An Efficient Building Block Layout Methodology For Compact Placement

Nikolaos G. Bourbakis-Mohammad Mortazavi
Binghamton University
T.J. Watson School of Applied Science
Dept. EE, AAAI Lab
Binghamton, NY 13902-6000

Abstract

In this paper, a new efficient methodology for building block layout is presented by using synthesis placement and compaction. The synthesis placement part of the methodology is based on a formal language called GEOMETRIA. The compaction part is based on geometric reshapings (gs) of blocks and the merging of the communication channels. Both reshaping and merging follow the VLSI regulations for legal layout placement and improve the overall functional performance of the integrated system, by reducing the average length of the connection lines and the size of the occupied chip area by retaining the functionality and the neighboring relations of the blocks. The main goal of the blocks' geometric reshaping is minimization of the wasted area (or dead space among the blocks) called "open holes". The channels merging process of compaction is based on the legal overlapping of the blocks' communication channels by reducing the layout placement at the local and global routing.

1 Introduction

One of the important task in the physical layout cycle of VLSI design automation is the compaction. Compaction is the positioning of various size of blocks under certain layout rules in order to mask the geometry of the available space for placement. There are many compaction methods that have been developed [1-11]. A 2-D compaction algorithm based on a planar triangulation graph produces satisfactory results by maintaining the neighboring relations among the blocks, runs efficiently and resolves blocks overlapping during the placement with wiring optimization [Sequin91]. Another category of compaction with satisfactory results is the geometric compaction during the placement. In particular, a linear time algorithm for placement of components in architectures with simple folding is given by [Gajski92]. The presented interleaved folding method showed satisfactory results. A channel compaction algorithm incorporating via minimization and lateral via shifting is presented [Cheng92]. This method presented 22.9% improvements in 1-D compaction for a sample of 24 circuits. Additional methods and algorithms have been developed to improve the placement process. Some of them are: An algorithm for placement of small ICs subject to performance constraints [Kuh91]. An optimization procedure, called DIFFUSION, which can use global circuit placement by suppressing inter-module and module-to-chip boundary overlaps [Kung90]. A performance driven placement algorithm based on convex programming and min-cut algorithms, and satisfying the timing requirements [Liu91].

We present a new compaction approach which is based on a geometric reshaping of some blocks and merging of some common channel routing boxes among the neighboring blocks. This approach also uses synthesis process to compact and improve the overall functional performance of the integrated without violation of the design rules.

The proposed compaction method will combine topological and functional features [Bourbakis 88] and can be applied either on hierarchical VLSI design methodologies, such as SHEDIO, or on other placement techniques, such as 2-D compaction [Tzeng91], [Lengauer90], by producing a more compact layout placement.

2 GEOMETRIA Methodology

2.1 GEOMETRIA Architecture

In this section we provide a brief discussion of the GEOMETRIA's architectural design and functionality, (see Figure 1). More specifically, the GEOMETRIA structure supports automatically three different VLSI tasks. The first task is a floor planning process of macro-blocks placed effectively in a specified area [Bourbakis94]. The second task is the block reshaping and channels merging process which minimizes the placement area by using a compaction support. The
2.2 GEOMETRIA Language

The manipulation of the graph representations of the layout placement blocks is handled efficiently by a context-free language called GEOMETRIA. The formal representation of the GEOMETRIA language is given in this section based on the model of a context-free language.

2.2.1 Definition

The GEOMETRIA language is generated by a context free grammar:

\[ G = (N_S, T_S, P_R, S_S) \]

- \( N_S \) is the set of non-terminal symbols and is defined as:
  \[ N_S = \{ G(x), S_S, \text{operations} \} \]
  where, \( z(\text{Sh}(x)) = G(x) \)
  \( x \in \{ N(i), H(j), CL(km), NF(n) \} \)

- \( T_S \) is the set of terminal symbols.
  \[ T_S = \{ \text{operators}, S_S, M_i, a_{ij}, x \} \cup \Sigma \]
    operators = \{ \( \ominus \), \( \oplus \), \( \otimes \), \( \Pi \), \( \Gamma \), \( \emptyset \), \( \cup \), \( \otimes \) \}

- \( S_S \) is the starting symbol.
- \( P_R \) is the set of production rules
  
  \[ S_S \rightarrow \epsilon \]
  
  \[ S_S \rightarrow G(x) \]
  
  \[ S_S \rightarrow G(x) \{ \text{operation} \} \]

\[ S_S \rightarrow G(x) \{ \text{operator} \} S_S \]

\[ S_S \rightarrow G(x) \{ \text{operation} \} S_S \]

\[ G(x) \rightarrow G(x_i) \text{, } i \rightarrow 1/2/3/4/ ... \backslash k \]

where \( \epsilon \) is the unique graph

\[ \Sigma \] is the primitive alphabet:

\[ \{ M_i / \text{M}_i = \{ h, w, v, u \} , i \in Z^+ \} \]

where \( h \) and \( v \) represent the vertical straight line segments with directions \( \uparrow \) and \( \downarrow \) respectively \( w \) and \( u \) represent the horizontal straight line segments with direction \( \rightarrow \) and \( \leftarrow \) respectively.

2.2.2 Operation

In this section descriptions of the basic operations of the GEOMETRIA model are provided.

- **Elimination** operation is shown by \( \ominus \) operator. This operation will act like a "subtraction" operation. It will create a new side by placing a smaller side on top of a bigger side, and the new side (function) will be generated. The graphical and mathematical representation of elimination operation is shown below.

  \[ M_x \ominus M_y \]

- **Extension** operation is shown by \( \oplus \) operator. This operation will act like a "addition" operation. It will create a new side by placing a smaller side on the right end-edge of a bigger side, and the new side (function) will be generated. The graphical and mathematical representation of extension operation is shown below.

  \[ M_x \oplus M_y \]

- **Positioning** operation is shown by \( \otimes \) operator. This operation will place one side of a block on top of the other. The graphical and mathematical representation of extension operation is shown below.

  \[ M_x \otimes M_y \]

- **Rotation** operation is shown by \( \Pi \) operator. The main purpose of rotation operation in GEOMETRIA language is to help fitting a block into an open hole. This operation will rotate a block with a
rotation by step of 90°. It should be also noted that by rotating a block, the functionality of the block will not be clockwise (-) or counter-clockwise (+) directions. It should be noted that in the future these angles will be change. The graphical and mathematical representation of rotation operation can be shown by: certain angles such as 90°, 180°, and 270° in either implemented and rotated by step of 45°. However, for the purpose of simplicity, our concern will be the

![Diagram](image)

**Synthesis** There is a set of four primitive synthesis patterns which makes two blocks connected to each other and will generate new shape of block. These are:

1. **Synthesis of two nodes (or blocks) into L shaped.**

   ![Diagram](image)

   \[
   M_{12} = M_{11}; \quad M_{22} = M_{12}; \quad M_{34} = (M_{33} \Theta M_{22}); \quad M_{45} = M_{42}; \quad M_{56} = (M_{41} \Theta M_{56}).
   \]

   \[
   G(N(1) \sigma T N(2)) = M_{11}a_{11}M_{12}a_{12}M_{13}a_{13}M_{14}a_{14}M_{15}a_{15}M_{16}a_{16}M_{17}a_{17}M_{18}a_{18}M_{19}a_{19}M_{11};
   \]

2. **Synthesis of two nodes (or blocks) into T shaped.**

   ![Diagram](image)

   \[
   M_{12} = M_{11} \Theta M_{12}; \quad M_{22} = M_{22}; \quad M_{33} = M_{32} \Theta M_{33}; \quad M_{44} = M_{42}; \quad M_{55} = M_{54} \Theta M_{55}.\]

   \[
   G(N(1) \sigma T N(2)) = M_{11}a_{11}M_{12}a_{12}M_{13}a_{13}M_{14}a_{14}M_{15}a_{15}M_{16}a_{16}M_{17}a_{17}M_{18}a_{18}M_{19}a_{19}M_{11};
   \]

3. **Synthesis of two nodes (or blocks) into I shaped.**

   ![Diagram](image)

   \[
   M_{12} = M_{11}; \quad M_{22} = M_{22}; \quad M_{33} = M_{32} \Theta M_{33}; \quad M_{44} = M_{42}; \quad M_{55} = M_{54} \Theta M_{55}.\]

   \[
   G(N(1) \sigma T N(2)) = M_{11}a_{11}M_{12}a_{12}M_{13}a_{13}M_{14}a_{14}M_{15}a_{15}M_{16}a_{16}M_{17}a_{17}M_{18}a_{18}M_{19}a_{19}M_{11};
   \]

4. **Synthesis of two nodes (or blocks) into I shaped.**

   ![Diagram](image)

   \[
   M_{12} = M_{11} \Theta M_{12}; \quad M_{22} = M_{22}; \quad M_{33} = M_{32} \Theta M_{33}; \quad M_{44} = M_{42}; \quad M_{55} = M_{54} \Theta M_{55}.\]
\[ M_{12}^{4} = M_{14}^{4}; \]
\[ M_{12}^{4} = M_{14}^{4}; \]
\[ G(N(1)) \cup \bar{G}(N(2)) = \]
\[ = M_{12}^{12} a_{12}^{12} M_{2}^{12} a_{24}^{12} M_{3}^{12} a_{34}^{12} M_{4}^{12} a_{41}^{12} M_{1}^{12} \]

**Partitioning** operation is shown by the \( \Pi \) operator. This operation will partition a large block into smaller blocks for the purpose of reshaping and re-synthesize it. There are two types of partitioning of blocks called *shape driven* and *request driven* partitioning.

**Block Partitioning Rules**

There will be cases where the shape of the candidate node and/or the shape of the examined open hole don't match at all. In these cases, we reshape the node (block) by partitioning it, making the new shape fit the area occupied by an open hole. In particular, the reshaping of a node \( N(i) \) may follow one or two types of partitioning. These types of partitioning are:

I. **Shape Driven Partitioning**

In this case we partition the node \( N(i) \) by first extending its sides under conditions, shown in Figure 2. This case becomes possible if an already reshaped node requires a new reshaping for possible extra compaction gain.

1. \( \bigcup_{k=1} E(N'(k)) = E(N(i)) \),
2. \( \bigcap_{k=1} E(N'(k)) = \emptyset \).

![Figure 2: The shape driven partitioning of a node \( N(i) \), where each \( E(N'(k)) \), \( k \in \mathbb{Z} \), must be a rectangular area.](image)

II. **Request driven partitioning**

In this case, either the size of the open hole allows us to fit only a portion of the available \( N(i) \), or for some reason (from the previous partitioning case) a \( E(N'(k)) \) does not fit in a particular sub-area of the open hole, but that area is adequate as a surface. For these cases, we partition \( N(i) \) or \( E(N'(k)) \) by using the shape of the open hole under the conditions below, as shown in Figure 3.

1. \( \bigcup_{k=1} E(N'(k)) = E(N(i)) \),
2. \( \bigcap_{k=1} E(N'(k)) = \emptyset \).

![Figure 3: Request driven partitioning of a node \( N(i) \), where each \( E(N'(k)) \) must be a rectangular surface and \( k \in \{1,2,3,\ldots\} \).](image)

**Overlapping:** If there are two sets of functional or non-functional planes and their crossing as the common area that belong to both blocks, this will be denoted by \( \otimes \) operator and it is called overlapping operation. This operation is similar to intersection operations of two sets of elements. It is mainly used in the lowest level of transistor layout. For example, overlapping of n-diffusion with polysilicon will make n-transistor, and overlapping of p-diffusion with polysilicon will create a p-transistor. Block representation of overlapping will be discussed later.

**Union:** If there are two sets of functional and non-functional planes and their union as the area that belong to either set of blocks, this will denoted by \( \cup \) operator and it is called union operation. This operation is similar to Union operations of two sets of elements. Block representation of union will be discussed later.

3. **Block Reshaping and Merging**

3.1 **Block Synthesis Rules After Reshaping**

The block partitioning will generate a number of smaller blocks (or subnodes), there is a need for their composition into a new shape (reshaping) fittable in the open hole area, by using synthesis rules.[12]

There is a basic synthesis rule in order to reshape a given shape of a block. This rule is applied to the block graph representations by converting them into new graph representations. Thus, the new graph expressions represent the new shapes of the blocks. The basic synthesis rule is given below, which is useful for the synthesis cases in combination with a set of primitive synthesis patterns (L, \( \Gamma \), T, I) described in the previous section.

**The Basic Synthesis Rule**

The synthesis of two graphs \( G(X_1) \) and \( G(X_2) \) starts with the union operation of them. Then a selected primitive synthesis pattern points out the pairs of graph nodes which must be "merged". The merging operation is either an "elimination" or an "extension" operation on the graph nodes, generating a new graph node from a
pair. The selected primitive synthesis pattern will assist in the reshaping of the new graph which represents the new shape of the block. The demonstration of an example will accommodate the understanding of the synthesis operation by using the primitive patterns.

**Compaction After Merging**

There are three possible ways two blocks can be connected:

I) If there is a full connection between two common edges, we will merge their routing channel into one (see Figure 4).

![Figure 4. Merging eight routing connection into four.](image)

Block Reshaping Conditions

1. The VLSI technology regulations must be followed.
2. The functionality of a macro-block must be unchanged after its reshaping.
3. The connectivity of the reshaped blocks with the other blocks remain the same.
4. If the block reshaping process either increases drastically the wiring, or does not provide significant gain, then the reshaping process is re-evaluated or stops.

**Examples**

A Hierarchical Compaction. Hierarchical compaction placement with reshaping process produces a new building blocks with 14.8% smaller than the original, and the reduction of the open holes area was 64.7% (see Figure 7).

B. 2-D Compaction

The second demonstration example is based on the layout placement generated by the 2-D compaction method [Sequin91]. In this particular case the reshaping process has been applied to only two blocks, as shown in Figure 8, and the reduction gain on the overall layout area was 10%. We did not continue the reshaping process further, at this moment, since our goal of demonstrating the efficiency of the proposed method was achieved.

C. Compaction After Merging

In this example (Figure 9), merging of routing channel approach is provided for eight blocks, with reduction of 15% in its height of SLP area by using the primitives given in Figure 10.

**Contributions and impacts**

1. It produces a more compact geometric placement of macro-blocks under the conditions that open holes exist inside the layout area.
2. It makes the reshaping and merging of the macro-blocks possible for beneficial layout placement under certain conditions mentioned above.
3. It can improve the results of other compaction approaches under the condition of the existence of open holes.
4. It works with any size of macro-blocks with fixed or variable size, not necessarily in a rectangular shape.
5. It maintains the neighbor relationship among the blocks.
Figure 7: The new layout area SLP after the new placement of the reshaped blocks.

Figure 8: (a) The layout placement of the macro-cell benchmark ami33 (b) The new layout placement of the macro-cell benchmark ami33 after the reshaping of two macro-blocks.

Figure 9. Compaction simulated results by using merging method when each block is considered as one of the five primitive blocks of figure 10.

Figure 10: Primitive Blocks with five different routing channel configurations.

References