レイアウト記述言語の構築とそれを用いたレイアウト再合成手法に関する研究

1994 年

重弘 裕二

内容梗概

本論文は, 筆者が平成元年から平成 5 年にかけて大阪大学大学院工学研究科後期課程に在学中ならびに大阪大学工学部情報システム工学科に在職中に行った大規模集積回路のマスクレイアウトデータの再合成手法に関する研究成果をまとめたものである.

集積回路技術の進歩に伴い, VLSI はますます大規模化し, もはや人手だけに頼る設計は不可能となっている. これに対処する計算機援用設計算法に関する研究も進み, 各設計工程における自動化手法の研究開発が急速に進行している. レイアウト設計は, VLSI 設計の全工程の中で依然としてかなりの部分を占めているが, その工程は基本的には機能セル内部のマスクレイアウト設計とそれらセル間の配置配線設計からなる. セル間の配置配線に関しては, これまで多くの研究が進められ, かなり実用化が達成されているが, 機能セルのレイアウト設計の自動化に関しては一層の研究開発が望まれている.

その一つの有力な手段として, プログラミング言語を用いて与えられたレイアウトを記述しておき, 必要に応じてその記述からマスクレイアウトデータを自動合成するという手法がある. 変更の可能性があるレイアウトパラメータを変数として記述しておけば, この変数に具体値を与えるだけで必要なレイアウトが得られるため, 設計変更が極めて容易になる. しかしながら, 設計者にとっては, 複雑なレイアウト記述の作成は非常に繁雑かつ困難な作業であり, この手法は実用化には到らなかった.

そこで, 本研究では, 既設計の機能セルのマスクレイアウトデータからレイアウト記述を自動生成する手法を考察し, 得られたレイアウト記述から新たにレイアウトを再合成するシステムを構築する.

本論文は 6 章から構成される. 第 1 章に序論を述べ, 第 2 章では対象とするレイアウトモデルについて述べ, 第 3 章では C 言語に基づくレイアウト記述言語を考案し, 第 4 章では既存のレイアウトデータからレイアウト記述を自動生成する手法を, 第 5 章では線形計画法を用いたレイアウト再合成手法を考察し, 第 6 章に以上の章の結論を述べる. 以下に各章の概要を述べる.

第 1 章では, VLSI の計算機援用設計, 特に機能セルのマスクレイアウト設計に関するこれまでの研究と課題について述べ, 本研究の背景, 目的を明らかにするとともに研究内容と成果について概説する.

第 2 章では, レイアウト記述についてまず概説し, 続いて, 一般的なレイアウトデータの構造について述べ, 本研究で対象とするレイアウトモデルについて考察する. さらに, 本研究で自動合成するレイアウトおよびその基となる既存のレイアウトの関連性に注目し, 本論文で扱う問題を定式化する.

第 3 章では, 機能セルのマスクレイアウト設計に有効なレイアウト記述手法について述べる.

まず, 設計規則と呼ばれるレイアウト要素の座標に関する制約条件に注目し, その変更や機能セルの動作速度等の性能上の変更に対処可能な, 汎用性の高いレイアウトの記述法について述べ, 得られたレイアウト記述に関する有効性について考察する.

次に, 新しいレイアウト記述言語を考案し, その詳細について述べる. 提案するレイアウト記述言語は, 3 種類のレイアウト記述用関数を C 言語に付加することによって得られるものであり, C 言語にレイアウト記述能力が備わったものであるといえる.

さらに, レイアウトデータを保持するための有効なデータベースについて述べた後, レイアウト記述をこのデータベース上のレイアウトデータに変換するレイアウト記述言語処理系の構築手法について考察し, 既存の C 言語を利用してレイアウト処理系を実現する.

第 4 章では, 既存のレイアウトデータから, 第 3 章で述べるような汎用性の高いレイアウト記述を自動生成する手法について述べる.

まず, レイアウト要素の座標間に存在する設計規則等の制約条件について注目し, 既存のレイアウトデータから, それらの制約条件を抽出する手法について考察する. 本手法と同様の処理はレイアウトコンパクション処理でも実行されるが, コンパクション問題と本論文で扱う問題との相違点を明確にし, コンパクション問題に対する手法が本論文で扱う問題には適さないことを指摘して, 新しい有用な手法を提案する.

次に, 制約グラフと呼ばれる抽出した制約条件を表現するグラフを用いて, レイアウト記述を生成する手法について考察する. まず, 処理の概要を述べ, ここで解決しなければならない問題を明らかにした後で, 制約グラフの最小木を用いてレイアウト記述を合成する手法を記述する. さらに, 提案する合成手法によってレイアウト再合成実験を行い, 本手法が実用上有効であることを示す.

第 5 章では, 第 3 章で述べるレイアウト記述言語処理系に線形計画法を導入する

ことにより, レイアウト記述の量を削減し, かつ最適レイアウトを再合成する手法について述べる.

まず, この再合成手法で用いる冗長部分を削減したレイアウト記述について述べ, その記述から線形計画問題の構成要素である連立一次不等式と目的関数を導く方法について考察する. さらに, レイアウト要素の抵抗率と素子の信号遅延に与える影響度に基づいた目的関数の設定法を考案し, 線形計画問題を構成する.

次に, 効率よく線形計画問題の解を求めるために, グラフ理論に基づくシンプレックス法について考察する. この手法では, 線形計画問題をグラフにより表現し, シンプレックス表における枢軸変換をグラフの木の初等変換に対応させて実行する. その際, その木の初期解として根付き木を用いることにより, 従来提案されている手法をさらに高速化している. 最後に, レイアウト再合成実験を行うことにより, 提案する手法が実用上有効であることを示す.

第 6 章では, 本研究で得られた成果を要約し, 今後に残された課題について述べる.

関連発表論文

I. 学会論文誌発表論文

(1) 重弘裕二, 白川功, 原嶋勝美, 神戸尚志: "C 言語によるレイアウト記述の一手法", 電子情報通信学会論文誌, Vol. J76-A, No. 4, pp. 618–627 (1993 年 4 月).

(2) Y. Shigehiro and I. Shirakawa: "A Recycling Scheme for Layout Patterns Used in an Old Fabrication Technology", *IEICE Trans.*, Vol. E76-A, No. 6, pp. 886–893 (June 1993).

II. 研究会等発表論文 (査読付)

(1) Y. Shigehiro, I. Shirakawa, and H. Takahashi: "A Recycling Scheme for Layout Patterns Once Generated for a Fabrication Technology", *Proc. 1992 Joint Technical Conference on Circuits/Systems, Computers and Communications* (JTC-CSCC '92), pp. 282–287 (July 1992).

(2) 長田岳史, 重弘裕二, 白川功: "線形計画法を用いた最適レイアウト再生成手法", 第 6 回 回路とシステム軽井沢ワークショップ, pp. 55–60 (1993 年 4 月).

(3) Y. Shigehiro, T. Nagata, and I. Shirakawa: "Layout Recycling Dedicatedly for Standard Cells Based on Graph Theoretic Simplex Approach", *Proc. 3rd International Design Automation Workshop* (Russian Workshop '93), pp. 129–136 (July 1993).

(4) Y. Shigehiro, I. Shirakawa, and T. Kambe: "A Recycling Scheme for Layout Patterns Used in an Old Fabrication Technology", *Proc. 11th European Conference on Circuit Theory and Design* (ECCTD '93), pp. 133–138 (August 1993).

(5) Y. Shigehiro, T. Nagata, I. Shirakawa, and T. Kambe: "Optimal Layout Recycling Based on Graph Theoretic Linear Programming Approach", *Proc. International Conference on Very Large Scale Integration* (VLSI '93), pp. 1.2.1–1.2.10 (September 1993).

(6) Y. Shigehiro, T. Nagata, and I. Shirakawa: "Optimal Layout Recycling", *Proc. Synthesis and Simulation Meeting and International Interchange* (SASIMI '93), pp. 255–263 (October 1993).

III. 研究会等発表論文

(1) 原嶋勝美, 若林謙二, 神戸尚志, 重弘裕二, 白川功: "C 言語によるレイアウト記述の一手法", 電子情報通信学会技術研究報告, CAS87-14 (1987 年 5 月).

(2) 重弘裕二, 白川功, 長尾明, 神戸尚志: "既存のレイアウトからのレイアウト記述の自動生成", 電子情報通信学会技術研究報告, CAS92-41, VLD92-41, DSP92-52 (1992 年 5 月).

(3) 重弘裕二, 白川功, 長尾明, 神戸尚志: "既存のレイアウトからのレイアウト記述の自動生成", 電子情報通信学会秋季大会, A-71 (1992 年 9 月).

(4) 長田岳史, 重弘裕二, 白川功: "線形計画法を用いた最適レイアウト変換手法", 電子情報通信学会秋季大会, A-72 (1992 年 9 月).

目 次

1 序論

集積回路技術の進歩に伴い, VLSI はますます大規模化, 複雑化し, もはや人手だけに頼る設計はほとんど不可能となっている. これに対処する計算機援用設計算法に関する研究も進み, 各設計工程における自動化が急速に進行している [1]. レイアウト設計は, VLSI 設計の全工程の中で依然としてかなりの部分を占めているが, その工程は基本的には機能セルのマスクレイアウト設計と機能セル間の配置配線設計からなる. セル間の配置配線に関しては, これまで多くの研究が進められ, かなり実用化が達成されているが [2–5], 最近では特に機能セルのマスクレイアウト設計の自動化に対する研究開発により多くの関心がよせられている. これは, 主として以下の事項に起因するものと思われる.

(i) VLSI 製造技術の急速な進歩によりプロセス技術に適合したセルの再設計が頻繁になったため, 多大な手間と労力を要するようになっていること.

(ii) サブミクロンプロセスでは各種の信号の遅延が回路動作に大きな影響を及ぼすため, 機能セルの出力信号駆動能力の最適化等の新たな問題が生じていること.

(iii) 大規模 VLSI を短期間に開発するために, CPU, PLA, ROM, RAM などの中規模以上の各種機能セルのライブラリの拡充が重要となっていること.

機能セルを人手設計する場合, それまでに設計済みのものをできるだけ再利用する方法が一般的である. その主な理由は, 回路のマスク構造に基本的な変更はなく, 製造技術の変化による設計規則変更への対応, 特定用途向き回路特性への対応が主たる設計変更の対象となるからである. ところが, 人手設計のためのグラフィック端末による対話型レイアウトエディタでは, レイアウト要素の座標値などのデータを具体的な数値で表現するため, 設計規則や素子サイズの変更に対応してデータを更新することは極めて煩わしい作業となる.

このような設計変更を自動的に実行するためには, 既設計の機能セルから基本的なレイアウト情報を抽出し, それに基づいて指定された変更条件に合致した機能セルを再合成することが実用上極めて重要な課題となる. その一つの有効な解決策として, まず, 機能セルのレイアウトパターンをレイアウト記述に自動変換し, レイアウト記述中の基本的なレイアウト情報 (長さ, 幅, 相対位置等) をパラメータ化し, 次

いで, パラメータ化されたレイアウト記述に設計変更条件を満たす数値を与えることにより, 要求された機能セルを自動合成する手法が考えられる [6–15].

本論文では, これを実現するために, レイアウト記述言語を考案し, 与えられたレイアウトデータからこの言語を用いたレイアウト記述を自動的に生成する手法, およびそのレイアウト記述からレイアウトデータを再合成する手法について考察する.

第 2 章では, レイアウト記述について概説する. ここでは, 一般的なレイアウトデータの構造に注目し, 本研究で対象とするレイアウトモデルについて記述し, さらに, 本研究で自動生成するレイアウト, およびその基となる既存のレイアウトに注目して本論文で扱う問題を定式化する.

第 3 章では, 機能セルのマスクレベルのレイアウト設計に有効なレイアウト記述手法 [16, 17] について考察する.

まず, 設計規則や機能セルの動作速度の変更に対処可能な, 汎用性の高いレイアウトの記述法について述べ, レイアウト記述の有効性について考察する.

次に, 新しいレイアウト記述言語を考案し, その詳細について述べる. 提案するレイアウト記述言語は, 3 種類のレイアウト記述用関数を C 言語に付加することにより, C 言語にレイアウト記述能力を持たせたものである. レイアウト記述言語については以前にも研究が行われ, いくつかの専用言語および処理系が報告されているが [16–25], いずれも専用のプリプロセッサやコンパイラなどを必要とするため, 言語の処理環境を構築するためには多大な手間と労力を要するものであった. 特に, 専用言語の場合, さらに設計者がそれを習得することも要求された. これを改善するため, レイアウト記述言語を C 言語に基づいて構築し, レイアウト設計者にとって習得が容易であるという特徴を持たせている. レイアウト記述を, 広く普及している C コンパイラと専用のライブラリを用意するだけで容易にレイアウトデータに変換するために, データベースを考案し, 与えられたレイアウト記述をこのデータベース上のレイアウトデータに自動的に変換するレイアウト記述言語処理系を構築することによって, 既存の C 言語処理系を活用したレイアウト記述生成システムを実現する.

設計者にはレイアウト記述からレイアウトの詳細な形状がわかりにくい場合が多く, したがって, レイアウト記述処理系をより一層有効に活用するためには, レイアウト記述を自動的に生成する必要がある. そこで, 第 4 章では既存のレイアウトデータから第 3 章で述べるような汎用性の高いレイアウト記述を自動生成する手法 [6–8, 11, 13] について考察する. アナログ回路を対象とした同様の試みが以前にも

報告されているが [26], 素子と配線が複雑に入り組んでいるディジタル回路に対しては, この試みは適用できない.

第 4 章では, まず, レイアウト要素の座標間に存在する設計規則等の制約条件に注目し, 既存のレイアウトデータから, それらの制約条件を抽出する手法について考察する. 同様の抽出法はレイアウトコンパクション処理でも用いられるが [27–30], コンパクション問題と本論文で扱う問題とは根本的に異なり, コンパクション問題に対する手法は本論文で扱う問題には適さない. そこで新たに斜め方向の配置をも考慮した制約グラフ [27] を導入して抽出した制約条件をすべて満たすようなレイアウト記述を自動的に生成する手法について考察する.

まず, 処理の概要を述べ, ここで解決しなければならない問題を明らかにした後で, 制約グラフの最小木を用いてレイアウト記述を合成する手法を考案する. さらに, 提案する手法を実現し, レイアウト再生成実験を行うことにより, 提案する手法が有効であることを示す.

しかしながら, 提案する手法では, レイアウト記述に対して面積や配線長の最小化を行うためには, レイアウト記述量がその分多くなるという問題点がある.

これを解決するために, 第 5 章では, 線形計画法を導入することによってレイアウト記述の量を削減し, かつ最適レイアウトを再合成する手法 [9, 10, 12, 14, 15] について考察する.

まず, 冗長部分を削減したレイアウト記述について述べ, その記述から線形計画問題の構成要素である連立一次不等式と目的関数を導く方法について考察する. さらに, レイアウト要素の抵抗率と素子の信号遅延に与える影響度に基づいた目的関数の設定法を考案し, 線形計画問題を定式化する.

次に, この線形計画問題の解を効率的に求めるための, グラフ理論に基づいたシンプレックス法 [31] について考察する. この手法では, 線形計画問題を長さとコストを各枝に付与したグラフにより表現し, シンプレックス表における枢軸変換をグラフの木の初等変換に対応させて実行する. その際, 木の初期解として根付き木を用いることにより, 従来提案されている手法をさらに高速化する手法についても言及する. 最後に, 提案する手法を用いてレイアウト再生成実験を行い, 提案する手法が実用上有効であることを示す.

最後に, 第 6 章では, 本研究で得られた成果を要約し, 今後に残された課題について述べる.

2 セルレイアウトの変換問題

2.1 緒言

本論文では, 機能セルに対するマスクレイアウト設計を自動化する目的で, 図 1 のように

(i) 既存のレイアウトデータをレイアウト記述に自動変換し, かつ

(ii) 設計規則や素子パラメータが具体的に定まったとき, そのレイアウト記述に基づいた最適レイアウトを自動合成する,

という手法について考察する. したがって, 本論文で考察する問題を明確にするためには, まず

(i) 与えられたレイアウトをレイアウト記述にいかにして変換しなければならないか,

(ii) 必要なレイアウトをレイアウト記述からいかにして再合成しなければならないか,

を明らかにしなければならない. 以下, レイアウト記述に変換される既存のレイアウトを**入力レイアウト**, レイアウト記述から再合成されるレイアウトを**出力レイアウト**と呼ぶ. 本章では, レイアウト記述について概説した後, 入力レイアウトの構造, および入力レイアウトに対する特性について述べ, 出力レイアウトに要求される性質に注目することにより, 本論文で考察する問題を明らかにする.

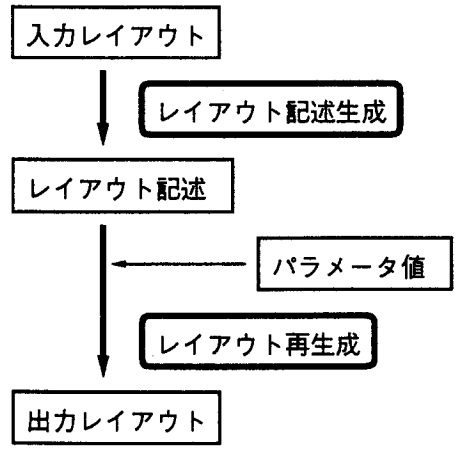


図 1: レイアウト記述によるレイアウト再生成

2.2 レイアウト記述

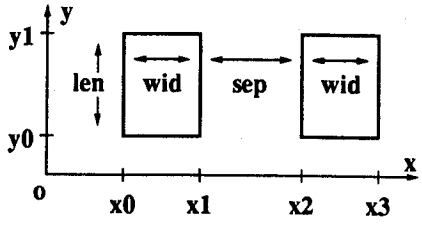
レイアウト記述とは, 言語により表現したレイアウトを意味する. 言語を用いることにより, 変数や制御文を用いた汎用性の高い表現が可能となり, 設計規則や素子サイズの変更が容易化できる.

例えば, 図 2 (a) の入力レイアウトをレイアウト記述により表現した例が図 2 (b) である. 最初の 4 行が座標の記述を表し, 残りの 2 行がレイアウト要素 (この場合は矩形) の記述を表している. 矩形の幅, 分離幅, 長さがパラメータ *wid*, *sep*, *len* により表現されているため, この記述はこれらのパラメータの具体値に依存しないものとなっている.

本論文で提案するレイアウト記述の詳細については 3.3 節で述べる.

2.3 レイアウトモデル

レイアウト設計とは, IC 内部のシリコンウェハ上に形成すべき酸化膜層やポリシリコン層などの形状と位置を決定することであり, レイアウトデータとは, 各層の形状を表すデータの集合体である. 各層は矩形や円などの単純な図形を組み合わせた形をしているため, レイアウトデータを, 層情報を持つ単純な図形 (以下, プリミティブと呼ぶ) の集合体とみなすこともできる. そこで本論文では, 図 3 に示すような, 素子およびセルからなる階層的なデータ構造を考える. すなわち, レイアウト設計とは, まず, 電気的に最小限の機能を持つように配置されたいくつかのプリミティブの集合をレイアウト要素と考え, それらを配置することである. このレイアウト要素を以下では単に素子と呼ぶ. 素子には, トランジスタ, コンタクト, 配線, 端子などがある. 回路としての機能を持つように配置されたいくつかの素子の集合をセルと呼ぶ.



(a) レイアウト

```
1:  x1 = x0 + wid;
2:  x2 = x1 + sep;
3:  x3 = x2 + wid;
4:  y1 = y0 + len;
5:  m_rectangle(x0, y0, x1, y1);
6:  m_rectangle(x2, y0, x3, y1);
```

(b) レイアウト記述例

図 2: レイアウト記述

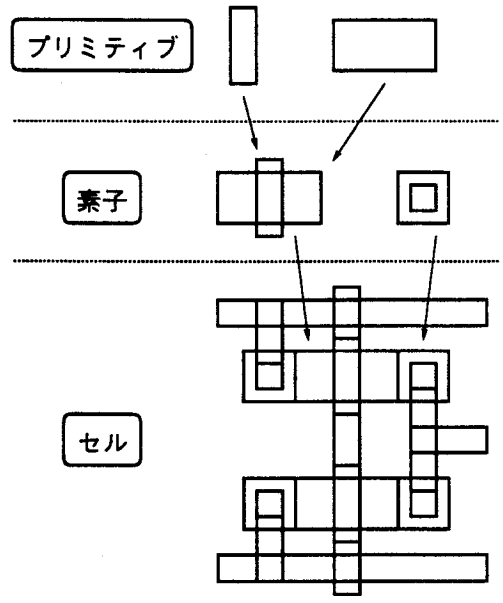


図 3: 機能セルのレイアウトの階層構造

さらに, いくつかのセルあるいは素子の集合をも一つのセルとみなす. 最終的には, IC の全回路を実現するレイアウトを, セルとして階層的に配置する.

2.4 入力レイアウト

本論文で提案するレイアウト再生成システムでは, 出力レイアウトが入力レイアウトと大きく異なることはない. したがって, 出力レイアウトの品質は入力レイアウトのそれとは本質的に同等である. そのため, 本研究では, 人手で入力され実際の利用に耐え得る品質を持ったレイアウトを処理の対象とする. すなわち, 入力レイアウトが以下の条件を満たすことを想定している.

(i) 設計規則に違反していないこと.

(ii) 必要に応じて配線に折れ曲がりを挿入するなど, セル面積を最小化するための種々のレイアウト処理が施されていること.

さらに, 問題を単純化するために入力レイアウトを以下のように限定する.

(iii) プリミティブの形状は, 各辺がチップの底辺に対して水平もしくは垂直な矩形のみとする.

(iv) 素子は, トランジスタ, コンタクト, 配線, 端子, およびウェルの 5 種類のみとする.

2.5 出力レイアウト

本研究の最終目的はレイアウトデータを再生成することにある. 出力レイアウトは, 設計規則に違反していないのは当然として, 面積が小さく, 電気的特性が良いものでなければならない. 本研究では, 以下のような特徴を持つレイアウト記述, もしくは出力レイアウトを再生成することを目標とする.

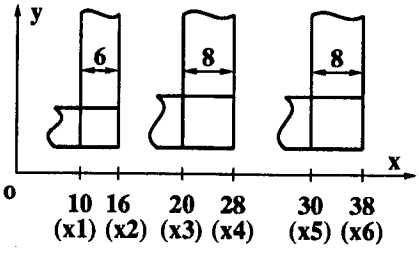
(i) 入力レイアウトにおいて大きさが異なるレイアウト要素は, 出力レイアウトにおいても異なる大きさに指定できる.

例えば, 図 4 (a) の入力レイアウトから生成した記述例が図 4 (b) であるが, このように, 入力レイアウトに 2 種類の幅 (6, 8) の配線が使われていた場合, 記述においても, 配線幅に対し 2 種類のパラメータ $w1, w2$ を割り当てる. これらのパラメータに異なる数値を与えることにより, 2 種類の幅の配線を持つ出力レイアウトを生成できる. これにより, 速度や集積度を向上させるために, 異なる太さの配線やゲート幅の異なるトランジスタを使い分けている入力レイアウトから, 同等の出力レイアウトを得ることができる.

(ii) 出力レイアウトにおいて配線が不必要に長くならない.

例えば, 図 5 (a) の入力レイアウトからは, 図 5 (b) の実線で示されるような冗長のない出力レイアウトを生成する. これにより, 配線長の増加に基づく電気抵抗の増加や動作速度の低下を避けることができる.

(iii) 入力レイアウトにおいて存在しないプリミティブは, 出力レイアウトに追加されない. 電気的な接続関係を保つために, 必要があればプリミティブが伸長する.

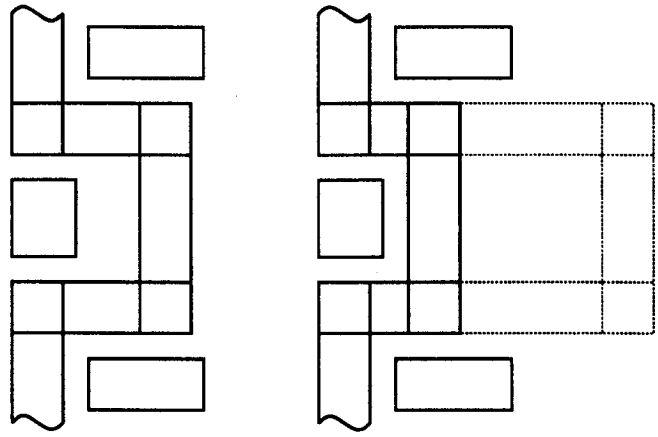


(a) 入力レイアウト

```
    ...
1:  x2 = x1 + w1;
2:  x4 = x3 + w2;
3:  x6 = x5 + w2;
    ...
```

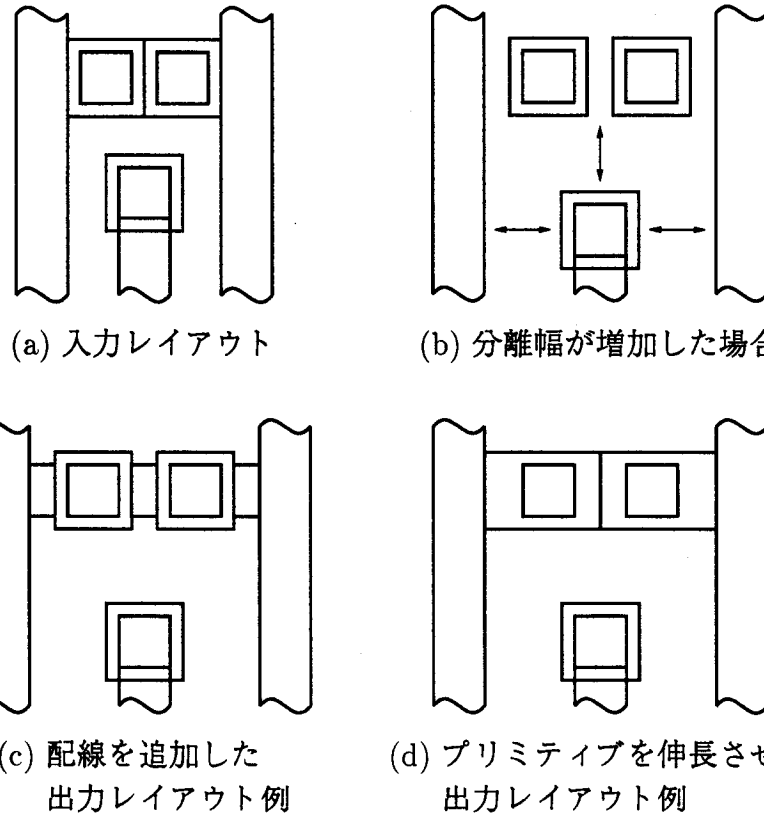
(b) レイアウト記述例

図 4: 異なるパラメータは異なる変数へ変換



(a) 入力レイアウト　　(b) 出力レイアウト例

図 5: 配線の最短化

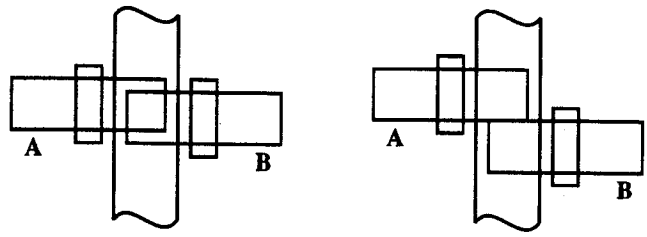


(a) 入力レイアウト　　(b) 分離幅が増加した場合

(c) 配線を追加した
出力レイアウト例

(d) プリミティブを伸長させた
出力レイアウト例

図 6: プリミティブ間の隣接関係を維持



(a) 入力レイアウト　　(b) 出力レイアウト例

図 7: プリミティブ間のオーバーラップを維持

例えば人手設計されたレイアウトでは, 隣接によりレイアウト要素を接続することがある. 図 6 (a) のレイアウトに対しプリミティブ間の分離幅を増加させるような設計規則の変更が生じた場合, 図 6 (b) のように電気的な接続関係が保てなくなることがある. 図 6 (c) のように配線を追加して対処すると, その部分の電気抵抗の増加により動作速度の低下を引き起こす可能性があるので, 図 6 (d) のようにプリミティブを伸長させ, なるべく動作速度に影響を与えないようにする.

(iv) 入力レイアウトにおいて共有部分を持つ同層のプリミティブは, 出力レイアウトにおいても共有部分を持つ.

例えば図 7 (a) におけるプリミティブ A と B のように近接して配置されているプリミティブ間には, 低抵抗な電気的接続が要求されている可能性があるため, 図 7 (b) のように, 出力レイアウトにおいてもそれらのプリミティブが離れないようにする.

2.6 結言

本章では, まず, レイアウト記述について概説し, 入力レイアウトがプリミティブ, 素子, セルという階層構造を持つことを述べた. 次に, 問題を単純化するために, 実用性を損なわない範囲で入力レイアウトに対して制限を設けることを述べた. さらに, 実用的な出力レイアウトについて考察し, 出力レイアウトに要求される特徴について述べた.

3 C 言語によるレイアウト記述言語

3.1 緒言

レイアウトの記述に用いる言語を**レイアウト記述言語**と呼ぶ. レイアウト記述言語については以前にも研究が行われ, いくつかの専用言語および処理系が報告されている [16–25].

それらは, 記述法に関して, レイアウト要素の相対的な位置の関係を指定する形式のもの [18, 20, 23] と, レイアウト要素の座標を具体的に指定する形式のもの [16, 17, 19, 21, 22, 24, 25] に分類することができる. 前者は, レイアウト要素の座標値の決定を処理系が行うため設計者の負担は軽減されるが, 座標を細かく指定することが困難となる. 本論文では, レイアウト記述を自動生成することを考慮し, 座標を細かく指定できる後者のレイアウト記述用言語について考察する.

処理系に関しては, 新たにコンパイラやインタプリタを必要とするもの [19, 22, 23], レイアウト記述を専用のプリプロセッサにより汎用言語に変換し, さらに汎用言語のコンパイラと専用のライブラリによりレイアウトデータに変換するもの [18, 21, 24], レイアウト記述を汎用言語のコンパイラと専用のライブラリにより中間言語記述に変換し, さらに専用のインタプリタなどによりレイアウトデータに変換するもの [20] などに分類できるが, いずれも専用のプリプロセッサやコンパイラなどを必要とするため, 言語の処理環境を構築するために多大な手間を要するものであった.

本章では C 言語に基づくレイアウト記述手法 [16, 17] について考察する. 提案するレイアウト記述法は C 言語を利用するため, レイアウトを記述する際に C 言語の変数, データ構造, および制御文などをそのまま利用できる. そのため, 専用言語と比較し, 設計者が容易に習得できる. さらに, 本章で提案する記述手法によるレイアウト記述は, 広く普及している C コンパイラと専用のライブラリを用意するだけで直接レイアウトデータに変換できる. その際, C 言語で書かれた既存のレイアウト自動設計用プログラムの直接的な利用も可能である.

以下, 本章では, レイアウト記述の有効性について述べた後, 提案するレイアウト記述法について述べ, 処理系の実現法について考察する.

3.2 レイアウト記述の有効性

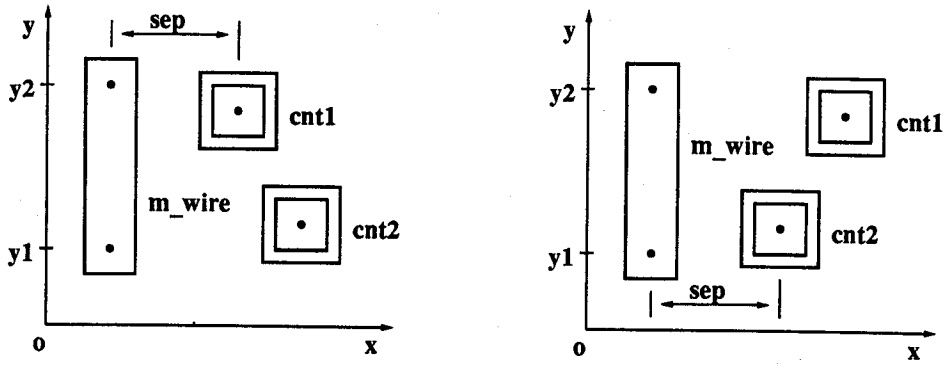
レイアウトの言語記述による表現には, 以下のような利点がある.

(i) 設計規則や素子サイズの具体値に依存しない形式でレイアウトを記述できる. したがって, 容易にそれらの変更に対処できる.

(ii) 規則的な構造を持つレイアウトが簡潔に記述できる. メモリ, データパス, PLA など, 中規模以上のレイアウトをも記述可能である.

本章で考察するレイアウト記述言語 **LDL** (Layout Description Language) は, C 言語にレイアウト記述用の関数群を付加したものである. LDL で必要となる記述能力の大半は C 言語が本来備えているため, それをそのまま利用すれば前述のような利点を持つ記述を実現できる.

例えば, 設計規則の具体値に依存しない形でレイアウトを記述するには, 変数や if 文を用いる. 配線を 2 つのコンタクトの左に設計規則を犯さないように配置した図 8 (a) のレイアウトの記述例である図 8 (b) では, 設計規則に依存する量であるコンタクトと配線の間隔を変数 (もしくは, C 言語の #define 機能によるマクロ) *sep*



(i) 上のコンタクトが配線に近い場合　(ii) 下のコンタクトが配線に近い場合

(a) レイアウト

```
1:  mgpcontact("cnt1", ..., ...);
2:  mgpcontact("cnt2", ..., ...);
3:  if (X("cnt1") < X("cnt2")) {
4:      m_wire( ..., X("cnt1") - sep, y1, X("cnt1") - sep, y2);
5:  } else {
6:      m_wire( ..., X("cnt2") - sep, y1, X("cnt2") - sep, y2);
7:  }
```

(b) レイアウト記述例

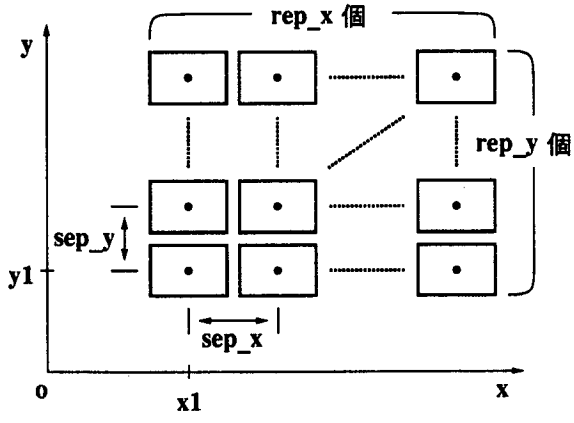
図 8: 設計規則に違反しないレイアウト例

で与えているため, *sep* の値を変更するだけで, 異なる設計規則に対応したレイアウトデータが生成できる. そのとき, どちらのコンタクトが左にあっても配線は 2 つのコンタクトと *sep* 以上の間隔を持つため, 生成されたレイアウトデータには設計規則違反が生じない.

同様に, トランジスタのゲート幅など, 回路の動作速度に影響を及ぼす値を変数を用いて記述しておけば, その変数に異なる値を与えることにより, 容易に特性の異なるレイアウトを得ることができる. さらに, 設計規則に依存する値を定義したファイルを用意し, C 言語の #include 機能により各レイアウト記述ファイルで取り込むことにすれば, そのファイルを変更するだけで設計規則の変更に対処できる.

また, 規則的な構造を持つレイアウトを記述するには, for 文を用いる. 例えば, セルが縦横に規則正しく配置された図 9 (a) のレイアウトの記述が図 9 (b) のように簡潔に記述できる.

本章で提案する手法には, C 言語を利用することにより生じる以下のような利点もある.



(a) レイアウト

```
1:  for (i = 0; i < rep_x; ++i) {
2:      for (j = 0; j < rep_y; ++j) {
3:          ntransistor( ..., ..., ...,
              x1 + sep_x * i, y1 + sep_y * j, ...);
4:      }
5:  }
```

(b) レイアウト記述例

図 9: セル配列のレイアウト例

(i) C 言語の形式でレイアウトを記述するため, 新たに言語仕様について習得する必要がない.

(ii) 本章で提案する手法と同様の手法により, 処理系を大幅に変更することなくレイアウト記述用の関数を追加できるため, 言語の拡張性が高い.

これまでに, 自動配置配線やレイアウト検証などのために多くのプログラムが開発されているが, それらに少し手を加え C 言語の関数として呼び出せるようにするだけで, それらの機能を利用したレイアウト記述をレイアウトデータに変換することが可能となり, 強力なセル・モジュール生成環境が構築できる. それらのプログラムが C 言語で書かれている場合も多いが, その場合, この拡張作業はさらに容易なものとなる.

3.3 レイアウト記述言語

本節では, 新たにレイアウト記述言語 LDL を考案し, その詳細について述べる. LDL では, C 言語と新たに考案するレイアウト記述用関数により機能セルのマスクレイアウトを記述する. そのレイアウト記述用関数は

(i) 素子を記述するための素子記述関数

(ii) セルの階層関係を管理するためのセル定義生成関数

(iii) レイアウト記述を容易化するためのレイアウト援用関数

から構成される. 以下, 順に詳細を述べる.

3.3.1 素子の記述

実際のレイアウト設計においては, 特殊な形状の素子が用いられることはあまりない. LDL では素子を最も基本的なレイアウト要素と考え, 図 10 のようにプリミティブを用いてあらかじめ定義しておくことにより, 設計者の負担軽減を図る. ただし, 設計に十分な自由度を持たせるため, 位置, 向き, 大きさは, 素子記述のパラメータとして指定可能とする.

ここで提案する手法では, 素子を記述するための関数をいくつか用意する. この関数を**素子記述関数**と呼ぶ. 素子記述関数の例を表 1 に示す. 例えば, n 型トランジス

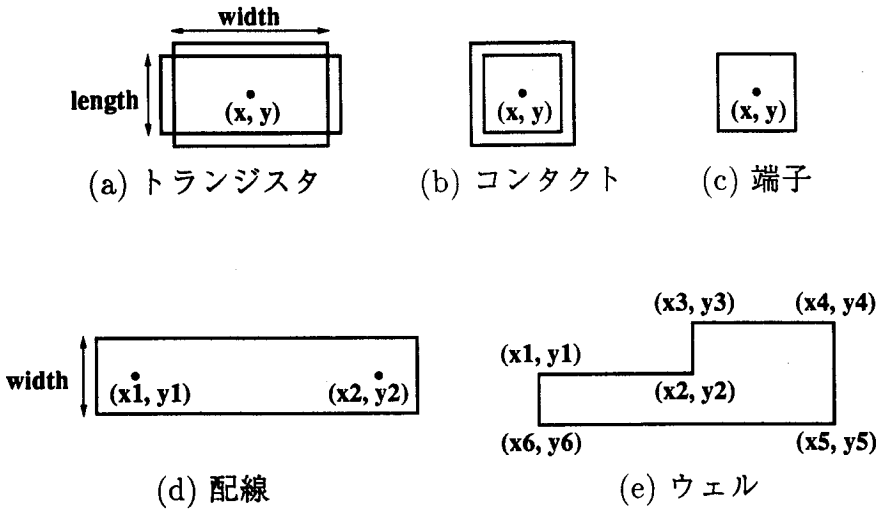


図 10: 素子の形状の例

表 1: 素子記述関数

関数名	機能
ntransistor	n 型トランジスタの記述
ptransistor	p 型トランジスタの記述
mdcontact	金属層-拡散層間コンタクトの記述
mgpcontact	金属層-ポリシリコン層間コンタクトの記述
mmcontact	金属層-第 2 金属層間コンタクトの記述
dmdcontact	第 2 金属層-拡散層間コンタクトの記述
m_terminal	金属層端子の記述
dm_terminal	第 2 金属層端子の記述
gp_terminal	ポリシリコン層端子の記述
m_wire	金属層配線の記述
dm_wire	第 2 金属層配線の記述
gp_wire	ポリシリコン層配線の記述
boundary	ウェルの記述
m(d, gp, dm, h, ...)_rectangle	金属 (拡散, ポリシリコン, 第 2 金属, コンタクトホール, ...) 層の矩形の記述

ntransistor(識別名, ゲート長, ゲート幅, x, y, 向き);
(a) トランジスタの記述

mdcontact(識別名, x, y);
(b) コンタクトの記述

m_terminal(識別名, ネット名, ネット識別子, x, y);
(c) 端子の記述

m_wire(幅, $x1$, $y1$, $x2$, $y2$);
(d) 配線の記述

boundary(頂点数, 頂点座標の配列);
(e) ウェルの記述

m_rectangle($x1$, $y1$, $x2$, $y2$);
(f) 矩形の記述

図 11: 素子の記述

タは, 図 11 (a) の形式で記述し, 引数により, 識別名 (後述のレイアウト援用関数で, 個々の素子を識別するための名前), ゲート長, ゲート幅, x 座標, y 座標, 向きを順次指定する. 金属層-拡散層間コンタクト, 金属層端子, 金属層配線なども, 同様に素子記述関数を用いて, 図 11 (b)–(d) の形式で記述する. また, CMOS 回路のレイアウトで必要となるウェルは, 多角形であり頂点数が不定である. そこで, 頂点数と頂点の座標を格納した配列を引数として図 11 (e) の形式で記述する. さらに, レイアウト記述自動生成処理において, レイアウトをより正確に指定できるように, 矩形を記述するための関数を用意した. 例えば, 金属層の矩形は図 11 (f) の形式で記述する.

3.3.2 セルの記述

セルのレイアウトを決定するには, セル自身の定義, およびセルを構成する下位セルや素子の配置の指定が必要である. ここで提案する手法では, C 言語の関数定義 [32] を利用してセルを記述する. 関数定義内で素子やセルの記述関数の呼び出し文を記述することにより, 下位セルや素子の配置を指定する. 通常 C 言語で, 下位の関数や標準ライブラリ関数の呼び出し文を記述することによりサブルーチンの利用を

```
1:  セル名 (引数)
2:  引数宣言
3:  {
4:      変数宣言
5:      openh(セル名, x 座標, y 座標, 向き);
6:      セル内部の記述
7:      closeh();
8:  }
```

図 12: セルの記述

指定するのと同様である.

図 12 がセル記述の形式である. まず, セル名を C 言語の関数名として記述し (図 12 , 1 行目), さらに必要に応じて, セルのパラメータおよびセルの記述に必要な局所変数を, 関数のパラメータ (図 12 , 2 行目) および関数の局所変数 (図 12 , 4 行目) として宣言する.

ここで提案する手法では, 階層関係を管理するための関数を用意する. この関数を**セル定義生成関数**と呼ぶ. セル定義生成関数 openh() ではセルの名前, 位置および向きを記述し (図 12 , 5 行目), 関数の本体でセルを構成する素子や下位セルの配置を記述する (図 12 , 6 行目). セルの構成要素が下位セルを含む場合は, その下位セルを記述している関数に対する関数呼び出し文によりその下位セルを記述し, 素子を含む場合は, 前述した素子記述関数によりその素子を記述する. 最後に, セル定義生成関数 closeh() を用いてそのセルの記述が終了したことを宣言し (図 12 , 7 行目), セルの記述が完了する.

3.3.3 レイアウト援用関数

ここで提案する手法では, レイアウト記述を容易化するために C 言語に関数を追加する. この関数を**レイアウト援用関数**と呼ぶ. その例を表 2 に示す. 例えば, 関数 X(), Y() を用いれば定義済みの素子やセルの座標を参照できる. ある素子またはセル a の x 座標を参照するには,

```
X("a")
```

とする. これにより, 他のセルや素子の位置を用いてセルや素子の座標値を相対的に

表現できる. 図 13 (a) のように 2 つのコンタクト間を配線で結ぶ場合, 図 13 (b) のように記述する. また, セル a の構成要素 b の x 座標を,

```
X("a.b")
```

と記述する. これは, セル内の素子に対してセル外部からの配線を記述するのに有効である. また, 関数 NAME() は, for 文などにおいてセルの配列 "A[1]", "A[2]", ..., 中のどれかを呼び出す際に用いる. 例えば, for 文中で $k = 1$ に対して

```
NAME("A", k)
```

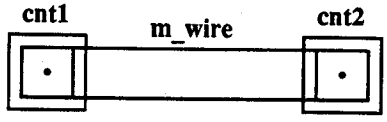
はセル "A[1]" を表す. さらに, モジュールリストとネットリストを引数として自動配置配線を行う関数 place_route() もある.

3.4 レイアウト記述処理系

階層構造を持つレイアウトデータを効率的に管理するためにはデータベースが用いられることが多い [33]. 本節でもデータベースに管理されたレイアウトデータを

表 2: レイアウト援用関数

関数名	機能
X, Y	引数で与えられた素子, セルの座標を返す
X_SIZE, Y_SIZE	引数で与えられた素子, セルの大きさを返す
NAME	第 1 引数の文字列と第 2 引数の値から合成したセル名を与える
place_route	第 1 引数のモジュールリストを, 第 2 引数のネットリストにしたがって, 1 次元に自動配置配線する



(a) レイアウト

```
1:  mgpcontact("cnt1", ..., ...);
2:  mgpcontact("cnt2", ..., ...);
3:  m_wire( ..., X("cnt1"), Y("cnt1"), X("cnt2"), Y("cnt2"));
```

(b) レイアウト記述例

図 13: コンタクト間の配線

対象にした処理について考察する. 以下では, まずレイアウトデータベースの構造について述べた後, レイアウト記述をレイアウトデータベース上のデータに変換する手法について考察する.

3.4.1 レイアウトデータベース

本節で対象とするデータベースの主な論理構造を図 14 に示す.

(i) 定義情報

定義情報は, 各レイアウト要素に関する設計情報を管理するためのものである. ここで管理される設計情報には, 図形情報, インスタンス情報, 端子情報, ネット情報, および定義自身の属性情報がある. 属性情報は, 定義情報の種類, 構成するデータを囲む最小の最外殻, 使用しているマスク層, バージョンなどの情報から成る.

LDL においては, C 言語の関数定義の形式にしたがって個々のセルに関する情報がまとめられているが, 同様の概念をデータベース上で実現するのが定義情報である.

(ii) 図形情報

図形情報は, 素子やセルのレイアウトパターンに関する情報である.

(iii) インスタンス情報

インスタンス情報は, 素子あるいはセルが実際に使用される場合の配置に関する個別の情報である. すなわちインスタンス情報は, セルを構成する下位セルや素子に関する配置情報から成り, セルの階層を表現する.

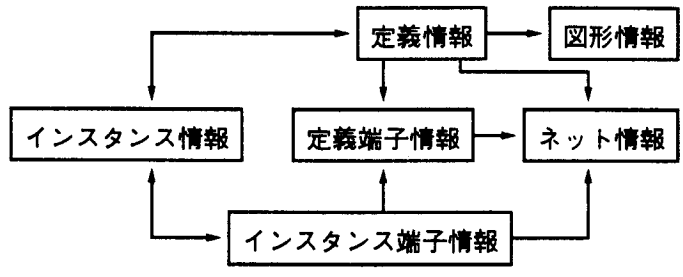


図 14: データベースの論理的構造

LDL においては, 下位セルを定義する関数や素子記述関数に対する関数呼び出しの形式を用いて配置情報を記述するが, データベース上でその情報を保持するのがインスタンス情報である.

(iv) 端子情報

端子情報には, 定義情報外でのネット情報を管理するための定義端子情報と, インスタンス情報を構成要素として持つ定義情報内でのネット情報を管理するための, インスタンス端子情報がある.

LDL では, 端子を生成する素子記述関数により記述される情報に相当する.

(v) ネット情報

ネット情報は, 各定義情報ごとに信号名を管理するためのもので, 定義端子情報, インスタンス端子情報から参照される.

3.4.2 記述からデータへの変換

LDL によるレイアウト記述は C 言語の文法に従っている. したがって, レイアウト記述用関数に対応するライブラリを用意すれば, C コンパイラによりコンパイルできる. 以下では, C コンパイラとレイアウト記述用関数のライブラリを用いて, LDL によるレイアウト記述をデータベース上のレイアウトデータに変換する手法について考察する. 提案する手法では, レイアウトに固有な処理はすべてレイアウト記述用関数のライブラリ内で実行する. そのため, 新たにプリプロセッサやコンパイラなどを作成する必要がない.

レイアウト記述は, 通常の C プログラムと同様, 次に示すような手順により処理される.

1° レイアウト記述を C コンパイラでコンパイルする (図 15 (1)).

2° その出力のオブジェクトファイルとレイアウト記述用関数のライブラリをリンクする (図 15 (2)).

3° その結果得られた実行ファイルを実行する (図 15 (3)).

3° の実行ファイルは **2°** でリンクされたレイアウト記述用関数のライブラリを含んでいる. 実行ファイルを実行するとそれが呼び出され, その内部で実際のレイアウトデータの生成が行われる.

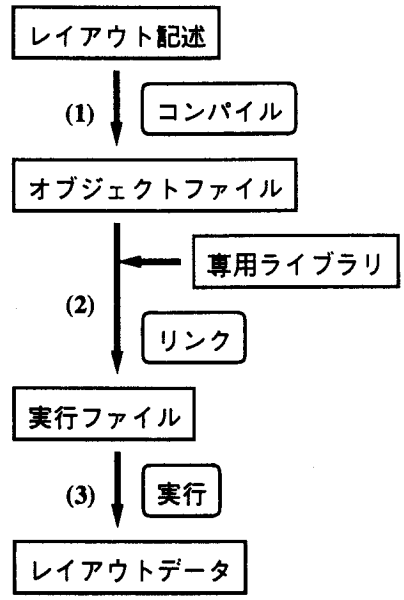


図 15: レイアウト記述処理系

LDL が記述の対象とするレイアウト要素には素子とセルがある. 素子は 3.3 節で述べたようにあらかじめ定義されているため, 内部構造もあらかじめ決定している. したがって, 素子の内部レイアウトデータはあらかじめ用意しておくことができる. 一方, セルはレイアウト記述により定義されるため, セルの内部レイアウトデータはレイアウト記述用関数により生成する必要がある.

レイアウト記述用関数の実行順序, すなわちレイアウトデータの生成順序は C 言語の実行順序に左右される. そのために 2 つの問題が生じる.

第 1 の問題は, セル内のレイアウト記述中に下位階層セルの記述関数への呼び出しを含む場合, セル定義の生成終了前に下位セルを記述した関数が実行され, その後再び制御が戻る処理順序となるため, 異なる階層のセルを同時に並行して定義する必要があることである. これは, 定義中のセルを階層的に含むすべての上位セルの情報を, スタック構造 (以下, **セル定義スタック**と呼ぶ) に保持しておくことにより解決できる. レイアウト記述用関数では, セル定義スタックにより対象となるセルを特定できるので, 複数セルを並行して定義しても混乱は生じない.

第 2 の問題は, 上位階層セルの記述関数から複数回呼び出されるセルの記述関数が, 複数回実行されることである. データの二重生成を避けるため, 2 度目以降の呼び出しではデータを生成しないようにする必要がある. これは, 各セルに対して, デー

タ生成の必要性を示すフラグ (以下, **データ生成フラグ**と呼ぶ) を用意しておくことにより解決できる.

また, 素子やセルの座標や大きさを参照するレイアウト援用関数 (X() や X_SIZE() など) を効率よく実現するために, セルや素子の位置や大きさ, 構成要素の識別名など, 関数を実現するために必要な情報をテーブル (以下, **レイアウト情報テーブル**と呼ぶ) に格納しておく. レイアウト情報テーブルをセル定義生成関数や素子記述関数によりデータベースと同時に更新しておけば, レイアウト情報テーブルからデータを読み出して関数の値として返すことにより, レイアウト援用関数を実現できる.

セル定義生成関数の実現法を図 16 に, 素子記述関数の実現法を図 17 に示す. 素子記述関数に関しては, ここでは ntransistor() について示したが, 他の関数もすべて同様である.

素子やセルの座標や大きさを参照するレイアウト援用関数の実現法を図 18 に示す. ここでは X() について示したが, X_SIZE() なども同様である.

```
 1:  openh(...) {
 2:      if (定義するセルが未登録である) {
 3:          定義情報に定義するセルを登録する;
 4:          レイアウト情報テーブルにセルの情報を追加する;
 5:          定義するセルのデータ生成フラグをセットする;
 6:      } else {
 7:          定義するセルのデータ生成フラグをリセットする;
 8:      }
 9:      if (上位セルが存在する
           and 上位セルのデータ生成フラグがセットされている) {
10:          上位セルに対してインスタンス情報を追加する;
11:          レイアウト情報テーブルにセルの情報を追加する;
12:      }
13:      セル定義スタックに定義するセルの情報を追加する;
14:  }

15:  closeh() {
16:      定義したセルの情報をセル定義スタックから削除する;
17:  }
```

図 16: セル定義生成関数の実現

```
1:  ntransistor(...) {
2:      if (その素子を含むセルのデータ生成フラグがセットされている) {
3:          そのセルに対してインスタンス情報を追加する;
4:          レイアウト情報テーブルに素子の情報を追加する;
5:      }
6:  }
```

図 17: 素子記述関数の実現

```
1:  X(...) {
2:      レイアウト情報テーブルから素子の情報を取り出す;
3:      return (取り出した情報);
4:  }
```

図 18: レイアウト援用関数の実現 (1)

```
1:  place_route(...) {
2:      パラメータを既存の配置配線プログラムの入力用に変換する;
3:      既存の配置配線プログラムを呼び出す;
4:      その結果に応じて, データベースとレイアウト情報テーブルを更新する;
5:  }
```

図 19: レイアウト援用関数の実現 (2)

一次元の自動配置配線を行う関数 place_route() は, 図 19 に示すように, 既存の配置配線プログラムを C 言語の関数として呼び出せるように修正することにより実現する. 同様な手法を用いれば, 二次元の自動配置配線プログラムなど, 他のレイアウト自動設計用プログラムをレイアウト援用関数として実現することも容易である.

3.5 結言

本章では, レイアウト記述の有効性について述べた後, C 言語に基づくレイアウト記述言語を考案し, その詳細について述べた. さらに既存の C コンパイラを利用し, レイアウト記述からレイアウトデータを生成する手法について考察した.

本章で提案した手法を用いれば, LDL 専用のコンパイラを作成することなく, レイアウト記述用関数のライブラリを作成するだけで, レイアウト記述をデータベー

ス上のレイアウトデータに変換する処理系が構築できる. 構築された処理系は, シリコンコンパイラなどにおける機能セルの自動生成処理への活用が期待できる.

4 記述による最適化に基づいたセルレイアウトの再生成手法

4.1 緒言

本章では, 機能セルのレイアウトパターンをレイアウト記述に自動変換する手法[6–8, 11, 13] について考察する. 提案する手法では, まず, 機能セルのレイアウトパターンからレイアウト情報を抽出し, レイアウト記述言語を用いてパラメータ化する. このように既設計のレイアウトパターンをレイアウト記述に変換しておけば, 設計変更条件を満たす数値をパラメータとして与えるだけで, 要求された機能セルを自動生成できる. また, 新しいレイアウト記述が必要な場合にも, グラフィックエディタを利用してレイアウトパターンを作成し, それを本章で提案する手法によりレイアウト記述に自動変換することにより, これまでと同等の手間でレイアウト記述を得ることができる.

従来, アナログ回路を対象としたものではあるが, 同様の試みが報告されている[26]. この手法では, 素子の記述とネット情報に関する記述のみが自動生成される. 記述からレイアウトパターンを再生成する際には, まず素子の位置が記述にしたがって決められ, その後, 自動配線手法により素子間配線が行われる. このため, 素子と配線が複雑に入り組んでいるディジタル回路に対しては, うまく適用できないと思われる. 本章で提案する手法では配線の記述も自動生成するので, このような問題は生じない.

提案する手法の概要を図 20 に示す. 以下では, まず, 入力レイアウトからレイアウト要素間に存在する制約条件を抽出する手法について考察し, 続いて, その制約条件に基づきレイアウト記述を生成する手法について考察する. 第 3 章に述べたレイアウト記述処理系には配線長の最短化等の最適化機能がないため, 最適化を考慮しながら記述を生成する. さらに, 提案する手法を実現し, レイアウト再生成実験を行うことにより, 提案する手法が有効であることを示す.

4.2 処理手法

IC のマスクレイアウトは可能な限り小さな方が望ましいが, レイアウト要素の大きさの下限やレイアウト要素間の距離の下限を定めた設計規則を犯してはならない[34]. したがって, レイアウト要素の位置は, 設計規則と周辺のレイアウト要素の位

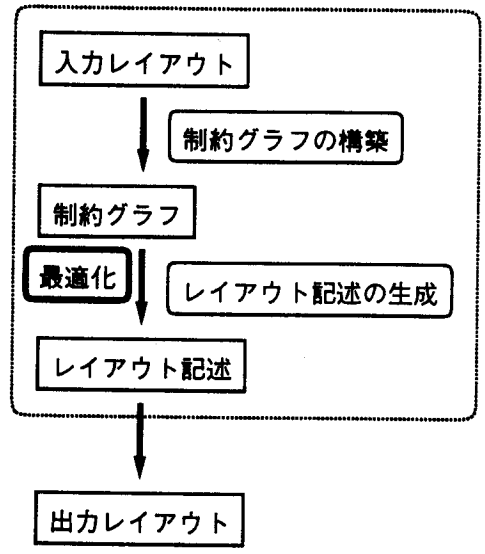


図 20: 記述による最適化に基づいたレイアウトの再生成手法

置によって表される制約条件を満たすように定めなければならない. あるレイアウト要素の座標を a, その周辺にあるレイアウト要素の座標を b とし, その座標間の距離の下限が r であるという設計規則が与えられたとき, $b < a$ とすれば, 制約条件は $b + r \leq a$ という不等式で表される. 以下, このような r を**制約パラメータ**と呼ぶ.

レイアウト記述において, 制約パラメータおよびレイアウト要素の座標を変数に割り当て, それらの変数を含む式で前述の制約条件を表現し, その式を用いてレイアウト要素の位置を記述することができれば, そのレイアウト記述は, 設計規則や素子サイズに依存しないものとなる.

レイアウト要素の座標を節点に, 座標間の制約条件を有向枝に対応させたグラフを**制約グラフ**と呼ぶが [27], 本章で提案する手法ではこの制約グラフを利用し処理を行う. 通常のコンパクション処理では, 制約パラメータが座標値の差の下限として具体的な数値で与えられるため, その値を重みとして枝に付加する. しかし本章で提案する手法では, レイアウト記述を生成する時点では設計規則や素子サイズが確定できないため, 制約パラメータを具体的な数値で表すことができない. そこで, 制約パラメータ自体を属性として枝に付加することとする. 処理の概要を以下に示す.

1° 入力レイアウトであるセルのデータの階層構造を展開する.

(以後, データはプリミティブ, すなわち矩形の集合体として扱うが, 必要に応じて, 素子やセルのデータをも参照する.)

2° 4.2.1 項に示す手順により制約グラフを構築する.

3° そのグラフから冗長な節点や枝を除去する.

4° 4.2.2 項に示す手順によりレイアウト記述を生成する.

4.2.1 制約グラフの構築

プリミティブの頂点の座標を節点に, その座標間の制約条件を有向枝に対応させた制約グラフ G_H と G_V を, x 方向と y 方向に対してそれぞれ構築する. 抽出する制約の種類は以下のように大きく分類される (実際には, 素子の種類などによりさらに細かく分類される) .

R1: プリミティブ間の最小分離幅の維持 (図 21 A).

R2: プリミティブの最小幅の維持 (図 21 B).

R3: プリミティブ間のオーバーラップ幅の維持 (図 21 C).

R4: 重なっているプリミティブの辺の間の順序関係の維持 (図 21 D).

R5: 配線長の最短化 (図 21 E).

抽出した制約の種類や, その制約が適用されるプリミティブの層の種類は, 4.2.2 項に示す手順において必要となるので, 有向枝に属性として付加しておく.

R2, R5 は, プリミティブ内の向かいあった辺の座標の間に存在する制約であり, 個々のプリミティブを調べれば容易に抽出できる. また, **R3, R4** は, 重なり合うプリミティブの辺の座標の間に存在する制約であり, 重なり合うプリミティブを調べれば容易に抽出できる. **R1** は, レイアウトのコンパクション処理においてよく利用

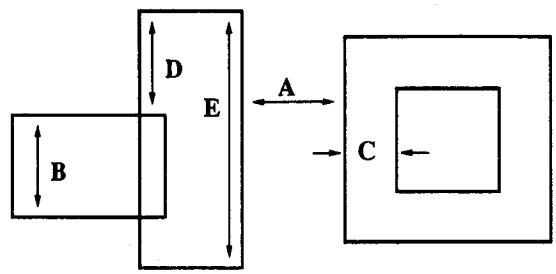
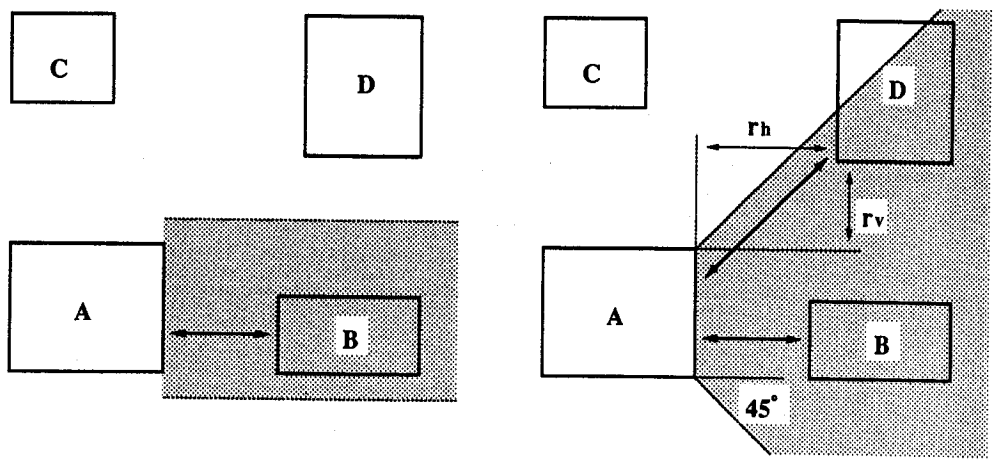


図 21: 制約条件

されるため, 抽出法についても多くの研究がなされているが [30], 本章で提案する手法では, これまでの抽出手法をそのままでは利用できない. これは, 通常のコンパクション処理では必要とならない制約も必要となることがあるためである. これを解決するために shadow-propagation 手法 [30] を修正した手法により制約グラフを構築する.

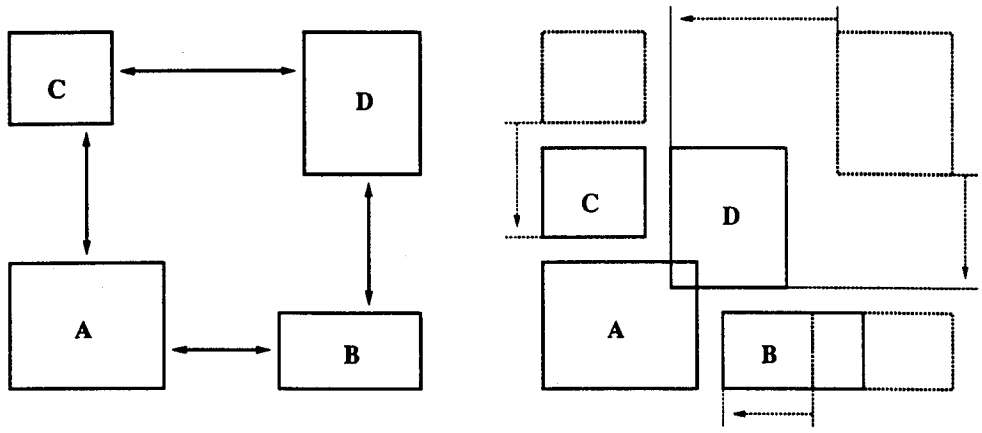
制約グラフを用いるコンパクション処理では, 通常は水平方向と垂直方向の処理を交互に行う. 例えば, 水平方向の処理を先に行い, 続いて垂直方向の処理を行う場合には, まず, y 座標を固定し, x 座標に関する処理を行い, x 座標を更新した後で, x 座標を固定し, y 座標に関する処理を行い, y 座標を更新する. したがって, 例えば x 座標に対する処理を行う際には, 図 22 (a) に示す範囲のプリミティブに対して制約を探索すれば十分である.

しかし, 本章で提案する手法では x 座標を更新, 固定せずに y 座標に関する処理を行わなければならない. なぜならば, 制約パラメータの値が与えられるのはレイアウト記述が生成された後であり, 記述生成処理の途中では座標値を更新できないためである. そのため, 斜め位置にあるプリミティブに対しても制約条件を考慮しておかなければ, 図 23 に示すように, 設計規則が犯されてしまう可能性がある. したがって, 本章で提案する手法では, 制約条件を抽出するときに, 水平および垂直方向だけではなく斜め方向をも同時に探索する. 例えば x 座標に対する処理を行う際には, 図 22 (b) に示す範囲のプリミティブに対して制約を探索する. 具体的には, プリ



(a) コンパクション　　(b) 本手法

図 22: プリミティブ間の制約を探索する範囲



(a) 入力レイアウト　　(b) 出力レイアウト

図 23: 斜め方向に位置するプリミティブ間の制約条件を考慮しない場合

ミティブ間の水平距離 r_h と垂直距離 r_v を比較し, $r_h \geq r_v$ であれば水平方向に制約を, そうでなければ垂直方向に制約を設け, プリミティブ間の距離が制約パラメータ以上に保たれるようにする. y 座標に対しても同様の処理を行えば, 周囲のプリミティブ全てに対して制約条件が考慮されるため, 出力レイアウトが必ず設計規則に従うことが保証される.

以上の考察に基づき, 水平/垂直制約グラフ G_H/G_V を図 24 の手順により構築する.

4.2.2 レイアウト記述の生成

通常のコンパクション処理において, 制約グラフに基づいてレイアウト要素の座標を決定するには, まず制約グラフのクリティカルパスを求め, その経路上にある節点に対応する座標の値を決定し, その後で, 配線長を考慮の上, 残りの座標の値を決定する [29, 30]. しかし, この手順は本章で提案する手法には適用できない. なぜなら, レイアウト記述を生成する時点では制約グラフの枝の重みの値, すなわち制約パラメータの値を確定できず, クリティカルパスを特定できないからである.

レイアウト記述は, 図 2 のように, 座標値を求めるための記述とプリミティブの記述からなる. 後者は自明であるので以下では前者について考察する.

例えば, 図 26 (a) の入力レイアウトに対して図 26 (b) のような制約グラフ G_H が構築されたとする. ここで, 辺 $A, \ldots, G$ の x 座標を表す変数をそれぞれ $x_a, \ldots, x_g$

0° グラフ G_H/G_V を空グラフに初期化する.

1° プリミティブの垂直/水平な各辺に対し, 対応する節点を G_H/G_V に加える.

2° **R1** の制約を表す枝を G_H/G_V に加えるために, 以下の手続きを, 入力レイアウトの左/下から右/上に向かって順に, 各プリミティブ A に対して適用する.

(1) 図 25 に示すように, A の右/上辺から 45 度の扇型の部分を探索する. その部分を A の **shadow** S と呼ぶ.

(2) 左/下辺が S に含まれるような各プリミティブに対し, 再び, その右/上辺から 45 度の扇型の部分を探索する. その部分を A の **double shadow** T と呼ぶ.

(3) もし, その左/下辺が S に含まれ, かつ T に含まれないようなプリミティブ B が存在するのであれば (例えば図 25 におけるプリミティブ B, D), A と B の間で, **R1** の制約条件を考慮する必要がある. したがって, そのような条件を満たすすべての B に対して以下の手続きを適用する.

(i) A の右/上辺と B の左/下辺に対応する節点 a'' と b' に対し, 有向枝 (a'', b') を G_H/G_V に加える.

(ii) $\mathbf{sep}(a'', b')$ を A と B の層間に定められた最小分離幅を表す変数とすると, 枝 (a'', b') のラベル $Lab(a'', b')$ を $\mathbf{sep}(a'', b')$ と定義する.

3° **R2** または **R5** の制約を表す枝を G_H/G_V に加えるために, 以下の手続きを各プリミティブ A に対して適用する.

(1) A の左/下辺と右/上辺に対応する節点 a' と a'' に対し, 有向枝 (a', a'') を G_H/G_V に加える.

(2) $\mathbf{wid}(a', a'')$, $\mathbf{len}(a', a'')$ をそれぞれ A の幅, 高さを表す変数とすると, 枝 (a', a'') のラベル $Lab(a', a'')$ を, A の向きに応じて $\mathbf{wid}(a', a'')$ もしくは $\mathbf{len}(a', a'')$ と定義する.

4° **R3** または **R4** の制約を表す枝を G_H/G_V に加えるために, 以下の手続きを, **R3** または **R4** の制約条件を考慮する必要がある二辺に対して適用する.

(1) その二辺に対応する節点 u, v に対し, 有向枝 (u, v) を G_H/G_V に加える.

(2) $\mathbf{d}(u, v)$ を **R3** または **R4** の制約パラメータを表す変数とすると, 枝 (u, v) のラベル $Lab(u, v)$ を $\mathbf{d}(u, v)$ と定義する.

図 24: 制約グラフ G_H/G_V の構築手法の概要

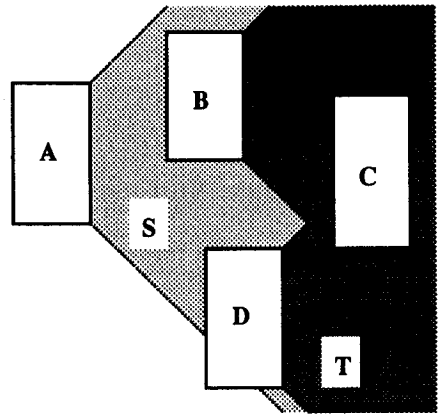
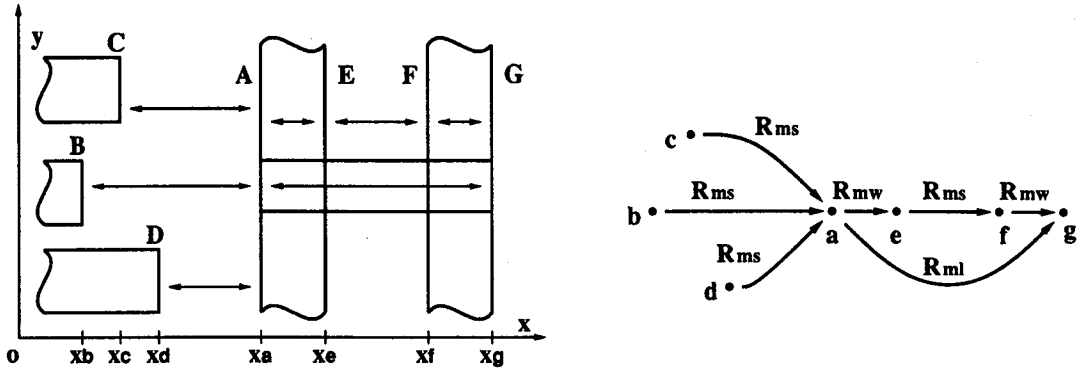


図 25: shadow S と double shadow T の伝搬



(a) 入力レイアウト　　(b) 制約グラフ

図 26: 複数の制約条件

とし, G_H において辺 $A, \ldots, G$ に対応する節点をそれぞれ $a, \ldots, g$ とする. また, R_{ml}, R_{ms}, R_{mw} がそれぞれ, 金属層配線の配線長の最短化, 金属層配線の最小幅の維持, 金属層のプリミティブ間の最小分離幅の維持という制約条件の制約パラメータを表しているとし, 枝 (a,g), (b,a), (c,a), (d,a), (a,e), (e,f), (f,g) のラベルがそれぞれ R_{ml}, R_{ms}, R_{ms}, R_{ms}, R_{mw}, R_{ms}, R_{mw} であるとする. もし, すべての辺を左詰めにするのであれば, 各辺の x 座標は図 27 (a) に示す記述により計算できる. しかし, 図 28 (b) のように配線幅を細くするためには, さらに x_f を再計算するために, 図 27 (b) に示す記述を追加しなければならない.

レイアウト全体の記述を得るためには, 同様にして, すべての座標変数に対して順番に具体値を代入しなければならない. しかし, 値を変数に代入する順序には注意を

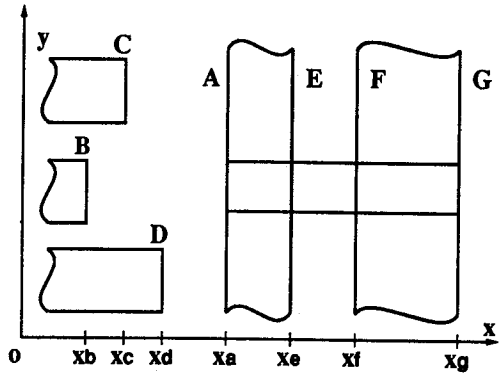
```
...
1:  Xa = Xb + Rms;
2:  Xa = max(Xa, Xc + Rms);
3:  Xa = max(Xa, Xd + Rms);
4:  Xe = Xa + Rmw;
5:  Xf = Xe + Rms;
6:  Xg = Xf + Rmw;
7:  Xg = max(Xg, Xa + Rml);
...
```

(a) x 座標を計算するためのレイアウト記述

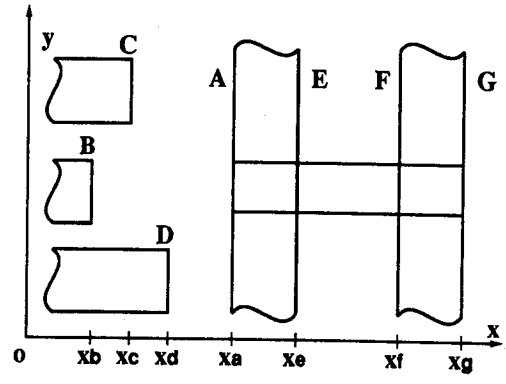
```
8:  Xf = Xg - Rmw;
```

(b) 配線幅を細くするために追加すべきレイアウト記述

図 27: 図 26 に対するレイアウト記述



(a) 図 27 (a) から再生成されるレイアウト



(b) 図 27 (a), (b) から再生成されるレイアウト

図 28: 図 27 から再生成されるレイアウト例

要する. なぜならば, 代入される値を計算する式に含まれるすべての変数は, 代入操作以前に定義されていなければならないからである. 例えばこの例の場合, x_a と x_f は x_g に対する代入以前に定義されていなければならない.

この例からもわかるように, レイアウト全体の記述を得るためには

(i) 各座標変数の値をどのようにして決定するか

(ii) 変数に値を代入する順序をどのようにして決定するか

という 2 つの問題を解決しなければならない. そのために, ここでは, 制約グラフ

G_H/G_V の最小木 T_H/T_V を用いた発見的手法を提案する. この手法では

> 枝 (u,v) が T_H/T_V に含まれているならば, u と v に対応する 2 つの辺の間の空間は, 他の空間に対し, 優先的に最小化されるべきである.

という基本方針を定めることにより, 座標変数の値を矛盾なく決定する. 図 29 において, 太い矢印は T_H/T_V に含まれる枝を, 細い矢印は T_H/T_V に含まれない枝を表しているとする. 例えば図 29 (a) の場合, $(u,v) \in T_H$, $(v,w) \notin T_H$ より, U と V の間の空間を V と W の間の空間よりも優先して最小化する.

各制約パラメータに対して, 空間を最小化する優先度を表す値 (以下, **優先順位数**と呼ぶ) を定義し, 各枝に対しその枝の表す制約条件に応じた優先順位数を付加する. 次に, その木に含まれる枝の優先順位数の総和を最小化するような最小木 T_H/T_V を求め, G_H/G_V および T_H/T_V を用いて, 以下に述べるような方針に基づきレイアウト記述を生成する.

(u,v) が T_H/T_V に含まれるとき, (u,v) に対応する空間を他の空間に優先して最小化して良いという方針より, 次のような指針が導かれる:

G1: u, v に対応する座標変数をそれぞれ x_u, x_v とする. もし, 既に x_u に具体的な値が代入されていれば, x_u と x_v の差を最小化するような値を x_v に割り当てて良い.

例えば図 29 (a) の場合, x_u の値が既に定まっているなら, x_w の値に関わらず x_v の値を定めることができる.

一方, 制約条件を守るためには, 次のような指針を守らなければならない:

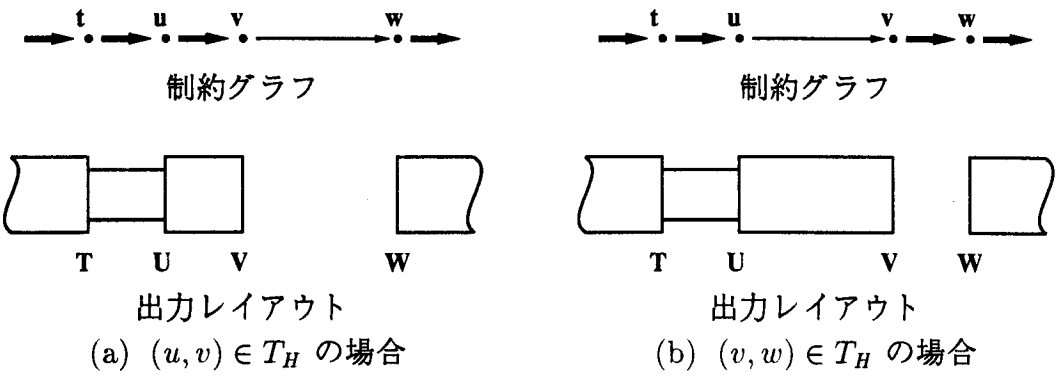


図 29: 空間の最小化に関する優先順位

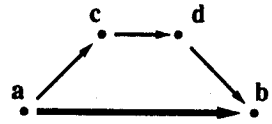
G2: a, b が G_H/G_V の節点で, それぞれ対応する座標変数が x_a, x_b とする. もし G_H/G_V 上に a から b への有向道が存在し, 既に x_a に具体的な値が代入されているのであれば, x_b に割り当てられる値は, 少なくとも, 有向道上の枝の持つ制約パラメータの総和以上, x_a の値より大きくなければならない.

例えば図 30 の場合, もし, x_a の値が定まり, **G1** にしたがって x_b の具体値を決定しようとするときには, a, c 間, c, d 間および d, b 間の制約条件も考慮しなければならない.

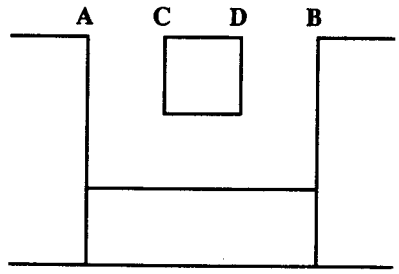
これらの指針を考慮し, G_H/G_V からレイアウト記述を生成するための手続きを構築した. その手続きを図 31 , 図 32 に示す.

図 31 , 図 32 に示す手続きにおいて, 節点 v に付けられた 'locked' というラベルは, v に対応する変数に対して既に適当な値が代入されていることを意味する. また, 節点 v に付けられた 'unlocked' というラベルは, v に対応する節点に対して, (他の節点の制約条件を満たすために) 既に仮の値が代入されているが, 後で適当な値を再代入する可能性があることを意味する. また, 枝 (v, u) が T_H/T_V に含まれているとし, v が 'locked' とラベル付けされ, かつ, u がラベル付けされていなければ, u を**代入可能節点**と呼ぶ.

手続き WRITE_A_VERTEX(v) は, ラベル付けされていない節点 v に対し, 'locked',



(a) 制約グラフ



(b) 出力レイアウト

図 30: a と b の間に有向道が存在する場合

```
    procedure GENERATE_LAYOUT_DESCRIPTION:
 1: begin
 2:     その木に含まれる枝の優先順位数の総和を最小化するような最小木
          T_H/T_V を求める;
 3:     G_H/G_V に含まれる任意の節点 r を選び, 基準節点とする;
 4:     x_r を r に対応する座標変数とする;
 5:     r に 'locked' というラベルを付け, レイアウト記述 LD に対して記述
          "x_r = 0" を追加する;
 6:     L:
 7:     for 'locked' とラベル付けされていない節点 v do
 8:         v のラベルを取り除く;
 9:     while 代入可能節点 v が存在する do
10:         WRITE_A_VERTEX(v);
11:     if 'locked' とラベル付けされていない節点が存在する then begin
12:         for G_H/G_V に含まれる有向枝 (u,v) do
13:             (u,v) を (v,u) に置換する;
14:         goto L;
15:     end
16: end
```

図 31: レイアウト記述生成手続き (1)

'unlocked', 'unspecified' のいずれかのラベルを付け, v に関する記述を生成するサブルーチンである. もし, 手続き WRITE_A_VERTEX(v) が呼ばれたとき, ラベル付けされていない節点 u から v への有向枝があれば, u は再帰的に手続き WRITE_A_VERTEX を呼び出すことによりラベル付けされる.

手続き GENERATE_LAYOUT_DESCRIPTION はメインルーチンである. 最初に, 基準節点 r に関する記述が出力され, r に 'locked' というラベルが付けられる. 続いて, 他の節点に対応する座標変数が r を基準として定義され, 記述として出力される. もし, 'locked' というラベルを付けられていない節点が残っていれば, G_H/G_V 内のすべての有向枝の向きを反転し, 上に述べた手続きを繰り返す.

4.3 実験結果

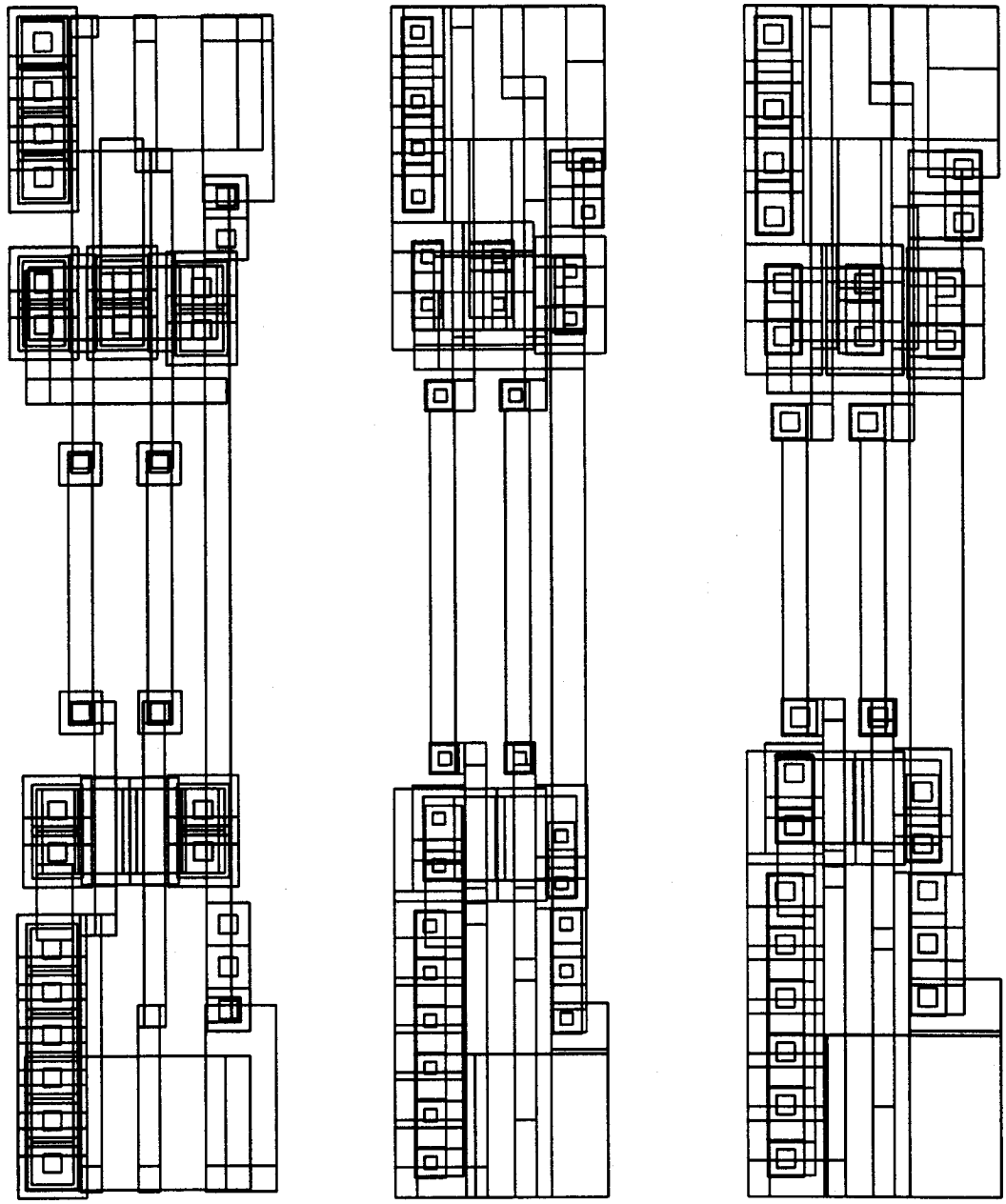
本章で提案した手法, ならびにレイアウト記述用言語処理系を UNIX ワークステーション (SUN SPARC station 2) 上で C 言語を用いて実現し, 実際のレイアウトに

```
      procedure WRITE_A_VERTEX(v):
17:   begin
18:       節点 v に 'checking' というラベルを付ける;
19:       for 有向枝 (u,v) do begin
20:           if u がラベル付けされていない then
21:               WRITE_A_VERTEX(u);
22:           if u が 'locked' か 'unlocked' とラベル付けされている then begin
23:               x_v と x_u をそれぞれ v と u に対応する座標変数とする;
24:               if LD が v に関する記述を含まない then
25:                   x_u と x_v の大小関係を考慮し,
                        LD に記述 "x_v = x_u + Lab(u,v)" もしくは
                        "x_v = x_u - Lab(u,v)" を追加する;
26:               else
27:                   x_u と x_v の大小関係を考慮し,
                        LD に記述 "x_v = max(x_v, x_u + Lab(u,v))" もしくは
                        "x_v = min(x_v, x_u - Lab(u,v))" を追加する;
28:           end
29:           if LD が v に関する記述を含まない then
30:               v に 'unspecified' というラベルを付ける;
31:           else if LD が T_H/T_V に含まれる有向枝 (u,v) に関する記述を含
                  む then
32:               v に 'locked' というラベルを付ける;
33:           else
34:               v に 'unlocked' というラベルを付ける;
35:       end
36:   end
```

図 32: レイアウト記述生成手続き (2)

対して実験を行った.

まず, 本章で提案した手法を用いて図 33 (a) のレイアウトからレイアウト記述を自動生成した. その際, 処理対象としてスタンダードセルを想定し, 上半分と下半分を別々に処理した. レイアウトに含まれる矩形の数は 153 であり, 記述の生成処理には 12 秒を要した. さらに第 3 章で述べた言語処理系を用いて, 生成した記述から図 33 (a) とは異なる素子サイズ, 最小分離幅のレイアウトを生成した. そのレイアウトを図 33 (b), (c) に示す. (b) と (c) は, コンタクトの大きさに対応する制約パラメータに異なる値を与えて生成したものであるが, 与えたパラメータ値に応じて, 正



(a) 入力レイアウト　(b) 出力レイアウト (1)　(c) 出力レイアウト (2)

図 33: 実験結果

しくレイアウトが再生成できた.

4.4 結言

本章では, 機能セルのレイアウトパターンをレイアウト記述に自動変換する手法について述べた. まず, 入力レイアウトからレイアウト要素間に存在する制約条件を

抽出し, 制約グラフを構築する手法について考察した. 次に, 制約グラフの最小木に基づき, 最適化を考慮したレイアウト記述を自動生成する手法について考察した. さらに, 提案する手法を実現し, レイアウト再生成実験を行い, 提案する手法の有効性を示した.

本章で提案した手法によって得られたレイアウト記述では, 素子サイズや最小分離幅が制約パラメータにより表現されているため, それらの具体値を与えるだけで利用目的にあったレイアウトを生成できる. 従来, このようなレイアウト記述が必要な場合にはテキストエディタによる人手設計に頼らざるをえなかったが, 本章で提案した手法を活用すれば, グラフィックエディタを利用した設計や既存のレイアウトの再利用などが可能となる.

5 線形計画法に基づいたセルレイアウトの再生成手法

5.1 緒言

第 4 章では, 既存のレイアウトからレイアウト記述を自動生成する手法について述べたが, その手法では, レイアウト記述により面積や配線長の最小化を行うため, 生成されるレイアウト記述の量が多いという問題点がある. これに対し本章では, 線形計画法を導入することにより, レイアウト記述の量を削減し, かつ, 最適なレイアウトを再生成する手法について考察する.

本章で提案する手法では, 配線長の最短化等の最適化処理を, 記述から出力レイアウトを再生成するときに行う. したがって, レイアウト記述生成時に最適化について考慮する必要がなく, レイアウト記述から冗長な部分を取り除くことができる.

提案する手法の概要を図 34 に示す. 以下では, まず, レイアウト記述から線形計画問題を導出する方法について考察し, レイアウト要素の抵抗率と素子の信号遅延に与える影響度に基づいた目的関数の設定法を考案する. 次に, 効率よく線形計画問題の解を求めるために, グラフ理論に基づくシンプレックス法について考察する. さらに, 提案する手法を実現し, レイアウト再生成実験を行うことにより, 提案する手法が有効であることを示す.

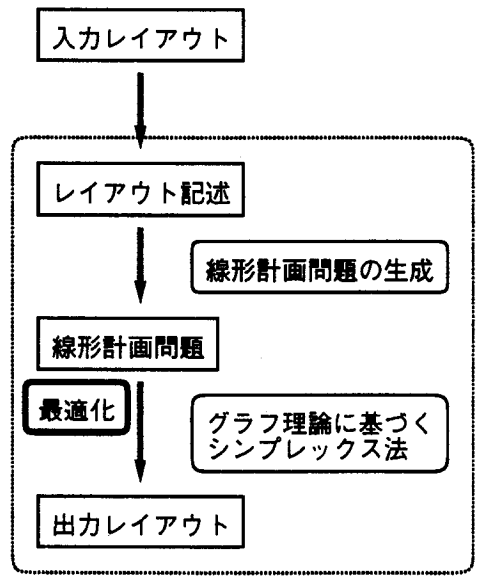


図 34: 記述による最適化に基づいたレイアウトの再生成手法

5.2 処理手法

レイアウト設計における一つの主要な目標は, レイアウト要素の幅や長さ等を与えられた制約条件の範囲内で最小化することであるが, これはレイアウト要素の座標を含む関数の最小化問題に置き換えることができる. また, 前述のように, レイアウト要素間の制約条件は 1 次不等式で表現できるので, レイアウト要素の座標決定問題は線形計画問題に帰着できる [35, 36].

このことに着目し, 本章ではレイアウト変換に線形計画法を導入する. 線形計画法を用いれば, 第 4 章で述べた手法のように, 同じ制約に関する記述を繰り返し出力する必要がなく, レイアウト記述の量を削減することができる.

具体的には, 単に, レイアウト記述の各行が制約グラフの各辺に対応している, 図 27 (a) のような記述からレイアウトを再生成する手法について考察する. まず, レイアウト記述を線形計画問題として定式化し, 次に, グラフ理論に基づくシンプレックス法によりレイアウト要素の座標を求める.

5.2.1 線形計画問題の生成

線形計画問題は連立一次不等式と目的関数により構成される. 以下では, レイアウト記述から連立一次不等式と目的関数を導く方法について考察する.

まず, 記述から連立一次不等式を導く. これは

(i) レイアウト記述の各行は制約グラフの各枝を表す

(ii) 制約グラフの各枝は 4.2 節で述べたように制約条件, すなわち一次不等式を表す

ことを考慮すると容易である. すなわち, レイアウト記述の各行をそれぞれ一次不等式に変換すれば, 連立一次不等式を得ることができる. 例えば図 27 (a) の記述からは, 連立一次不等式

$$\begin{cases} x_a \geq x_b + R_{ms} \\ x_a \geq x_c + R_{ms} \\ x_a \geq x_d + R_{ms} \\ x_e \geq x_a + R_{mw} \\ x_f \geq x_e + R_{ms} \\ x_g \geq x_f + R_{mw} \\ x_g \geq x_a + R_{ml} \\ \cdots \end{cases} \tag{1}$$

が導かれる.

次に, 記述から目的関数を構成する. 本章では,

> 各プリミティブの幅や長さに, そのプリミティブの層や制約の種類に対応した重み (以下, コストと呼ぶ) を乗じたものの総和

を, 線形計画法の目的関数とする. このためには, プリミティブの幅や長さを座標変数により表現する必要があるが, これは

(iii) レイアウト記述に含まれる制約パラメータが, プリミティブの層や制約の種類を表している

ことを考慮すると容易である.

例えば, Rmw というパラメータが金属層配線の配線幅を表していることを考慮すると, 図 27 (a) の 4 行目の

$$x_e = x_a + R_{mw}; \tag{2}$$

という記述から,

> 辺 A と辺 E が金属層配線の幅方向の 2 つの座標を表している

ことがわかる. したがって, その配線の幅は $x_e - x_a$ と表現できる.

同様にして, レイアウト記述全体から, すべてのプリミティブの幅や長さを表現する式を抽出し, コストを乗じて総和を求めることにより, 目的関数を得ることができる. 例えば図 27 (a) の記述から目的関数

$$\begin{aligned} Z &= C_{ms}(x_a - x_b) \\ &+ C_{ms}(x_a - x_c) \\ &+ C_{ms}(x_a - x_d) \\ &+ C_{mw}(x_e - x_a) \\ &+ C_{ms}(x_f - x_e) \\ &+ C_{mw}(x_g - x_f) \\ &+ C_{ml}(x_g - x_a) \\ &+ \cdots \end{aligned} \tag{3}$$

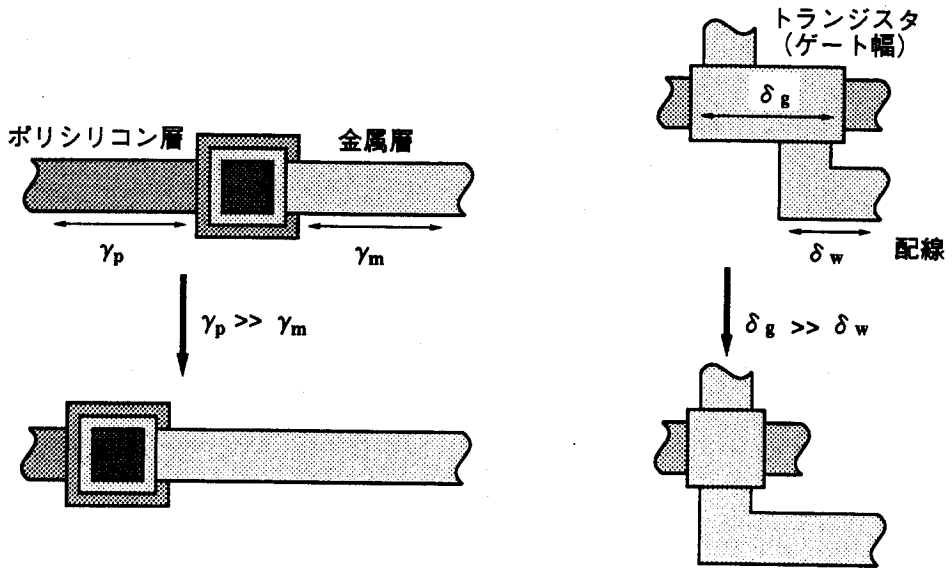
が導かれる. ここで C_{ms}, C_{mw}, C_{ml} はそれぞれ R_{ms}, R_{mw}, R_{ml} に対応したコストである. このコストを定義する方法については 5.2.2 項で述べる.

5.2.2 目標関数の設定

レイアウト問題に線形計画法を適用する場合, 目的関数を総配線長や素子の大きさの総和とすることが多い [35]. しかし, 本論文で対象とするレイアウト生成では金属層やポリシリコン層, 拡散層といった何種類もの層を同時に扱う必要があるので, 信号遅延に大きく影響する電気抵抗率を考慮し, 抵抗率の大きな層の配線長ほど優先的に短くするような目的関数を設定する必要がある. そのために, まず各層 l に対して, その層の抵抗率に比例したパラメータ γ_l を定義する. 例えば, ポリシリコン層は金属層の数百倍の抵抗率を持つので, 図 35 (a) に示すように, ポリシリコン層に対するパラメータ γ_p は金属層に対するパラメータ γ_m の数百倍の値とする.

さらに, プリミティブに対する制約の種類が異なると, その大きさの変化が回路の動作特性に及ぼす影響も異なることを考慮しなければならない. 例えば, 図 35 (b) のようにポリシリコン層がトランジスタのゲートと配線に使われている場合, トランジスタのゲート幅の方が配線長よりも信号遅延に及ぼす影響が大きい. したがって, 各制約に対し, その制約を持つプリミティブの単位長さ当たりの影響度に比例したパラメータ δ_c を導入する.

制約パラメータ R_{lc} に対して, それに対応するパラメータ γ_l と δ_c を用いて, コス



(a) 層に対するパラメータ　　(b) 制約の種類に対するパラメータ

図 35: コストパラメータ

ト C_{lc} を

$$C_{lc} = \gamma_l \times \delta_c \tag{4}$$

と定義する.

5.2.3 グラフ理論に基づくシンプレックス法

線形計画法では, 問題の規模の増大, すなわち変数や制約条件の数の増加にともない, 計算時間が著しく増加する. 従って, 大規模なレイアウトに対して線形計画法を適用するためには, 効率のよい算法が要求される. このような目的に対し, [31] では, 制約グラフで表現される線形計画問題に対し, 行列を扱わずにシンプレックス法を実行する効率のよいアルゴリズムを提案している. このアルゴリズムの概要を図 36 に示す. ただし, 図 36 において, 長さ $L(u,v)$ とは有効枝 (u,v) に属性として付加されている制約パラメータの具体値である. また, コスト $C(u,v)$ としては 5.2.1 項で定義したものを用いる.

この手続き中の **2°** における T_H/T_V は図 37 に示すアルゴリズム [37] により求められる. ここで s および t は, レイアウト全体の左/上端および右/下端に対応する

1° 与えられた線形計画問題を基に, 各枝 (u,v) が長さ $L(u,v)$ とコスト $C(u,v)$ の具体値を持つような, 制約グラフ G_H/G_V を構築する.

2° すべての基本ループの値が非正となるような木 T_H/T_V を求める. ここで**基本ループの値**とは, 枝 (u,v) が基本ループに順向きに含まれるとき $+L(u,v)$ を, 逆向きに含まれるとき $-L(u,v)$ を加算して求めた, そのループに含まれる枝の長さの代数和である. [このような T_H/T_V は実行可能解に対応する.]

3° すべての基本ループの値が非正であり, かつ, すべての基本カットセットの値が非負となるように, T_H/T_V に対し, 繰り返し木の初等変換を行う. ここで**基本カットセットの値**とは, 枝 (u,v) が基本カットセットに順向きに含まれるとき $+C(u,v)$ を, 逆向きに含まれるとき $-C(u,v)$ を加算して求めた, そのカットセットに含まれる枝のコストの代数和である. [このような T_H/T_V は最適解に対応する.]

4° 基準となる節点から各節点 u までの木 T_H/T_V 上における距離 $d(u)$ を計算する. 各プリミティブの垂直/水平な辺の x/y 座標は, 対応する節点 u の距離 $d(u)$ として求められる.

図 36: グラフ論的シンプレックス法の概要

```
     procedure LongestPathTree1:
 1:  begin
 2:      与えられた制約グラフ G_H/G_V において, 節点 s から節点 t への有向
           道 P を含むような木 T を求める;
 3:      L:
 4:      for 補木 T̄ に含まれる枝 e_c do begin
 5:          e_c により定まる基本ループ C を求める;
 6:          基本ループ C の値 l(C) を計算する;
 7:          if l(C) > 0 then begin
 8:              dir(e_i) = −1 であるような C の枝 e_i の集合 C_N を求める;
 9:              if P が C_N のすべての枝を含む then begin
10:                  C_N に含まれる任意の枝 e_b を選ぶ;
11:                  T ← T − {e_b} + {e_c}
                       (T に対して木の初等変換を行う);
12:                  T の変換に応じて P を修正する;
13:              end else begin
14:                  C_N に含まれ P に含まれない任意の枝 e_b を選ぶ;
15:                  T ← T − {e_b} + {e_c}
                       (T に対して木の初等変換を行う);
16:              end
17:              goto L;
18:          end
19:      end
20:      return T;
21:  end
```

図 37: 最長経路木を求める手続き

節点である. また, $dir(\cdot)$ はそれぞれ基本ループ C に対する枝の向きを表し, 順向きのとき $+1$, 逆向きのとき -1 である.

この手続きをさらに高速化するために, 図 37 の 2 行目における木 T として, s を根とする根付木を選ぶことにする. このとき, 以下の命題が成り立つ.

(i) 補木枝 e_c と共通の終点を持つ木枝 e_t が一つだけ存在する.

(ii) その木枝 e_t は e_c によって定まる基本ループに逆向きに含まれる.

(iii) e_t が P に含まれるとき, 基本ループに逆向きに含まれる他の枝もすべて P に含まれる.

(iv) 補木枝 $e_c = (u, v)$ の長さを $len(e_c)$, s から木枝を通って節点 u, v に到る経路の長さをそれぞれ $d(u)$, $d(v)$ とすると, 補木枝 e_c によって定まる基本ループ C の値 $l(C)$ は,

$$l(C) = len(e_c) + d(u) - d(v) \tag{5}$$

により求めることができる.

(ii), (iii) により, (i) における e_t が P に含まれるか否かにかかわらず, e_t は常に図 37 の 10 行目もしくは 14 行目で選ばれる e_b の候補となる. さらに, e_t と e_b により木の初等変換を行った結果得られる木も, やはり s を根とする根付木である. したがって, (i) における e_t を 10, 14 行目の e_b として選ぶことにより,

(i) 9 行目の処理

(ii) 2 および 12 行目の P を求める処理

を省略することができ, さらに (iv) により 6 行目の $l(C)$ の計算を簡単化できる.

以上の考察に基づき, 図 37 の手続きを図 38 のように効率化できる.

```
    procedure LongestPathTree2:
 1: begin
 2:     与えられた制約グラフ G_H/G_V において, 節点 s を根とする根付き木
          T を求める;
 3:     L:
 4:     for 補木 T̄ に含まれる枝 e_c = (u, v) do begin
 5:         l(C) ← L(u, v) + d(u) − d(v)
              (e_c により定まる基本ループ C の値を計算する);
 6:         if l(C) > 0 then begin
 7:             v を終点に持つ木 T の枝 (u', v) を求める;
 8:             T ← T − {(u', v)} + {e_c}
                  (T に対して木の初等変換を行う);
 9:             goto L;
10:         end
11:     end
12:     return T;
13: end
```

図 38: 図 37 を改良した手続き

5.3 実験結果

本章で提案した手法を UNIX ワークステーション (SUN SPARC station 2) 上で C 言語を用いて実現し, 実際のレイアウトに対して実験を行った.

まず, 本章で提案した手法を用いて, 4 つの既存のレイアウトをレイアウト記述に変換した. 次に, 目的関数を操作することにより素子の大きさが制御できることを確認するために, トランジスタのゲート幅に関するパラメータ δ を変えてレイアウト記述からレイアウトを再生成した. その一例として, 図 39 に入力となった既存のレイアウト, 図 40 に生成されたレイアウトを示す. 図 40 では図 40 (b) のレイアウトの方がトランジスタのゲート幅が大きくなっているのがわかる.

さらに, 第 4 章で述べた手法を用いて既存のレイアウトをレイアウト記述に変換し, それらの記述からレイアウトを再生成し, レイアウト記述量および処理時間に関して本章で提案した手法と比較した. その結果を表 3 , 表 4 に示す. ただし, レイアウト再生成に関しては, 5.2.3 項で述べた手法と同様に座標計算のみを行うプログラムを新たに作成し, 実験を行った. 第 3 章で述べた手法ではデータベース処理などに多くの時間が費やされるため, そのまま 5.2.3 項で述べた手法と比較するのは適当ではないからである.

本章で提案した手法では, 第 4 章で述べた手法の約 40 – 65% の記述量でレイアウト変換を実行できた. 実行時間に関しては, 第 4 章で述べた手法と比較して, 記述生成に必要な時間は減少し, レイアウト再生成に必要な時間は増加しているが, いずれも, 実用的な計算時間で実行できた.

第 4 章で述べた手法では記述生成時に, 本章で提案した手法ではレイアウト再生成時に最適化処理を行うことを考慮すると, これらの手法の違いによる記述量や実行時間の違いは容易に理解できる.

5.4 結言

本章では, 線形計画法を用いたレイアウト再生成手法について述べた. まず, レイアウト記述から線形計画問題を導出する方法について考察し, レイアウト要素の抵抗率と素子の信号遅延に与える影響度に基づいた目的関数の設定法を考案した. 次に, 効率よく線形計画問題の解を求めるために, グラフ理論に基づくシンプレックス法について考察し, 根付き木を用いることにより従来の手法を高速化した. さらに, 提案

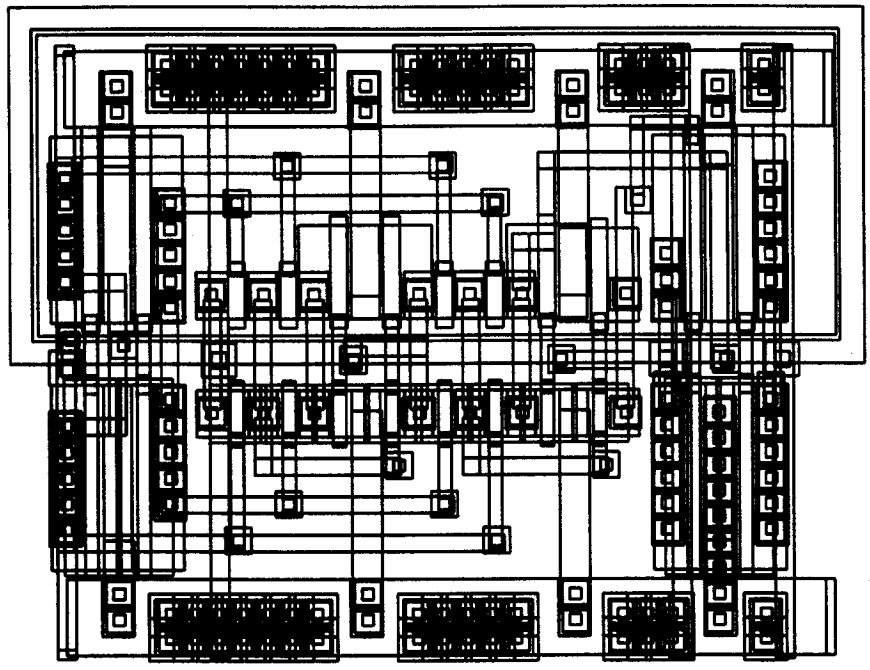


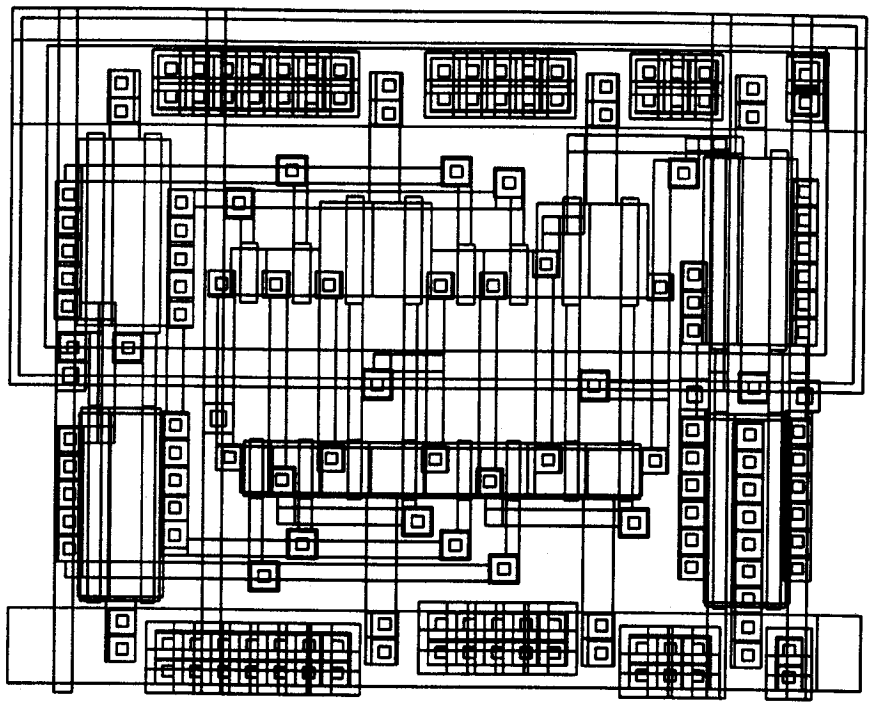
図 39: 入力レイアウト

表 3: 抽出された制約グラフ

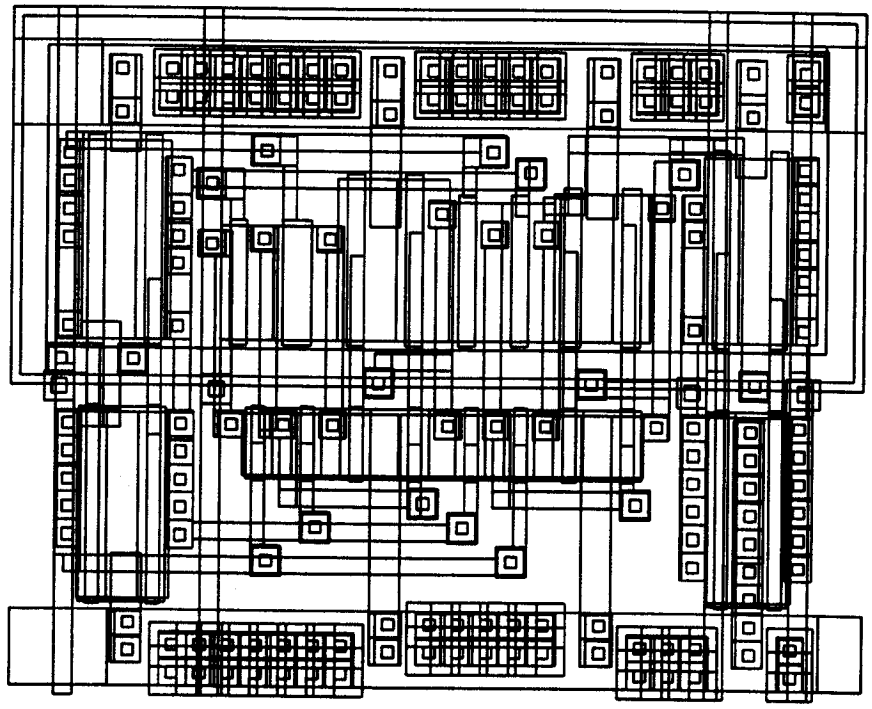
	矩形数	頂点数 (G_H/G_V)	枝数 (G_H/G_V)
#1 (2 入力 NAND)	153	151/ 256	374/ 773
#2 (複合ゲート)	300	400/ 416	1327/1637
#3 (ラッチ)	695	872/ 891	3360/4136
#4 (フリップフロップ)	1057	1094/1685	4194/6533

表 4: 処理時間と記述量

	記述による最適化			線形計画法による最適化		
	処理時間 (秒)		記述量	処理時間 (秒)		記述量
	記述生成	再生成	(行)	記述生成	再生成	(行)
#1	11.9	1.3	2992	11.3	1.7	1300
#2	32.8	2.3	5146	30.5	8.4	3264
#3	207.0	7.6	17867	196.1	92.4	8191
#4	1813.9	10.4	23244	1792.0	139.2	11784



(a) 出力レイアウト (1)



(b) 出力レイアウト (2)

図 40: 出力レイアウト

する手法を実現し, レイアウト再生成実験を行い, 提案する手法の有効性を示した.

本章で提案した手法により, 従来よりも少ないレイアウト記述から実用レベルのレイアウトを得ることができた. しかも, グラフ論的考察によるシンプレックス法を用いることにより, 実用的な計算時間で最適レイアウトを生成することができた. 本章で提案した手法は既存のレイアウトの再利用にだけでなく, 通常のシンボリックレイアウトにおけるコンパクション処理等に対しても応用できると考えられる.

6 結論

本論文では, レイアウト記述言語に基づくレイアウト再生成手法について述べた. 以下に, 本研究で得られた成果と, 今後に残された課題について述べる.

第 2 章では, 本論文で扱う問題を明確にするために, レイアウト記述について概説し, 入力レイアウトの構造や入力レイアウトに対する制限, および出力レイアウトに要求される特徴について考察した.

第 3 章では, C 言語に基づくレイアウト記述言語を構築した. これは, 3 種類のレイアウト記述用関数を C 言語に付加することにより, C 言語にレイアウト記述能力を持たせたものである. したがって, レイアウトを記述する際に, C 言語の変数, データ構造, および制御文などを用いることができる. これらを利用すれば, 設計規則や素子サイズの具体値に依存しない形式でレイアウトを記述でき, また, 規則的な構造を持つレイアウトを簡潔に記述できる. さらに, C 言語に基づくため, 言語の拡張性が高い, C 言語を習得している設計者には容易に理解できる, レイアウト記述言語処理系を容易に構築できる, といった特徴を持つ.

第 4 章では, 記述による最適化に基づいたセルレイアウトの再生成手法について述べた. まず, 入力レイアウトから制約グラフを抽出する手法について考察し, 続いて, 制約グラフから最小木に基づきレイアウト記述を再生成する手法について考察した. ここで提案した手法を用いれば, 既存のレイアウトデータライブラリやグラフィックエディタで作成したレイアウトデータをレイアウト記述に自動的に変換することができ, さらに, その記述にパラメータを与えることにより, 異なる設計規則等に対応したレイアウトを再生成することができる. したがって, 集積回路の製造技術の進歩に伴う設計規則の変更処理や, シリコンコンパイラにおけるセル生成処理への応用が期待できる. 得られたレイアウト記述では自動的に最適化が行われるため, 第 3 章で述べたような最適化機能を持たない処理系でも実用的なレイアウトを得ることができる. 実際に処理系を構築し, レイアウト再生成実験を行うことにより, 提案した手法の有効性を確認した.

第 5 章では, 線形計画法に基づいたセルレイアウトの再生成手法について述べた. まず, レイアウト記述から線形計画問題を導出する手法について考察し, 続いて, 効率よく線形計画問題の解を求めるために, グラフ理論に基づくシンプレックス法について考察した. ここで提案した手法を用いれば, 第 4 章で述べた手法と同等のこ

とを, 少量のレイアウト記述により実現できる. したがって, 第 4 章で述べた手法と同様に, 設計規則の変更処理やシリコンコンパイラへの応用が期待できる. また, グラフ理論に基づくシンプレックス法を用いれば, 制約グラフからレイアウト要素の座標を求めることができるが, これは, コンパクション処理への応用が期待できる. 実際に処理系を構築し, レイアウト再生成実験を行うことにより, 提案した手法の有効性を確認した. 第 4 章で述べた手法と第 5 章で述べた手法を比較すると, レイアウト再生成の実行時間に関しては前者が, レイアウト記述の量に関しては後者が優れている. これを考慮すれば, 状況に応じてより適した手法を選択できる.

本論文では, プリミティブの形状を, 各辺がチップの底辺に対して水平もしくは垂直な矩形のみと限定したが, 最近では, 斜め向きの矩形を用いたレイアウトが実用化されつつある. これを処理できるように, 入力レイアウトに対する制限を取り除くことは今後の課題である. また, より大きなレイアウトを処理するためには, レイアウトの階層構造や反復構造を抽出してレイアウト記述により表現することが必要であると考えられるが, これを実現する手法の構築も今後の課題である.

謝辞

本研究の全過程を通じて, 直接理解ある御指導を賜わり, つねに励ましていただいた白川功教授に心から感謝する. また石浦菜岐佐講師には適切な御指導, 御討論をいただき, 心から感謝する.

大学院前期, 後期両課程に在学中ならびに大阪大学工学部に在職中に御指導, 御教示を賜わった電子工学教室吉野勝美教授, 濱口智尋教授, 西原浩教授, 尾浦憲治郎教授, 児玉慎三教授, 情報システム工学教室寺田浩詔教授, 藤岡弘教授, 西尾章治郎教授, 薦田憲久教授, 鈴木胖教授, 大型計算機センター熊谷貞俊教授, 産業科学研究所溝口理一郎教授, 基礎工学部谷内田正彦教授, 中村勝吾名誉教授, 角所収名誉教授, 塙輝雄名誉教授, ならびに裏克己名誉教授に対し厚く御礼申し上げる.

第 2 章, 第 3 章について適切な御指導, 御教示をいただいた中央大学築山修治教授に厚く感謝する.

本研究に関し, 適切な御教示, 御助言をいただいたシャープ株式会社神戸尚志博士に厚く感謝する.

第 3 章について適切な御教示, 御助言をいただいた大阪府立大学原嶋勝美助手, 第 4 章について適切な御教示, 御助言をいただいたシャープ株式会社長尾明氏, 第 5 章について適切な御教示, 御助言をいただいた日本電気株式会社吉村猛部長に厚く感謝する.

本研究に関し, 基礎工学部荒木俊郎助教授, 大型計算機センター出口弘講師, 沖電気工業株式会社福永茂氏には, 本学大学院に在学中に有益な御助言, 御討論をいただき, 心から感謝する.

筆者の属している白川研究室の尾上孝雄助手, 大学院学生崔溟鎔氏, 豊永昌彦氏 (松下電器産業株式会社より留学中), 山田晃久氏 (シャープ株式会社より留学中), 鈴木等氏 (シャープ株式会社より留学中), 小原隆司氏, 長田岳史氏, 渡辺健司氏, 楊世宗氏, 安齋勝矢氏, 高津正道氏, 正城敏博氏, 山崎年樹氏, 同研究室の鹿島有紀子嬢, 大阪電気通信大学の中道和則氏には種々の面で御協力いただいた. ここに記して感謝する次第である.

参考文献

[1] 小林勉, 須藤常太, 細田泰弘: "LSI-CAD [I]", 電子情報通信学会誌, Vol. 70, No. 12, pp. 1291–1297 (1987).

[2] J. Soukup: "Circuit layout", *Proc. IEEE*, Vol. 69, No. 10, pp. 1281–1304 (1981).

[3] A. Sangiovanni-Vincentelli: "Automatic layout of integrated circuits", *Design Systems for VLSI Circuits*, pp. 113–196, Martinus Nijhoff Publishers (1987).

[4] 上田和宏, 須藤常太: "LSI-CAD [II]", 電子情報通信学会誌, Vol. 71, No. 1, pp. 80–87 (1988).

[5] E. S. Kuh and T. Ohtsuki: "Recent advances in VLSI layout", *Proc. IEEE*, Vol. 78, No. 2, pp. 237–263 (1990).

[6] 重弘裕二, 白川功, 長尾明, 神戸尚志: "既存のレイアウトからのレイアウト記述の自動生成", 電子情報通信学会技術研究報告, CAS92-41, VLD92-41, DSP92-52 (1992).

[7] Y. Shigehiro, I. Shirakawa, and H. Takahashi: "A Recycling Scheme for Layout Patterns Once Generated for a Fabrication Technology", *Proc. 1992 Joint Technical Conference on Circuits/Systems, Computers and Communications* (JTC-CSCC '92), pp. 282–287 (1992).

[8] 重弘裕二, 白川功, 長尾明, 神戸尚志: "既存のレイアウトからのレイアウト記述の自動生成", 電子情報通信学会秋季大会, A-71 (1992).

[9] 長田岳史, 重弘裕二, 白川功: "線形計画法を用いた最適レイアウト変換手法", 電子情報通信学会秋季大会, A-72 (1992).

[10] 長田岳史, 重弘裕二, 白川功: "線形計画法を用いた最適レイアウト再生成手法", 第 6 回 回路とシステム軽井沢ワークショップ, pp. 55–60 (1993).

[11] Y. Shigehiro and I. Shirakawa: "A Recycling Scheme for Layout Patterns Used in an Old Fabrication Technology", *IEICE Trans.*, Vol. E76-A, No. 6, pp. 886–893 (1993).

[12] Y. Shigehiro, T. Nagata, and I. Shirakawa: "Layout Recycling Dedicatedly for Standard Cells Based on Graph Theoretic Simplex Approach", *Proc. 3rd International Design Automation Workshop* (Russian Workshop '93), pp. 129–136 (1993).

[13] Y. Shigehiro, I. Shirakawa, and T. Kambe: "A Recycling Scheme for Layout Patterns Used in an Old Fabrication Technology", *Proc. 11th European Conference on Circuit Theory and Design* (ECCTD '93), pp. 133–138 (1993).

[14] Y. Shigehiro, T. Nagata, I. Shirakawa, and T. Kambe: "Optimal Layout Recycling Based on Graph Theoretic Linear Programming Approach", *Proc. International Conference on Very Large Scale Integration* (VLSI '93), pp. 1.2.1–1.2.10 (1993).

[15] Y. Shigehiro, T. Nagata, and I. Shirakawa: "Optimal Layout Recycling", *Proc. Synthesis and Simulation Meeting and International Interchange* (SASIMI '93), pp. 255–263 (1993).

[16] 原嶋勝美, 若林謙二, 神戸尚志, 重弘裕二, 白川功: "C 言語によるレイアウト記述の一手法", 電子情報通信学会技術研究報告, CAS87-14 (1987).

[17] 重弘裕二, 白川功, 原嶋勝美, 神戸尚志: "C 言語によるレイアウト記述の一手法", 電子情報通信学会論文誌, Vol. J76-A, No. 4, pp. 618–627 (1993).

[18] R. J. Lipton, S. C. North, R. Sedgewick, J. Valdes, and G. Vijayan: "ALI: A procedural language to describe VLSI layouts", *Proc. 19th Design Automation Conf.*, pp. 467–474 (1982).

[19] P. A. D. Powell and M. I. Elmasry: "The icewater language and interpreter", *Proc. 21st Design Automation Conf.*, pp. 98–102 (1984).

[20] J. M. Mata: "ALLENDE: A procedural language for the hierarchical specification of VLSI layouts", *Proc. 22nd Design Automation Conf.*, pp. 183–189 (1985).

[21] W. E. Cory: "Layla: A VLSI layout language", *Proc. 22nd Design Automation Conf.*, pp. 245–251 (1985).

[22] M. R. Buric and T. G. Matheson: "Silicon compilation environments", *Proc. Custom Integrated Circuits Conf.*, pp. 208–212 (1985).

[23] J. A. Solworth: "GENERIC: A silicon compiler support language", *Proc. 23rd Design Automation Conf.*, pp. 524–530 (1986).

[24] W. Kao, D. Tsou, D. Chi, and M. Movahed-Ezaki: "C3: A system and methodology for the development of module generators", *Proc. Int. Conf. on Computer-Aided Design*, pp. 78–81 (1986).

[25] 神原弘之, 小野寺秀俊, 田丸啓吉: "アナログモジュール自動生成のためのレイアウト記述の一手法", 1989 年電子情報通信学会春季全国大会講演論文集, SA-7-1 (1989).

[26] 佐藤寿倫, 小野寺秀俊, 田丸啓吉: "モジュール・ジェネレータ開発容易化の一手法", 電子情報通信学会技術研究報告, CAS90-100 (1990).

[27] M. Y. Hsueh, "Symbolic layout and compaction of integrated circuits", *ERL Memo. UCB/ERL M79/80*, Univ. of California, Berkeley (1979).

[28] L. Z. Liao and C. K. Wong: "An algorithm to compact a VLSI symbolic layout with mixed constraints", *IEEE Trans. Computer-Aided Design*, Vol. CAD-2, No. 2, pp. 62–69 (1983).

[29] Y. E. Cho: "A subjective review of compaction", *Proc. 22nd Design Automation Conf.*, pp. 396–404 (1985).

[30] D. G. Boyer: "Symbolic layout compaction review", *Proc. 25th Design Automation Conf.*, pp. 383–389 (1988).

[31] 吉村猛: "ネットワーク問題におけるシンプレックス法", 電子情報通信学会論文誌, Vol. J70-A, No. 2, pp. 156–163 (1987).

[32] B. W. Kernighan and D. M. Ritchie: "The C Programming Language", Prentice-Hall (1987).

[33] 神戸尚志, 谷貞宏, 小嶋格, 富田常雄, 森本清巳: “VLSI レイアウト設計のための統合化支援システム” 情報処理学会論文誌, Vol. 31, No. 3, pp. 351–360 (1990).

[34] N. H. E. Weste and K. Eshraghian: “Principles of CMOS VLSI Design”, Addison-Wesley Publishing Company (1985).

[35] 本間祐次, 宮野浩, 梶谷洋司: “線形計画法によるレイアウトコンパクションについて”, 電子情報通信学会技術報告, CAS88-105 (1988).

[36] 奥田亮輔, 佐藤寿倫, 小野寺秀俊, 田丸啓吉: “対称性保持の制約を扱えるレイアウトコンパクションアルゴリズム”, 電子情報通信学会論文誌, Vol. J73-A, No. 3, pp. 536–543 (1990).

[37] 後藤敏, 大附辰夫: “グラフ理論におけるシンプレックス法”, 電子通信学会論文誌, Vol. 57-A, No. 11, pp. 810–817 (1974).