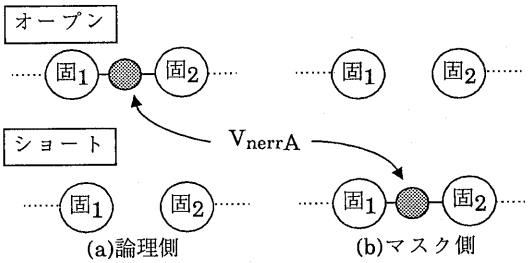
b)ネット頂点のエラー

エラーネット頂点V_{nerr}に対して距離1の頂点集合を$V_{d=1}(V_{nerr})$とする(ネット頂点からの距離1の頂点は素子頂点である)。頂点集合$V_{d=1}(V_{nerr})$がすべて固有化されている場合のみを考える。論理側のエラーネット頂点の集合を$\{V_{nerrlogc}\}$、マスク側のエラーネット頂点の集合を$\{V_{nerrmask}\}$とする。Case-3、Case-4に関して図13、図14に示す。

(Case-3) A側の$V_{d=1}(V_{nerrA})$はV_{nerrA}を中心として連結しているが、これと対応するB側の$V_{d=1}(V_{nerrA})$は中心となるネット頂点がないため連結していないとき。

◆A側=論理側、B側=マスク側
→頂点集合$V_{d=1}(V_{nerrA})$のオープン

◆B側=論理側、A側=マスク側
→頂点集合$V_{d=1}(V_{nerrA})$のショート



$V_{d=1}(V_{nerrA}) \equiv \{固1, 固2\}$

図13 オープン及びショート(Case-3)例

(Case-4)　A側の$V_{d=1}(V_{nerrA})$はV_{nerrA}を中心として連結しているが、これと対応するB側の$V_{d=1}(V_{nerrA})$は連結しているが、中心となるネット頂点V_nが他のA側の頂点と既に対応し固有化しているとき。

◆A側＝論理側、B側＝マスク側
→ $V_{d=1}(V_{nerrA})$と$V_{d=1}(V_n)$間がショート

◆B側＝論理側、A側＝マスク側
→ $V_{d=1}(V_{nerrA})$と$V_{d=1}(V_n)$間がオープン

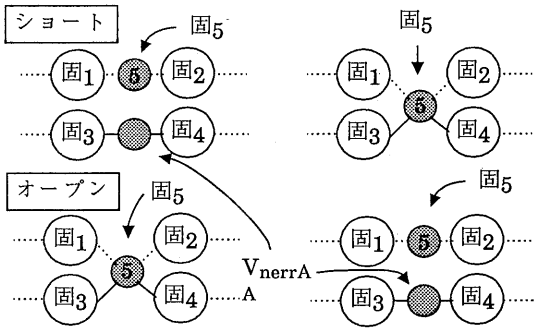


図14　オープン及びショート(Case-4)例

以上をまとめると、表2のようにエラー頂点の分類ができる。

表2　エラー頂点の分類

	素子頂点		ネット頂点	
	Case1	Case2	Case3	Case4
論理側	素子割り付けミス	素子抜け	オープン	ショート
マスク側	素子割り付けミス	素子余り	ショート	オープン

c) 連結エラー

2個以上連結した素子頂点を含む連結エラーが発生した場合は、この連結したエラー頂点部をまとめて回路接続情報にもどし、エラーとして出力する。

②エラー枝の分類

固有化された頂点間に残っている枝のみを対象とし、エラー頂点に連結して残っている枝は、エラー頂点の一部と考える。

ここで、固有化された素子頂点固$_S$、ネット頂点固$_N$を両端とした未対応の枝を$E_{err(S,N)}$とする(但し、固$_S$固$_N$は論理側マスク側それぞれに一対一に存在する)。ここで、以下に示す条件に基づいてエラーの分類を行う。

(Case-5)　論理側$E_{err(S,N)}$(以後$E_{el(S,N)}$と呼ぶ)は固$_S$と$V_{d=1}$(固$_N$)間の接続オープン

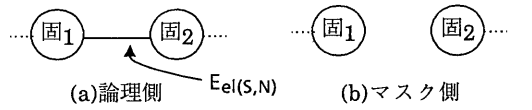


図15　接続オープン(Case-5)例

(Case-6)　マスク側$E_{err(S,N)}$(以後$E_{em(S,N)}$と呼ぶ)は固$_S$と$V_{d=1}$(固$_N$)間の接続ショート

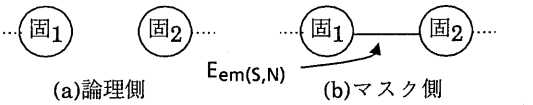


図16　接続ショート(Case-6)例

3.3.3　StepB3:　エラーの統合

論理接続照合を行い得られたエラー報告に従って設計者はマスクパターンを修正するが、この修正作業は配線パターンを間違った場所から正しい場所へ付け替えたり、未配線箇所の配線を行ったり、短絡配線箇所の修正を行うなど配線パターンを修正する場合がほとんどである。これは、エラーを含む回路グラフに対して、枝を付け替えることにより、いかに効率的に同形のグラフに修正できるかを考える問題に置き換えることができる。この際、従来のようにグラフの不一致箇所を列挙したエラー報告では、エラー箇所が多い場合などは効率的に修正を行えない。そこで、効率的なエラー報告を行うため、StepB2で分類したエラー情報を統合する。以下に、統合の条件①～③を示す。

① 端子名の誤り

$E_{el(S,N)}$及び$E_{em(S,N)}$が存在するが枝のラ

ベルがそれぞれL_A、L_Bで一致しない場合$E_{el(S,N)}$による固$_S$と固$_N$の接続オープンエラーと、$E_{em(S,N)}$による固$_S$と固$_N$の接続ショートエラーを統合し端子名誤り($L_B \rightarrow L_A$)とする。

② 配線先エラー
固$_S$と頂点集合$V_{d=1}$(固$_{NA}$)の接続オープンエラーと、固$_S$と頂点集合$V_{d=1}$(固$_{NB}$)の接続ショートエラーが同時に報告された場合、上記2つのエラーを統合し固$_S$の配線先エラーとする。そして頂点集合$V_{d=1}$(固$_{NB}$)への接続から頂点集合$V_{d=1}$(固$_{NA}$)への接続へ配線先を付け替える。以後固$_S$の配線先エラー$V_{d=1}$(固$_{NB}$)$\rightarrow V_{d=1}$(固$_{NA}$)と呼ぶ。

③ 逆接続エラー
固$_{S1}$の配線先エラーと固$_{S2}$の配線先エラーの配線先を付け替えた頂点集合とが逆の関係($V_{d=1}$(固$_{NB}$)$\leftrightarrows V_{d=1}$(固$_{NA}$))にあれば、これを逆接続エラーとして統合する。

3.4 エラー処理の具体例

図15は逆接続エラーの例であり、固$_5$,固$_6$がネット頂点、残りは素子頂点を表す。これを用いて具体的な説明を行う。

まず、エラーの分類に従い、

ERR1 論理側余り(オープン) 固$_5$ー固$_2$

ERR2 論理側余り(オープン) 固$_6$ー固$_4$

ERR3 マスク側余り(ショート) 固$_5$ー固$_4$

ERR4 マスク側余り(ショート) 固$_6$ー固$_2$

が得られる。

ERR1とERR4に注目すると、固$_S$≡固$_2$、$V_{d=1}$(固$_{NB}$)≡固$_6$、$V_{d=1}$(固$_{NA}$)≡固$_5$。ERR2とERR3に注目すると、固$_S$≡固$_4$、$V_{d=1}$(固$_{NB}$)≡固$_5$、$V_{d=1}$(固$_{NA}$)≡固$_6$。これを3.3.3の統合条件②に従い、ERR1とERR2をERR1′に、ERR3とERR4をERR2′に統合する。

ERR1′ 固$_5$の配線先エラー 固$_4\rightarrow$固$_2$

ERR2′ 固$_6$の配線先エラー 固$_2\rightarrow$固$_4$

次に、ERR1′とERR2′では、
固$_{S1}$≡固$_5$、固$_{S2}$≡固$_6$、$V_{d=1}$(固$_{NB}$)≡固$_4$、$V_{d=1}$(固$_{NA}$)≡固$_2$。これを統合条件③に従い、逆接続エラーとして統合する。

ERR1″ 固$_5$の配線と固$_6$の配線が逆接続エラー(固$_4\leftrightarrows$固$_2$)

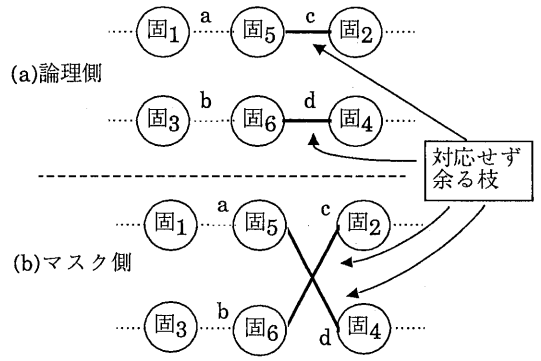


図15 エラーの統合例

4. 適用結果

(1)エラー検出

エラーとして、素子割り付けミス、素子抜け、素子余り、接続オープン、接続ショート、配線先エラー、端子誤り、逆接続を一度に付加した回路で、本システムを用いて接続照合を行った結果、疑似エラーを出力すること無く、的確にエラーを検出することが確認できた。

(2)処理時間

図16に、本手法を実品種に適用した場合の、グラフ頂点数に対する処理時間を示す。計算機は、IBM4341を用いた。

表3 処理結果

セル数	頂点数	CPU時間(sec)
108	284	16
514	1150	27
930	1834	36
1188	2422	42
3231	6597	80
3954	7645	136
5537	11122	130

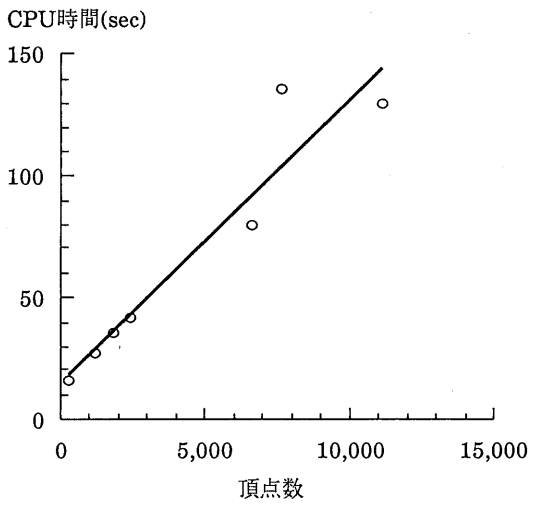


図16　処理時間

図16から、本手法の処理時間が実用的な回路規模のデータに対して、ほぼ頂点数に比例していることがわかる。

5.　まとめ

以上、VLSI論理回路設計とマスクレイアウト設計間の接続検証において、回路接続表現をラベルによる対応付けを考慮した**CCF(Connection Condition Flag)**による回路グラフ表現を用い、高速で的確な対応付けと容易なエラー箇所の限定が可能なヒューリスティック接続照合アルゴリズムを開発、その手続きと評価について報告した。

本手法を実開発品種に適用した結果、疑似エラーを出力することなく、高速かつ的確にエラー検出できることが確認できた。

また、本システムのエラー出力は、重複が無く統合化されているものであるため、設計ミスの修正作業の効率化につながる有効なものであることが確認できた。

今後、より多くの品種に適用しアルゴリズムの改善を図っていきたい。また、本システムをEWSへ移植し、より設計者に受け入れやすいエラー出力形式を考えていく予定である。

参考文献

1) P. M. Mauer and A. D. Schapira: A Logic-to-Logic Comparator for VLSI Layout Verification, IEEE Trans. Computer-Aided Design, vol.7, No.8, pp.897-907 (1988).

2) E. Bark: A Network comparison algorithm for layout verification of integrated circuits, IEEE Trans. Computer-Aided Design, vol.CAD-3, No.4, pp.135-141 (1984).

3) R. L. Spickelmier and A. R. Newton: Wombat: A new netlist comparison algorithm, Proc. ICCAD, pp.170-171 (1983).

4) D. C. Schmidt and L. E. Druffel: A fast Backtracking algorithm to test directed graphs for isomorphism using distance matrices, J. ACM, vol.23, No.3, pp.433-445 (1976).

5) Y. Shiran: YNCC: A new algorithm for device-level comparison between two functionally isomorphic VLSI circuits, Proc. ICCAD, pp.298-301 (1986).

6) C. Ebeling and O. Zajicek: Validating VLSI circuit layout by wirelist comparison, Proc. ICCAD, pp.172-173 (1983).

7) T. Watanabe, M. Endo and N. Miyahara: A new automatic logic interconnection verification system for VLSI design, IEEE Trans. Computer-Aided Design, vol.CAD-2, No.2, pp.70-81 (1983).

8) W. Maeda: Simple test for a class of isomprphic graphs, 信学会技法、回路とシステム研究会、CAS83-107, pp.85-89 (1983).

9) 寺尾淳子、田中節子、井上雅夫、羽山繁: 超LSIレイアウト検証システムFRIENDにおける階層的な接続検証、信学会技法、回路とシステム研究会、CAS86-204, pp.9-15 (1987).