Non-slicing Floorplans with Boundary Constraints Using Generalized Polish Expression

De-Sheng Chen

Department of Information Engineering
and Computer Science
Feng Chia University
100 Wen-hwa Road, Taichung, Taiwan
Tel: +886-4-24517250 Ext. 3746
Fax: +886-4-24516101
e-mail: dschen@pine.iecs.fcu.edu.tw

Chang-Tzu Lin, Yi-Wen Wang

Department of Information Engineering
and Computer Science
Feng Chia University
100 Wen-hwa Road, Taichung, Taiwan
Tel: +886-4-24517250 Ext. 3719, 3763
Fax: +886-4-24516101
e-mail: p8993852@knight.fcu.edu.tw,
ywang@fcu.edu.tw

Abstract- In this paper, we address the problem of VLSI floorplanning with considering boundary constraints. The problem is practical and crucial in physical design since architects decide to arrange some I/O involved modules along the chip boundary to minimize both chip area and off-chip connections. By using a new representation called Generalized Polish Expression, we propose an efficient algorithm to handle the boundary constraints on non-slicing floorplans. In addition, a new fixing heuristic based on modular similarity is also presented to effectively fix the generated infeasible floorplans during the process. The experimental result is good in commonly used MCNC benchmark circuits.

I. INTRODUCTION

In the application of the floorplanning method, it will be useful if the users are allowed to specify some floorplan constraints in the final layout. The floorplan constraint we consider here is called boundary constraint: some modules are constrained to be packed along one of the four sides of the chip boundary: on the left, on the right, at the bottom, or at the top of the final floorplan. The constraint considered here is very valuable because architects may want to place some modules along the chip boundary to minimize both chip area and off-chip connections. The boundary constraint problem on slicing floorplans, using Polish expression, had been studied in [5]. The boundary constraint problem on non-slicing floorplans, using different non-slicing representations, had also been discussed in articles [3, 6].

Recently, an easy and efficient representation of non-slicing structure, called *Generalized Polish Expression* (GPE for short), is proposed [2]. Basically, GPE is the generalization of Polish expression, and it inherits the elegant properties of Polish expression in handling floorplanning problems. In this paper, by using GPE, we propose an efficient algorithm to handle the boundary constraints on non-slicing floorplans. Furthermore, a new fixing heuristic based on modular similarity is presented to effectively fix the generated infeasible floorplans during the process. We tested our algorithm with some benchmark data and the experimental result is good.

The paper is organized as follows. Preliminaries are given in section II. Section III describes our proposed method. Section IV gives the experimental results on MCNC benchmarks. We make some conclusions in the last section.

II. PRELIMINARIES

A module B is a rectangle of height h_B, width w_B, and area $area_B$. A super-module consists of several modules. A floorplan for n modules consists of an enveloping rectangle R subdivided by horizontal lines and vertical lines into n non-overlapping rectangles such that each rectangle must be large enough to accommodate the module assigned to it.

In our problem, we are given two kinds of hard modules $M = F \cup C$. The modules in F have freedom to move while the modules in C are constrained to be packed along one of the four sides of the final floorplan. A module B which is constrained to right boundary is denoted as B^R, and the rest may be deduced by analogy. A *feasible* packing is a packing in the first quadrant such that all the modules in C are placed on the boundaries as required. Our objective is to construct a feasible floorplan R to minimize the total area of the floorplan R.

A. Polish Expression

This representation can only present slicing structure of a floorplan. Each packing is encoded by a sequence, including module name and two relational operators. As illustrated in Fig. 1, every leaf corresponds to a basic module and is marked by a module name. Every internal node of the tree is labeled by a + or a *, corresponding to a vertical or a horizontal cut respectively. We can obtain a Polish expression [1] of length $2n - 1$ with n modules in the slicing floorplan by traversing the slicing tree.

B. Generalized Polish Expression (GPE)

GPE (the abbreviation for *Generalized Polish Expression*) [2] is the generalization of Polish expression. GPE can efficiently reuse some area that cannot be utilized anymore if only having vertical and horizontal operators defined in Polish expression, and is able to present non-slicing structural floorplan.

GPE uses a sequence of modules to reflect a physically

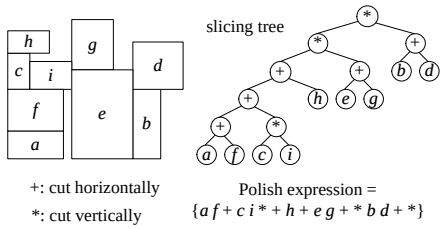


Fig. 1. Slicing tree representation and its corresponding Polish expression of a slicing floorplan.

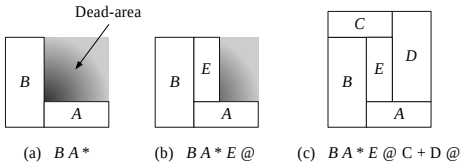


Fig. 2. (a) dead area is no longer utilized by only horizontal or vertical operators (b) corner operator, @, can effectively reuse the dead area, and (c) a floorplan of wheel structure, where *A*, *B*, *C*, *D* and *E* can be a module or a super-module. Notice that the part of shadow is dead area.

non-slicing floorplan by proposing a new relational operator @. As illustrated in Fig. 2(a), if there are only geometrically vertical and horizontal operators, the utilization of *dead area* is not achievable. The corner operator @, however, will arrange a module or a super-module in a corner formed by the other modules. As shown in Fig. 2(b), the corner operator will arrange *E* in the corner, i.e. the dead area constructed by *A* and *B*, where *A*, *B* and *E* can be a module or a super-module. Through corner operator, the dead area can be effectively reused by the other modules that have not been arranged yet. Furthermore, with the proposed corner operator, the new encoding scheme GPE can express the structure of wheel, as illustrated in Fig. 2(c).

C. Young-Wong Algorithm

In [5], Young and Wong handle the boundary constraints by adding an algorithm to find the information of each module. If a new generated Polish expression does not satisfy the boundary constraints in the progress of simulated annealing process, it can be fixed as much as possible by shuffling the modules. An example is shown in Fig. 3. In the figure, boundary constraint is violated in Fig. 3(a) since module *e* is not packed at the right, as required. To fix this, *e* is exchanged with *g*, where *g* is the module closest to *e* in the Polish expression and that *g* is packed on the right boundary. The result after shuffling the two modules is shown in Fig. 3(b).

III. Our Proposed Method

In [5], the boundary information of each module is found by scanning the Polish expression from right to left. The relationship of topology of + and * operators are checked to obtain the boundary information of each module. However,

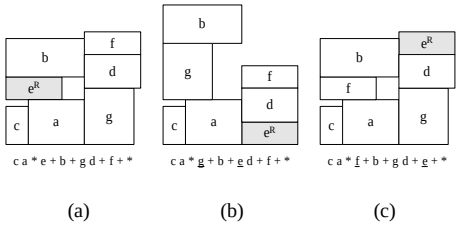


Fig. 3. Suppose module *e* is constrained to be packed along the right boundary. (a) an infeasible floorplan, (b) feasible by shuffling *e* with *g*, (c) feasible by shuffling *e* with *f*.

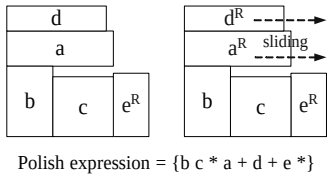


Fig. 4. The boundary constraints at right side are, (a) only set to module *e*, (b) set to modules *e*, *a* and *d*, where module *a* and *d* can be slid to right boundary.

modules that can be slid to the chip boundary may be not found due to the order of mergence of Polish expression. Hence, the method used in [5] may lose a certain chances to fix an infeasible floorplan that could lead to a good result at the end. An example of the situation is shown in Fig. 4. A floorplan is shown in Fig. 4 (a), and the boundary information of right side is only set to module *e*. Although, both of the module *a* and *d* can be slid to the right side of the chip (Fig. 4 (b)), the situation is not checked due to the appearing order of Polish expression. Hence, it may lose the chances to repair the violated modules. Therefore, it is important to come up with a more accurate method to find the boundary information of a floorplan.

A. Boundary Information Checking Algorithm (*BICA*)

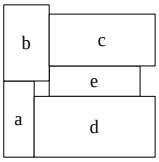
In our boundary information checking algorithm, the boundary information of a floorplan can be established by scanning the GPE once. This is done by recording four lists: L, B, R and T when entirely scanning the GPE from left to right, and by using a stack. The modules exist in list L mean there are no modules *cover* them in final floorplan, i.e. the modules are at or can be slid to left boundary of the floorplan, and the modules in lists B, R and T may be deduced by analogy. Each element *N* that is pushed into stack has four lists: *N*.left, *N*.below, *N*.right, *N*.above. The modules exist in list *N*.left mean there are no modules *cover* them in a sub-floorplan, i.e. the modules are at or can be slid to left boundary of the sub-floorplan, and the modules in lists *N*.below, *N*.right and *N*.above may be deduced by analogy. We push an element into the stack whenever we see a module or a super-module. We pop both the top and the top - 1 elements in the stack, and then push a new element which the four lists of the new element are obtained from the

Boundary Information Checking Algorithm
Input: A Generalized Polish expression $\Psi = \{\lambda_1, \lambda_2, \ldots, \lambda_{2n-1}\}$
Output: Four boundary lists L, B, R and T in the final floorplan.

```
1.    top = 0.
2.    For i = 1 to 2n -1
3.      If λ_i is + operator:
4.        N.left = stack[top].left union stack[top-1].left
5.        N.right = stack[top].right union stack[top-1].right
6.        N.above = x_cover(stack[top-1].left, stack[top].right,
          TOP)
7.        N.below = x_cover(stack[top].bottom,
          stack[top-1].bottom, BOTTOM)
8.        Pop the top two elements of stack
9.        Push N to stack
10.     If λ_i is * operator:
11.       N.left = x_cover(stack[top-1].left,stack[top].left, LEFT)
12.       N.right = x_cover(stack[top].left,stack[top-1].left,
          RIGHT)
13.       N.above = stack[top].above union stack[top-1].above
14.       N.below = stack[top].below union stack[top-1].below
15.       Pop the top two elements of stack
16.       Push N to stack
17.     If λ_i is @ operator:
18.       N.left = x_cover(stack[top-1].left, stack[top].left, LEFT)
19.       N.right = x_cover(stack[top].left, stack[top-1].left, RIGHT)
20.       N.above = x_cover(stack[top-1].left, stack[top].right, TOP)
21.       N.below = x_cover(stack[top].bottom,
          stack[top-1].bottom, BOTTOM)
22.       Pop the top two elements of stack
23.       Push N to stack
24.     If λ_i is a module name:
25.       N.left = λ_i
26.       N.right = λ_i
27.       N.above = λ_i
28.       N.below = λ_i
29.       Push N to stack
30.   L = stack[top].left
31.   B = stack[top].below
32.   R = stack[top].right
33.   T = stack[top].above
```

two popped elements with reference to both ***union*** and ***x_cover***(***retain***, ***alter***, ***direct***) operations whenever we see an operator. For an expression ***AB****, we arrange super-module ***A*** left to super-module ***B***. The ***union*** operation which acts on an expression ***AB**** directly append the list of ***B***.above to both the list of ***A***.above and form a new element ***N***.above, and so does ***N***.below. The lists of ***retain*** and ***alter*** in ***x_cover*** operation mean the list of ***retain*** replace the list of ***alter*** which is covered by the lists of ***retain*** in one of the four direction: LEFT, RIGHT, TOP, BOTTOM. The rest of situations of ***x_cover*** may be deduced by analogy. An example of the boundary information checking algorithm is given in Fig. 5. We scan the GPE from left to right once. Suppose we scan from the module ***a*** to the first operator *, i.e. GPE = {a d *}. A new element ***N*** will be pushed into stack, where ***N***.left will be the module ***a***, ***N***.right will be the module ***d***, and ***N***.above and ***N***.below will be the modules ***a*** and ***d***. The rest of processing can be deduced by analogy. Finally, we will obtain the boundary information of the floorplan as follows: L = {a-b}, B = {a-d}, R = {d-e-c}, T = {b-c}.

Because we use more accurate information to find out the boundary information of a floorplan, we can easily fix an infeasible solution to obtain a feasible one.



GPE = {a d * b + e c + @}
CCL = {(b, d)}

Scanned GPE = {a d *}
N.left = *x_cover*({a}, {d}, LEFT) = {a}
N.right = *x_cover*({d}, {a}, RIGHT) = {d}
N.above = {a} union {d} = {a-d}
N.below = {a} union {d} = {a-d}

Scanned GPE = {a d * b +}
N.left = {a} union {b} = {a-b}
N.right = {d} union {b} = {d-b}
N.above = *x_cover*({b}, {a-d}, TOP) = {b}
N.below = *x_cover*({a-d}, {b}, BOTTOM) = {a-d}

Scanned GPE = {e c +}
N.left = {e} union {c} = {e-c}
N.right = {e} union {c} = {e-c}
N.above = *x_cover*({c}, {e}, TOP) = {c}
N.below = *x_cover*({e}, {c}, BOTTOM) = {e}

Scanned GPE = {a d * b + e c + @}
N.left = *x_cover*({a-b}, {e-c}, LEFT) = {a-b}
N.right = *x_cover*({d-b}, {e-c}, RIGHT) = {d-e-c}
N.above = *x_cover*({b}, {c}, TOP) = {b-c}
N.below = *x_cover*({a-d}, {e}, BOTTOM) = {a-d}

Final boundary information of the floorplan:
L = {a-b}, B = {a-d}, R = {d-e-c}, T = {b-c}

Fig. 5. Some snapshots of BICA.

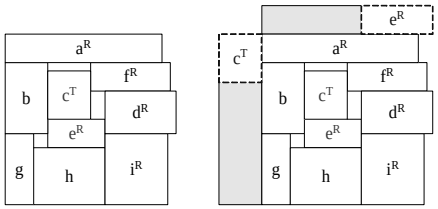
B. Fixing an Infeasible Floorplan

If the generated floorplan violates the boundary constraints, we are going to fix it as much as possible by shuffling the modules with reference to the ***similarity heuristic***. The value, ***D***, of similarity between module ***A*** and module ***B*** can be defined as follows.

$$D = |h_A - h_B| + |w_A - w_B|$$

The smaller the value ***D*** is, the similar the two modules are. We will choose the module which has the smallest value to perform the fixing. An example is shown in Fig. 3. In the figure, boundary constraint is violated in Fig. 3(a) since module ***e*** is not packed at the right, as required. To repair this, we exchange ***e*** with ***f*** where ***f*** is the module most similar to ***e*** in the GPE and that ***f*** is packed on the right boundary, as shown in Fig. 3 (c). Generally, if a module ***X*** is not packed along the boundary as required, we will shuffle it with another module ***Y*** which is most similar to ***X*** in the GPE and that ***Y***'s position satisfies the boundary constraint of ***X***. Obviously, the shuffling with reference to similarity heuristic (Fig. 3(c)) will be better or equal to the shuffling with the closest module (Fig. 3(b)).

C. Cost Function



(a) (b)

Fig. 6. (a) A floorplan with two violated modules *c* and *e*, (b) Added penalty (shaded region) for the two violated modules.

(a) (b)

Fig. 7. Final results with 10 boundary constraints, (a) ami49, (b) ami33.

TABLE I
RESULTS OF THE BOUNDARY CONSTRAINTS TESTING

			PE		GPE	
	# of Constraints	Constraint modules {(L), (B), (R), (T)}	Area (mm^2)	Time (sec)	Area (mm^2)	Time (sec)
ami33-bc	6	{(3, 11), (7), (1, 9), (5)}	1.21	41.4	1.21	80.57
	8	{(3, 13), (7, 17), (1, 11), (5, 15)}	1.25	46.61	1.21	113.12
	10	{(9, 27), (15, 19, 32), (3, 17, 30), (6, 24)}	1.28	50.02	1.23	137.74
ami49-bc	6	{(4), (15, 19), (14, 22), (1)}	37.48	73.03	37.11	232.71
	8	{(13), (15, 19), (3, 42), (1, 9, 35)}	38.2	69.77	37.4	189.56
	10	{(13, 14, 22), (15, 19), (3, 42), (1, 9, 35)}	39.56	65.52	37.24	270.61

It is possible that some constraints are still violated after all the possible shufflings. Hence, the cost function is defined as $A + P$, where A is the total area of the final layout and P is the penalty term for the violated boundary constraints. The penalty term is the new total dead area produced by the modules, which are virtually put at the boundaries of the floorplan along which they should be packed. For instance, if module *e* is constrained to be packed on the right, the penalty term for *e* will be the new produced dead area caused by virtually putting *e* to the right boundary of the final floorplan and the same situation for calculating *c* constrained to be packed on the right, as shown in Fig. 6 (b). The boundary constraint terms P will drop to zero as the process proceeds.

IV. Experimental Results

Based on the simulated annealing method [4], we implemented the GPE representation in the C++ programming language on a PC with Intel PIII 800MHz CPU and 256 MB memory. For comparison, we also implemented the algorithms presented in [5]. The experimental result is shown in Table I. Note that all the modules used in the experiment are hard and the modules with boundary constraints are randomly chosen. The area of a floorplan is measured by that of the minimum bounding box enclosing the floorplan. As shown in the Table I, the first column is the number of constraint modules. The second column means the labels of modules, which have boundary constraints in the order of {(Left-Boundary), (Bottom-Boundary), (Right-Boundary), (Top-Boundary)}. It is clear that GPE achieves promising area utilization with reasonable runtime to satisfy the boundary constraints, as required. The final circuit layouts of ami49, ami33 with 10 boundary constraints are shown in Fig. 7(a) and Fig. 7(b), respectively.

V. Conclusions

We successfully propose an efficient algorithm to handle non-slicing floorplan with boundary constraints by using the *Generalized Polish Expression* [2]. In addition, we can effectively fix the expression based on the accurate boundary information of a floorplan and the similarity heuristic. The experimental results show that the performance is good.

References

[1] D. F. Wong, and C. L. Liu, "A New Algorithm for Floorplan Design," *Proc. DA*C, pp.101–107, 1986.
[2] Chang-Tzu Lin, De-Sheng Chen and Yi-Wen Wang, "GPE: A New Representation for VLSI Floorplan Problem," *Proc. ICCD*, pp. 42-44, 2002.
[3] Jianbang Lai, Ming-Shiun Lin, Ting-Chi Wang and Li-C. Wang, "Module Placement with Boundary Constraints Using the Sequence-Pair Representation," *Proc. ISCAS*, pp. 341-344, 2001.
[4] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Scienc*e, vol. 220, no. 4598, pp.671–680, May, 1983.
[5] F. Y. Young, D. F. Wong, and H. H. Yang, "Slicing floorplans with boundary constraints," *IEEE Trans. on CAD*, pp. 1385–1389, 1999.
[6] Yuchun Ma, Sheqin Dong, Xianlong Hong, Yici Cai, Chung-Kuan Cheng, Jun Gu, "VLSI Floorplanning with Boundary Constraints Based on Corner Block List," *Proc. ASP-DAC*, pp. 509-514, 2001.