UTPlaceF 2.0: A High-Performance Clock-Aware FPGA Placement Engine

WUXI LI, YIBO LIN, MENG LI, SHOUNAK DHAR, AND DAVID Z. PAN,
The University of Texas at Austin

Modern field-programmable gate array (FPGA) devices contain complex clock architectures on top of configurable logics. Unlike application specific integrated circuits (ASICs), the physical structure of clock networks in an FPGA is pre-manufactured and cannot be adjusted to different applications. Furthermore, clock routing resources are typically limited for high-utilization designs. Consequently, clock architectures impose extra clock constraints and further complicate physical implementation tasks such as placement. Traditional ASIC placement techniques only optimize conventional design metrics such as wirelength, routability, power, and timing without clock legality consideration. It is imperative to have new techniques to honor clock constraints during placement for FPGAs. In this paper, we propose a high-performance FPGA placement engine, UTPlaceF 2.0, that optimizes wirelength and routability while honoring complex clock constraints. Our proposed approaches consist of an iterative minimum-cost-flow-based cell assignment as well as a clock-aware packing for producing clock-legal yet high-quality placement solutions. UTPlaceF 2.0 won the first place in ISPD'17 clock-aware FPGA placement contest organized by Xilinx, outperforming the second- and the third-place winners by 4.0% and 10.0%, respectively, in routed wirelength with competitive runtime, on a set of industry benchmarks.

CCS Concepts: • **Hardware → Reconfigurable logic and FPGAs**; Design Aids;

Additional Key Words and Phrases: FPGA, Placement, Packing, Clock Legalization

ACM Reference format:
WUXI LI, YIBO LIN, MENG LI, SHOUNAK DHAR, and DAVID Z. PAN. 2017. UTPlaceF 2.0: A High-Performance Clock-Aware FPGA Placement Engine. *ACM Trans. Des. Autom. Electron. Syst.* 0, 0, Article 0 (April 2017), 23 pages.
https://doi.org/0000001.0000001

1 INTRODUCTION

Placement is one of the most important and time-consuming optimization steps in FPGA implementation flow and it can significantly affect the efficiency and quality of mapped designs on FPGA devices. The placement problem has attracted great attention from both academia and industry and has been intensively studied during the last two decades. However, with the increasing complexity and scale of modern FPGA devices, today's FPGA architecture imposes various intricate constraints, which have not yet been paid enough attention to, during placement stage. These constraints have great impact on placement quality in terms of traditional design metrics such as wirelength,

Author's addresses: W. Li, Y. Lin, M. Li, S. Dhar, and D. Z. Pan; Electrical and Computer Engineering Department, University of Texas at Austin, Austin, TX 78712; emails: wuxi.li@utexas.edu, dpan@ece.utexas.edu.

1084-4309/2017/4-ART0 $15.00
https://doi.org/0000001.0000001

routability, power, and timing. Therefore, it is imperative to have new placement techniques to honor these complicated architecture constraints.

Clock rules, among various FPGA architecture constraints, are of great importance, not only because of their significant impact on timing closure and power dissipation but their imposed layout constraints that affect how efficiently other logics can be mapped. With the consideration of clock rules, placement turns out to be much more difficult for FPGAs even to find a feasible solution. For modern FPGAs, clock network planning must be involved during or even before placement stage due to their pre-manufactured clock networks, which cannot be adjusted to different designs, as well as their limited clock routing resources. The interdependency between clock sink placement and clock network planning makes the clock-aware placement for FPGAs a challenging chicken and egg problem.

Besides placement, clock rules also impose severe difficulties to FPGA packing. Traditional logical packing techniques without considering physical information are blind to layout constraints introduced by clocks, which often produce unplaceable solutions. So it becomes critical and necessary to incorporate clock rules together with physical information for packing algorithms to deliver clock- and placement-friendly netlists.

While many existing works have proposed fairly mature placement and packing techniques for FPGAs to optimize conventional design metrics such as wirelength, routability, power, and timing [3–5, 7, 8, 10, 11, 13–18, 20], there has been limited research effort on placement and packing with the awareness of clock rules. The closest related previous work is [6], which proposes a cost function for cell swapping that penalizes high-clock-usage placement and integrates it into a conventional simulated-annealing-based placement engine to produce clock-legal solutions. However, their approach suffers from slow annealing process and its quality heavily depends on the cost function tuning. Moreover, their approach can easily get stuck in some local swappings and hence unable to resolve global clock congestions.

To address the clock legalization challenges in FPGA placement for large-scale state-of-the-art commercial FPGAs, we proposed a high-performance clock-aware placement engine, UTPlaceF 2.0, which simultaneously optimizes the conventional placement objectives of wirelength and routability and honors complicated global and detailed clock constraints. The major contributions of this work are highlighted as follows.

- An iterative minimum-cost-flow-based cell assignment technique producing clock-legal solutions with minimized perturbation to placements optimized for other design metrics (e.g., wirelength and routability).
- A clock-aware packing technique with probabilistic clock distribution estimation for producing clock- and placement-friendly netlists.
- Won first place in ISPD'17 clock-aware FPGA placement contest [22] on industry-strength benchmarks released by Xilinx [19] and outperforms the second- and third-place winners by 4.0% and 10.0%, respectively, in routed wirelength with competitive runtime.

The rest of this paper is organized as follows. Section 2 reviews the preliminaries and presents the UTPlaceF 2.0 framework overview. Section 3 gives the details of UTPlaceF 2.0 algorithms. Section 4 shows the experimental results, followed by the conclusion in Section 5.

2 PRELIMINARIES AND OVERVIEW

In this section, we will briefly introduce the targeted FPGA clocking architecture and give the constraints and problem formulation for clock-aware placement. At the end of this section, an overview of the proposed UTPlaceF 2.0 framework will be exposed.

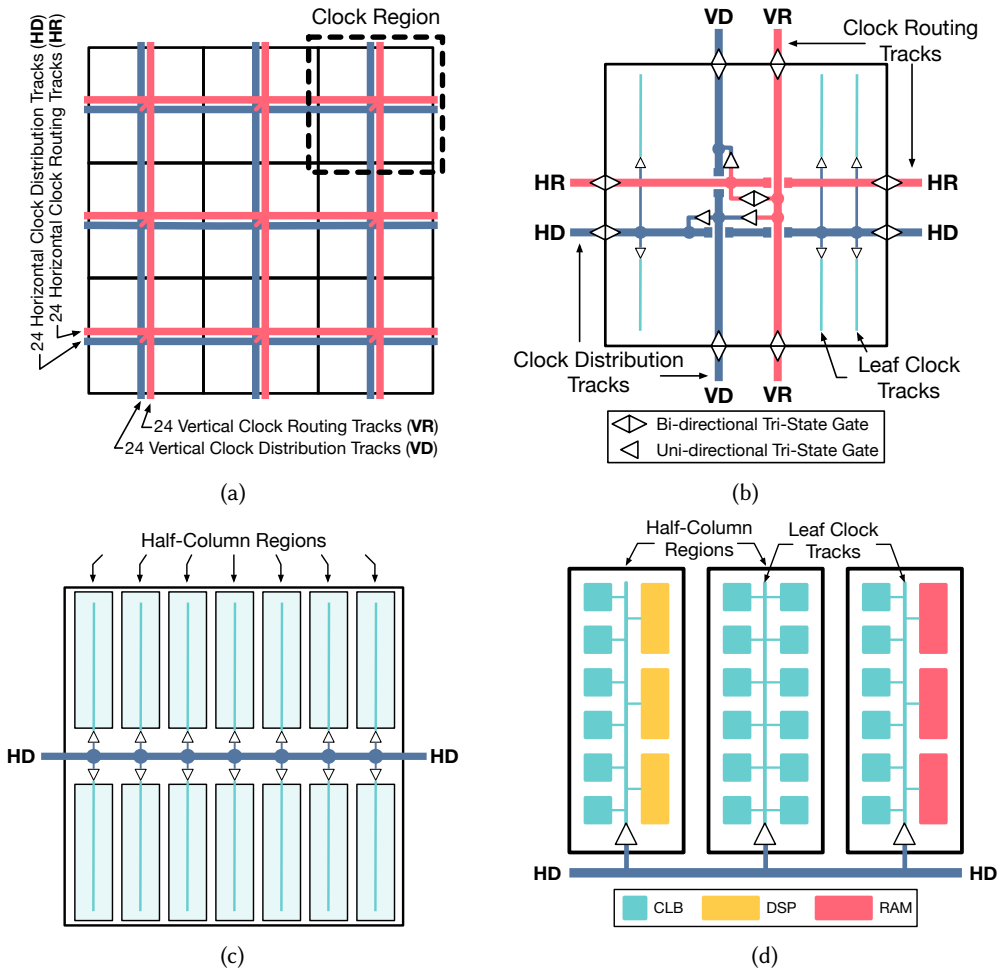


Fig. 1. Clocking architecture of Xilinx Ultrascale. (a) A global view of 3 by 3 of 9 clock regions, the clock routing network (red), and the clock distribution network (blue). (b) A detailed view of clock routing tracks, clock distribution tracks, and leaf clock tracks within a clock region. (c) A global view of half-column regions within a clock region. (d) A detailed view of three adjacent half-column regions with different logic resources.

2.1 Clocking Architecture

UTPlaceF 2.0 is targeted to Xilinx UltraScale VU095 [19], which was adopted in both ISPD'16 and ISPD'17 FPGA placement contests [21, 22]. In this particular architecture, each FPGA device is divided into 5 by 8 of 40 clock regions and each clock region contains synchronous elements such as configurable logic blocks (CLBs), digital signal processors (DSPs), random-access memories (RAMs), and so on. Each CLB further consists of lookup tables (LUTs) and flip-flops (FFs).

The global clocking architecture is a two-level structure containing a clock routing network on top of a clock distribution network. As shown in Figure 1(a), both routing and distribution networks have 24 horizontal and 24 vertical tracks running through each clock region. A detailed view of clocking architecture within a clock region is exposed in Figure 1(b). The connection between any

horizontal and vertical clock routing tracks (HR and VR) is bidirectional but the same connection (HD and VD) is unidirectional (from VD to HD) in clock distribution network. Besides, a clock can hop onto VD through their corresponding HR and VR, but there is no path back vice versa. Therefore, once a clock is hooked on the clock distribution network (HD and VD), it can only stay on it or go to the leaf-level clock tracks.

For the leaf clock networks, a clock region is further divided into multiple half-column regions of two-site width and half-clock-region height as shown in Figure 1(c). Figure 1(d) illustrates a detailed view of three adjacent half-column regions. Different half-column regions might contain distinct logic resources, such as CLBs, DSPs, and RAMs, but they have similar leaf clock networks to directly drive clock sinks within them. More detailed clocking architecture can be found in [22].

2.2 Clock Constraints for Placement

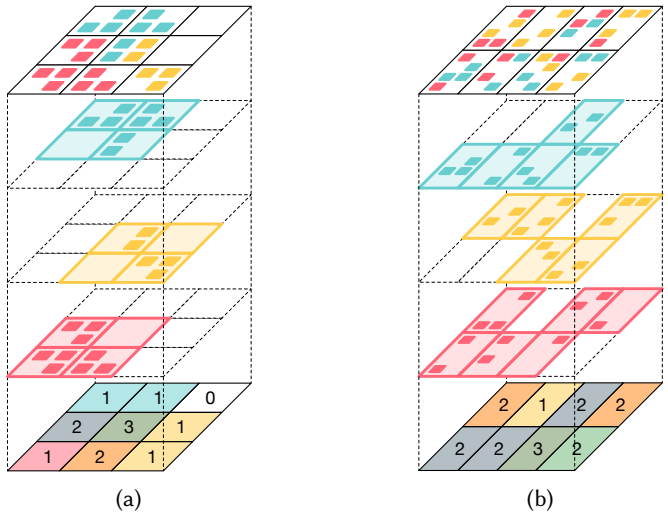


Fig. 2. Illustration of clock demand calculation for clock regions and half-column regions. Different colors represent different clocks. (a) Global clock demand calculation for clock regions. (b) Clock demand calculation for half-column regions within a clock region.

Restricted by the clocking architecture, two major constraints, namely clock region constraint and half-column region constraint, are imposed in the placement stage.

Clock region constraint is introduced by the limited clock routing/distribution tracks and it restricts the global clock demand in each clock region must be equal to or less than 24 in our targeted FPGA architecture. For a clock region, its global clock demand is defined as the total number of clock nets that have their bounding boxes intersected with it. Figure 2(a) illustrates how global clock demands are calculated for a simple placement with three clock nets. The top layer shows the sink distribution for each clock, the three middle layers represent the spanning clock regions of each clock and the layer in the bottom gives the final global clock demand in each clock region.

Similarly, imposed by the limited leaf clock tracks, half-column region constraint requires that each half-column region can only contain at most 12 clocks. Figure 2(b) illustrates the clock demand calculation for half column regions within a clock region. Different from clock regions, the clock

demand in a half-column region is only the number of clocks inside it, regardless of clock net bounding boxes.

2.3 Problem Formulation

In modern FPGA placement, the optimization usually includes multiple objectives, such as wirelength, which is measured by Half-Perimeter Wirelength (HPWL), and routability. Wirelength is still regarded as the major objective, since it is a good first-order approximation of many other metrics, e.g., power and timing. However, pure wirelength-driven placement often results in routing quality degradation and even unroutable solutions. Therefore, in UTPlaceF 2.0, wirelength and routability are optimized simultaneously.

To produce legal placement solutions, apart from clock constraints, packing rules also need to be satisfied when multiple LUTs and FFs are placed into the same CLB site. The detailed packing rules for our targeted FPGA are elaborated in [21].

With all constraints and objectives defined, we now define our clock-aware placement problem as follows.

Problem 1 (Clock-Aware Placement). *Given a netlist of LUTs, FFs, DSPs, RAMs and I/Os, produce a legal placement solution with minimized routed wirelength, meanwhile packing rules, clock region constraint, and half-column region constraint are satisfied.*

2.4 UTPlaceF 2.0 Overview

The overall flow of UTPlaceF 2.0 is shown in Figure 3. UTPlaceF 2.0 is a natural extension of UTPlaceF [7] and consists of five major steps: 1) flat initial placement (FIP), 2) packing, 3) CLB-level global placement, 4) legalization, and 5) detailed placement. On top of conventional wirelength and routability optimizations performed in UTPlaceF, clock constraints are explicitly considered throughout the UTPlaceF 2.0 framework.

FIP is responsible for producing physical location and routing congestion information of each cell to better guide decision making in the packing stage. It consists of two phases: 1) pure wirelength-driven phase and 2) clock- and routability-driven phase. In each iteration of the first phase, a quadratic analytical placement is solved to minimize wirelength, followed by a rough legalization [9] for cell overlapping removal. After that, a sequence of global moves are performed to further refine rough-legalized placement while preserving cell density. In the second phase, besides the conventional routability optimization, an additional clock region assignment (see Section 3.1) step is called after the quadratic placement. It produces a cell-to-clock-region assignment solution that satisfies the clock region constraint. Then, the rough legalization and heterogeneous cell (e.g., DSPs, RAMs, and I/Os) legalization are only performed within each clock region to honor the assignment result. UTPlaceF 2.0 stops FIP once the wirelength converges.

In the packing stage, a probability-based estimation is performed to predict the clock distribution in the final placement solution. Then, by incorporating the estimated clock information into the original packing algorithm in UTPlaceF, the packing in UTPlaceF 2.0 is enhanced to be clock-aware (see Section 3.2).

CLB-level global placement is performed immediately after packing to further optimize the placement for the post-packing netlist. It shares the same framework with the second phase of FIP, and use the final solution of FIP as the starting point to speed-up placement convergence.

In legalization stage, while minimizing the conventional objective of pin movement, clock region constraint and half-column region constraint are rigorously respected by clock region assignment technique and half-column region assignment technique (see Section 3.3), respectively.

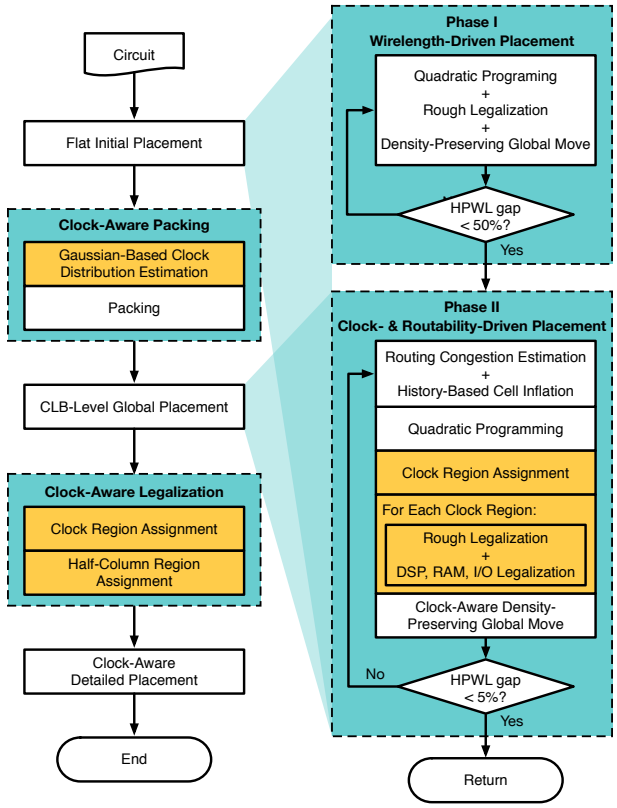


Fig. 3. The overall flow of UTPlaceF 2.0. Yellow-shaded blocks indicates major new/modified steps that differ from UTPlaceF.

Finally, detailed placement techniques in UTPlaceF is extended to be clock-aware and maintain the clock legality throughout the wirelength and routability optimization process before the final solution is produced.

3 UTPLACEF 2.0 ALGORITHMS

In this section, we will explain UTPlaceF 2.0 algorithms, including clock region assignment, clock-aware packing, and half-column region assignment, in details.

3.1 Clock Region Assignment

Clock region assignment is a key step in UTPlaceF 2.0 to ensure the satisfaction of clock region constraint. It is intensively called throughout major steps, such as FIP, CLB-level global placement, and legalization, in UTPlaceF 2.0. Therefore, it has a significant impact on both solution quality and runtime. Here we formally define the clock region assignment problem as follows.

Problem 2 (Clock Region Assignment). *Given a rough-legalized placement and logic resource capacity of each clock region, assign cells to clock regions to minimize total pin movement without logic resource and global clock overflow.*

Given the notations defined in Table 1, Problem 2 can be written as a binary minimization problem with linear and boolean logical constraints as shown in Formulation (1).

Table 1. Notations Used in Clock Region Assignment

$\mathcal{V}$	The set of cells.
$\mathcal{V}^{(s)}$	The set of cells of resource type $s \in$ {CLB, DSP, RAM}.
P_i	The number of pins in cell i.
$A_i^{(s)}$	The cell i's demand for resource type $s \in$ {CLB, DSP, RAM}.
$\mathcal{R}$	The set of clock regions.
$C_j^{(s)}$	The clock region j's capacity for resource type $s \in$ {CLB, DSP, RAM}.
$D_{i,j}$	The physical distance between cell i and clock region j.
$\mathcal{E}$	The set of clock nets.
$Z_{i,e}$	A binary value represents whether cell i is in clock net e.
$\mathcal{L}_j^+$	The set of clock regions that are left to or in the same column of clock region j.
$\mathcal{R}_j^+$	The set of clock regions that are right to or in the same column of clock region j.
$\mathcal{B}_j^+$	The set of clock regions that are below or in the same row of clock region j.
$\mathcal{T}_j^+$	The set of clock regions that are above or in the same row of clock region j.

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad \sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{R}} P_i \cdot D_{i,j} \cdot \boldsymbol{x}_{i,j}, \tag{1a}$$

$$\text{subject to} \quad \boldsymbol{x}_{i,j} \in \{0,1\}, \forall i \in \mathcal{V}, \forall j \in \mathcal{R}, \tag{1b}$$

$$\sum_{j\in\mathcal{R}} \boldsymbol{x}_{i,j} = 1, \forall i \in \mathcal{V}, \tag{1c}$$

$$\sum_{i\in\mathcal{V}} A_i^{(s)} \cdot \boldsymbol{x}_{i,j} \le C_j^{(s)}, \forall j \in \mathcal{R}, \forall s \in \{\text{CLB, DSP, RAM}\} \tag{1d}$$

$$\sum_{e\in\mathcal{E}} \Big[\Big(\bigvee_{q\in\mathcal{L}_j^+} Z_{i,e} \cdot \boldsymbol{x}_{i,q} \Big) \wedge \Big(\bigvee_{q\in\mathcal{R}_j^+} Z_{i,e} \cdot \boldsymbol{x}_{i,q} \Big) \wedge \Big(\bigvee_{q\in\mathcal{B}_j^+} Z_{i,e} \cdot \boldsymbol{x}_{i,q} \Big) \wedge \Big(\bigvee_{q\in\mathcal{T}_j^+} Z_{i,e} \cdot \boldsymbol{x}_{i,q} \Big) \Big] \le 24, \forall j \in \mathcal{R}. \tag{1e}$$

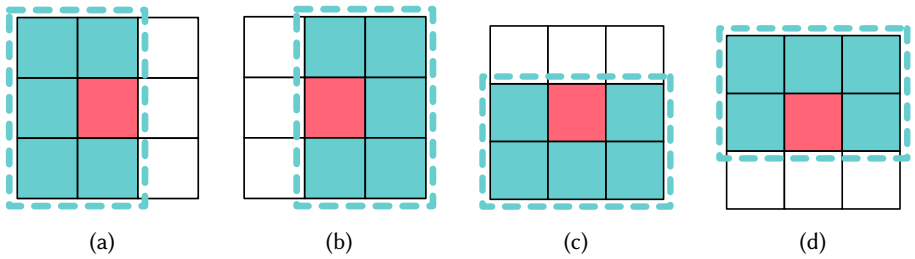


Fig. 4. Illustration of the sets of clock regions (dashed regions) (a) $\mathcal{L}_j^+$, (b) $\mathcal{R}_j^+$, (c) $\mathcal{B}_j^+$, and (d) $\mathcal{T}_j^+$ of clock region j (red) defined in Table 1.

Formulation (1) is optimized over binary variables $\boldsymbol{x}_{i,j}$ to minimize the objective (1a) of total pin movement. If cell $i \in \mathcal{V}$ is assigned to clock region $j \in \mathcal{R}$, then $\boldsymbol{x}_{i,j} = 1$, otherwise $\boldsymbol{x}_{i,j} = 0$. The constraint (1c) ensures that each cell is assigned to one and only one clock region. The constraint (1d) guarantees the demands are no greater than the capacities for all logic resource types (e.g., CLB, DSP, and RAM) in each clock region. The boolean logical constraint (1e) ensures the satisfaction of clock region constraint. The sets of clock regions $\mathcal{L}_j^+$, $\mathcal{R}_j^+$, $\mathcal{B}_j^+$, and $\mathcal{T}_j^+$ of clock region j are illustrated in Figure 4. Intuitively, for a given clock region j, if and only if a clock net has cells located in all $\mathcal{L}_j^+$, $\mathcal{R}_j^+$, $\mathcal{B}_j^+$, and $\mathcal{T}_j^+$, its bounding box would intersect with j and occupy global clock resource in it. The constraint (1e) sums up all such clock nets for each clock region and restricts the clock usage no more than the clock capacity, which is 24 in our target FPGA architecture.

3.1.1 The Relaxation Algorithm. With proper modeling, the boolean logical constraint (1e) can be transformed to a set of linear constraints and then Formulation (1) can be optimally solved utilizing integer linear programming techniques. However, integer linear programming is computationally expensive and suffers from unaffordable runtime for our application. Therefore, here we relax Formulation (1) to an easier problem that can be efficiently solved, as shown in Formulation (2).

$$\underset{\boldsymbol{x},\lambda}{\text{minimize}} \quad \sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{R}}(\mathrm{P_i}\cdot\mathrm{D_{i,j}}+\lambda_{\mathrm{i,j}})\cdot\mathrm{x_{i,j}}, \tag{2a}$$

$$\text{subject to} \quad \boldsymbol{x}_{i,j} \in \{0,1\}, \forall i \in \mathcal{V}, \forall j \in \mathcal{R}, \tag{2b}$$

$$\sum_{j\in\mathcal{R}} \boldsymbol{x}_{i,j} = 1, \forall i \in \mathcal{V}, \tag{2c}$$

$$\sum_{i\in\mathcal{V}} A_i \cdot \boldsymbol{x}_{i,j} \le C_j, \forall j \in \mathcal{R}. \tag{2d}$$

Compared to the original Formulation (1), Formulation (2) has two major differences: 1) instead of considering capacity constraint (1d) simultaneously for all logic resource types, here we only consider the capacity constraint (2d) for one resource type at a time (i.e., we consider three resource types separately); 2) we remove the boolean logical constraint (1e) and add a penalty multiplier $\lambda_{i,j}$ for each potential cell $i \in \mathcal{V}$ to clock region $j \in \mathcal{R}$ assignment $\boldsymbol{x}_{i,j}$ to the objective (2a).

The main idea of our relaxation can be explained as follows. We first ignore the clock region constraint (1e) and only respect each logic resource constraint separately. Each time after solving Formulation (2), we properly penalize assignments causing clock overflow by updating the corresponding penalty multipliers $\lambda_{i,j}$ in the objective (2a). By repeating the process of solving and updating Formulation (2), the assignment will progressively converge to a clock-feasible solution.

The proposed relaxation-based clock region assignment algorithm is summarized in Algorithm 1. Cells are first divided into three groups based on their resource types, including CLB, DSP, and RAM, in line 1. Then all penalty multipliers $\lambda_{i,j}, \forall i \in \mathcal{V}, \forall j \in \mathcal{R}$ are initialized to zero in line 2, which degenerates the Formulation (2) into a pure logic-resource-constrained pin-movement minimization problem without clock legality consideration. Within the loop from line 3 to line 8, we first solve Formulation (2) for each cell group under current penalty multiplier settings, and then in line 7, penalty multipliers $\lambda_{i,j}$ are updated according to the assignment solutions produced from line 4 to line 6. The penalty multipliers are incrementally updated to penalize assignments causing clock overflow and new assignments produced by the Formulation (2) with these updated penalty multipliers will be progressively closer to clock-legal solutions after each iteration. The process of solving and updating Formulation (2) from line 3 to line 8 is repeated until a clock-overflow-free assignment is reached.

Algorithm 1 Clock Region Assignment with Relaxation

Input: A rough-legalized placement.
Output: A movement-minimized assignment without logic resource and clock overflow.

```
Divide cells V into three groups, V^(CLB), V^(DSP), and V^(RAM), based on their logic resource types.
λ_{i,j} ← 0, ∀i ∈ V, ∀j ∈ R;
while clock overflow exists do
    for each U ∈ {V^(CLB), V^(DSP), V^(RAM)} do
        Solve Formulation (2) for U;                    ▷ See Section 3.1.2
    end for
    Update penalty multiplier λ;                        ▷ See Section 3.1.3
end while
```

3.1.2 Minimum-Cost Flow Transformation. One notable merit of our relaxation in Formulation (2) is that it can be transformed into a minimum-cost-flow problem, which is a fairly mature field with many efficient algorithms [1].

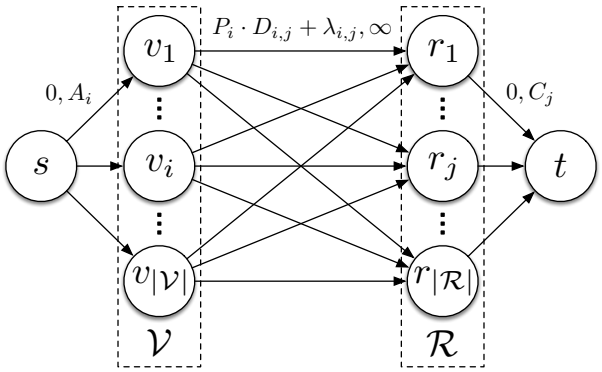


Fig. 5. The minimum-cost flow representation of Formulation (2). Pair of numbers (e.g., $P_i \cdot D_{i,j} + \lambda_{i,j}, \infty$) on each edge represents cost and capacity, respectively, and ∞ means unlimited capacity.

Figure 5 illustrates the minimum-cost flow representation of Formulation (2). If all cells in $\mathcal{V}$ have unit resource demand ($A_i = 1, \forall i, j \in \mathcal{V}$), optimally solving Formulation (2) is equivalent to computing the minimum-cost flow of amount $\sum_{i \in \mathcal{V}} A_i$ on the graph. For cases with non-unit resource demands, however, the assignment solutions corresponding to their minimum-cost flow results may contain cells that are split into multiple clock regions, which require extra post-processing steps and slight relaxation on constraint (2d) to guarantee solution feasibility. In UTPlaceF 2.0, we always apply unit cell resource demand ($A_i = 1$) to ensure the solutions produced by our minimum-cost flows transformation are feasible.

3.1.3 Penalty Multiplier Updating. Algorithm 1 is a generalized relaxation framework for solving the clock region assignment problem. One key step within this framework is how to update the penalty multiplier λ after each assignment iteration. Various updating strategies can converge to different feasible solutions. In this section, we will only discuss one specific updating method, which is applied in UTPlaceF 2.0, and its effectiveness is demonstrated by our experiments.

The guiding idea of our λ updating method is that we hope clock overflow can be mitigated by forbidding some clocks from occupying the overflowed clock regions. We observe that to block a clock net from taking clock resources in a given clock region, all the cells in this clock net must be strictly restricted in the region below, above, left, or right to the clock region, as shown in Figure 6. Otherwise, the clock bounding box must intersect with the clock region. Thus, intuitively, clock congestion can be resolved by iteratively pushing clock nets to these four escaping regions corresponding to overflowed clock regions.

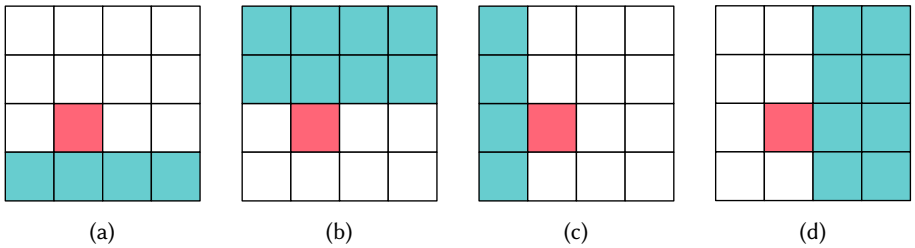


Fig. 6. The four escaping regions (green) that are defined as the regions (a) below, (b) above, (c) left, and (d) right to a given clock region (red), respectively. To avoid employing clock resources of the given clock region, all cells of a clock net must be simultaneously placed into one of these four escaping regions.

Inspired by this observation, we propose our penalty multiplier updating method summarized in Algorithm 2. We first choose the most overflowed clock region as our target clock region to resolve in line 1 and locate the four escaping regions illustrated in Figure 6 for the target clock region from line 2 to line 5. Then from line 6 to line 11, for clock nets occupying clock resources in the target clock region, we compute their moving costs to each of these four escaping regions and store the calculation results as candidates for later overflow resolving. Within the loop from line 16 to line 30, these candidates are accessed in ascending order of their costs. For each candidate, in line 17, we first ensure that no other candidate associated with the same clock net has been chosen and then call function `IsFeasible` to reject candidates leading to infeasible solutions. The feasibility checking and function `IsFeasible` will be further discussed later in this section. If a candidate is feasible, we block all assignments between cells in the candidate clock net and clock regions outside the candidate escaping region by setting the corresponding penalty multipliers to infinity from line 20 to line 24. This operation prevents the target clock region from being further employed by the candidate clock in the later assignment iterations. The loop from line 16 to line 30 is repeated until all overflow in the target clock is fully resolved or the number of resolved overflow reaches the limit I_{max}, which is 2 in UTPlaceF 2.0 by default. Intuitively, I_{max} is the maximum descent step for clock overflow resolving. Under smaller I_{max}, clock congestion can be resolved more smoothly and evenly among different clock regions with the cost of more assignment iterations.

Now we explain the function `Cost`(e, b) that computes the cost of pushing clock e to escaping region b. The objective of our assignment is to minimize total pin movement. In addition, each cell movement may lead to logic resource overflow in the target escaping region. Thus the cost consists of two components: pin movement cost and logic resource cost, shown as follows,

$$Cost(e, b) = \sum_{i \in \mathcal{V}(e)} (P_i \cdot d_{i,b} + \alpha \cdot A_i^{(\mathrm{CLB})} + \beta \cdot A_i^{(\mathrm{DSP})} + \gamma \cdot A_i^{(\mathrm{RAM})}). \tag{3}$$

Algorithm 2 Penalty Multiplier Updating For Clock Region Constraint

Input: Penalty multiplier λ. The maximum clock overflow descent step I_{max} for each clock region in each iteration.

Output: An updated penalty multiplier λ that will results in less clock overflow.

Find the clock region r with the largest clock overflow O_{max};
$B_r \leftarrow$ the region below to r;
$T_r \leftarrow$ the region above to r;
$L_r \leftarrow$ the region left to r;
$R_r \leftarrow$ the region right to r;
$l \leftarrow \emptyset$;
for each clock $e \in \mathcal{E}$ occupying clock resource of r **do**
    for each region $b \in \{B_r, T_r, L_r, R_r\}$ **do**
        $l \leftarrow l \cup \{\texttt{Cost}(e, b)\}$; ▷ See Equation (3)
    end for
end for
Sort candidates in l by ascending order of their cost;
$I \leftarrow \texttt{min}(O_{max}, I_{max})$;
$i \leftarrow 0$;
$chosen[e] \leftarrow false, \forall e \in \mathcal{E}$;
for each $\texttt{Cost}(e, b) \in$ sorted l **do**
    if $chosen[e]$ **or** $\neg\texttt{IsFeasible}(e, b, \lambda)$ **then**
        Continue;
    end if
    for each cell i in clock e **do**
        for each clock region j that is not in region b **do**
            $\lambda_{i,j} \leftarrow \infty$;
        end for
    end for
    $chosen[e] \leftarrow true$;
    $i \leftarrow i + 1$;
    if $i \geq I$ **then**
        Return;
    end if
end for

where $\mathcal{V}(e)$ denotes the set of cells in clock net e, and $d_{i,b}$ denotes the physical distance between cell i and box region b. The trade-off weights α, β, and γ are set to 5, 50, and 50, respectively, in our experiments.

The only thing left now is the feasibility checking function $\texttt{IsFeasible}(e, b, \lambda)$. For an intermediate solution with the logic resource constraint satisfied, arbitrarily blocking a set of assignments cannot further guarantee the existence of feasible solutions. For example, if clock nets are restricted in small regions without enough logic resources to accommodate all the cells, the corresponding assignment problem will be infeasible. One straightforward method to avoid this issue is to solve the updated minimum-cost flow for the graph in Figure 5 to check if a feasible solution exists before actual applying the new assignment blockings. However, this method is far too cumbersome to

be applied in our iterative framework, which motivates us to propose a light-weight yet effective feasibility checking method.

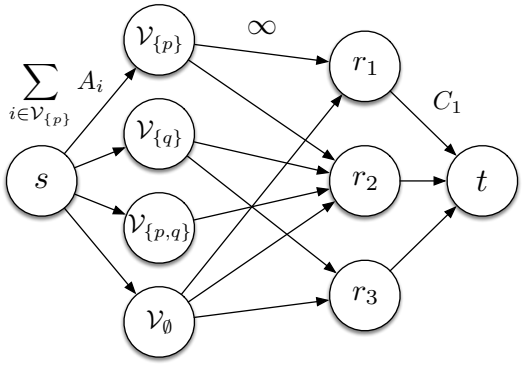


Fig. 7. Illustration of our maximum-flow-based feasibility checking. Numbers on edges represent their capacities.

Instead of solving the minimum-cost flow for the graph in Figure 5, we construct a maximum-flow graph, as illustrated in Figure 7, to perform fast feasibility checking. Here we are trying to assign a netlist with two clock nets p and q to three clock regions r_1, r_2, and r_3. We divide all cells into four mutually exclusive groups $\mathcal{V}_{\{p\}}$, $\mathcal{V}_{\{q\}}$, $\mathcal{V}_{\{p,q\}}$, and $\mathcal{V}_{\emptyset}$, based on the clock nets they belong to, where $\mathcal{V}_p$ and $\mathcal{V}_q$ denote the groups of cells belonging only to p and q, respectively, $\mathcal{V}_{\{p,q\}}$ denotes the group of cells belonging to both p and q, and $\mathcal{V}_{\emptyset}$ is the set of cells without clock nets. In this graph construction, we only introduce edges for unblocked assignments ($\lambda_{i,j} \neq \infty$), and specific to this example, clock p cannot be assigned to r_3 and clock q cannot be assigned to r_1. Then, with the proper edge capacity settings as shown in Figure 7, checking the assignment feasibility of Formulation (2) is equivalent to computing the maximum flow of amount $\sum_{i \in \mathcal{V}} A_i$ in this graph. If the resulting maximum flow value is equal to $\sum_{i \in \mathcal{V}} A_i$, a feasible solution with the set of new assignment blockings applied must exist, otherwise, the assignment problem becomes infeasible.

It should be noted that our maximum-flow-based checking are performed for three resource types (CLB, DSP, and RAM) separately each time and we only conclude the updated problem is feasible when all three checks are passed.

Compared with the minimum-cost flow graph in Figure 5, the problem size is dramatically reduced by the clock grouping in the graph construction in Figure 7. In addition, edge costs are removed in the maximum-flow graph since we solve it only for feasibility checking purpose. Therefore, the proposed maximum-flow-based technique enables a much faster yet reliable feasibility checking process.

3.1.4 Speed-up Techniques. The minimum-cost flow formulation shown in Figure 5 is optimal for given λ and unit logic resource demand but may suffer from long runtime for large designs. Instead of solving flat minimum-cost flow problems, we here propose a geometric clustering technique to reduce the problem size by grouping cells that are physically close and share the same set of clock nets together. Our proposed geometric clustering technique can significantly speed up the minimum-cost flow solving process while preserving good solution quality.

Here we use a simple example in Figure 8 to illustrate our clustering idea. In this example, there are twelve cells and two clock nets p and q. The "$\mathcal{S}, A$" pair (e.g., $\{p, q\}$, 1) on each cell denotes

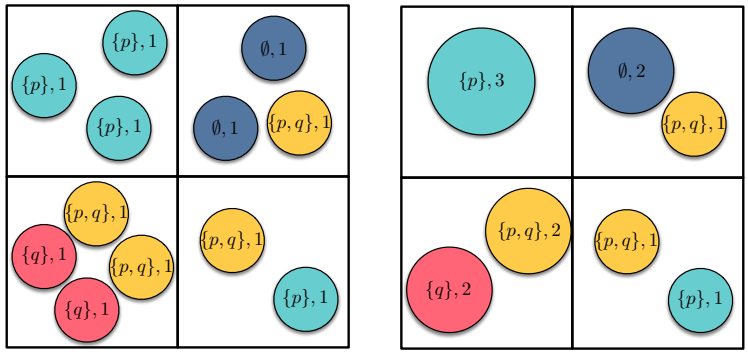


Fig. 8. Illustration of our geometric clustering technique. (a) Twelve cells need to be assigned in the flat minimum-cost flow without the clustering technique. (b) With our geometric clustering applied, the number of objects needs to be assigned is reduced to seven. "$\mathcal{S}, A$" pair (e.g., $\{p, q\}$, 1) on each cell denotes the set of clock nets it belongs to and its logic resource demand, respectively. $\emptyset$ means the cell does not belong to any clock nets.

the set of clock nets it belongs to and its logic resource demand, respectively. If the flat minimum-cost-flow-based assignment in Figure 5 is directly applied, we need to assign twelve objects, as shown in Figure 8(a), in the problem. However, if we divide all twelve cells into four bins based on the two-by-two geometric partitioning shown in Figure 8 and cluster cells that have the same clock signatures within each partition, the number of objects to be assigned will reduce to seven, as shown in Figure 8(b).

$$\underset{\boldsymbol{x},\lambda}{\text{minimize}} \quad \sum_{\mathrm{k}\in\mathcal{K}}\sum_{\mathrm{j}\in\mathcal{R}}\left(\frac{\mathrm{P_k}}{\mathrm{A_k}}\cdot \mathrm{D_{k,j}} + \lambda_{\mathrm{k,j}}\right)\cdot \mathrm{x_{k,j}}, \tag{4a}$$

$$\text{subject to} \quad \boldsymbol{x}_{k,j} \in \{0, 1, 2, \dots\}, \forall k \in \mathcal{K}, \forall j \in \mathcal{R}, \tag{4b}$$

$$\sum_{j\in\mathcal{R}} \boldsymbol{x}_{k,j} = A_k, \forall k \in \mathcal{K}, \tag{4c}$$

$$\sum_{k\in\mathcal{K}} \boldsymbol{x}_{k,j} \le C_j, \forall j \in \mathcal{R}. \tag{4d}$$

With our clustering technique applied, instead of solving Formulation (2), we solve a very similar Formulation (4), where $\mathcal{K}$ denotes the set of clusters, P_k and A_k denote the total pin count and total logic resource demand of cells in cluster k, and $D_{k,j}$ denotes the physical distance between the average location of cells in cluster k and clock region j. Besides, unit logic resource demand is assumed for all cells (not clusters).

Formulation (4) can still be solved utilizing the minimum-cost flow transformation illustrated in Figure 5. However, comparing to Formulation (2), several key differences need to be emphasised.

Firstly, since inaccuracy is injected to pin movement calculation by using averaged pin count $\frac{P_k}{A_k}$ and averaged cell locations for $D_{k,j}$ in the objective (4a), less optimal solutions will be obtained as the cluster size increases. On the other hand, a larger cluster size can dramatically reduce the problem size, which results in much faster runtime. Therefore, with different partition sizes, we

can achieve different tradeoffs between quality and runtime. In UTPlaceF 2.0, the partition width and height are empirically set to 5 CLB sites.

Secondly, in the minimum-cost flow graph with our clustering technique applied, each assignment object represents a cluster of cells and does not have unit logic resource demand anymore (i.e., $A_k \geq 1$). Thus, the final minimum-cost flow solution of Formulation (4) might assign non-zero flows to more than one edge of an assignment object. It is physically equivalent to splitting a cluster into subgroups and assigning them to multiple clock regions. For each of these split cases, to decide which cell goes to which clock region, one more level of minimum-cost flow corresponding to Formulation (2) is needed. Note that, this second-level minimum-cost flow is only for cells within the split cluster and the set of clock regions that this cluster has been assigned to. Since we assume unit resource demand for all cells, split assignment would not happen again in these second-level minimum-cost flows and the solution feasibility can be guaranteed.

3.2 Clock-Aware Packing

Packing is responsible for clustering LUTs and FFs into CLBs that satisfy all packing rules, meanwhile, optimizing various design metrics such as power, timing, and channel width. In UTPlaceF, packing is performed by pairwisely merging cells based on a set of attraction functions, which grant higher priority to cell pairs that are physically closer and share more small nets. Although the packing algorithm in UTPlaceF is effective for wirelength and routability optimization, it is no longer able to guarantee solution quality with the newly introduced clock region constraint.

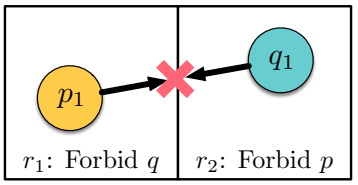


Fig. 9. An example to show the necessity of clock-aware packing. p and q denote two clock nets. p_1 and q_1 are two cells belonging to clock p and q, respectively.

One example to illustrate the necessity of clock awareness for packing is shown in Figure 9. In this example, clock q is forbidden in clock region r_1 and clock p is forbidden in clock region r_2 according to the clock region assignment result in FIP. If we, unfortunately, cluster p_1 and q_1 together, the packing solution would become infeasible, since no clock region can simultaneously contain clock p and q. Therefore, it is critical to making packing algorithms clock-aware for solution feasibility.

3.2.1 Probability-Based Clock Distribution Estimation. One common idea to avoid the case shown in Figure 9 is to let packing algorithms respect the clock region assignment solution after FIP. However, this idea imposes another question: how to deal with clock regions with spare global clock resources. In general, not all the global clock resources are employed after clock region assignment, so if we can further allocate these spare resources to some clock nets, packing algorithms might be able to explore a larger solution space and produce better packing solutions. Motivated by this reason, we propose a probabilistic model to estimate the probability of each clock occupying each clock region in the final placement solution. The estimation results will be incorporated into our new packing algorithm (See Section. 3.2.2) to help us smartly make use of those spare clock resources.

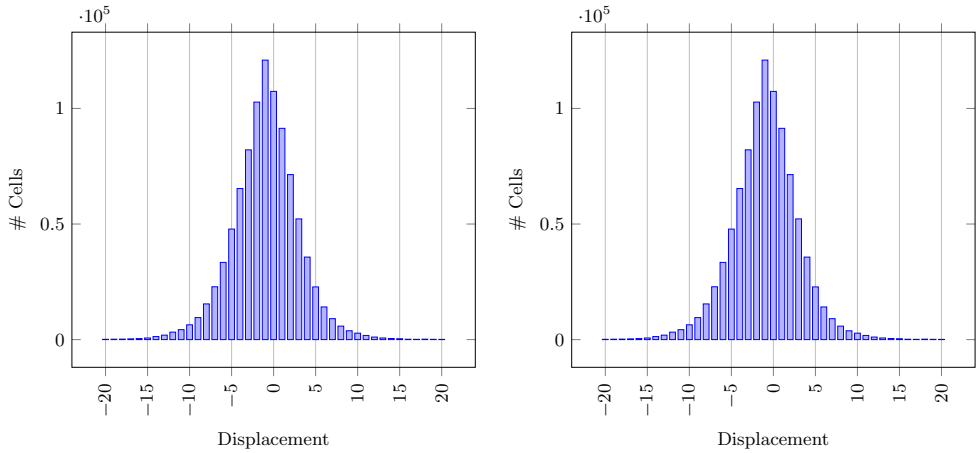


Fig. 10. Distributions of cell displacement in the final placement relative to FIP in (a) x direction and (b) y direction for a representative benchmark, CLK-DESIGN5 (0.94M cells), in ISPD'17 contest example benchmark suite. All placement solutions are generated by UTPlaceF without clock legality consideration.

The potential clock distribution mismatch between FIP and the final placement is introduced by the cell movement between them. We collect the displacement of all cells in the final placement solution relative to their locations in the FIP for a representative benchmark and the result is shown in Figure 10. It can be seen that the cell displacement in both x and y directions have a bell-shaped distribution around zero. So it is reasonable to make the assumption that the cell displacement satisfies the Gaussian distribution with mean value of zero in both x and y directions, which can be written as follows,

$$X' - X \sim \mathcal{N}(0, \sigma_x^2), \tag{5a}$$

$$Y' - Y \sim \mathcal{N}(0, \sigma_y^2). \tag{5b}$$

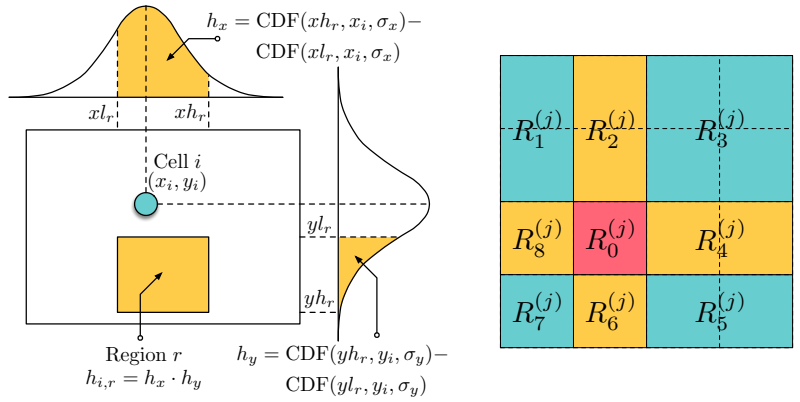
where (X', Y') and (X, Y) denote (x, y) coordinates of cells in the final placement and FIP, respectively, $\mathcal{N}(\mu, \sigma^2)$ represents a Gaussian distribution with mean value μ and standard deviation σ, and σ_x and σ_y are estimated standard deviations for cell displacement in x and y directions.

Given the assumption in Equation (5), a cell i at location (x_i, y_i) in FIP, and any region r with bounding box (xl_r, yl_r, xh_r, yh_r), the probability of cell i being placed into the region r in the final placement, denoted by $h_{i,r}$, can be written as follows,

$$h_{i,r} = (\mathrm{CDF}(xh_r, x_i, \sigma_x) - \mathrm{CDF}(xl_r, x_i, \sigma_x)) \cdot (\mathrm{CDF}(yh_r, y_i, \sigma_y) - \mathrm{CDF}(yl_r, y_i, \sigma_y)). \tag{6}$$

where $\mathrm{CDF}(x, \mu, \sigma)$ is the cumulative distribution function of the Gaussian distribution $X \sim \mathcal{N}(\mu, \sigma^2)$ evaluated at x. It represents the probability that X takes value less than or equal to x. CDF is defined in Equation (7), where $\mathrm{erf}(x)$ denotes the error function [2]. Figure 11(a) gives a visual illustration for the cell-to-region probability calculation process.

$$\mathrm{CDF}(x, \mu, \sigma) = \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{x - \mu}{\sigma\sqrt{2}}\right)\right]. \tag{7}$$



(a) (b)

Fig. 11. (a) A visual illustration for the cell-to-region probability calculation shown in Equation (6). (b) Nine sets of clock regions, $R_0^{(j)}, R_1^{(j)}, \ldots, R_8^{(j)}$, corresponding to clock region j (red).

Now, we show the method to calculate the probability of clock region j having clock demand from clock net k in the final placement. Given a clock region j, we divide all clock regions into nine sets (as shown in Figure 11(b)), $\mathcal{R}_0^{(j)}, \mathcal{R}_1^{(j)}, \ldots, \mathcal{R}_8^{(j)}$, where $\mathcal{R}_0^{(j)}$ contains only clock region j. It is easy to see that the bounding box of any clock net k does not intersect with the clock region j if and only if all cells in the clock net k are placed in one of the four regions that are above, below, to the left of, and to the right of the clock region j ($\mathcal{R}_0^{(j)}$) (i.e., $\mathcal{R}_1^{(j)} \cup \mathcal{R}_2^{(j)} \cup \mathcal{R}_3^{(j)}$, $\mathcal{R}_5^{(j)} \cup \mathcal{R}_6^{(j)} \cup \mathcal{R}_7^{(j)}$, $\mathcal{R}_7^{(j)} \cup \mathcal{R}_8^{(j)} \cup \mathcal{R}_1^{(j)}$, and $\mathcal{R}_3^{(j)} \cup \mathcal{R}_4^{(j)} \cup \mathcal{R}_5^{(j)}$). Therefore, the probability of clock region j not being occupied by clock net k, denoted by $\overline{H_{j,k}}$, can be defined using Equation (8).

$$\overline{H_{j,k}} = \sum_{\substack{(l,m,n) \in \\ \{(1,2,3),(3,4,5),\\(5,6,7),(7,8,1)\}}} \prod_{i \in \mathcal{V}(k)} (h_{i,\mathcal{R}_l^{(j)}} + h_{i,\mathcal{R}_m^{(j)}} + h_{i,\mathcal{R}_n^{(j)}}) - \sum_{l \in \{1,3,5,7\}} \prod_{i \in \mathcal{V}(k)} h_{i,\mathcal{R}_l^{(j)}}. \tag{8}$$

Finally, the probability of clock region j having clock demand from clock net k in the final placement, denoted by $H_{j,k}$, can be defined as follows,

$$H_{j,k} = 1 - \overline{H_{j,k}}. \tag{9}$$

3.2.2 Clock-Aware Packing Attraction Function. UTPlaceF 2.0 adopts the UTPlaceF packing framework, which consists of maximum-weighted-matching-based and BestChoice-based [12] clustering algorithms. Both algorithms rely on attraction functions to evaluate the goodness of any pairwise clustering and iteratively merge high-attraction object pairs to form CLBs. To produce clock-friendly CLB-level netlists for placement, the packing attraction functions in UTPlaceF are enhanced to be clock-aware in UTPlaceF 2.0.

The original attraction functions in UTPlaceF for connected objects consist of two components. The first component is the distance score, which exponentially penalizes objects that are physically far away in FIP. The second component is the connectivity score, which grants higher priority for objects that share more small nets [7]. The original packing attraction function for two connected objects i and j then can be generalized as follows,

$$\phi_{i,j} = \left(1 - e^{\gamma \cdot (D_{i,j} - \overline{\lambda})}\right) \cdot \sum_{e \in \text{Net}(i) \cap \text{Net}(j)} \frac{k}{P_e - 1}. \tag{10}$$

where $D_{i,j}$ denotes the physical distance between i and j in FIP, $\text{Net}(i) \cap \text{Net}(j)$ denotes the set of nets that are shared by i and j, P_e represents the number of pins in net e, k is 2 for two-pin nets and 1 for other nets, and γ and $\overline{\lambda}$ are tuning parameters determined experimentally.

The proposed new attraction function is same as the original one, except that a new term is introduced to account for the clock legality. It is described by the following expression,

$$\phi_{i,j} = \left(1 - e^{\gamma \cdot (D_{i,j} - \overline{\lambda})}\right) \cdot \sum_{e \in \text{Net}(i) \cap \text{Net}(j)} \frac{k}{P_e - 1} \cdot \prod_{k \in \mathcal{E}(i) \cup \mathcal{E}(j)} H_{t,k}. \tag{11}$$

where $\mathcal{E}(i) \cup \mathcal{E}(j)$ denotes the set of clock nets that contain at least one of cell i and cell j, t represents the clock region that the center of gravity of i and j falls into, and $H_{t,k}$ is the probability defined in Equation (9). Here we use clock region t as the estimated target clock region to place the cluster of i and j. Intuitively, the new term represents the probability that a cluster can be placed into its estimated target clock region without any clock conflicts. So integrating this new term to the original attraction function helps to block out clustering cases that are likely to violate clock region constraint.

The same clock penalty term is applied to all attraction functions in UTPlaceF to make the whole packing flow clock-aware. With our enhanced clock-aware attraction functions in UTPlaceF 2.0, better wirelength in the final solutions is demonstrated by our experiments.

3.3 Half-Column Region Assignment

As shown in Figure 3, half-column region constraint is explicitly respected in legalization stage by our half-column region assignment technique. Here we formally define the half-column region assignment problem as follows.

Problem 3 (Half-Column Region Assignment). *Given a rough-legalized placement, logic resource capacity of each half-column region, and a feasible clock region assignment solution, assign cells within each clock region to half-column regions to minimize total pin movement without logic resource and clock overflow.*

Problem 3 is very similar to the clock region assignment problem defined in Problem 2. It can also be formulated into Formulation (1) but with a simpler clock constraint (1e). So the same relaxation and minimum-cost flow framework described in Section 3.1.1 and Section 3.1.2 can be seamlessly applied to solve half-column region assignment problem as well.

The only notable difference from the clock region assignment is the method to update penalty multipliers for clock overflow resolving. Our proposed penalty multiplier updating algorithm for half-column region constraint is summarized in Algorithm 3. It should be noted that, given a feasible clock region assignment solution, the half-column region assignment can be performed within each clock region independently without impacting clock legality. So the scope of Algorithm 3 is only limited in a single clock region.

Within a clock region, we first find the target half-column region with the largest clock overflow in line 1. Then, the cost of moving each clock out of the target half-column region is calculated and stored in a list from line 2 to line 4. The cost of moving clock e out of half-column r is defined as follows,

Algorithm 3 Penalty Multiplier Updating For Half-Column Region Constraint

Input: Penalty multiplier λ.
Output: An updated penalty multiplier λ that will results in less clock overflow.

```
1: Find the half-column region r with the largest clock overflow O_max;
2: for each clock e in r do
3:     l ← l ∪ {Cost(e, r)};                          ▷ See Equation (12)
4: end for
5: Sort candidates in l by ascending order of their cost;
6: i ← 0;
7: for each Cost(e, r) ∈ sorted l do
8:     if ¬IsFeasible(e, r, λ) then
9:         Continue;
10:    end if
11:    for each cell i in clock e do
12:        λ_{i,r} ← ∞;
13:    end for
14:    i ← i + 1;
15:    if i ≥ O_max then
16:        Return;
17:    end if
18: end for
```

$$Cost(e, r) = \sum_{i \in \mathcal{V}(e) \cap \mathcal{V}(r)} P_i, \tag{12}$$

where $\mathcal{V}(e) \cap \mathcal{V}(r)$ denotes the set of cells that are simultaneously in clock e and half-column region r, and P_i represents the total number of pins associated with cell i.

In the loop from line 7 to line 18, we iteratively pick the clock with the smallest moving cost and block it out of the target half-column region in the subsequent assignment iterations. A feasibility check similar to the one illustrated in Figure 7 is performed in line 8 to ensure the existence of feasible assignments after the edge blocking. This process is repeated until the number of iterations hits the overflow limit in line 15.

4 EXPERIMENTAL RESULTS

UTPlaceF 2.0 was implemented in C++ and compiled by g++ 4.7.2. The official contest evaluation results conducted by Xilinx is used here to demonstrate the effectiveness of UTPlaceF 2.0.

The characteristics of ISPD'17 benchmark suite are listed in Table 2. This benchmark suite consists of industry-strength designs with gate counts ranging from 0.45 million to about 1 million. Several designs in this suite have extremely high resource utilization and clock usage.

As UTPlaceF 2.0 won the first place in ISPD'17 placement contest, we here compare our results with the second- and third-place contest winners. Table 3 shows the routed wirelength comparison results that reported by Xilinx Vivado v2016.4. It can be seen that UTPlaceF 2.0 achieves the best overall routed wirelength and outperforms by 4.0% and 11.0% in routed wirelength compared with the other two contest winners, respectively.

The runtime comparison is shown in Table 4. Running in a single thread, UTPlaceF 2.0 completes the largest benchmark CLK-FPGA13 (0.96M cells) in 20 minutes. We also report the runtime

Table 2. ISPD'17 Placement Contest Benchmarks Statistics

Benchmark	#LUT	#FF	#RAM	#DSP	#Clocks
CLK-FPGA01	215K	236K	170	75	30
CLK-FPGA02	215K	236K	170	75	30
CLK-FPGA03	242K	270K	255	112	33
CLK-FPGA04	268K	300K	340	150	36
CLK-FPGA05	295K	325K	425	187	39
CLK-FPGA06	322K	354K	510	225	42
CLK-FPGA07	350K	384K	595	262	45
CLK-FPGA08	376K	414K	680	300	48
CLK-FPGA09	392K	431K	765	337	51
CLK-FPGA10	408K	449K	850	375	54
CLK-FPGA11	424K	450K	900	397	55
CLK-FPGA12	440K	484K	950	420	56
CLK-FPGA13	456K	503K	1000	442	57
Resources	538K	1075K	1728	768	N/A

breakdown of UTPlaceF 2.0 in Figure 12(a). On average, the majority (68.4%) of the total runtime is taken by FIP, while detailed placement, CLB-level global placement, packing, and legalization respectively consume 20.0%, 5.3%, 3.3%, and 0.3% of the total runtime. We further divide the runtime of FIP into four components, as shown in Figure 12(b), where the preconditioned conjugate gradient for quadratic programming takes 88.8% of the FIP runtime, followed by 4.9% and 4.2% for rough legalization and density-preserving global move, respectively, and only 0.7% is taken by the clock region assignment.

Table 3. Routed Wirelength Comparison with ISPD'17 Contest Winners

Benchmark	2nd Place		3rd Place		UTPlaceF 2.0 (1st Place)	
	Routed WL	Ratio	Routed WL	Ratio	Routed WL	Ratio
CLK-FPGA01	2209328	1.001	2268532	1.027	2208170	1.000
CLK-FPGA02	2273729	0.998	2504444	1.099	2279171	1.000
CLK-FPGA03	6229292	1.164	5803110	1.084	5353071	1.000
CLK-FPGA04	3817377	1.032	4085670	1.105	3697950	1.000
CLK-FPGA05	4995177	1.065	5180916	1.104	4692356	1.000
CLK-FPGA06	5605573	1.003	6216898	1.112	5588507	1.000
CLK-FPGA07	2504544	1.024	2676088	1.095	2444837	1.000
CLK-FPGA08	1989632	1.055	2057117	1.091	1885632	1.000
CLK-FPGA09	2583442	0.995	2813538	1.084	2596654	1.000
CLK-FPGA10	4770168	1.069	4839765	1.084	4464341	1.000
CLK-FPGA11	4207699	1.006	4777177	1.142	4184233	1.000
CLK-FPGA12	3376930	1.002	3739517	1.110	3368698	1.000
CLK-FPGA13	3920965	1.019	4320345	1.123	3847832	1.000
Norm.	1.040	-	1.100	-	1.000	-

4.1 Efficiency Validation of Maximum-Flow-Based Feasibility Checking

We validate the efficiency of our maximum-flow-based feasibility checking proposed in Section 3.1.3 by experiments shown in Table 5. The column 2 - 5 separately list the number of solvings (# Solving) and total solving time (ST) of minimum-cost flow (MCF) and maximum flow (MF) for each benchmark. The column 6 gives the total solving time taken by MCF and MF. Without the maximum-flow-based feasibility checking, a MCF instead of a MF solving is required to check if the

Table 4. Runtime (Seconds) Comparison with ISPD'17 Contest Winners

Benchmark	2nd Place		3rd Place		UTPlaceF 2.0 (1st Place)	
	Runtime	Ratio	Runtime	Ratio	Runtime	Ratio
CLK-FPGA01	3023	5.68	354	0.67	532	1.00
CLK-FPGA02	3153	6.15	333	0.65	513	1.00
CLK-FPGA03	4066	3.91	666	0.64	1039	1.00
CLK-FPGA04	3077	4.33	464	0.65	711	1.00
CLK-FPGA05	3631	3.87	680	0.72	939	1.00
CLK-FPGA06	3836	3.60	695	0.65	1066	1.00
CLK-FPGA07	3953	4.68	410	0.49	845	1.00
CLK-FPGA08	4395	8.31	277	0.52	529	1.00
CLK-FPGA09	5428	6.45	414	0.49	842	1.00
CLK-FPGA10	3305	3.39	516	0.53	974	1.00
CLK-FPGA11	4341	4.06	548	0.51	1068	1.00
CLK-FPGA12	4949	6.39	413	0.53	774	1.00
CLK-FPGA13	3748	3.20	548	0.47	1172	1.00
Norm.	4.63	-	0.57	-	1.00	-

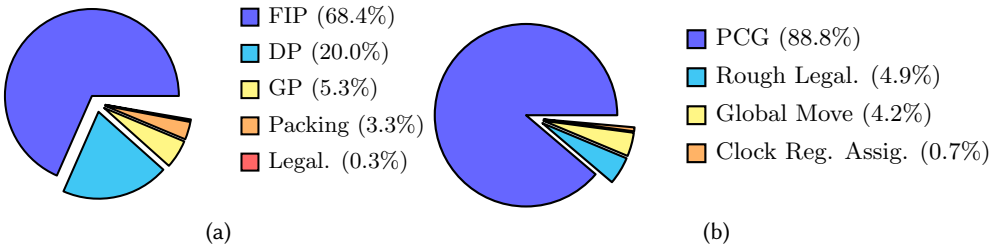


Fig. 12. Runtime breakdown of (a) UTPlaceF 2.0 and (b) flat initial placement (FIP).

updated penalty multipliers are feasible. In this case, our clock region assignment would become a pure MCF-based algorithm. We project the required solving time of it by Equation (13) and report the results in the column 7 of Table 5. The overall speedup of "MCF + MF" over "Pure MCF" for each benchmark is listed in the last column of Table 5.

$$\text{Proj. Pure MCF ST} = \frac{\text{MCF ST}}{\text{\# MCF Solving}}(\text{\# MCF Solving} + \text{\# MF Solving}) \tag{13}$$

It can be observed that MF is about one to two orders of magnitude faster than MCF in our experiments. By applying the maximum-flow-based feasibility checking, we can achieve up to ×2.6 overall speedup. Note that the runtime of MCF is dependent to the partition sizes mentioned in Section 3.1.4 but the MF-based checking is not. Therefore, even more speedup can be achieved if smaller partitions are used in the clock region assignment.

4.2 Trade-offs of Different Partition Sizes

Figure 13 gives the trade-offs between HPWL and clock region assignment runtime under different partition sizes. We perform experiments on a representative benchmark, CLK-FPGA05, with partition width and height set to 1, 5, 10, 15, 25, and 30, respectively. All the HPWL and runtime are normalized to the result with height and width of 5. As can be seen, with larger partition sizes, the runtime drops quickly and saturates at around 0.4 (normalized runtime) while the wirelength increases steadily but slowly. Considering the wirelength is not very sensitive to the partition size,

Table 5. Runtime Speedup by Applying Maximum-Flow-Based Feasibility Checking

Benchmark	MCF		MF		MCF + MF	Proj. Pure MCF	Speedup
	# Solving	ST (sec)	# Solving	ST (sec)	ST (sec)	ST (sec)	
CLK-FPGA01	90	0.96	0	0.00	0.96	0.96	×1.00
CLK-FPGA02	120	1.19	60	0.01	1.20	1.79	×1.49
CLK-FPGA03	738	22.10	1227	0.34	22.44	58.84	×2.62
CLK-FPGA04	219	4.25	221	0.05	4.30	8.54	×1.99
CLK-FPGA05	171	4.57	168	0.05	4.62	9.06	×1.96
CLK-FPGA06	282	9.58	342	0.11	9.69	21.20	×2.19
CLK-FPGA07	84	0.91	0	0.00	0.91	0.91	×1.00
CLK-FPGA08	96	0.77	0	0.00	0.77	0.77	×1.00
CLK-FPGA09	87	0.95	0	0.00	0.95	0.95	×1.00
CLK-FPGA10	222	4.15	159	0.03	4.18	7.12	×1.70
CLK-FPGA11	216	4.79	153	0.03	4.82	8.18	×1.70
CLK-FPGA12	123	1.88	42	0.01	1.89	2.52	×1.33
CLK-FPGA13	93	1.61	12	0.01	1.62	1.82	×1.12

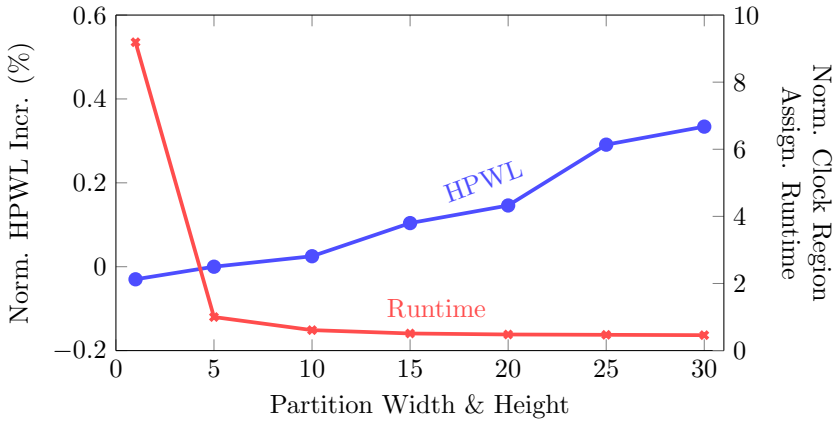


Fig. 13. Trend of HPWL and clock region assignment runtime with different partition sizes for clustering (Section 3.1.4) on benchmark CLK-FPGA05. All HPWL and runtime are normalized to the result of partition width and height = 5.

we experimentally choose partition width and height = 5 to achieve fast runtime and reasonably good solution quality.

4.3 Effectiveness Validation of Clock-Aware Packing

Figure 14 shows the normalized HPWL increases (less is better) of placements with and without clock-aware packing compared to the lower-bound placements that ignore all clock constraints. As can be seen, clock-aware packing, for most benchmarks, delivers better wirelength over original non-clock-aware packing. Another key observation is that, with both proposed clock region assignment and clock-aware packing techniques applied, UTPlaceF 2.0 can satisfy all clock constraints by only paying a little wirelength overhead (< 1%).

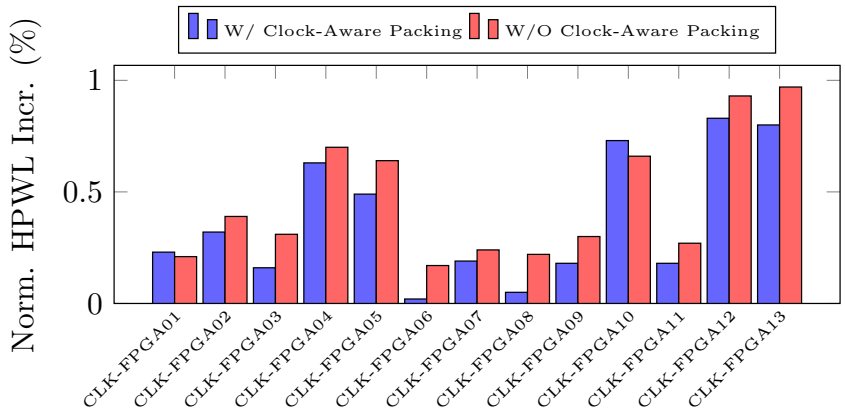


Fig. 14. Normalized HPWL increases (%) of placements w/ and w/o clock-aware packing compared to placements without any clock constraint considerations.

5 CONCLUSION

With the increasing complicated FPGA clocking architecture, respecting clock rules is becoming a fundamental issue in modern FPGA implementation flow. In this paper, we have proposed a high-performance clock-aware FPGA placement engine, UTPlaceF 2.0, which is capable of satisfying clock rules for state-of-the-art FPGA devices while still maintaining good wirelength and routability. An iterative minimum-cost-flow-based clock region assignment framework, a probability-based clock distribution estimationm method, and a clock-aware packing technique are proposed for better honoring clock legality throughout the whole placement and packing process. As the first place winner of ISPD'17 clock-aware FPGA placement contest, UTPlaceF 2.0 can achieve legal and high-quality placement solutions efficiently, which outperforms all other contest placers in routed wirelength with competitive runtime.

REFERENCES

[1] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. 1993. *Network Flows: Theory, Algorithms, and Applications.* Prentice-Hall, Inc.

[2] Larry C. Andrews. 1992. *Special Functions of Mathematics for Engineers.* SPIE Optical Engineering Press.

[3] Vaughn Betz and Jonathan Rose. 1997. VPR: A new packing, placement and routing tool for FPGA research. In *IEEE International Conference on Field Programmable Logic and Applications (FPL).* 213–222.

[4] Yu-Chen Chen, Sheng-Yen Chen, and Yao-Wen Chang. 2014. Efficient and effective packing and analytical placement for large-scale heterogeneous FPGAs. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD).* 647–654.

[5] Marcel Gort and Jason H. Anderson. 2012. Analytical placement for heterogeneous FPGAs. In *IEEE International Conference on Field Programmable Logic and Applications (FPL).* 143–150.

[6] Julien Lamoureux and Steven J. E. Wilton. 2008. On the Trade-off Between Power and Flexibility of FPGA Clock Networks. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 1, 3 (2008), 13:1–13:33.

[7] Wuxi Li, Shounak Dhar, and David Z. Pan. 2017a. UTPlaceF: A routability-driven FPGA placer with physical and congestion aware packing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* (2017).

[8] Wuxi Li, Meng Li, Jiajun Wang, and David Z. Pan. 2017b. UTPlaceF 3.0: A Parallelization Framework for Modern FPGA Global Placement. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD).* 908–914.

[9] Tao Lin, Chris C.N. Chu, Joseph R. Shinnerl, Ismail Bustany, and Ivailo Nedelchev. 2013b. POLAR: Placement based on novel rough legalization and refinement. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD).* 357–362.

[10] Tzu-Hen Lin, Pritha Banerjee, and Yao-Wen Chang. 2013a. An efficient and effective analytical placer for FPGAs. In

ACM/IEEE Design Automation Conference (DAC). 10:1–10:6.

[11] Alexander S. Marquardt, Vaughn Betz, and Jonathan Rose. 1999. Using cluster-based logic blocks and timing-driven packing to improve FPGA speed and density. In *ACM Symposium on FPGAs*. 37–46.

[12] Gi-Joon Nam, Sherief Reda, Charles J. Alpert, Paul G. Villarrubia, and Andrew B. Kahng. 2006. A fast hierarchical quadratic placement algorithm. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 25, 4 (2006), 678–691.

[13] Ryan Pattison, Ziad Abuowaimer, Shawki Areibi, Gary Gréwal, and Anthony Vannelli. 2016. GPlace: A congestion-aware placement tool for ultrascale FPGAs. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 68:1–68:7.

[14] Chak-Wa Pui, Gengjie Chen, Wing-Kai Chow, Ka-Chun Lam, Jian Kuang, Peishan Tu, Hang Zhang, Evangeline F.Y. Young, and Bei Yu. 2016. RippleFPGA: A routability-driven placement for large-scale heterogeneous FPGAs. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 67:1–67:8.

[15] Senthilkumar Thoravi Rajavel and Ali Akoglu. 2011. MO-Pack: Many-objective clustering for FPGA CAD. In *ACM/IEEE Design Automation Conference (DAC)*. 818–823.

[16] Amit Singh, Ganapathy Parthasarathy, and Malgorzata Marek-Sadowska. 2002. Efficient circuit clustering for area and power reduction in FPGAs. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 7, 4 (2002), 643–663.

[17] Love Singhal, Mahesh A. Iyer, and Saurabh Adya. 2017. LSC: A Large-Scale Consensus-Based Clustering Algorithm for High-Performance FPGAs. In *ACM/IEEE Design Automation Conference (DAC)*. 30:1–30:6.

[18] Marvin Tom, David Leong, and Guy Lemieux. 2006. Un/DoPack: re-clustering of large system-on-chip designs with interconnect variation for low-cost FPGAs. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 680–687.

[19] Xilinx Inc. 2017. http://www.xilinx.com. (2017). Accessed: 2017-03-17.

[20] M Xu, Gary Gréwal, and Shawki Areibi. 2011. StarPlace: A new analytic method for FPGA placement. *Integration, the VLSI Journal* 44, 3 (2011), 192–204.

[21] Stephen Yang, Aman Gayasen, Chandra Mulpuri, Sainath Reddy, and Rajat Aggarwal. 2016. Routability-Driven FPGA Placement Contest. In *ACM International Symposium on Physical Design (ISPD)*. 139–143.

[22] Stephen Yang, Chandra Mulpuri, Sainath Reddy, Meghraj Kalase, Srinivasan Dasasathyan, Mehrdad E. Dehkordi, Marvin Tom, and Rajat Aggarwal. 2017. Clock-Aware FPGA Placement Contest. In *ACM International Symposium on Physical Design (ISPD)*. 159–164.

Received April 2017; revised September 2017; accepted December 2017