Modern floorplanning with boundary clustering constraint*

Li Li[1], Yuchun Ma[2], Ning Xu[1], Yu Wang[2], Xianlong Hong[2]
[1] WuHan University of Technology, WuHan, China
[2]Tsinghua University, Beijing, China
Email: myc@mail.tsinghua.edu.cn

Abstract

With the development of SOC designs, modern floorplanning typically needs to provide extra options to meet the different emerging requirements in the hierarchical designs, such as boundary constraint for I/O connection, clustering constraint for performance and reliability, etc. This paper addresses modern floorplanning with boundary clustering constraint. It has been empirically shown that the modern constraints extremely restrict the solution space; that is, a large number of randomly generated floorplans might be infeasible. In order to effectively search the feasible solutions, the feasible conditions based on B-tree representation with boundary clustering constraint are investigated. The properties, coupled with an efficient simulated annealing algorithm, provide the way to produce feasible floorplans by dynamic repairing, which can transform an infeasible solution into a feasible one if the constraint is violated. Our algorithm is verified by using the MCNC and GSRC benchmarks, and the empirical results show that our algorithm can obtain promising solutions in acceptable time.*

1. Introduction

Classical floorplanning formulation roughly determines the layout of a given set of modules, such that no modules overlap, and the enclosing layout region has minimum area and interconnection. After entering SOC era, however, modern floorplanning takes more care of providing extra options to place dedicated modules in the hierarchical designs. So it is common that designers will put additional placement constraints on the final packing to get meaningful designs for different purposes, such as abutment, boundary and fixed-outline constraints, etc. For instance, in mixed mode placement, the macros are always packed on the boundaries or clustered to the boundaries first so that the standard cells can be placed in the center. Therefore, we define a special kind of constraints named boundary clustering constraint which requires constrained modules being on the boundaries or clustering to the boundaries. Boundary clustering constraint can help to minimize wire length and routing space fragmentation in hierarchical design for SOC designs [10] [11]. In mixed-size placement, before the detailed placement of standard cells, the macro modules are first packed along or clustered to the boundaries which satisfy the boundary clustering constraint. As shown in the Figure 1, the macro blocks are all placed along or clustered to the boundaries before the standard cells are placed [10].

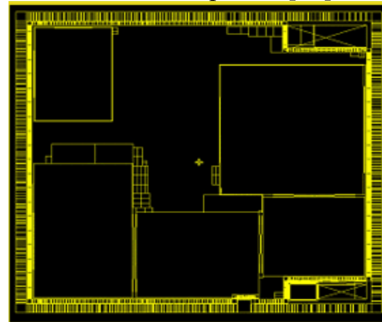

Figure 1: Placement of hard macro blocks

Recently, several floorplan representations are proposed, e.g., sequence pair (SP) [4], BSG [5], O-tree [2], B*-tree [1], corner block list (CBL) [6] and TCG [7]. Each representation has its own characteristics. Based on different representations, several works were proposed to handle various constraints including boundary constraint. In [8], it handled the boundary constraint using sequence pair. In [12], CBL was used to handle boundary constraints by proposing several boundary conditions in CBL representation. In [3], it handled boundary constraint using B*-tree. In boundary constraint, the constrained modules are restricted to be along the boundaries. But the boundary clustering constraint is different from boundary constraint since the constrained modules may not be located along boundaries but be clustered to the constrained blocks which are along the boundaries. As

*This work is supported by NSFC 60606007 and 60720106003, Tsinghua Basic Research Fund JC20070021, and Tsinghua National Laboratory for Information Science and Technology (TNList) Cross-discipline Foundation.

978-0-7695-3684-2/09 $25.00 © 2009 IEEE
DOI 10.1109/ISVLSI.2009.24

IEEE computer society

Authorized licensed use limited to: Tsinghua University Library. Downloaded on July 10,2010 at 13:27:57 UTC from IEEE Xplore. Restrictions apply.

shown in Figure1, some small macros are not along the boundaries, but clustered near to the big macros which are along the boundaries. Dealing with boundary clustering constraint is more meaningful for the macro placement in hierarchical designs. T.C. Chen, etc proposed a MP-tree to solve macro modules in mixed-size design in [9]. They handled the standard cells and macro modules separately by packing them in two steps. The macros are placed along four boundaries alone so that the center region is left to the standard cells. Although the two-step method ensures maximal contiguous (minimally fragmented) space for standard cells, it may result in long interconnects and therefore inferior timing results since the connections between macros and other cells are ignored in the first step. Therefore, to obtain the global optimization, it is necessary to pack the macros with the consideration of other cells. In the high level of hierarchical designs, the standard cells can be clustered into virtual modules so that a floorplanning methodology with boundary clustering constraints is very necessary.

In this paper, we deal with the floorplan design with boundary clustering constraint using the B*-tree representation. B*-tree has been proved a superior representation due to its simple, yet effective binary tree structure. In this paper, the feasible conditions based on B*-tree representation with boundary clustering constraint are investigated. The properties, coupled with an efficient simulated annealing algorithm, provide the way to produce floorplans with boundary clustering constraint. Through dynamic repairing, it is guaranteed that every solution in each iterative process is feasible. The effect and efficiency of our algorithm are verified by using the MCNC and GSRC benchmarks, and the empirical results show that our algorithm can obtain promising solutions in acceptable time.

The remainder of this paper is organized as follows. Section2 gives the problem definition. Section3 briefly introduces the B*-tree representation.Section4 explores the feasible conditions based on B*-tree for the boundary clustering constraint. Section 5 presents our overall algorithm. Experimental results are reported in Section 6. Finally, we conclude our work in Section 7.

2. Problem Formulation

In this section, we first define the boundary clustering constraint and then formulate the placement problem.

2.1. Boundary Clustering Constraint

The boundary clustering constraint means that the constrained modules must be placed along or clustered to the boundaries in the final placement. We conclude the definitions of this constraint as follows.

Definition 1: *Modules are clustered along the bottom (left) boundary iff these modules abut each other and there exists no other module below (left to) the clustered modules in the final placement.*

Definition 2: *Modules are clustered along the top (right) boundary iff these modules abut each other and there exists no other module above (right to) the clustered modules in the final placement.*

For instance, as shown in Figure2, b_3, b_4 and b_5 are defined as constrained modules. The packing in Figure 2 represents an admissible placement. In Figure 2(a), we can see that b_3 is on the left boundary. b_5 and b_4 are clustered with b_3 to the left boundary. So b_3, b_4 and b_5 satisfy the boundary clustering constraint. However, in Figure 2(b), the placement is illegal. Since b_3, b_4 and b_5 are not placed along or clustered to any boundaries.

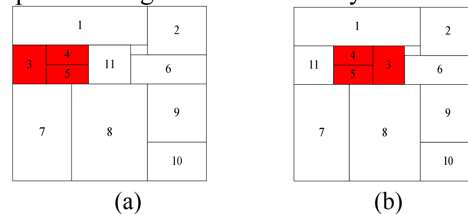

Figure 2: A placement with boundary clustering constraint modules (b_3, b_4 and b_5) (a) A feasible placement; (b) An infeasible placement

2.2. Problem Definition

Let $B = \{b_1, b_2, \ldots, b_n\}$ be a set of n rectangular modules whose respective width, height and area are denoted by W_i, H_i and A_i, $1 \leq i \leq$ n. Let (x_i, y_i) denote the coordinate of the bottom-left corner of module b_i, $1 \leq i \leq$ n, on a chip. A placement P with the boundary clustering constraint is an assignment of the coordinate (x_i, y_i) for each b_i, $1 \leq i \leq$ n, such that no two modules overlap and the constrained modules meet the boundary clustering constraint. The goal of floorplanning/placement is to optimize a predefined cost metric, such as the area and wire length, induced by the assignment of b_i's on the chip.

3. B*-tree Representation

B*-trees [1] are an ordered binary tree for modeling a compacted floorplan. Given an admissible placement [2] (in which no module can move left or bottom) P, we can represent it by a unique B*-tree T. (See Figure 3(b) for the B*-tree representing the placement of Figure 3(a)). The root of B*-tree corresponds to the block on the bottom-left corner. Similar to the DFS procedure, we construct the B*-tree T for an admissible placement P in a recursive fashion:

Authorized licensed use limited to: Tsinghua University Library. Downloaded on July 10,2010 at 13:27:57 UTC from IEEE Xplore. Restrictions apply.

Starting from the root, we first recursively construct the left subtree and then the right subtree. Let R_i denote the set of blocks located on the right-hand side and adjacent to b_i. The left child of the node n_i corresponds to the lowest block in R_i that is unvisited. The right child of n_i represents the lowest block located above and with its x-coordinate equal to that of b_i.

Based on the definition, the root of T represents the block on the bottom-left corner, and thus the x- and y-coordinates of the block associated with the root (x_{root}, y_{root}) = (0, 0). If node n_j is the left child of node n_i, block b_j is placed on the right-hand side and adjacent to block b_i in the placement; i.e., $x_j = x_i + w_i$. Otherwise, if node n_j is the right child of n_i, block b_j is placed above block b_i, with the x-coordinate of b_j equal to that of b_i; i.e., $x_j = x_i$. With the contour structure, we can compute the y-coordinate of a block in amortized constant time.

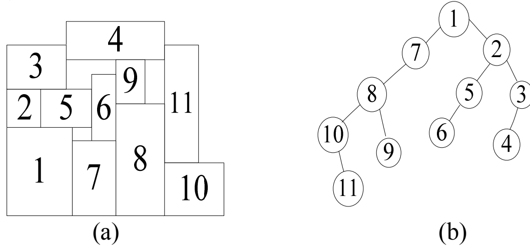

Figure 3: (a) An admissible placement; (b) the B*-tree representing the placement

4. Feasible conditions for the boundary clustering constraint

In this paper, we proposed the feasible conditions based on B*-tree for boundary clustering constraint so that we can search the feasible solutions efficiently. The feasible solutions may guide the process of the randomly optimization and promote the efficiency of our algorithm.

4.1. Feasible conditions of B*-tree

From the construction of B*-tree, we can conclude that if a node is not on a boundary but satisfied the boundary clustering constraint, the parent of the node must be a satisfied boundary clustering constrained module. So our algorithm firstly checks the boundary modules, and then finds out the modules clustering to the boundary constrained modules.

Let the leftmost branch (rightmost branch) of a B*-tree denote the path formed by the root and its leftmost (rightmost) descendants. By the definition of the B*-tree, all nodes in the leftmost (rightmost) branch are not right (left) children of others, which means there exists no module below (left to) the corresponding modules. We conclude the following conditions for the feasible conditions of a B*-tree with the bottom and left boundary clustering constraint.

- ***Bottom boundary clustering condition****: The nodes corresponding to the bottom clustering blocks must be on the leftmost branch of a B*-tree and their children.*
- ***Left boundary clustering condition:*** *The nodes corresponding to the left clustering modules must be on the rightmost branch of a B*-tree and their children.*

For example, we can see in Figure 3(b), b_1, b_7, b_8 and b_{10} are in the leftmost branch, so they are on the bottom boundary. It also has clustered condition. If b_8 is boundary clustering constrained module, b_9 is also a boundary clustering module. Because the b_9 clusters with b_8 to the bottom boundary. Similarly, the left boundary modules are b_2 and b_3. We can see that b_5 is a child of b_2, it means that b_5 is clustered with b_2 to the left boundary. If the b_2 is a constrained module, the b_5 satisfies the boundary clustering constraint, and so on.

Let the bottom-left (bottom-right) branch of a B*-tree denotes the path formed by the end of the leftmost (rightmost) branch and its rightmost (leftmost) descendants. If the nodes in the bottom-left (bottom-right) branch have left (right) children, it is very difficult to judge the right (top) boundary conditions. In order to guarantee that modules can be placed along the right (top) boundary, their left (right) children are moved. By the definition of the B*-tree, all nodes in the bottom-left branch of a B*-tree which left children are moved are with the same x- coordinate with the module placed at the bottom-right corner. So there is no module placed right to these modules. Further, no module is placed above these modules since the right children of the nodes in the bottom-right branch are moved. Thus, we conclude the following conditions for the feasible conditions of a B*-tree with the right and top boundary clustering constraint.

- ***Right boundary clustering condition:*** *The nodes corresponding to the right clustering modules must be on the bottom-left branch of a B*-tree which left children are moved.*
- ***Top boundary clustering condition:*** *The nodes corresponding to the top clustering modules must be on the bottom-right branch of a B*-tree which right children are moved.*

In Figure 3(b), b_{11} and b_{10} are on the bottom-left branch and their left children are moved. So they are on right boundary. Similarly, the top boundary is including b_3 and b_4.

As we all know, the final placement not only depends on B*-tree structure, but also depends on the dimension of the modules. Handling floorplanning with boundary clustering constraint, we propose some sufficient conditions for B*-tree which are mentioned above. It means that the placement is legal if the sufficient conditions are satisfied. So the sufficient conditions may help us search the feasible solutions

Authorized licensed use limited to: Tsinghua University Library. Downloaded on July 10,2010 at 13:27:57 UTC from IEEE Xplore. Restrictions apply.

efficiently and easily help us judge the validity of the solutions. The sufficient conditions also can guide the process of the randomly optimization.

4.2. Repair an infeasible B*-tree

Based on the proposed conditions, we can transform an infeasible B*-tree into feasible one. Three kinds of operations are devised to change the violated nodes to meet the boundary clustering constraint. If a module *b* is a constrained module, but it violates the constraint in the packing such as the module n_4 in Figure 4. We can remedy this violation by taking one of the following three kinds of operations.

Op1: Swap with a free node in leftmost (rightmost) branch or bottom-left (bottom-right) branch of a B*-tree which is not constrained.

Op2: Insert into leftmost (rightmost) branch or bottom-left (bottom-right) branch of a B*-tree.

Op3: Choose a constrained node in leftmost or rightmost branch or a child of a node which satisfies the constrained as parent.

Suppose a placement has nine modules and the boundary clustering constrained modules are n_0, n_1 and n_4. In Figure 4(a), the B*-tree represents one of the placement. n_0 and n_1 satisfy the boundary clustering constraint, but n_4 is a violated node. Doing **Op1**, we can choose n_5 which is a free boundary node to swap with. So Figure 4(b) is the result of swapping. We can insert n_4 into the rightmost branch of B*-tree showing in Figure 4(c). We also can choose the n_1 as parent to do **Op3** and be right child of it. Because n_1 is a boundary clustering constrained node. Figure 4(d) shows the legal result. In some cases, certain operations are not available. When there has no free boundary modules, we cannot do **Op1.** And when no nodes can satisfy the **Op3** condition, we must choose to do **Op1** or **Op2**. Also in some particular condition, we only can do **Op2**. In any conditions, one of the three operations can be used at least. These operations are based on modification on tree structure, **Op1** only takes *O(1)* times. The **Op2** and **Op3** take *O(n)* time, where *n* is the number of modules.

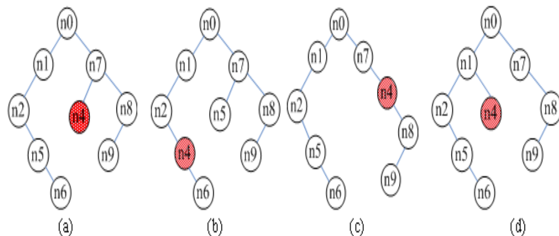

Figure 4: (a) an infeasible B*-tree in which n_4 is violating; (b) swapping with a free boundary node; (c) inserting into rightmost branch of the B*-tree; (d) choosing a left boundary constrained node as parent.

5. SA-based optimization flow

The flow of our algorithm is summarized in Figure 5. With B*-tree representation we develop a simulated annealing based algorithm [1] for handling the placement with boundary clustering constraint. Given an initial B*-tree, we firstly check whether the B*-tree satisfies the feasible conditions for the boundary clustering constraint. If it violates the constraint, we should repair it. In the simulated annealing process, the *perturb()* function perturbs the B*-tree to get a new one and roughly judges whether it changes the feasible conditions of B*-tree. It returns a boolean variable. If the perturbations do not change the validity of the B*-tree, it returns false. It can avoid redundant scanning and improve the efficiency of our algorithm. The detail methods will be introduced in section 5.2. On the contrary, if the *perturb()* function returns true, we need to recheck the feasible conditions of B*-tree. We transform an infeasible B*-tree into feasible one if any condition is violated. The perturbation process repeats until the termination conditions are met.

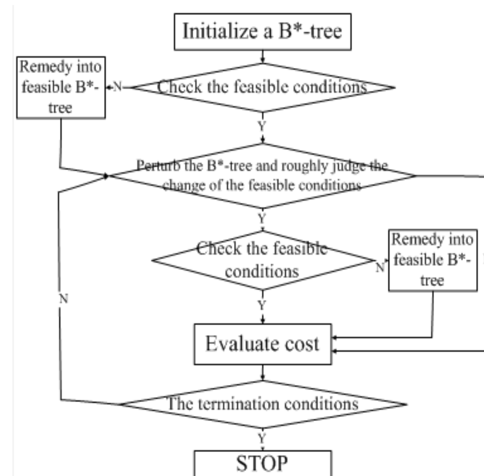

Figure 5: Design flow graph of our algorithm

5.1. Perturbation in SA

We use simulated annealing to search for an optimal solution. We apply the following three operations to perturb a B*-tree:

Op1: Rotate a module.

Op2: Swap two modules.

Op3: Move a module to another place in B*-tree.

Op1 only exchanges the width and height of a module without changing a B*-tree while **Op2** and **Op3** do. Only two nodes in a B*-tree are exchanged for **Op2**. The time complexity of **Op2** takes *O(1)* time. However, the topology of a B*-tree is changed for **Op3** since we need to delete and insert nodes into the B*-tree. The operations for deleting and inserting nodes are described in the following.

Authorized licensed use limited to: Tsinghua University Library. Downloaded on July 10,2010 at 13:27:57 UTC from IEEE Xplore. Restrictions apply.

For node deletion, three types of nodes must be considered: leaf nodes, nodes with one child, and nodes with two children. For a leaf node, it can be removed from a B*-tree directly without affecting other nodes. For a node with one child, it is replaced by its child. The subtree rooted by the child remains unchanged after the deletion. This tree update can be performed in *O(1)* time. The process to delete a node with two children is a bit more complex. One of its two children is chosen to replace the target node. Then we move a child of the node to the position of the node. The procedure continues until the corresponding leaf node is processed. This operation takes *O(h)* time, where h is the height of B*-tree. As shown in Figure 6, three cases of a node deletion are represented. In figure 6(a), the node deleted is a leaf node. And in Figure 6(b), it has one child. The node with two children is deleted showing in Figure 6(c).

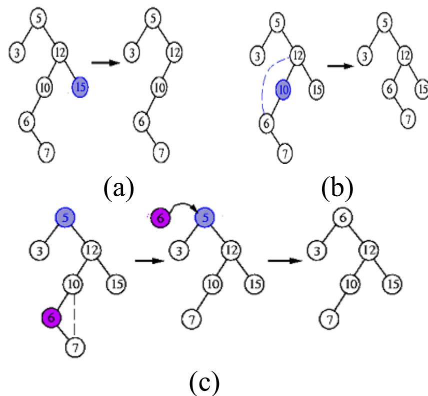

Figure 6: The operation of a node deletion (a) a leaf node; (b) a node with one child; (c) a node with two children

When we insert a node n_i into a B*-tree, we randomly choose a node n_j as its new parent. Then, n_i is inserted into the left (or right) of n_j and the original left (or right) child of n_j becomes the left (or right) child of n_i. The operation takes *O(1)* time. According to the above analysis, Op3 takes *O(n)* time, where n is the number of modules.

5.2. Maintaining a Feasible B*-tree

The feasible condition of a B*-tree may be destroyed after perturbation. However, some perturbations do not influence the legality of B*-tree. In these conditions, we needn't recheck the B*-tree. It is very helpful to reduce our running time especially when the number of modules is very large. For example, it is obvious that rotating modules do not change the validity. And when we choose two free or constrained nodes to do the perturbation, the validity of the result will not be affected and so on.

Further, to guarantee a feasible B*-tree during perturbation, we do not move nodes to left (right) children of the nodes in the bottom-left (bottom-right) branch. It can be controlled in the *perturb()* function.

6. Experimental Results

We implemented our algorithm using C++ programming language on a Pentium 4 2.4 GHz computer with 512 MB memory. We compared our algorithm with floorplanner based on B*-tree with no constraint in [1] based on the MCNC and GSRC benchmark circuits listed in Table 1. Columns 1, 2, 3 in the table give the respective names of circuits, number of modules and number of constrained modules.

Table1: the information of the circuit

Circuit	#of modules	#of constrain t number
apte	9	4
xerox	10	4
hp	11	4
ami33	33	10
ami49	49	10
n100	100	11
n200	200	20
n300	300	20

In the hierarchical designs, we can cluster standard cells into virtual blocks which have the similar size of the macro modules. We simultaneously handle macro modules and the virtual blocks in the floorplanning. Using the above mentioned benchmarks, the constrained modules denote the macro modules and the remaindering modules denote the virtual blocks in SOC design.

The area and time comparisons between the B*-tree with no constraint in [1] and ours are listed in Table 2. The column "Time ratio" gives the ratio of the runtime in [1] and the runtime of our algorithm. The empirical results show that even with certain amount of boundary clustering constrained modules, our algorithm can obtain promising solutions in acceptable time. According to the above mentioned sufficient conditions in section 4, the solution space may be reduced. But from the results, we can see that the coverage of the solution space is quite extensive and the quality of the solution is also promising. As shown in Table 2, the average area utilization is 95.902%, the decrease is only 0.635% comparing with the algorithm in [1] with no constrained modules. For xerox circuit, the area utilization has increased from 95.91% to 97.036%. The time over-head by checking and repairing B*-tree is linear to the number of modules. For apte circuit, the time ratio is 1:1.47. However, when the size of circuits are between ten and a hundred, such as xerox, hp, ami33, ami49 and n100, the time cost increases about three or four times. When we use the circuits of n200 and n300, the running time has increased about 8 times. Figure 7 shows the resulting layout for ami33 with ten constrained modules. Figure

Authorized licensed use limited to: Tsinghua University Library. Downloaded on July 10,2010 at 13:27:57 UTC from IEEE Xplore. Restrictions apply.

8 shows the resulting placements for n100 with the eleven constrained modules shaded.

7. Conclusion

We have explored the feasible conditions of a B*-tree with boundary clustering constraint and developed a simulated annealing based floorplan algorithm. Also, we have proposed an efficient algorithm to transform an infeasible solution into a feasible one if the boundary clustering constraint is violated. Unlike most previous works, our algorithm can repair infeasible solutions rather than directly rejecting them during optimization process. Our algorithm is verified by using the MCNC and GSRC benchmarks, and the empirical results show that our algorithm can obtain promising solutions in acceptable time.

Table 2: comparison between B*-tree based algorithm used in [1] with no constraints and ours

circuit	Total area of blocks	B*-tree with no constraint			B*-tree with constraint			Runtime ratio
		Resulting area (mm^2)	Area utilization(%)	Runtime (s)	Resulting area (mm^2)	Area utilization %)	Runtime (s)	
apte	46.56	46.92	99.23	0.82	46.92	99.23	1.13	1 : 1.47
xerox	19.32	20.26	95.91	0.98	19.940	97.036	3.66	1 : 3.73
hp	8.92	9.11	96.90	1.03	9.110	96.903	3.36	1 : 3.25
ami33	1.16	1.19	97.04	16.84	1.213	95.377	57.59	1 : 3.42
ami49	35.43	36.78	96.36	27.34	37.200	95.281	156.6	1 : 5.72
n100	0.1795	0.1854	96.8	119.22	0.1886	95.178	476.6	1 : 3.50
n200	0.1757	0.1842	95.41	178.77	0.1850	94.950	1450.3	1 : 8.11
n300	0.2732	0.2874	95.05	362.2	0.2928	93.260	3000.7	1 : 8.29
average			96.51			95.902		

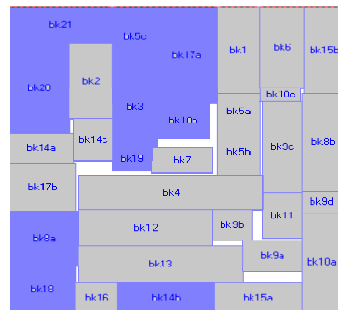

Figure 7: The placement result of ami33, where the number of constraint modules is ten, the dead space is 4.623%.

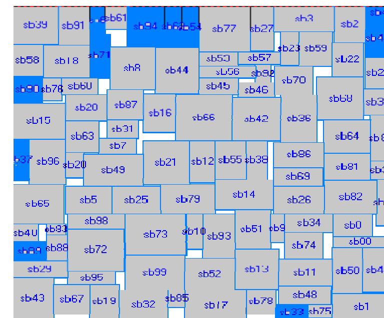

Figure 8: The placement result of n100, where the number of constraint modules is eleven, the dead space is 4.822%.

References

[1] Y.C. Chang, Y.W. Chang, G.M. Wu, and S.W. Wu, "B*-trees: A new representation for non-slicing floorplans, " Proc. DAC, 2000, pp. 458-463.

[2] P.N. Guo, C.K. Cheng, and T. Yoshimura, "An O-tree representation of non-slicing floorplans and its applications," Proc. DAC, 1999, pp. 268-273

[3] J.M. Lin, H.E. Yi, and Y.W. Chang, "Module placement with boundary constraints using B*-trees," IEEE Proceedings Circuits, Devices and Systems, 2002, pp. 251--256.

[4] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "Rectangle-Packing Based Module Placement," International Conference on Computer Aided Design, 1995, pp. 472–479

[5] S. Nakatake, K. Fujiyoshi, H. Murata, and Y. Kajitani, "Module Placement on BSG-Structure and ICLayout Applications," International Conference on Computer Aided Design, 1996, pp. 484–491

[6] X. Hong, G. Huang, T. Cai, J. Gu, S. Dong, C.-K. Cheng, and J. Gu, "Corner Block List: An effective and efficient topological representation of non-slicing floorplan," International Conference on Computer Aided Design, 2000, pp. 8–12.

[7] J.M. Lin and Y.W. Chang, "TCG: A Transitive Closure Graph-Based Representation for Non-Slicing Floorplans, " 38th Design Automation Conference, 2001, pp. 764–769.

[8] J. Lai, M.S. Lin, T.C. Wang and Li.C. Wang, "Module Placement With Boundary Constraints Using the Sequence Pair Representation," Asia and South Pacific Design Automation Conference, 2001, pp. 515–520.

[9] Tung-Chieh Chen, Ping-Hung Yuh, Yao-Wen Chang, Fwu-Juh Huang and Denny Liu, "MP-trees: A Packing-Based Macro Placement Algorithm for Mixed-Size Designs,", DAC'07, 2007, pp. 447-452

[10] Enno Wein and Jacques Benkoski, "Hard macros will revolutionize SOC design," EETimes, 2004

[11] Synopsys, Inc., "Hard Macro Placement in Complex SOC Design," SOC Central, 2007

[12] Yuchun Ma, Sheqin Dong, Xianlong Hong, Yici Cai, Chung-Kuan Cheng, Jun Gu: VLSI floorplanning with boundary constraints based on corner block list. ASP-DAC 2001, pp. 509-514

Authorized licensed use limited to: Tsinghua University Library. Downloaded on July 10,2010 at 13:27:57 UTC from IEEE Xplore. Restrictions apply.