Dataflow-Aware Macro Placement Based on Simulated Evolution Algorithm for Mixed-Size Designs

Jai-Ming Lin, *Member, IEEE*, You-Lun Deng, Ya-Chu Yang, Jia-Jian Chen, and Po-Chen Lu

***Abstract*— This article proposes a novel approach to handle macro placement. Previous works usually apply the simulated annealing (SA) algorithm to handle this problem. However, the SA-based approaches usually have difficulty in handling preplaced macros and require longer runtime. To resolve these problems, we propose a macro placement procedure based on the corner stitching data structure and then apply an efficient and effective simulated evolution algorithm to further refine placement results. In order to relieve local routing congestion, we propose to expand areas of movable macros according to the design hierarchy before applying the macro placement algorithm. Finally, we extend our macro placement methodology to consider dataflow constraint so that dataflow-related macros can be placed at close locations. The experimental results show that our approach obtains a better solution than a previous macro placement algorithm and a tool. Besides, placement quality can be further improved when the dataflow constraint is considered.**

***Index Terms*— Design hierarchy (DH), macro placement, mixed size, physical design, simulated evolution.**

I. Introduction

As PROCESS technologies advance, a circuit may contain billions of transistors. To reduce design complexity, reuse of intellectual property (IP) has become modern trend. Hence, a modern system-on-chip (SoC) contains more and more macros, which makes macro placement become the most important stage in the physical design.

Macro placement has a great impact on wirelength and routability since the locations of standard cells are affected by its result. However, macro placement is much harder because the dimension of a macro is much larger than that of a standard cell. Besides, more and more preplaced macros (i.e., obstacles) also make the problem more difficult. However, the industry still relies on experienced engineers to adjust the locations and orientations of macros. This is quite inefficient since we may repeatedly modify locations of macros, place standard cells, and route nets before a design converges to a good result. Note that cell placement or routing may take more than one day in a large-scale design by a commercial tool. Hence, product development cycle time will prolong indefinitely if proper macro locations cannot be found.

The three-stage placement flow is considered as the most suitable design methodology for large-scale mixed-size circuits since it can be easily integrated into a practical physical design flow. For example, Chen *et al.* [7] and Lu *et al.* [18] proposed a three-stage methodology based on the bell-shaped function and electrostatics-based placement algorithm, respectively. They first perform placement prototyping to distribute standard cells and macros over a placement region while optimizing some objectives such as wirelength or routability. Since there still exist overlaps among standard cells and macros after this stage, the macro placement stage legalizes macros according to the placement prototyping result. Finally, standard cells are placed in the remaining space after macro locations are fixed.

Several macro placement approaches have been proposed in recent years, such as Chang *et al.* [5], MP-tree [6], CP-tree [8], ECS [9], and ePlace-MS [18]. All these approaches are based on the simulated annealing (SA) algorithm. We can divide these works into three categories: B*-tree [22]-based algorithm, circular contour-based [9] algorithm, and mix of these two representations. MP-tree [6] uses a B*-tree with four subtrees to represent macros that are packed in the four corners of a placement region. Then, CP-tree [8] modifies B*-tree to handle preplaced macros in the four boundaries of a chip, where preplaced macros are modeled as packing anchors in the four branches of a binary tree. Chiou *et al.* [9] extend Corner Sequence (CS) proposed by Lin *et al.* [16] and use a sequence of two tuples and a circular contour to enable each macro to be placed in any corner of the contour, call ECS. Based on the circular contour data structure [9], Chang *et al.* [5] proposed a macro placement scheme according to the damped-wave constructive multilevel framework recently. To reduce complexity and enhance the speed of the SA algorithm, they cluster and decluster macros based on B*-tree in each wave, where a cluster size becomes smaller as the amplitude of oscillation decreases with time. In the methodology of ePlace-MS [18], they also develop a macro placement algorithm based on SA to control macro motion directly. In each perturbation of SA, the annealer randomly picks a macro and randomly determines its motion vector within a search range.

Most of existing macro placement works only focus on optimization of wirelength and routability without consider-

Manuscript received June 22, 2020; revised November 5, 2020; accepted January 27, 2021. Date of publication February 23, 2021; date of current version April 28, 2021. This work was supported in part by Himax Technologies Inc. and in part by the National Science Council of Taiwan under Grant MOST 108-2221-E-006-147-MY2. *(Corresponding author: Jai-Ming Lin.)*

The authors are with the Department of Electrical Engineering, National Cheng Kung University, Tainan 70101, Taiwan (e-mail: jmlin@ee.ncku.edu.tw; s10932626341@yahoo.com.tw; gimi3302@gmail.com; n26070114@gs.ncku.edu.tw; borson432@gmail.com).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TVLSI.2021.3057921.

Digital Object Identifier 10.1109/TVLSI.2021.3057921

ing special constraint such as dataflow. However, a design usually contains a sequential circuit, which is composed of two elements: one is control unit and the other is datapath. Datapath performs actual operations such as addition and multiplication, while the control unit determines the sequence of operations in a datapath. Since design quality can be further improved if dataflow is considered during placement, some placement works [10], [11], [15], and [21] begin to study this issue. However, these works all focus on standard cells and try to align the cells in datapaths neatly without considering macros. However, macro placement considering dataflow between modules has a broader impact than standard cells. Recently, Vidal-Obiols *et al.* [20] use a hierarchical multilevel flow to handle macro placement considering dataflow. However, they may place macros in the center of a chip because their approach adopts the SA algorithm. Moreover, their approach cannot deal with preplaced macros.

A. Our Contribution

Previous works [5], [6], [8], [9], [18] all use the SA-based approach to handle the macro placement. Because SA perturbs solutions randomly and places macros from scratch in every perturbation, it normally requires longer time to find a good solution, especially when a design contains hundreds of macros and millions of standard cells. These works neither consider preplaced macros nor guarantee nonoverlap between macros when preplaced macros exist because using existing representations to check whether two macro overlaps sometimes is very complex. Moreover, they need to fill empty space between a preplaced macro and a chip boundary if the preplaced macros do not abut to chip boundaries.

This article proposes a novel approach to handle macro placement. The characteristics are summarized as follows.

1) Novel Macro Placement Approach: We apply a placement method based on the corner stitching [19]. Since empty space can be found easily according to this data structure, we do not always pack a macro to the contour such as the SA-based algorithm. Moreover, preplaced macros that do not abut to chip boundaries can be handled directly. More importantly, it is quite easy to check whether a macro can be placed at a location even though there exist many obstacles around the location. Furthermore, we apply an efficient and effective simulated evolution algorithm to refine a placement solution. Unlike SA that changes a solution arbitrarily, the simulated evolution algorithm perturbs a solution according to the placement quality of macros. Only partial placement will be modified in each iteration, which facilitates it converge to a good solution quickly. In addition, the algorithm can avoid a solution getting stuck in a local optimal solution. Experimental results show that our methodology can complete a large design, which has more than 500 macros and around 4 million standard cells, in less than 6 min. However, the SA-based approach fails to obtain a legal placement in one day.

2) Macro Grouping According to Design Hierarchies: Once related macros could be placed at a local region, the associated standard cells will be placed close to them. Then, small wirelength is obtained spontaneously. Since macros in similar subcircuits have stronger relation than others, this article

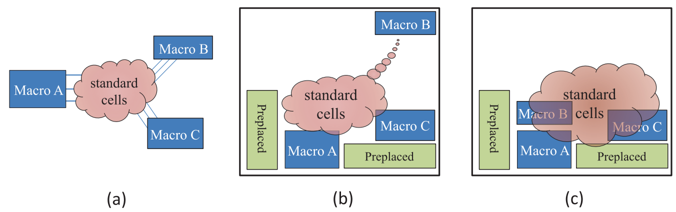


Fig. 1. (a) A set of three macros in the same DH without direct net connection. (b) Longer wirelength when the three macros are placed far away. (c) Local congestion due to placement of three macros in a local region without allocating space for standard cells.

proposes to group macros according to the design hierarchy (DH) as well as their original locations before these macros are placed. Our experimental result shows that wirelength (routing congestion) generated by our methodology without considering DH is 13% longer (2.14 times larger) than that is considered.

3) Reservation of Placement Space for Standard Cells: Even though macros with strong relationships can be placed at a local region, we may fail to place the associated standard cells at close locations due to insufficient space or have to pack them inside a very small region, which induces longer wirelength and worse routing congestion. In order to reserve a proper placement area for standard cells during the macro placement stage, this article proposes a novel idea that expands macros according to the area of related standard cells.

Fig. 1 shows an example to illustrate the phenomenon mentioned earlier. Fig. 1(a) shows three macros A, B, and C that connect to an identical set of standard cells, but there exists no direct connection between each other. If we place these macros far away from each other, it may lead to longer wirelength induced by standard cells, as shown in Fig. 1(b). On the contrary, it may induce local congestion if these macros are placed at close locations without allocating a proper region for placing the associated standard cells, as shown in Fig. 1(c).

4) Consideration of Dataflow: Unlike most of the previous macro placement works that only consider wirelength and routability, this article considers the dataflow constraint during macro placement. In order to reduce dataflow winding, we apply the Fiduccia-Mattheyses (FM)-based algorithm [13] to swap two dataflow-oriented macro groups in two regions when we allocate macro groups over placement regions. Moreover, we determine a better placement ordering before placing these macros such that macros with stronger relation in dataflow have a higher probability to be placed at close locations.

Fig. 2(a) shows the result of macro placement without considering dataflow, whereas Fig. 2(b) shows the result considering it, where the green arrows denote dataflow. This figure demonstrates that macro placement without considering dataflow will induce a longer wirelength. Hence, it may lead to serious routing congestion.

The remaining sections are organized as follows. Section II overviews our methodology. Sections III and IV introduce a macro grouping algorithm and a recursive partition algorithm, respectively. Then, a macro legalization algorithm and a macro refinement approach are illustrated in Sections V and VI, respectively. Section VII illustrates a dataflow-aware macro

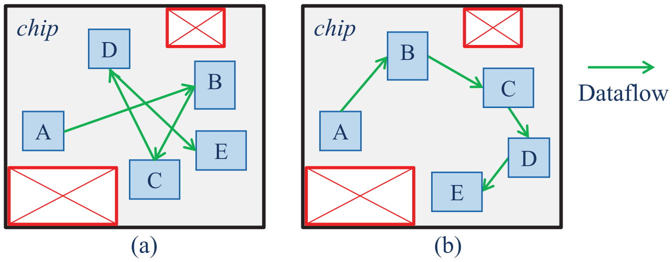


Fig. 2. (a) Macro placement without considering dataflow. (b) Macro placement considering dataflow.

placement approach. The experimental results are shown in Section VIII. Finally, Section IX concludes our work.

II. Overview of Our Methodology

Fig. 3 shows the flowchart of our methodology. It is composed of five stages: the preprocessing, macro grouping, macro group redistribution, macro legalization, and macro refinement stages.

First, the preprocessing stage merges placement blockages to decrease the number of obstacles to reduce placement complexity. In order to reserve enough placement regions for standard cells, we construct a DH tree and expand macros according to the tree. Then, macro grouping stage clusters strongly related macros such that it is possible to reduce wirelength by placing associated standard cells around these macro groups after they are fixed (see Section III). In order to find a proper region for placing macros in each macro group, the macro group redistribution stage redistributes macro groups over a placement region according to the recursive partition algorithm (see Section IV). After the placement region for each macro group is determined, our macro legalization algorithm places each macro into a specified region based on the corner stitching data structure (see Section V). Finally, the simulated evolution algorithm is applied to find a better solution, where the algorithm rips up some macros in each iteration and finds new locations to place these macros (see Section VI).

III. Macro Grouping

This section illustrates a macro grouping algorithm to cluster strongly related macros in order to make these macros be placed at close locations. This stage consists of two steps. First, a hierarchy tree is constructed according to a DH. Next, macros that have similar DHs and stronger connection or are placed at close locations are grouped together according to the best choice algorithm [4].

A. Construction of a Hierarchy Tree

This section shows the procedure to construct a hierarchy tree according to the instance names of macros and standard cells (i.e., names that are described in the Verilog of a design) as follows.

Step 1: Initialize a root for a design.

Step 2: For each macro, we trace the tree from the root according to the hierarchies in its instance name. If the hierarchy does not exist, we will insert a new node into the tree and then connect the previous node to the new node by an edge; otherwise, we go to its child node directly. The procedure repeats until the node corresponding to the last hierarchy is inserted and the macro will be recorded in the node.

Step 3: Then, each standard cell is inserted into an existing node of the tree, which is constructed in the previous step. We trace the tree from the root according to the hierarchies in its instance name until the node corresponding to a hierarchy does not exist. Then, the cell is recorded to the last node, which can be found in the tree.

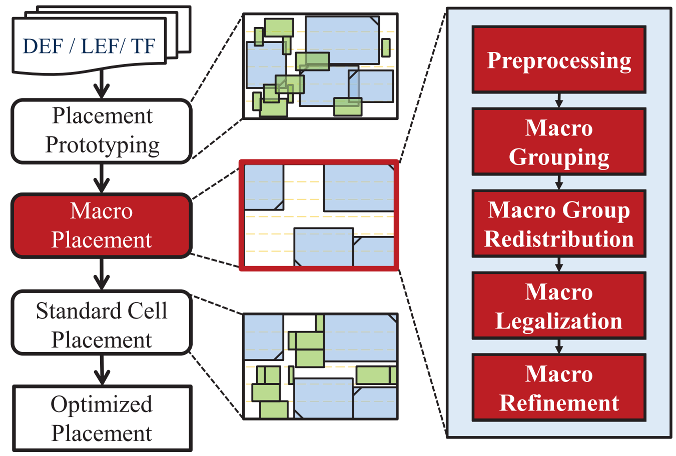


Fig. 3. Flowchart of our methodology.

According to the procedure, the topology of the tree is determined by macros, where each internal node represents a hierarchy in a design and each leaf belongs to a unique macro. After the tree is constructed, the total area of standard cells in each internal node is equally allocated to the leaves in its descendants. Then, the macro in each leaf is expanded according to the area allocated to it such that a placement area for standard cells is preserved.

Fig. 4 shows an example of a hierarchy tree, where blue (green) rectangles denote macros (standard cells). Each node in the tree represents a hierarchy in a design. For instance, the leaf in the leftmost branch of the tree denotes a macro whose instance name is A1/B1/C1. For a standard cell whose instance name is A1/B1/C4, it is inserted into the node B1 in the leftmost branch, which is the deepest node that can be found in the path from the root. Finally, each macro in a leaf is expanded according to the areas of standard cells in the path from the root.

B. Macro Grouping Algorithm

This section proposes a macro grouping algorithm based on the best choice [4]. Let g_u (or g_v) denote a macro group, and each macro group is composed of one macro in the beginning.

Since an original clustering algorithm only considers the connectivity and the combined area of two groups g_u and g_v, a new score function $\phi(g_u, g_v)$ is proposed to consider other

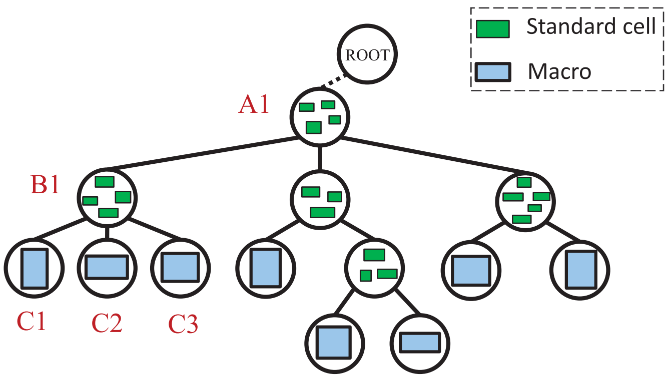


Fig. 4. Hierarchy tree.

issues such as the similarity of DHs as follows:

$$\phi(g_u, g_v) = \gamma \frac{1}{\Delta D(g_u, g_v)} + \delta \mathcal{H}(g_u, g_v) + \epsilon C(g_u, g_v) + \kappa \frac{1}{\Delta A(g_u, g_v) + 1} \quad (1)$$

where $\Delta D(g_u, g_v)$ denotes the distance between g_u and g_v in an initial placement. The geometric center of macros in a group is considered as the location of the group. Let $\mathcal{H}(g_u, g_v)$ denote the number of common hierarchies in the instance names of g_u and g_v, for example, if the instance names of g_u and g_v are A/B/C1 and A/B/D, respectively. The common hierarchies of the two instance names are A/B, so $\mathcal{H}(g_u, g_v)$ is set to 2. A large $\mathcal{H}(g_u, g_v)$ implies that the two groups have a stronger relationship. $C(g_u, g_v)$ represents the number of connectivity between g_u and g_v, and $\Delta A(g_u, g_v)$ is the area difference between g_u and g_v. γ, δ, ϵ, and κ are user-specified parameters (we set the values as 0.2, 0.3, 0.3, and 0.2, respectively).

We have two additional hard constraints in our grouping algorithm.

1) *Area Constraint:* In order to avoid generating an extremely large group, the combined area of two groups should be smaller than α times of an allowable placement area, where α is a user-specified parameter (α is set to 0.3 in our experiment).
2) *Hierarchy Constraint:* The common hierarchy number $\mathcal{H}(g_u, g_v)$ must be larger than β, where β is a user-specified parameter (β is set to 2 in our experiment).

Two groups with a small $\mathcal{H}(g_u, g_v)$ means that they have a weak relation from the point of view of a design. Hence, it is not necessary to place them in the vicinity.

IV. Recursive Partition Algorithm

The section introduces a recursive partition algorithm to redistribute macro groups in order to determine a proper placement region for each macro group.

The pseudocode of the recursive partition algorithm is shown in Algorithm 1. Let B and G_B denote a placement region and a set of macro groups in B, respectively. We first initialize Q by B and G_B (Line 1), and the procedure continues until Q is empty (Lines 2–24). After B and G_B

Algorithm 1 Macro Group Redistribution

R: A set of non-sliced regions ($R = \emptyset$)
Q: Queue of the pairs of regions and macro groups ($Q = \emptyset$)
B: placement region
G_B: macro groups in B

1: Q.Enqueue(B, G_B);
2: **while** (!Q.Empty()) **do**
3: (B, G_B)=Q.Dequeue();
4: **if** ($Width(B) > Height(B)$) **then**
5: $CutDirect$ = VERT;
6: **else**
7: $CutDirect$ = HORI;
8: **end if**
9: ($B_0, B_1, G_{B_0}, G_{B_1}$) = Eval_best_cut($B, G_B, CutDirect$);
10: **if** ($|G_{B_0}| > 1$ && $A_{B_0} > \tau$) **then**
11: Q.Enqueue(B_0, G_{B_0});
12: **else**
13: **for** ($g_i \in G_{B_0}$) **do**
14: $r_{g_i} = B_0$;
15: **end for**
16: **end if**
17: **if** ($|G_{B_1}| > 1$ && $A_{B_1} > \tau$) **then**
18: Q.Enqueue(B_1, G_{B_1});
19: **else**
20: **for** ($g_i \in G_{B_1}$) **do**
21: $r_{g_i} = B_1$;
22: **end for**
23: **end if**
24: **end while**

are dequeued (Line 3), B is cut vertically (horizontally) if its width (height) is larger than its height (width) (Lines 4–8). In order to partition G_B into two parts, we assume that there exists a cutline l_j, which runs through the gravity center v_j of each macro group g_j in G_B. Then, the best cutline can be found according to the function Eval_best_cut (Line 9). The function to evaluate the cost of a cutline will be introduced in the next paragraph. After G_B is partitioned into two groups G_{B_0} and G_{B_1}, their new placement regions B_0 and B_1 are determined. The recursive procedure continues if the number of macro groups in G_{B_0} (G_{B_1}), denoted by $|G_{B_0}|$ ($|G_{B_1}|$), is larger than 1 and the area of its placement region, denoted A_{B_0} (A_{B_1}), is larger than τ, where τ is a user-specified parameter (Lines 10, 11, 17, and 18). Otherwise, the placement region r_{g_i} for each macro group g_i in G_{B_0} (G_{B_1}) is found (Lines 13–15 and 20–22).

The function $\Psi(l_j)$ used to evaluate the cost of a cutline l_j is shown in the following:

$$\Psi(l_j) = \rho E + \zeta \Delta A(l_j) + \mu D_{\text{Shift}} \quad (2)$$

where E denotes the total weight of the nets that are cut by the partition. $\Delta A(l_j)$ denotes the difference of macro areas in the two partitions. For each horizontal (vertical) cutline l_j, we have to shift l_j to a new location l'_j in the y-axis (x-axis) such that the two regions B_0 and B_1 have the most uniform utilizations for the macro groups inside them. The procedure is introduced in the next paragraph. Let D_{Shift} denote the moving distance of a cutline (i.e., $D_{\text{Shift}} = |l_j - l'_j|$). ρ, ζ, and μ are user-specified parameters (we set the values as 0.5, 0.6, and 0.15, respectively).

Since it is time-consuming to determine a new location for each cutline l_j, we propose a method to speedup the procedure. We first divide a placement region B into several stripes and estimate the nonoccupied area in each stripe. The number of stripes in a region is equal to the number of

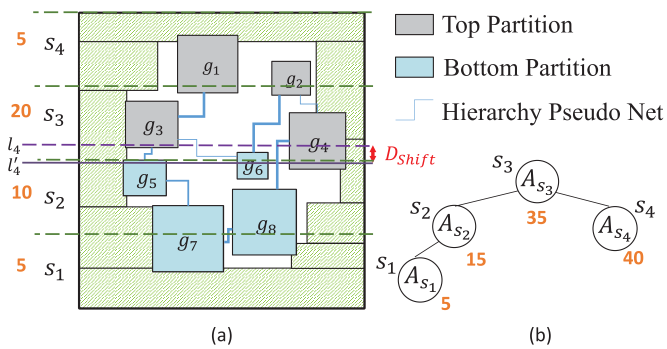


Fig. 5. Fast procedure to determine a new location l_4' of a cultine l_4. (a) Division of the placement region into four stripes (i.e., s_1 to s_4). (b) l_4' is determined by traversing the binary tree constructed from s_1 to s_4 in (a).

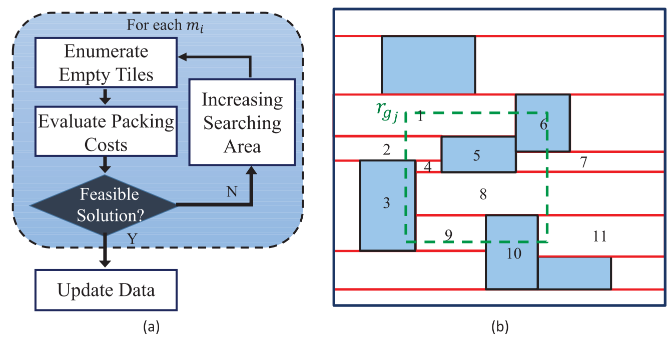


Fig. 6. (a) Flowchart of our macro legalization algorithm. (b) Legalization of m_i in r_{g_j}.

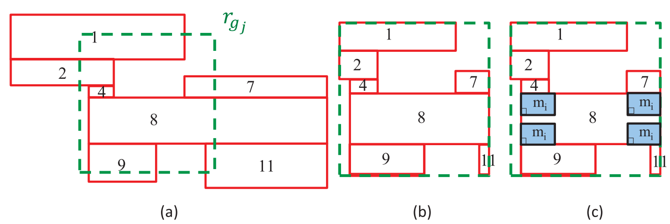


Fig. 7. Legalization of m_i in r_{g_j}. (a) Enumeration of empty tiles. (b) Trimming down the areas of empty titles outside r_{g_j}. (c) Packing m_i to four corners of title 8.

groups in the region. Let s_k and A_{s_k} denote a stripe and the nonoccupied area in s_k, respectively. We gradually add A_{s_k} to the next $A_{s_{k+1}}$ from bottom to top (left to right) (i.e., $A'_{s_{k+1}} = A'_{s_k} + A_{s_{k+1}}$, where $A'_{s_k} = \sum_{i=1}^{k} A_{s_i}$), and the result is recorded in a balanced tree [14], where each node denotes a stripe. Then, a new location for each l_i can be determined quickly by searching the tree according to the total areas of macro groups in the two partitions determined by l_j.

Please see Fig. 5 for example. Assume that the region is divided into four uniform stripes (i.e., s_1–s_4) as shown in Fig. 5(a), where the free area in each stripe is shown in the left side of the stripe (i.e., $A_{s_1} = 5$, $A_{s_2} = 10$, $A_{s_3} = 20$, and $A_{s_4} = 5$). After adding each A_{s_k} to $A_{s_{k+1}}$ in sequence, a binary tree can be constructed according to the result, as shown in Fig. 5(b). The values of nodes in the tree are $A'_{s_1} = 5$, $A'_{s_2} = 15$, $A'_{s_3} = 35$, and $A'_{s_4} = 40$. Suppose that we want to find a new location of l_4, where the total areas of macro groups in the two partitions are 10 and 16, respectively. The macro groups in the same partition are denoted by the same color in the figure. By traversing the search tree from its root, we can consider s_2 as the new location of l_4' since it makes the two regions have the most uniform utilizations (i.e., $10/15 \simeq 16/25$, where 15 and 25 are the free areas in the bottom and the top regions, respectively).

V. Macro Legalization

After each macro group g_j has been allocated to a proper region r_{g_j}, this section introduces our macro legalization algorithm, which places each macro m_i in g_j into r_{g_j} or as close to r_{g_j} as possible.

A. Macro Legalization Based on Corner Stitching

We first introduce the procedure to record a placement according to the corner stitching data structure [19]. For each placed macro, we will extend the top and bottom boundaries of the macro until they touch other obstacles. Then, empty and occupied rectangular regions are formed, where each rectangular region is called a tile. We connect a tile to its neighboring tiles through the links in the bottom-left and top-right corners of the tile. Thus, empty regions can be found easily through these links.

Fig. 6(a) shows the flow of our macro legalization algorithm. In the beginning, we create a window whose size is equal to r_{g_j}. Then, all empty tiles inside the window are enumerated according to the operation of the corner stitching, named directed area enumeration. Any empty tile whose region exceeds the boundaries of the window is trimmed down. For each empty tile, we will pack m_i into one of its four corners. If a legal location is found, a packing cost is evaluated through the cost function Π_i in (3) shown in Section V-B. Note that even when the size of m_i is larger than a tile, we can easily check the feasibility of a location by finding its neighboring tiles through the links of the current tile. Finally, m_i will be placed to the location with the smallest cost if there exist several feasible locations in the window. On the contrary, the window size will be increased and the same procedure is repeated in the new window. The loops continue until m_i is placed.

Fig. 7 shows the procedure to place a macro m_i into a region r_{g_j} in Fig. 6(b), where r_{g_j} is enclosed by a dotted line. First, all empty tiles in r_{g_j} are found, as shown in Fig. 7(a). Then, the areas of empty tiles that exceed the boundaries of the region are trimmed down and the resulting tiles are shown in Fig. 7(b). Finally, we place m_i to each corner of any empty tile, where Fig. 7(c) shows the result when m_i is placed at each corner of tile 8.

B. Cost Evaluation

The function to evaluate the quality of placing m_i in a location is shown as follows:

$$\Pi_i = \nu D_{\text{Group}} + \chi \frac{1}{D_{\text{Chip}}} + \omega D_{\text{Disp}} + \sigma W \tag{3}$$

where D_{Group} is the distance between the location of m_i and the gravity center of the group g_j of m_i, where the gravity center is computed according to the areas and the locations of placed macros in g_j. We hope that m_i can be placed as close to placed macros in g_j as possible. D_{Chip} is the distance between m_i and the gravity center of empty regions in a chip where all standard cells are excluded. The gravity center of empty regions will be computed before we place macros. A larger D_{Chip} means that m_i is placed away from the center of placeable regions in a chip. Then, a more complete region can be reserved for placement of standard cells after macros are placed. D_{Disp} is the displacement between the current and the initial locations of m_i. W denotes the wirelength of the nets between m_i and preplaced macros or preplaced pins. ν, χ, ω, and σ are user-specified parameters (we set the values as 0.05, 0.2, 0.1, and 0.1, respectively).

VI. Macro Refinement

This section gives a macro refinement approach based on the simulated evolution algorithm [12]. This approach optimizes a solution by ripping up some macros and finding their new locations. In order to avoid solutions getting stuck at a local optimal solution, macros that are placed at good locations still have a chance to be removed.

A. Score Function

The following function F_i is used to evaluate the placement quality of m_i:

$$F_i = \begin{cases} \mathcal{Q}, & \text{if } D_{\text{Group}} = 0 \\ \mathcal{Q} + \dfrac{\lambda}{D_{\text{Group}} \times \log(k+1)}, & \text{otherwise} \end{cases} \tag{4}$$

where D_{Group} denotes the distance between m_i and the gravity center of its own group g_j. The value D_{Group} equals zero when m_i is placed at the gravity center directly, and this usually happens when there exists only one macro in g_j. $\mathcal{Q}$ denotes the placement quality of m_i that is measured by the following equation:

$$\mathcal{Q} = W + D_{\text{Group}} \tag{5}$$

where W denotes the wirelength of the nets between m_i and other placed macros or fixed pins.

According to (5), m_i with larger wirelength or being placed away from the gravity center of g_j is more likely to be ripped up. In order to get a better solution, we give a higher probability to rip up a macro with good placement quality in the first few iterations. However, the probability is reduced as iteration increases to facilitate the convergence of a program. To achieve this goal, the second term $\lambda/(D_{\text{Group}} \times \log(k+1))$ of the second equation in (4) reverses the value D_{Group} to make a good placement have a larger penalty than a bad placement, where λ is a user-specified parameter and k is the number of iterations (we set λ as 0.1). Since D_{Group} is multiplied by a log function that has a small value in the first few iterations, this term can take effect only when k is small. However, the value of the log function becomes larger and stable as the iteration k increases, and the value of $\lambda/(D_{\text{Group}} \times \log(k+1))$ becomes significantly small, which makes it useless.

Algorithm 2 Simulated Evolution-Based Macro Refinement

M: A set of macros
G: A set of macro groups
Q: Queue for recording ripped-up macros
1: **for** ($m_i \in M$) **do**
2: F_i = Eq. (4);
3: **end for**
4: **for** ($m_i \in M$) **do**
5: $\hat{F}_i$=Normalize(F_i);
6: **end for**
7: **for** ($m_i \in M$) **do**
8: $rand$ = Random();
9: **if** ($\hat{F}_i > rand$) **then**
10: Rip-up(m_i);
11: Q.Enqueue(m_i);
12: **end if**
13: **end for**
14: Q.Sorting();
15: **while** (!Q.Empty()) **do**
16: m_i = Q.Dequeue();
17: Compute v_j according to placed macros in g_j;
18: Corner_stitching_packing(m_i, v_j);
19: **end while**

B. Macro Refinement Algorithm

The pseudocode of our simulated evolution-based algorithm is shown in Algorithm 2. First, the score F_i for each placed macro m_i is evaluated (Lines 1–3). Then, F_i is normalized to a new value $\hat{F}_i$, where the range of the value is between 0.1 and 0.9 (Lines 4–6). Next, we generate a random number *rand*, whose value is in the range between 0.0 and 1.0. If $\hat{F}_i$ is larger than *rand*, m_i is ripped up and enqueued in a queue Q (Lines 7–13). Next, all macros in Q are sorted according to their normalized values in the nonincreasing order (Line 14). Finally, all macros in Q are placed again by the macro legalization algorithm illustrated in Section V (Lines 15–19). Note that since some macros are removed, we have to adjust the gravity center v_j of g_j in order to place m_i close to the remaining macros in g_j (Line 17).

VII. Dataflow-Aware Macro Placement

Sections III–VI have illustrated our macro placement methodology. This section extends this methodology to handle dataflow constraint.

A. Preliminary

Given dataflow constraint from a circuit designer, we first build dataflow graphs (DFGs) to facilitate us searching the relationship of two components in the constraint. N_p denotes a data node in a DFG, which corresponds to a module, and H denotes the DH depth of the module. All nodes in a DFG have an identical H; hence, a DFG is represented by $\mathcal{F}_k^H$, where k is the index of the graph. For two nodes N_p and N_q in a DFG, the following conditions hold.

1) $N_p \mapsto N_q$ represents a direct flow from N_p to N_q.
2) $N_p \rightleftharpoons N_q$ represents no direct data flow between N_p and N_q. However, they have a direct common source and a direct common destination.

Fig. 8 shows an example of DFGs, where the instance name of N_1 in Fig. 8(a) (see Fig. 8(b)) is $A1$ ($A3/B1$). Thus, the hierarchy depth of the DFG in Fig. 8(a) (see Fig. 8(b)) is 1 (2). In Fig. 8(b), $N_4 \mapsto N_7$ because there exists a direct

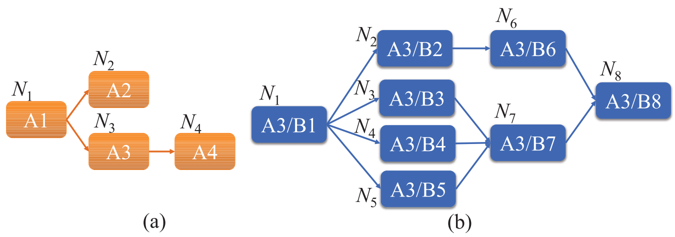


Fig. 8. Example of DFGs. (a) $H = 1$. (b) $H = 2$.

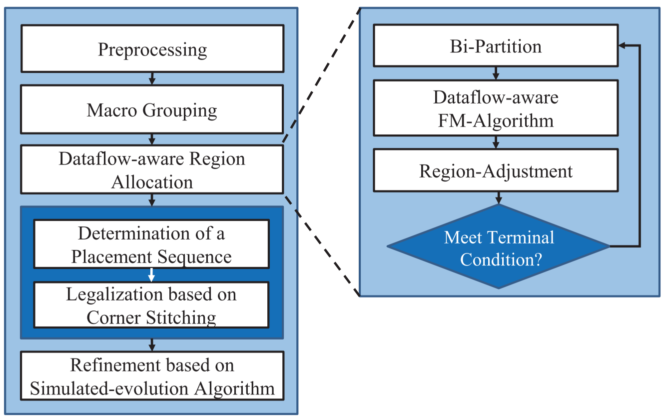


Fig. 9. Flowchart of dataflow-aware macro placement methodology.

flow from N_4 to N_7. Moreover, $N_4 \rightleftharpoons N_5$ since they have a direct common source N_1 and a direct common destination N_7 without any flow between N_4 and N_5.

B. Overview of Dataflow-Aware Macro Placement Methodology

Fig. 9 shows the flowchart of dataflow-aware macro placement methodology. After the preprocessing stage and macro grouping stage, the dataflow-aware region allocation stage allocates macro groups to different placement regions such that the utilization in each region is uniform by the recursive partition algorithm, as mentioned in Section IV. In order to further reduce flows passing through two regions, we apply the FM-based algorithm to swap dataflow-oriented macro groups in two regions in each partition (see Section VII-C). After all macro groups are assigned to regions, macros in each region are legalized according to the placement algorithm based on corner stitching. Since the result of legalization is greatly affected by a placement ordering, we first determine a proper sequence considering dataflow constraint in order to favor those macros defined in DFGs. Finally, the legalization result is further optimized by the simulated evolution refinement procedure. Since our macro placement methodology has been introduced in the previous sections, the following subsections only focus on the procedure that is related to the dataflow constraint.

C. Dataflow-Aware Region Allocation

This subsection introduces a recursive partition algorithm to allocate macro groups to their placement regions. As mentioned in Section IV, we will separate a placement region into several stripes by a set of cutlines. The cutline that has the smallest cost is selected to partition the macro groups into two parts. Then, we apply the FM-based algorithm [13] to swap dataflow-oriented macro groups in two regions in each partition in order to reduce dataflow winding. The same procedure continues until terminal conditions are met.

1) Construction of a Pseudo Net: For each pair of macro groups g_i and g_j in the different sides of a region, we will add a pseudo net to represent their similarity in DHs and strength of relationship in a DFG. The weight $\varphi(g_i, g_j)$ of a pseudo net connecting g_i and g_j is defined as follows:

$$\varphi(g_i, g_j) = \mathcal{H}(g_i, g_j) + \mu\mathcal{P}(g_i, g_j) \tag{6}$$

where μ is a user-specified parameter (we set the value as 1.6). $\mathcal{H}(g_i, g_j)$ denotes the similarity of DHs between g_i and g_j, which is determined according to the number of common hierarchies of the instance names in g_i and g_j. Since each group may contain several components, the instance name of a group is based on the common hierarchy of the instance names of the components in the group.

$\mathcal{P}(g_i, g_j)$ represents the strength of relationship between g_i and g_j if g_i and g_j are related dataflow-oriented groups, which means that each of them belong to a node in the same DFG $\mathcal{F}_k^H$. $\mathcal{P}(g_i, g_j)$ is estimated by one of the following equations:

$$\begin{cases} \dfrac{2}{|N_p| \times (|N_p| - 1)}(H-1) & \text{if } N_p = N_q \\ \dfrac{1}{|N_p| \times |N_q|}\left((H-1) + H \times \dfrac{1}{d_{s-\text{out}} \times d_{d-\text{in}}}\right), & \text{if } N_p \rightleftharpoons N_q \\ \dfrac{1}{|N_p| \times |N_q|}\left((H-1) + H \times \dfrac{1}{d}\right) & \text{if } N_p \mapsto N_q \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

The components in a data node may be divided into different groups, where $|N_p|$ ($|N_q|$) represents the number of groups formed by the components in N_p (N_q). Hence, we have to construct a pseudo net for each group pair (g_i, g_j) (i.e., $\forall g_i \in N_p$ and $\forall g_j \in N_q$, where N_p and N_q in $\mathcal{F}_k^H$). If g_i and g_j belong to the same node, it is computed by the first equation; otherwise, it is, respectively, computed by the second and third equations when $N_p \rightleftharpoons N_q$ and $N_p \mapsto N_q$. Each equation is divided by the total number of pseudo nets in N_p (between N_p and N_q) to avoid overweighting by many pseudo nets when many groups belong to the same data node. Since a large common hierarchy depth implies a stronger relationship, we add the value $H-1$ in each equation. Besides, the relationship between two groups is declined as the numbers of flows related to the common source and destination of N_p and N_q increase, and we divide H by $d_{s-\text{out}} \times d_{d-\text{in}}$ in the second term of the second equation, where $d_{s-\text{out}}$ ($d_{d-\text{in}}$) denotes the out-degree (in-degree) of the common source (destination) of N_p and N_q. Similarly, we divide H by d in the second term in the third equation, where d represents the summation of the out-degree of N_p and the in-degree of N_q.

For example, the weight $\varphi(g_i, g_j)$ of the pseudo net connecting groups g_i is computed, where g_i and g_j belong to N_4 and N_5 in Fig. 8(b), respectively. $\mathcal{H}(g_i, g_j) = 1$ because the common hierarchy of the instance names of N_4 and N_5 is A3. Then, the strength of relationship between g_i and g_j [i.e., $\mathcal{P}(g_i, g_j)$] is computed. $N_4 \rightleftharpoons N_5$ since there exists no direct data flow between N_4 and N_5, and $\mathcal{P}(g_i, g_j) = 1/(|N_4| \times |N_5|)((H-1)+H\times 1/(d_{s-\text{out}} \times d_{d-\text{in}}))$ by the second equation in (7). The relation of g_i and g_j declines as the numbers of flows related to the common source and destination of N_4 and N_5 increase. $d_{s-\text{out}} = 4$ ($d_{d-\text{in}} = 3$) because the common source (destination) of N_4 and N_5 is N_1 (N_7) and the out-degree (in-degree) of N_1 (N_7) is 4 (3). Assume that N_4 and N_5 are divided into 3 and 2 groups, respectively (i.e., $|N_4| = 3$ and $|N_5| = 2$). Therefore, $\mathcal{P}(g_i, g_j) = 1/(|3| \times |2|)(1 + 2 \times 1/4 \times 1/3) = 7/(36)$. Then, $\varphi(g_i, g_j)$ is equal to $\mathcal{H}(g_i, g_j) + \mu\mathcal{P}(g_i, g_j) = 1 + 7/(36) = 43/36$ if $\mu = 1$. In another example, g_i and g_j, respectively, belong to N_4 and N_7 in Fig. 8(b) when we compute $\varphi(g_i, g_j)$. $\mathcal{H}(g_i, g_j) = 1$ because the common hierarchy of the instance names of N_4 and N_7 is A3. Since $N_4 \mapsto N_7$, $\mathcal{P}(g_i, g_j) = 1/(|3| \times |2|)(1+2\times 1/(1 + 3)) = 1/4$ by the third equation of (7) (we assume $|N_7| = 2$). $d = 1+3$ because the out-degree of N_4 is 1 and the in-degree of N_7 is 3. Hence, $\varphi(g_i, g_j)$ equals $1 + 1 \times 1/4 = 5/4$.

2) Macro Group Exchanging by FM Algorithm: The FM-based algorithm is used to swap dataflow-oriented macro groups in two subregions to minimize the cutsize and the total cost of pseudo nets. Unlike a traditional FM algorithm that only moves an object in each pass, our approach swaps two objects since we need to determine their locations after they are exchanged. The procedure in each iteration is shown in the following.

1) Calculate the gain for each pair of groups g_i and g_j, where the value is the summation of the changes of cutsize of signal nets and cost of pseudo nets after they are swapped.
2) Exchange the pair of groups with the maximal gain and lock them after they are swapped.
3) Repeat step 2) until all groups are locked.

Because the area utilizations in two subregions become unbalanced after some groups are exchanged, we have to determine a new location of the cutline in the region.

D. Dataflow-Aware Macro Ordering

Since the quality of legalization is greatly affected by a placement ordering, we have to determine a proper sequence for macros when the dataflow constraint is considered.

The pseudocode of the dataflow-aware macro ordering algorithm is shown in Algorithm 3. First, all macro groups are sorted according to values computed by (8) in the nonincreasing order (Line 1) and stored in a list $L_{\text{init\_}g}$

$$\Gamma(g_i) = A_{g_i} + \frac{1}{D_{b,g_i}} + \mathcal{N}_{g_i} \tag{8}$$

where A_{g_i} is the area of group g_i. D_{b,g_i} denotes the minimum distance between a chip boundary and g_i after group redistribution. $\mathcal{N}_{g_i}$ is the number of nets of g_i. Then, Lines 3–14 determine a macro group ordering according to the

Algorithm 3 Dataflow-Aware Macro Ordering

M: A set of macros
G: A set of macro groups
L_{init_g}: List of an initial ordering of macro groups
L_g: List of the resulting ordering of macro groups
L_{tp_g}: List for temporarily macro group ordering
L_m: List of the resulting ordering of macros
L_{tp_m}: List for temporarily macro ordering

1: L_{init_g} = Sorting_by_Gama(G);
2: $L_g = \emptyset$;
3: **while** ($L_{init\_g} \neq \emptyset$) **do**
4: $g_i = L_{init\_g}$.Pop();
5: **if** (g_i is already in L_g) **then**
6: Continue;
7: **end if**
8: **if** (g_i is not a dataflow-oriented group) **then**
9: L_g.Push(g_i);
10: **else**
11: L_{tp_g} = BFS(g_i);
12: L_g.append(L_{tp_g});
13: **end if**
14: **end while**
15: $L_m = \emptyset$;
16: **for** ($g_i \in L_g$) **do**
17: L_{tp_m} = Sorting_by_Psi(g_i);
18: L_m.append(L_{tp_m});
19: **end for**

sequence in $L_{\text{init\_}g}$, and the result is stored into a list L_g. We iteratively pop a macro group g_i from $L_{\text{init\_}g}$ (Lines 4). If g_i is already in L_g, we do nothing (Lines 5–7). Otherwise, g_i is inserted into L_g directly if g_j is not a dataflow-oriented macro group (Lines 8 and 9). On the contrary, we have to extract the related macros groups of g_i from $\mathcal{F}_k^H$, where $\mathcal{F}_k^H$ is the DFG that the data node N_p of g_j belongs to (Lines 10–13). The function BFS in Line 11 uses a breadth-first search algorithm to visit the nodes in $\mathcal{F}_k^H$ from node N_p and store the macro groups associated with each node into L_{tp_g}. Note that $\mathcal{F}_k^H$ is considered as an undirected graph when we perform BFS. Next, L_{tp_g} is appended after L_g (Line 12). After the sequence L_g of macro groups is obtained, we iteratively sort the macros in each group g_i according to the values computed by (9) and then store the result into an output list L_m (Lines 16–19)

$$\psi(m_i) = \max(w, h) + A_{m_i} + \mathcal{N}_{\text{pin}} \tag{9}$$

where w and h denote the width and height of a macro m_i, respectively. A_{m_i} and $\mathcal{N}_{\text{pin}}$ denote area and pin number of m_i, respectively. Lin *et al.* [17] have highlighted the benefit of regular placement of macros because power planning can become easier and better routability could be obtained if macros are placed regularly. According to (9), macros with identical types will be placed in serial so that we have a higher chance to place them regularity. When two macros have the same value ψ, they are sorted by D_{b,m_i}, where D_{b,m_i} is the minimum distance between a chip boundary and m_i.

VIII. Experimental Results

Our algorithm was implemented in C++ programming language and ran on IBM x3250 M2 Linux server with Intel Xeon 2.27-GHz CPU and 90-GB memory. We tested it on the circuits designed by Himax Technologies Inc. [3]. Table I shows the information about benchmarks. The largest design has about four million cells, and the number of macros is

TABLE I
OUR BENCHMARKS

Cir.	Movable Macros	Preplaced Macros	I/O Pads	Standard Cells	Nets	DFGs	Data Nodes
Cir1	30	13	130	157K	181K	0	0
Cir2	71	47	365	1098K	1126K	0	0
Cir3	55	15	219	232K	235K	0	0
Cir4	38	15	169	321K	327K	0	0
Cir5	32	12	351	347K	352K	3	21
Cir6	66	3	471	214K	220K	1	5
Cir7	184	8	0	3843K	4166K	1	4
Cir8	549	5	0	3746K	4515K	1	5

549, where four benchmarks (i.e., Cir5–Cir8) have dataflow constraint.

Our design flow is as follows: we dump a netlist from a database of IC compiler (ICC), whose formats include DEF and LEF. Next, cells and macros are distributed over a placement region by NTUplace3 [7]. Then, the macros are legalized by our methodology. Finally, locations of macros are fed back to ICC by TCL files, and the placement of standard cells and signal net routing is completed by ICC.

Our experiment is divided into four parts. First, we show that placing macros following the DH is beneficial to wirelength and routability by removing the function from our methodology and then compare it with original methodology. The experimental results are shown in columns 2–7 in Table II, where columns 2–4 and 5–7 show the results when DH is ignored and considered, respectively. Wirelength and routing overflow (denoted by O.F.) are measured by ICC. The table shows that our methodology with DH gets better results no matter in wirelength or in routability. Wirelength of the methodology without DH is increased by 13%, and overflow is about two times larger than those when DH is considered. In the largest case Cir8. The wirelength of our methodology with DH is 13.1% shorter than that without DH. More importantly, its routing overflow is 41.1% lower. The experimental result demonstrates that to place macros with similar DH in the vicinity is helpful not only in wirelength but also in routability.

To demonstrate the effectiveness of our methodology, the second part of the experiment compares it with CP-tree [8] and the tool (i.e., ICC [1]). We do not compare with Chang *et al.* [5] and ECS [9] since Chang *et al.* [5] cannot release their benchmarks and codes for the confidential issue and ECS [9] only can provide their executable file. However, this file cannot work correctly in our cases. To meet the restriction of CP-tree [8] that all preplaced macros have to abut to the boundaries of a placement region, we modified their source codes by enlarging the dimensions of preplaced macros to fill empty space between the macros and chip boundaries. The results of CP-tree [8] and the tool are shown in columns 8–10 and 11–12 in Table II, respectively. Note that CP-tree and our methodology are both based on the same placement prototyping results of NTUplace3 [7]. NA in the table denotes that CP-tree failed to obtain legal placements in one day for the two large cases Cir7 and Cir8. In the results of the six small cases (i.e., Cir1–Cir6), our wirelength and routing overflow are, respectively, 13% and 65% smaller than those by CP-tree. Moreover, our runtime is significantly faster than theirs even though the runtime of the two large cases is ignored. Since some values are NA in a column, the normalized value of the column is computed according to the available values and we mark the value with the symbol "*." Similarly, the results of our methodology are better than the tool. The table reveals that wirelength and overflow by tool are 8% and 9.41 times larger than our results, respectively. Besides, our approach is much faster.

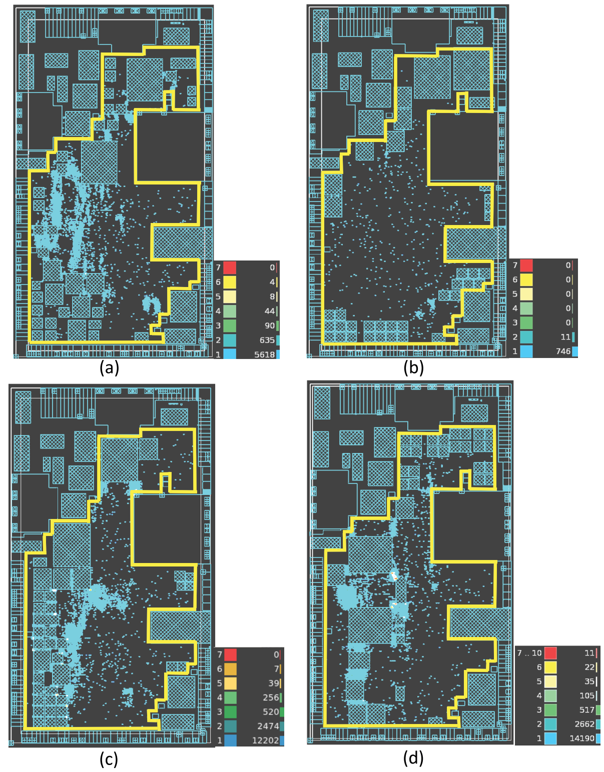


Fig. 10. Placement and global routing congestion map of Cir4 by (a) our methodology without DH, (b) our methodology with DH, (c) CP-tree, and (d) tool.

Fig. 10(a)–(d) shows the placements and global routing congestion (reported by ICC) of Cir4 according to our methodology without DH, our methodology with DH, CP-tree, and tool, respectively. For clarity, we mark preplaced macros by a bold line in the figure. The small picture beside each placement is the legend to show the status of routing congestion. For example, the placement in Fig. 10(a) has 5618 violations with overflow 1 and has 635 violations with overflow 2 and so on. Unlike the result in Fig. 10(b) which arranges macros with an equal dimension neatly, macros are placed arbitrarily in our methodology without considering DH in Fig. 10(a) even though they have an identical shape. Although CP-tree also arranges macros orderly when these macros have an identical shape and are placed in a local region in Fig. 10(c), the congestion is still worse than our result in Fig. 10(b) because too many macros are placed on the left side of the chip. Tool obtains the worst result because several macros are not pushed to the boundaries of the chip in Fig. 10(d) and it induces serious routing congestion between macros.

The third part shows the impact of considering dataflow in macro placement. We compare our methodology with

TABLE II

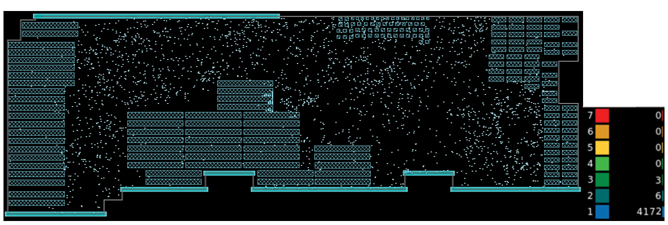
COMPARISONS OF OUR METHODOLOGY WITH DH AND WITHOUT DH, CP-TREE, AND TOOL IN WIRELENGTH AND ROUTABILITY

	Our methodology						CP-tree [8]			Tool [1]		
	w/o design hierarchy			w/ design hierarchy								
Cir.	# of O.F.	WL (10^7 μm)	T (s)	# of O.F.	WL (10^7 μm)	T (s)	# of O.F.	WL (10^7 μm)	T (s)	# of O.F.	WL (10^7 μm)	T (s)
Cir1	1500	1.08	27.2	507	1.12	25.3	1041	1.20	28188	574	1.18	206
Cir2	1622	6.99	75.6	1294	6.55	79.6	84554	8.91	37908	97061	6.91	1468
Cir3	1003	1.36	29.4	775	1.28	34.5	1256	1.30	23256	891	1.34	281
Cir4	7398	1.88	41.8	768	1.70	44.2	19971	1.94	27360	21869	1.88	326
Cir5	160	0.89	33.3	230	0.81	35.6	1212	0.90	28728	1502	0.91	417
Cir6	199	1.16	42.1	161	1.07	47.6	214	1.35	30276	179	1.21	233
Cir7	8865	25.7	230	4193	22.5	245	NA	NA	$>$ 1 day	7985	23.8	6715
Cir8	14300	16.9	336	8431	14.7	347	NA	NA	$>$ 1 day	23805	16.3	6681
Nor.	2.14	1.13	0.95	1	1	1	28.98*	1.20*	658.60*	9.41	1.08	19.01

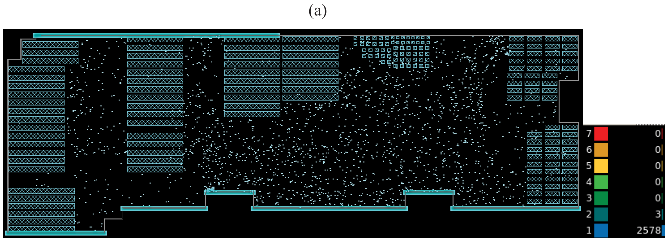
TABLE III

COMPARISONS OF OUR METHODOLOGY WITHOUT DFGS AND WITH DFGS

	w/o DFGs			w/ DFGs		
Cir.	# of O.F.	WL (10^7 μm)	T (sec)	# of O.F.	WL (10^7 μm)	T (sec)
Cir5	230	0.81	35.6	211	0.75	40.6
Cir6	161	1.07	47.6	170	0.97	55.5
Cir7	4193	22.5	245	3838	23.4	421.5
Cir8	8431	14.7	347	5138	13.9	924.7
Nor.	1	1	1	0.87	0.95	1.67

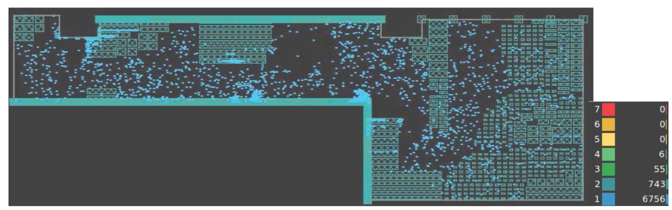


(a)

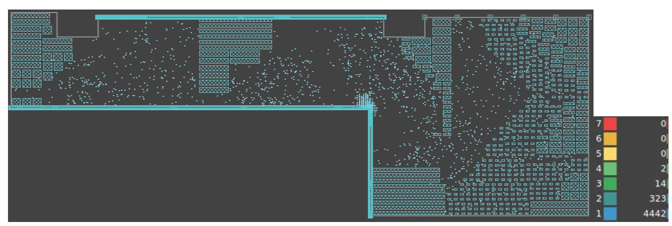


(b)

Fig. 11. Resulting placements and global routing congestion maps of Cir7 by our methodology. (a) Without DFGs. (b) With DFGs.



(a)



(b)

Fig. 12. Resulting placements and global routing congestion maps of Cir8 by our methodology. (a) Without DFGs. (b) With DFGs.

TABLE IV

COMPARISONS OF OUR METHODOLOGY AND RTL-AWARE MACRO PLACEMENT

	RTL-aware Macro Placer [19]			Our Methodology w/ DFGs		
Cir.	# of O.F.	WL (10^7 μm)	T (sec)	# of O.F.	WL (10^7 μm)	T (sec)
Cir5	644	0.9	45.6	238	0.81	17.2
Cir6	377	1.38	24.7	353	1.08	19.2
Cir7	54451	26.16	297	4846	23.24	357.8
Cir8	662904	16.15	597	544039	16.08	393.7
Nor.	1	1	1	0.56	0.89	0.76

DFGs with that without DFGs and show the experimental results in Table III, where columns 2–4 and 5–7 show the results of our methodology without DFGs and with DFGs, respectively. The table shows that our methodology with DFGs can obtain better results, where its wirelength and overflow are, respectively, 5% and 13% smaller than the results without DFGs. This demonstrates that the consideration of DFGs is helpful since DFGs will force macros to be placed at more appropriate positions. Fig. 11 (Fig. 12) shows the placements and global routing congestions of Cir7 (Cir8) based on our methodology without and with DFGs, respectively. The small picture beside each placement is the legend to show the status of routing congestion.

In the last part, we compare our dataflow-aware macro placement methodology with a recently published work, named RTL-aware macro placer [20]. Since their method cannot handle preplaced macros, we modified our test cases to form empty rectangle placement regions by moving preplaced macros to other locations. To make situation of overflow more obvious, we reduce one routing layer for each test case. The experimental results are shown in Table IV, and the values show that our methodology is better than RTL-aware macro placer [20] no matter in wirelength or in routing congestion. Our wirelength is 11% shorter than theirs. More importantly, our routing overflow is 44% smaller than their result. RTL-aware macro placer gets worse result because they

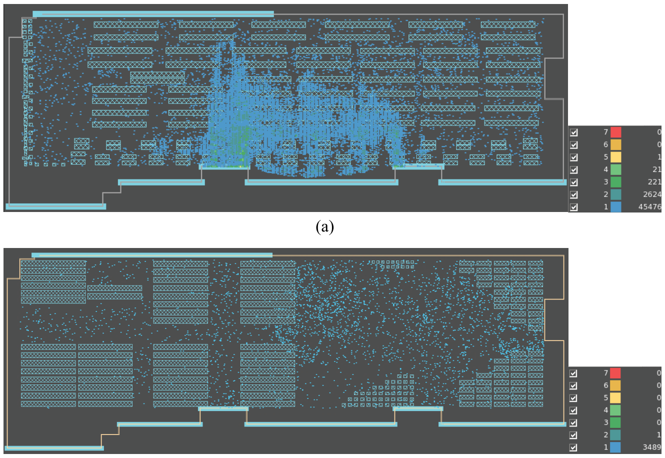


Fig. 13. Placements and global routing congestion maps of Cir7 by (a) RTL-aware macro placer [20]. (b) Our methodology.

may place macros in the center of a placement region, while our approach tends to place macros around boundaries. Note that the results in Table IV are worse than those in Table III because preplaced macros are placed at bad locations in the modified cases and placement area is smaller after the modification.

Fig. 13 shows the resulting placements and global routing congestions of Cir7 by RTL-aware macro placer [20] and our methodology, respectively. The result of [20] shown in Fig. 13(a) has serious routing congestion because macros are evenly distributed over a placement region. This causes many signals passing through macros since standard cells can only be placed between macros. On the contrary, our result keeps complete placement regions for standard cells in Fig. 13(b).

IX. Conclusion

This article has proposed an efficient and effective methodology to place macros. Even though a design contains several preplaced macros and they are placed at arbitrary locations in a chip, the proposed approach can be applied. We have extended our macro placement methodology to handle dataflow constraint. Experimental results have demonstrated that we can obtain better results than an SA-based approach and a tool in real designs. Moreover, wirelength and routing congestion can be further improved after the dataflow constraint is considered according to our experiment.

References

[1] *Synopsys*. Accessed: May 7, 2020. [Online]. Available: https://www.synopsys.com/company.html

[2] *Cadence*. Accessed: May 7, 2020. [Online]. Available: https://www.cadence.com/

[3] *Himax Technologies*. Accessed: May 7, 2020. [Online]. Available: https://www.himax.com.tw/zh/company/about-himax/

[4] C. Alpert, A. Kahng, G.-J. Nam, S. Reda, and P. Villarrubia, "A semi-persistent clustering technique for VLSI circuit placement," in *Proc. Int. Symp. Phys. Design (ISPD)*, 2005, pp. 200–207.

[5] C.-H. Chang, Y.-W. Chang, and T.-C. Chen, "A novel damped-wave framework for macro placement," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD)*, Nov. 2017, pp. 504–511.

[6] T.-C. Chen, P.-H. Yuh, Y.-W. Chang, F.-J. Huang, and T.-Y. Liu, "MP-trees: A packing-based macro placement algorithm for modern mixed-size designs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 9, pp. 1621–1634, Sep. 2008.

[7] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang, "NTUplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 7, pp. 1228–1240, Jul. 2008.

[8] Y.-F. Chen, C.-C. Huang, C.-H. Chiou, Y.-W. Chang, and C.-J. Wang, "Routability-driven blockage-aware macro placement," in *Proc. 51st Annu. Design Autom. Conf. Design Autom. Conf. (DAC)*, 2014, pp. 1–6.

[9] C.-H. Chiou, C.-H. Chang, S.-T. Chen, and Y.-W. Chang, "Circular-contour-based obstacle-aware macro placement," in *Proc. 21st Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2016, pp. 172–177.

[10] M. Cho, H. Xiang, H. Ren, M. M. Ziegler, and R. Puri, "LatchPlanner: Latch placement algorithm for datapath-oriented high-performance VLSI designs," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD)*, Nov. 2013, pp. 342–348.

[11] S. Chou, M.-K. Hsu, and Y.-W. Chang, "Structure-aware placement for datapath-intensive circuit designs," in *Proc. 49th Annu. Design Autom. Conf. (DAC)*, 2012, pp. 3–7.

[12] K.-R. Dai, W.-H. Liu, and Y.-L. Li, "NCTU-GR: Efficient simulated evolution-based rerouting and congestion-relaxed layer assignment on 3-D global routing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 3, pp. 459–472, Mar. 2012.

[13] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Proc. 19th Design Autom. Conf.*, 1982, pp. 175–181.

[14] R. Hinze, "Constructing red-black trees," in *Proc. WAAAPL*, 1999, vol. 22, no. 3, pp. 89–99.

[15] C.-C. Huang, B.-Q. Lin, H.-Y. Lee, Y.-W. Chang, K.-S. Wu, and J.-Z. Yang, "Graph-based logic bit slicing for datapath-aware placement," in *Proc. 54th Annu. Design Autom. Conf.*, Jun. 2017, pp. 18–20.

[16] J.-M. Lin, Y.-W. Chang, and S.-P. Lin, "Corner sequence—A P-admissible floorplan representation with a worst case linear-time packing scheme," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 11, no.4, pp. 679–686, Aug. 2003.

[17] J.-M. Lin, Y.-L. Deng, S.-T. Li, B.-H. Yu, L.-Y. Chang, and T.-W. Peng, "Regularity-aware routability-driven macro placement methodology for mixed-size circuits with obstacles," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 1, pp. 57–68, Jan. 2019.

[18] J. Lu *et al.*, "EPlace-MS: Electrostatics-based placement for mixed-size circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 5, pp. 685–698, May 2015.

[19] J. K. Ousterhout, "Corner stitching: A data-structuring technique for VLSI layout tools," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 3, no. 1, pp. 87–100, Jan. 1984.

[20] A. Vidal-Obiols, J. Cortadella, J. Petit, M. Galceran-Oms, and F. Martorell, "RTL-aware dataflow-driven macro placement," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2019, pp. 186–191.

[21] S. I. Ward *et al.*, "Keep it straight: Teaching placement how to better handle designs with datapaths," in *Proc. ACM Int. Symp. Int. Symp. Phys. Design*, 2012, pp. 79–86.

[22] M.-C. Wu and Y.-W. Chang, "Placement with alignment and performance constraints using the B∗-tree representation," in *Proc. ICCD*, Oct. 2004, pp. 568–571.

Jai-Ming Lin (Member, IEEE) received the B.S., M.S., and Ph.D. degrees from the National Chiao Tung University, Hsinchu, Taiwan, in 1996, 1998, and 2002, respectively, all in computer science.

From 2002 to 2007, he was an Assistant Project Leader with the CAD Team, Realtek Corporation, Hsinchu, Taiwan. He is currently a Professor with the Department of Electrical Engineering, National Cheng Kung University, Tainan, Taiwan. His research interests include physical design for 3-D or 2.5-D integrated circuits (ICs), macro placement and cell placement, and power planning.

You-Lun Deng received the B.S. degree in electrical engineering from the National Kaohsiung Normal University, Kaohsiung, Taiwan, in 2016, and received the M.S. degree in electrical engineering from the National Cheng Kung University, Tainan, Taiwan, in 2018.

His research interest includes physical design for macro placement.

Ya-Chu Yang received the B.S. degree in electrical engineering from Feng Chia University, Taichung, Taiwan, in 2017, and the M.S. degree in electrical engineering from the National Cheng Kung University, Tainan, Taiwan, in 2019.

Her research interest includes physical design for macro placement.

Jia-Jian Chen received the B.S. degree in electrical engineering from the National Taipei University of Technology, Taipei, Taiwan, in 2018, and the M.S. degree in electrical engineering from the National Cheng Kung University, Tainan, Taiwan, in 2020.

His research interest includes physical design for macro placement.

Po-Chen Lu received the B.S. degree in electrical engineering from the National Chung Hsing University, Taichung, Taiwan, in 2019. He is currently pursuing the master's degree in electrical engineering with the National Cheng Kung University, Tainan, Taiwan.

His research interest includes physical design for macro placement.