システムＬＳＩ設計技術　97－10
（２０００．９．８）

Slicing-TreeとBox Packingを組み合わせた自動フロアプラン手法

澁谷利行　金澤裕治　横丸敏彦　†高橋猛
(株)富士通研究所　†(株)エンティティ
〒211-8588 川崎市中原区上小田中4-1-1
TEL: (044)754-2663, FAX: (044)754-2664
E-mail: shibu@flab.fujitsu.co.jp

あらまし

Embedded ArrayやGate Arrayにおいて、配置や論理合成ツールに対して、配置位置のdirectiveを指定するためのdirective floorplanを自動生成するためのSTBP(Slicing-Tree with Box Packing)手法を提案する。slicing-treeで求めたブロック形状をそのままRAMなどの固定形状のハードブロックに適用させると、dead spaceが生成されてしまう。そこで、STBP法では、物理的に実現不可能なハードブロックに対して、他のソフトブロックとのマージを行い、box packingを実行することにより、dead spaceを発生せずにフロアプランを自動で生成する。実データを用いた評価を行い、人手相当の品質が、実用上問題の無い時間内に生成できることを示した。

キーワード：フロアプラン、slicing-tree、box packing、最適化、レイアウト

Floorplanner Based On Slicing-Tree With Box Packing

Toshiyuki SHIBUYA　Yuzi KANAZAWA　Toshihiko YOKOMARU　†Takeshi TAKAHASHI
FUJITSU LABORATORIES LTD.　†ENTITY COMPUTER SYSTEM
4-1-1, Kamikodanaka Nakahara-Ku, Kawasaki 211-8588, Japan
TEL: (044)754-2663, FAX: (044)754-2664
E-mail: shibu@flab.fujitsu.co.jp

Abstract

In this paper, we present an efficient floorplanner, called STBP(Slicing-Tree with Box Packing), that combines a slicing-tree based floorplanner with a box packing algorithm so as not to generate any dead space. We have evaluated STBP on the practical layout data, and show the quality of the results is comparable to that of the manual layout.

Keywords : floorplan, slicing-tree, box packing, optimization, layout

1 はじめに

近年のVLSI設計におけるテクノロジの微細化と回路の大規模化により、フロアプラン設計の重要性は益々高まっている。テクノロジの微細化の影響としては、ゲートの遅延よりも、配線の遅延の方が支配的になってきているために、ブロック間の配線遅延がチップのパフォーマンスに大きな影響を及ぼす様になってきている。このために、ブロック間をつなぐネットがタイミングエラーを起こさない距離に各ブロックを配置する必要がある。また、大規模化の影響として、実際にフロアプランすべきブロックやRAMの数が100個を越えるものが出て来ており、フロアプラン設計に要する設計期間が増加している。このような状況から、高品質なフロアプランを自動で生成するフロアプランシステムによる設計の期間の短縮が求められている。

VLSI設計におけるフロアプラン設計を、目的別に以下の2つに分類する。

1. formal floorplan

   主な目的として、階層設計を実現するためのフロアプランとして用いられる。配置されるブロックの境界は、厳密に決められており、2つ以上のブロックの境界が重なり合うことはない。隣接するブロックとブロックの間には、ブロック間配線のための配線領域を確保する必要がある。ブロックの形状は矩形で、縦横比が固定であるハードブロックと、可変であるソフトブロックが同時に存在する。

   コストとしては、基本的には配線長とチップ面積が使用される。フロアプランにより配置すべきブロックの面積の総和は変化しないので、チップ面積を最小化するためには、ブロック間に発生するdead spaceを減らす必要がある。また、今後、ブロック間のタイミング制御が重要になってくる。

2. directive floorplan

   このフロアプランの目的は、配置や論理合成ツールに対して、配置位置のdirectiveを指定することである。基本的に、配置ツールでフラットに扱う単位に対して、directive floorplanは指定されると考える。従来、チップ全体を一括で配置可能だった場合には、チップ全体に対して、directive floorplanを行っていたが、回路の大規模化にともない、formal floorplanで配置されるブロック単位において、directive floorplanを行うようになってきている。

   配置されるブロックの形状は、基本的に矩形であるが、formal floorplanに比べてその位置や形状に厳密性は必要ない。与えられたdirective floorplanを配置や論理合成ツールがどのように解釈するかは、ツール毎に異なっている。例えば、配置を例に取ると、ある配置ツールは、フロアプランの境界を厳密に意識するが、別のツールでは指定されたブロックの近辺に、そのブロックに属するセルを配置するが、その境界を無視したりする。

   本論文では、できるだけ解釈による違いを無くすため、directive floorplanによって指定されたブロック形状は次のように解釈するとする。

   (a) あるブロックに属するセルは、必ず指定されたブロック領域内に配置ツールによって配置される。

   (b) どのブロックにも覆われていない領域は、dead spaceと考え、フロアプラン指定されていないセルのみ、その領域は配置可能とする。ここでは、問題を簡単にするために、全てのセルは、あるブロックに属していると考え、フロアプラン指定されていないセルは存在しないとする。

formal floorplanを自動生成するための手法として提案されている従来手法のほとんどが、フロアプラン問題をブロックのパッキング問題として扱っている。従って、dead spaceの最小化がコスト関数となっている。これらの手法では、slicing-tree[17]、sequence-pair[11]、bounded-slicing grid[12]、O-tree[6]などの構造的な手法から得られた相対的な位置情報を基にパッキングを行う順序と開始場所を指定してパッキングを行っている。コストの最適化には、simulated annealing[10]などの最適化手法と組み合わせて、繰り返しパッキングを行う。また、最近では、force-directed法を用いて、配線長の二乗和の最適化と、ブロックの重なりを取り除くことを同時に行う手法[4]も注目を集めている。ただし、この手法では完全にブロックの重なりを取り除くことが難しいことが知られており、その改善方法[1]も提案されている。

directive floorplanを自動生成するための手法としては、formal floorplanのために開発された手法をそのまま使っても実現可能である。しかし、この場合、ハードブロックが多数含まれているとdead spaceが多くなってしまう。今後、IP-reuseにより、ハードマクロの数が増えることから、益々この問題は顕著になると予想される。Embedded ArrayやGate Arrayのようにチップなどのサイズに制約がある場合には、directive floorplanにおいて、dead spaceの量が多くなると、実際にセルが配置されているブロック部分のセル使用率が高くなり、配線率の低下を招く恐れがある。従って、directive floorplanにおいては、できるだけdead spaceの少ないフロアプランを求めることが、配線率を高めるために重要である。

本論文では、Embedded ArrayやGate Array向きのdirective floorplanの自動生成手法として、STBP(Slicing-Tree with Box Packing)手法を提案する。基本的なブロックの形状や配置位置は、全てのブロックをソフトブロックと見なして、slicing-tree

を用いて決定する。すると、ハードブロックは、slicing-treeで決定された形状とは異なっているため、そのままslicing-treeがフロアプランの解にはならない。そこで、slicing-treeにおいて、そのハードブロックをleaf nodeとした時に、一つ上のparent nodeであるinternal nodeを、新なleaf nodeとして実現し、その下のdescendantであるnodeをマージする。実際にハードブロックと他のブロックがマージ可能かどうかは、box packingを行い物理的にレイアウト可能かどうか調べる。もしも、box packingが不可能なら、さらに上のnodeにマージすることを繰り返すことにより、物理的に実現可能なフロアプランを生成する。

以下、第2章では、directive floorplan問題の定義とSTBP手法で用いるマージの考え方を説明する。第3章では、STBP手法の説明を行う。第4章では、その性能評価を行なう。また、以下の章において、特に断りが無い限りEmbedded ArrayやGate Array向きのdirective floorplanを前提に議論を行っている。

2 準備

2.1 Directive Floorplan問題

[13][16][17]を参考にして、directive floorplaningのモデルを一般化したものを以下に示す。

2.1.1 入力

1. フロアプランすべき領域の矩形形状：$H \times W$

   階層設計が行われていない場合には、これはチップ形状と同じになる。

2. ハードブロックの集合：HB

   $HB = \{hb_1, hb_2, \cdots, hb_i, \cdots, hb_n\}$

3. ハードブロックの制約条件：HBC

   ハードブロックiの面積(A_i)、回転可能なコードの集合、高さ(p_i)、幅(q_i)の4つの要素を一組としたn組のリストをHBCとする。

4. ソフトブロックの集合：SB

   $SB = \{sb_1, sb_2, \cdots, sb_i, \cdots, sb_m\}$

5. ソフトブロックの制約条件：SBC

   ソフトブロックiの面積(A_i)、縦横比の下限(r_i)、上限(s_i)の3つの要素を一組としたm組のリストをSBCとする。

6. ブロック全体の集合：B

   $B = HB \cup SB,\ b = n + m$

7. ネットリスト：$H = (B, E)$

   Eは、各ネットによって接続されるBのsubsetをhyperedgesで表現したものである。

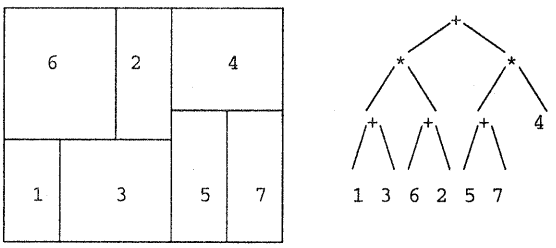
2.1.2 出力

dead space無しにHBC、SBCの制約条件を満たしたフロアプランとしてブロックの中心位置の座標と、ソフトブロックの縦横比を出力する。

2.2 Slicing-Tree

次に、STBP法と基本アルゴリズムであるslicing-treeによるフロアプラン生成手法の概要を説明する。

フロアプラン生成において、slicing-treeを用いる最大の利点は、フロアプランの構造表現が計算機で扱い易いことである。slicing-treeとは、与えられたフロアプランすべき領域の矩形形状において、再帰的に縦または横への分割を繰り返しすことによって得られるフロアプランである。slicing-treeにおいて、leaf nodeはフロアプランされる個々のブロックを表し、internal nodeは分割しているカットラインを表している。縦方向、横方向に分割してるカットラインを、それぞれ「+」、「*」という記号で表し、internal nodeにラベル付けすると、図1(a)のslicingフロアプランは、図1(b)のようなslicing-treeで表現することができる。



(a) slicingフロアプラン　　(b) slicing-tree

図 1: フロアプランの一例

このslicing-treeをPolish expression representationとして表現する方法が[16]によって提案された。図1(b)のtreeをPolish expressionで表現すると式1のようになる。

$$13 + 62 + *57 + 4 * + \qquad (1)$$

数字はブロックのIDを表すoperand、「+」、「*」を分割方向を表すoperatorと考える。2つ以上の同じoperatorが連続していないPolish expressionをnormalized Polish expressionとすると、一つのslicing-treeに一つのnormalized Polish expressionに一対一で対応する。

このPolish expressionを用いてslicing-treeの構造を変えるために、次の3つのオペレーションが存在する[17]。

M1: 2つの隣接するoperandを交換する。

M2: ある一定の長さの*, +の文字列の補集合(complement)と取る。

M3: 隣接する2つのoperandとoperatorを交換する。

ただし、M3のオペレーションを適用させるとnormalized Polish expressionの条件が崩れてしまう。

最適化方法としては、与えられたコスト関数に対して、simulated annealingを適用する。

2.3 Box Packing

box packingは、本来、二次元コンパクションのために開発された技術である。与えられた矩形の中に、ハードブロックのような形状固定の図形を重なり無く詰め込む技術である。今回実装したのは、zone refining[15]と呼ばれる技術をハードブロックの詰め込み用に改良したものである。

zone refiningでは、ceilingに並べたブロックを一つづつ外し、floorに詰め込む処理を行う。その際に、ブロック間の隣接関係を隣接グラフとして表現し、ブロックがceilingからfloorに移動するごとに隣接グラフを更新する。

ハードブロックを詰め込むためのbox packingは、slicing-treeのsimulated annealingによる最適化の過程において、頻繁に呼び出される。このため、できるだけ処理が単純で、パッキングが効率的にできる必要がある。そのため、以下の改良を行った。

1. 隣接グラフの代わりに、floorに配置されたブロックの表面が探索できるブロックのリスト(ブロック表面リスト)を作成する。
2. 配置すべきブロックは、ブロック表面リストの中から最もコストの低い配置位置に配置し、その配置に従ってブロック表面リストを更新する。
3. ceilingは設けず、マージする前のオリジナルのslicing-treeで得られたハードブロックの座標を用いてceilingからfloorのパッキング方向にソートを行い、floorに近いものからパッキングの候補とする。

このような改良により、高速にハードブロックのパッキングを実現した。

3 フロアプラン自動生成手法(STBP法)

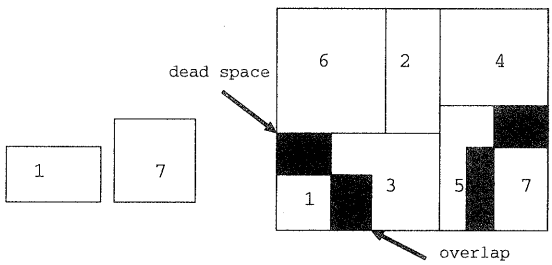
STBP法のアルゴリズムは[16]によって提案されたslicing-treeによるフロアプラン生成手法を基本にしている。ハードブロックを含んだ回路において、slicing-treeはdead spaceを生成するという欠点がある。ここで提案する手法では、box packingの手法と組み合わせることにより、その欠点を解決し、Embedded ArrayやGate Array向きのdirective floorplanに適用する。

最初に本手法の独自の考え方であるブロックマージの考え方について説明し、全体のアルゴリズムの説明に移る。

3.1 ブロックのマージ

ハードブロックが回路に存在する場合に、dead space無しにdirective floorplanを実現する目的で、ブロックのマージという考え方をここで導入する。従来のフロアプラン自動生成手法では、与えられたブロックBを必ず一つ一つ別なブロックとして並べていったのに対し、ここでは、必要に応じて複数のブロックを一つのブロックにマージすることを許す。

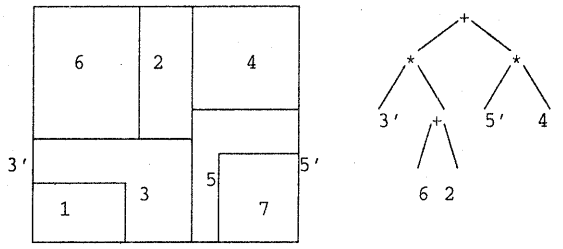
directive floorplanにおいて、ブロックのマージを許す利点は次の通りである。図1(a)において、全てのブロックがソフトブロックであるなら、ソフトブロックの制約条件SBCを満たしている限り、このまま配置へのdirective floorplanと使用することができる。しかし、仮にブロック1と7が、図2(a)のような形状のハードブロックであるとすると、図1(a)で与えられたブロックの形状と一致していないために、図2(b)のようにdead spaceが生成されたり、周辺のブロックに重なりができたりしてしまう。黒く塗り潰された矩形はdead spaceを表しており、斜線の矩形はハードブロックとソフトブロックが重なった領域を表している。この結果の場合、ブロック3と5のセル使用率が高くなり、配線し難くなると考えられる。



(a)ハードブロック　　(b)dead space

図 2: dead spaceが生じる例

そこで、ブロック1と3をマージし、ブロック3'とする。ブロック3'の形状はブロック1と3を含んだ大きさと見なす。そして、ブロック1は、新に生成されたブロック3'の中を自由に移動できるものとする。同様に、ブロック7とブロック5をマージし、それぞれハードブロックの形状として適当な位置に配置した結果を図3(a)に示す。また、その時のマージされたslicing-treeを図3(b)に示す。



(a) box packingの例　　(b) マージされた例

図 3: ブロックマージの例

このように、ハードブロックとソフトブロックをマージすることにより、ハードブロックの配置位置の自由度が上がり、dead space無しにdirective floorplanが生成できることが分かる。

しかし、新たな問題として、マージされたブロック内でのハードブロックのパッキングを行う必要がある。その方法については、次の章で説明する。

3.2 アルゴリズムの概要

[16]の手法に改良を加えた点は、コスト関数の計算方法とneighborhood structureの生成方法である。

3.2.1 コスト関数の計算方法

neighborhood structureとして生成されたPolish expressionにおいて、次の処理手順でコストを計算する。

STEP1: Polish expressionを基に、面積のバランスを保ちながら、全てのブロックBに対し、rectangular dissectionを行い、slicing-treeのフロアプランを作成する。この時、ハードブロックもソフトブロックと同じようにブロックの縦横比は可変と見なす。

STEP2: STEP1で生成されたブロック形状と実際の形状が一致していないハードブロックHBのleaf nodeにフラグをセットする。

STEP3: フラグがセットされたnodeの一つ上のparent nodeをsuper nodeとし、そのdescendantであるnodeを全てをマージし、フラグを外す。

STEP4: super nodeにマージされたハードブロックが物理的にsuper nodeの領域で実現可能かどうかbox packingを行いチェックする。box packingを行う方向は、乱数を発生して決める。box packingが出来なかった時には、このsuper nodeにフラグをセットしてSTEP3へ行く。

STEP5: 全てのnodeにおいてフラグが外れたなら、STEP6へ行き、さもなければSTEP3へ行く。

STEP6: box packingされたslicing-treeにおいて、第3.3章で定義するコスト関数を適用してコストを求める。

3.2.2 neighborhood structureの生成方法

ハードブロックにおいては、外部のブロックから見える端子位置まで決定されているので、ハードブロックの回転方向の決定も重要な要素である。そこで、neighborhood structureを生成するために、第2.2章で説明した3つのオペレーションに対し、モジュールの回転方向を操作するための次のオペレーションを加えている。

M4: 任意のハードブロックの回転方向を、そのブロックのHBCの制約に従い、変化させる。

M1～3のオペレーションは、Polish expressionを直接変形させるものだが、M4では各ハードブロックに用意されたテーブルの変数を書き換える処理となる。

3.3 コスト関数

フロアプランの品質を評価する指標として、チップ面積、配線長、タイミングが考えられる。第2.1.2章で述べているように、directive floorplanでは面積は最適化のコストとは考えない。以下の議論や実験では、話を簡単にするために、配線長を中心にコストの議論を行う。

回路の配線長を見積もる際に、考えなくてはならない要因が2つ存在する。一つは、ブロック間をつなぐネットの配線長であり、もう一つは、ブロック内のネットの配線長である。

3.3.1 ブロック間ネットの配線長コスト

まず、ブロック間ネットの配線長のコスト($Cost_{inter}$)見積もり方法を定義する。

$$Cost_{inter} = \sum_{k=1}^{|E|} FUNC1(k) \qquad (2)$$

ここで、$|E|$はネットリスト$H=(B,E)$におけるネットの本数、$FUNC1(k)$はネットkにつながっているソフトブロックの中心とハードブロックの端子位置を囲む最小矩形のhalf-perimeterである。

3.3.2 ブロック内ネットの配線長コスト

次に、ブロック内ネットの配線長について考察する。フロアプランを生成する段階では、各ソフトブロック内のセル配置は決定していない。このような場合には、Rent parameter[3][5]やlocal neighbor analysis[7]を用いて予測する手法が提案されている。これらの見積もり方法は、ブロックの縦横比の項目は

入力パラメータとして存在しない。従って、フロアプランを自動生成した際に変化するブロックの縦横比に応じてブロック内ネットの配線長は変化せず、一定であると言える*。

このような理由から、一般的にフロアプランの評価コストには、ブロック内ネットの配線長を考慮することはあまりない。しかし、STBP法では2つ以上のブロックが一つにマージすることを許しているため、$Cost_{inter}$だけを評価したのでは、ブロックをマージすればする程$Cost_{inter}$は小さくなり、好ましくないフロアプランとなってしまう。

そこで、本論文の実験では、簡易的に以下のようにブロック内ネットの配線長($Cost_{intra}$)を定義する。

$$Cost_{intra} = \sum_{i=1}^{m}(FUNC2(i) \times NUM\_NET(i) \times \alpha) \tag{3}$$

ここで、mはソフトブロックの個数、$FUNC2(i)$はブロックiの周囲のhalf-perimeter、$NUM_NET(i)$はブロックiのブロック内ネットの本数である。αは$Cost_{inter}$と$Cost_{intra}$のバランスを保つためのパラメータで、今回の実験では$\alpha = 1/1000$を使用した。

3.3.3 ブロックマージ時の配線長コスト

2つ以上のブロックがマージされた場合、$Cost_{inter}$と$Cost_{intra}$の計算は次のように行う。

1. $Cost_{inter}$

   一つのブロックにマージされたブロック間のネットkの$FUNC2(k)$を0にする。

2. $Cost_{intra}$

   あるブロックにマージされ、含まれたブロックiの$FUNC2(i)$は、マージ後のブロックの周囲のhalf-perimeterを用いる。

この計算方法により、接続の強い2つのソフトブロックがマージされると$Cost_{inter}$が下がり易くなる。また、極端に大きさの異なるソフトブロックがマージされると小さい方のブロックの$Cost_{intra}$が大きく増加してしまうことから、大きなブロックが小さなブロックを吸収してしまうような現象は起こらないと考えている。

3.3.4 ブロックマージによるペナルティコスト

ネットリストの特性によっては、人手によって細かく分割された多数のソフトブロックがマージされる場合がある。評価として人手との比較を行う場合には、あまりマージが行われると、フロアプランそのものの特性が変わり、比較が難しくなってしまう。また、実運用上の問題においても、元のブロック分割をできるだけ保持したいという要求が起こる場合もある。

そこで、マージされるブロック数を制御するために、ブロックマージによるペナルティコストを設定した。

今回の実装では、ペナルティコスト($Cost_{merge}$)を以下のように定義した。

$$Cost_{merge} = \sum_{i=1}^{b}(MERGED(i) \times FUNC3(i) \times \beta) \tag{4}$$

ここで、bはブロックの個数、$FUNC3(i)$はブロックiの面積を返す関数である。また、$MERGED(i)$はブロックiがマージされたなら1を、されていないなら0を返す関数である。βは$Cost_{merge}$と他のコストとのバランスを取るためにパラメータで、今回の実験では1を用いた。

マージされるブロック数の制御を行うために、別なアプローチとして、第3.2.1章のSTEP3において、super nodeを生成する際に、元のleaf nodeからの深さに応じて、super nodeを作らないようにする方法が考えられる。しかし、本論文での実装では、ブロックマージによるペナルティコストを採用している。

4 実験結果

STBP法によって自動生成されたフロアプランと人手によって作成されたフロアプランを比較することにより、その実用性を検証する。

実験データには、表1の3種類のASIC向けEmbedded Arrayを使用した。また、ソフトブロックの制約条件SBCは無いものとした。実験においては、Sun UltraSPARC-II 360MHzを計算機として使用した。

simulated annealingの最適化においては、初期温度をT0=100.000とし、終了条件の温度をT=0.01に設定し、温度の下げる関数として$T' = 0.95 \times T$を用いた。最適化に用いたコストは、第3.3章で定義したものを用いている。具体的には、$Cost_{total} = Cost_{inter} + Cost_{intra} + Cost_{merge}$である。

今回使用したテクノロジでは、配置されたハードブロックの内、RAMやROMにおいては、その回りに特別な電源ラインをフロアプラン後に発生する必要がある。したがって、フロアプラン自動生成システムにおいて、この電源ラインを幅を正確に見積もらないと、電源の重なりや、ソフトブロックの使用率が不均一になってしまうという問題点がある。しかし、その電源ラインの太さは、ハードブロックが配置される位置によって決まるため、複雑な計算が必要となる。自動フロアプラン生成システムの中で、そのまま適用するには、計算コストが高いと考え、今回実装したSTBP法では、簡易的な見積もりを行った。その結果生じた多

*実際には、面積一定とした場合には、ブロックの縦横比によってブロックの周囲の長さは変化するため何らかの影響はあると考えられる。

表 1: 評価データ

回路名	ソフトブロック数	ハードブロック数	ブロック間ネット数
F1	4	1	3,328
F2	8	14	38,246
F3	38	25	16,303

少のズレは、マニュアルによって修正した。この修正によるコストへの影響はほとんど無いと考えている。

評価において、社内CADシステムとして開発された配置[14][2]配線[9][8]ツールを使用し、a) 配置実行後の論理配線長、b) 配置配線実行後の実配線長と配線エラー数、を評価した。評価における制約条件としては、ソフトブロックに対しては縦横比の上限下限は設定しなかった。ハードブロックの回転可能なコードは、実際のRAM、ROMに設定されているコードを使用した。

表1のデータに対し、人手で設計者によって設定されたフロアプランの評価結果と、STBP法で自動生成したフロアプランの評価結果を表2に示す。人手の結果と比較して、ほぼ同等な品質が得られている。また、処理時間においても、配置配線に要する処理時間と比べると、1/10以下になっており、実用上問題ないと考えている。また、simulated annealingのパラメータをチューニングすることにより、さらに高速化が実現できると考えている。

STBP法では、ハードブロックをbox packingするために、ソフトブロックとのマージを行っているために、ほとんど全てのハードブロックは他のソフトブロックにマージされる。また、ソフトブロック間でもマージは発生されるが、その個数はブロック分割の細かさや回路の特徴、コストの設定の仕方で変る。今回の評価の最終結果において、ソフトブロック間でマージされている個数は、F1, F2, F3、それぞれ、0、2、14個である。F3のデータでは、マージされたソフトブロックの個数がかなり多いが、その原因は、細かいサイズのブロックが多数存在したために、細かいブロック間でのマージが起きたためである。

F3のデータにおいて、STBP法により生成したフロアプランを基に、配置まで行った結果を図4に示す。slicingフロアプランの特徴として、サイズの極端に違うソフトブロックが隣り合うと、小さい方のブロックが細長くなってしまう。slicingフロアプランを効率的に活用するためには、できるだけサイズの等しいブロックにパーティショニングする技術が必要と考えている。

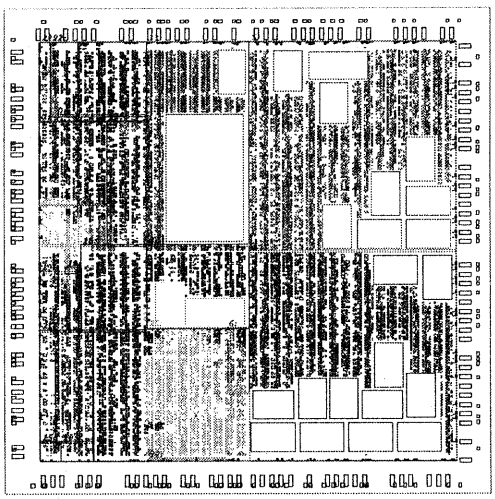


図 4: F3のレイアウト結果

5 まとめ

本論文において、slicing-treeとbox packingを組み合わせることにより、directive floorplan向きのフロアプランを自動生成するSTBP法を提案した。実データを用いた評価を行い、人手相当の品質が、実用上問題の無い時間内に生成できる見通しが得られたことを報告した。

今後の予定としては、ブロック間のタイミング制約も考慮する方法や、既配置のハードブロックを取り扱う方法への拡張を検討している。また、本アプローチでの経験から、formal floorplan向きのフロアプラン自動生成の効率的な手法も検討を行っている。

謝辞

本研究に当たり、多大なる御協力を頂いた富士通CAD本部、および富士通ソーシャルサイエンスラボラトリ寺田殿に深謝いたします。

参考文献

[1] 岡本 匠, 吉村 猛, 高永 潤, 水牧 俊博, and 水 沼 貞幸. 2次計画法と矩形パッキングに基づくVLSI フロアプラン の一手法. In *DA シンポジウム 2000*, pages 79–84, 2000.

[2] J. Cong, P. Li, S. K. Lim, T. Shibuya, and D. Xu. "Large scale circuit partitioning with loose/stable net removal and signal flow based clustering". In *Proc. Int'l Conf. on Computer-Aided Design*, pages 441–446, 1997.

[3] W.E. Donath. Placement and average interconnection lengths of computer logic. *IEEE Trans. on Circuits and Systems*, CAD-26(4):272–277, 1979.

表 2: 人手との比較

	人手			STBP法			
	論理配線長	実配線長	#未配線	論理配線長	実配線長	#未配線	配置時間(sec.)
F1	723,250	725,940	0	624,491	625,614	0	10.6
F2	14,211,117	15,480,569	282	13,551,238	14,731,169	289	921.0
F3	4,453,917	5,641,697	24,586	4,487,518	5,427,815	39,078	3,655.4

[4] H. Eisenmann and F.M. Johannes. Generic Global Placement and Floorplanning. In *Proceedings of the 35rd ACM/IEEE Design Automation Conference*, pages 269–274, 1998.

[5] M. Feuer. Connectivity of random logic. *IEEE Trans. on Computers*, C-31(1):29–33, 1982.

[6] P.N. Guo, C.K. Cheng, and T. Yoshimura. An O-Tree representation of Non-Slicing Floorplan and its Applications. In *Proceedings of the 36rd ACM/IEEE Design Automation Conference*, pages 268–273, 1999.

[7] T. Hamada, C.K. Cheng, and P. Chau. A wire length estimation technique utilizing neighborhood density equations. In *Proceedings of the 29rd ACM/IEEE Design Automation Conference*, pages 57–61, 1992.

[8] K. Kawamura, S. Fueki, and H. Miwatari. Generalized Touch and Cross Router. *Fujitsu Scientific & Technical Journal*, 31(2):208–214, 1995.

[9] K. Kawamura, T. Shindo, T. Shibuya, H. Miwatari, and Y. Ohki. Touch and Cross Router. In *Proc. ICCAD*, pages 56–59, 1990.

[10] S. Kirkpatrick, C. D. Gelatt. Jr., and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220:671–680, 1983.

[11] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani. Rectangle-Packing-Based Modue Placement. In *Proceedings IEEE International Conference on Computer-Aided Design*, pages 472–479, 1995.

[12] S. Nakatake, K. Fujiyoshi, H. Murata, and Y. Kajitani. Module Placement on BSG-Structure and IC Layout Applications. In *Proceedings IEEE International Conference on Computer-Aided Design*, pages 484–491, 1996.

[13] Sadiq M. Sait and Habib Youssef. *VLSI physical design automation.* McGraw-Hill, 1995.

[14] T. Shibuya, I. Nitta, and K. Kawamura. SMINCUT: VLSI Placement Tool Using Min-Cut. *Fujitsu Scientific & Technical Journal*, 31(2):197–207, 1995.

[15] H. Shin, A. L. Sangiovanni-Vincentelli, and C.H. Sequin. Two-Dimensional Compaction by "Zone Refining". In *Proceedings of the 23rd ACM/IEEE Design Automation Conference*, pages 115–122, 1986.

[16] D.F. Wong and C.L. Liu. A new algorithm for floorplan design. In *Proceedings of the 23rd ACM/IEEE Design Automation Conference*, pages 101–107, 1986.

[17] F.Y. Young and D.F. Wong. Slicing floorplans with pre-placed modules. In *Proceedings IEEE International Conference on Computer-Aided Design*, pages 252–258, 1998.