Multi-Electrostatic FPGA Placement Considering SLICEL-SLICEM Heterogeneity and Clock Feasibility

Jing Mai[1,2], Yibai Meng[2], Zhixiong Di[3], Yibo Lin[2]*
[1]School of Computer Science, Peking University [2]School of Integrated Circuits, Peking University
[3]School of Information Science and Technology, Southwest Jiaotong University
{jingmai,mengyibai,yibolin}@pku.edu.cn, zxdi@home.swjtu.edu.cn

ABSTRACT

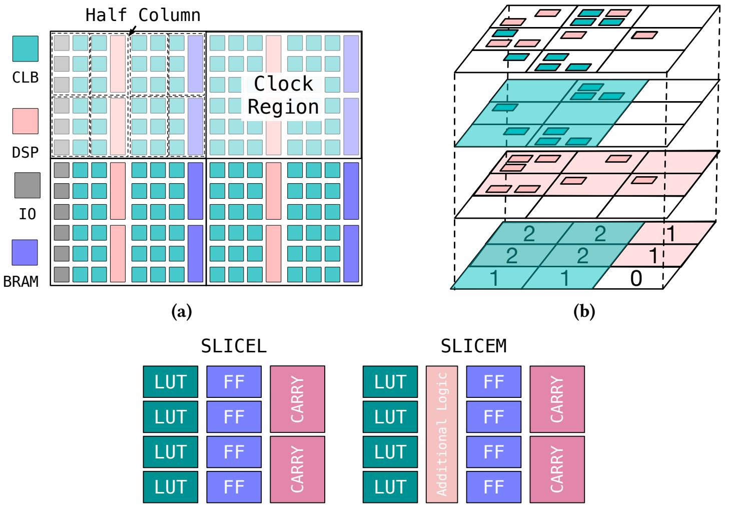
Modern field-programmable gate arrays (FPGAs) contain heterogeneous resources, including CLB, DSP, BRAM, IO, etc. A Configurable Logic Block (CLB) slice is further categorized to SLICEL and SLICEM, which can be configured as specific combinations of instances in {LUT, FF, distributed RAM, SHIFT, CARRY}. Such kind of heterogeneity challenges the existing FPGA placement algorithms. Meanwhile, limited clock routing resources also lead to complicated clock constraints, causing difficulties in achieving clock feasible placement solutions. In this work, we propose a heterogeneous FPGA placement framework considering SLICEL-SLICEM heterogeneity and clock feasibility based on a multi-electrostatic formulation. We support a comprehensive set of the aforementioned instance types with a uniform algorithm for wirelength, routability, and clock optimization. Experimental results on both academic and industrial benchmarks demonstrate that we outperform the state-of-the-art placers in both quality and efficiency.

1 INTRODUCTION

Modern FPGA placement has two major challenges: 1) the heterogeneity of FPGA architecture and 2) the strict clock routing constraints. The heterogeneity of FPGA architecture comes from a variety of instance types and asymmetric slice compatibility from SLICEL-SLICEM heterogeneity. Such heterogeneity results in discrete mathematical problems, challenging the analytical placement algorithms, usually based on continuous optimization. Besides, the complicated clock architectures to achieve low clocking skew impose challenging clock routing constraints on FPGA placement, lowering the solution quality of placement algorithms.

Modern FPGA placers are usually based on analytical approaches. Recent studies develop quadratic programming-based algorithms [1–9] and nonlinear optimization-based algorithms [10–12] for wirelength and routability optimization. Table 1 summarizes features of the published state-of-the-art FPGA placers. Among them, the current state-of-the-art solution quality is achieved by nonlinear placers such as elfPlace [10] and NTUfplace [12]. They propose to spread instances by minimizing the potential energy in a multi-electrostatic system [10] or a hand-crafted mathematical field system [12]. However, most existing FPGA placers only consider a simplified FPGA architecture with a subset of instance types, i.e., {LUT, FF, BRAM, DSP}, ignoring the SLICEL-SLICEM heterogeneity [2–8, 10–12]. Among these placers, only a few consider the widely-adopted clock routing constraints in real architectures [6–8, 11, 12].

In this paper, we tackle the heterogeneous FPGA placement considering both SLICEL-SLICEM heterogeneity and clock feasibility based on a new multi-electrostatic formulation. We handle a comprehensive set of instance types, i.e., {LUT, FF, BRAM, DSP, distributed RAM, SHIFT, CARRY}, which are commonly used in FPGA design [15]. Our major contributions are summarized as follows.



CLB Slice	LUTs	FFs	CARRY	Distributed RAM	SHIFT
SLICEL	8	16	1	NA	NA
SLICEM	8	16	1	512 bits	512 bits

(c)

Figure 1: (a) A simplified column-based FPGA architecture example like *Xilinx UltraScale*, with a 2×2 clock regions and half columns (dash line). (b) An illustration of the calculation of clock demand. Different colors represent different clocks, and the numbers represent the clock demand of each CR. (c) An example of CLB slices categorized to SLICEL and SLICEM with asymmetric compatibility. LUT blocks in a SLICEL can be configured to LUTs. All LUT blocks in a SLICEM can only be configured to *one* mode: LUT, distributed RAM, or SHIFT. No mixing between LUTs, distributed RAMs, and SHIFTs in a CLB is allowed.

- We propose a new multi-electrostatic formulation to handle asymmetric slice compatibility from SLICEL-SLICEM heterogeneity as well as techniques to handle carry chains.
- We propose a nested *Lagrangian relaxation*-based technique for wirelength, routability, and clock optimization with a dynamically-adjusted preconditioning technique to stabilize the convergence.
- We propose a quadratic penalization technique to eliminate violations of the discrete clock constraints.

Experiments on *ISPD 2017 contest benchmarks* [16] demonstrate 14.2%, 11.7%, 9.6%, and 7.9% improvement in routed wirelength, compared to the recent cutting-edge FPGA placers [6–8, 12], respectively. Further experiments on *industrial* benchmarks demonstrate that the proposed techniques can bring 19.3%, 10.7%, and 6.6% wirelength improvement with better stability. Our placer also support GPU acceleration and can achieve 1.5–6× speedup over the baselines.

The rest of the paper is organized as follows. Section 2 introduces the preliminary knowledge of the FPGA architecture and modern FPGA placement. Section 3 describes the core placement algorithms. Section 4 shows the experimental results, followed by the conclusion in Section 5.

*Corresponding author

DAC '22, July 10–14, 2022, San Francisco, CA, USA

ACM ISBN 978-1-4503-9142-9/22/07…$15.00
https://doi.org/10.1145/3489517.3530568

Table 1: Features of the published state-of-the-art FPGA placers.

Placer		`RippleFPGA` [13]	`GPlace` [4]	`UTPlaceF` [2]	`elfPlace` [10]	`GPlace 3.0` [14]	`RippleFPGA Clock-Aware` [8]	`UTPlaceF 2.0&2.X` [6, 7]	`NTUfPlace` [12]	Ours
Clock Constraints		×	×	×	×	×	✓	✓	✓	✓
Resources Supported	LUT, FF, BRAM, DSP	✓	✓	✓	✓	✓	✓	✓	✓	✓
	CARRY, SHIFT, Distributed RAM	×	×	×	×	×	×	×	×	✓
GPU-Acceleration		×	×	×	✓	×	×	×	×	✓
Algorithm Category		Quadratic	Quadratic	Quadratic	Nolinear	Quadratic	Quadratic	Quadratic	Nonlinear	Nonlinear

2 PRELIMINARIES

In this section, we introduce the FPGA architecture and the background of multi-electrostatics based FPGA placement.

2.1 Device Architecture

In this work, we target *Xilinx UltraScale* series [15, 17], e.g., *UltraScale VU095*, as a representative architecture (illustrated in Figure 1(a)). A simplified version of this architecture is also used in ISPD 2016&2017 FPGA placement contests with a subset of instance types, i.e., {LUT, FF, BRAM, DSP} [16, 18].

2.1.1 SLICEL-SLICEM heterogeneity. Apart from regular LUTs, the architecture also has other LUT-like instances: distributed RAM and shift registers (SHIFT). CLBs have two categories of slices, SLICEL and SLICEM, the structures of which are shown in Figure 1(c). SLICEL and SLICEM have slightly different logic resources and thus they support different configurations. Different from other regular instances like DSP and BRAM, these LUT-like instances owns another asymmetrical placement constraints, i.e., LUTs, FFs, and CARRYs can be placed in both SLICEL and SLICEM, whereas distributed RAMs and SHIFTs can only be placed in SLICEM. Moreover, if a SLICEM is configured as LUTs, then it cannot be used as distributed RAMs or SHIFTs; vice versa.

2.1.2 Carry Chain Alignment Constraints. We also need to consider carry chain alignment for CARRY instances as a placement constraint. The CARRY instances are placed in CLB slices, and a carry chain consists of a series of sequential CARRY instances that are connected by cascading wires from lower to upper bits. The alignment constraint imposes that the CARRY instances in a chain need to be placed in the same column, and in successive CLB slices in proper order according to the cascading wires.

2.1.3 Clock Constraints. The target FPGA device is divided into 5 × 8 rectangular-shaped clock regions (CRs), as shown in Figure 1(a) [1]. Each CR consists of columns of site resources. Each CR can be further subdivided into pairs of lower and upper half columns (HCs) of half-clock-region height horizontally. The width of each HC is the same as that of two site columns except for some corner cases, as shown in Figure 1(a).

The clock architecture imposes two clock constraints on placement, *the clock region constraint*, and *the half column constraint*, as shown in Figure 1(b). The clock region constraint restricts the clock demand of each clock region to be at most 24, where the clock demand is defined as the total number of clock nets whose bounding boxes intersect with the clock region. The half clock region constraint restricts the number of clock nets within the half column to be at most 12.

2.2 Multi-Electrostatics based FPGA Placement

Electrostatics-based placement models each instance as an electric particle in an electrostatic system, as illustrated in Figure 2. It is originally proposed in ASIC placement [19], leveraging the basic physical insight that balanced charge distribution in an electrostatic system contributes to low potential energy, so *minimizing the potential energy can resolve density overflow and help spread instances in the layout.* This approach is then extended to multiple electrostatic fields such that multiple resource types in FPGA placement like LUT, FF, DSP, and BRAM can be handled [10]. Figure 3 illustrates the multi-electrostatic formulation for LUT and DSP resources. By minimizing the total potential energy of multiple fields, we can reduce the density overflow, as low energy means balanced distribution of instances. The problem can be written as,

$$\min_{\boldsymbol{x},\boldsymbol{y}} \widetilde{W}(\boldsymbol{x},\boldsymbol{y}) \quad \text{s.t. } \Phi_s(\boldsymbol{x},\boldsymbol{y}) = 0, \forall s \in S, \tag{1}$$

where $\boldsymbol{x}, \boldsymbol{y}$ denote instance locations, $\widetilde{W}(\cdot)$ denotes the wirelength objective, S is the field type set, and $\Phi_s(\cdot)$ denotes the electric potential energy of the field for field type $s \in S$. We formally constrain the target energy $\Phi_s(\boldsymbol{x},\boldsymbol{y})$ to 0, as the energy is usually nonnegative. The constraints can be relaxed to the objective and solved by gradient descent method. In practice, we stop the optimization when the energy is small enough; or equivalently, the density overflow is low enough. It needs to mention that the formulation in Figure 3 assumes one instance occupies the resources of only one field, which cannot handle the complicated SLICEL-SLICEM heterogeneity in Section 2.1.1.

In this work, we focus on wirelength and routability optimization, considering SLICEL-SLICEM heterogeneity and clock constraints. We define the FPGA placement problem as follows.

Problem 1 (FPGA Placement)**.** *Given a netlist consisting of instances in {LUT, FF, DSP, BRAM, distributed RAM, SHIFT, CARRY}, produce a feasible FPGA placement solution with optimized wirelength and routability, satisfying the clock constraints.*

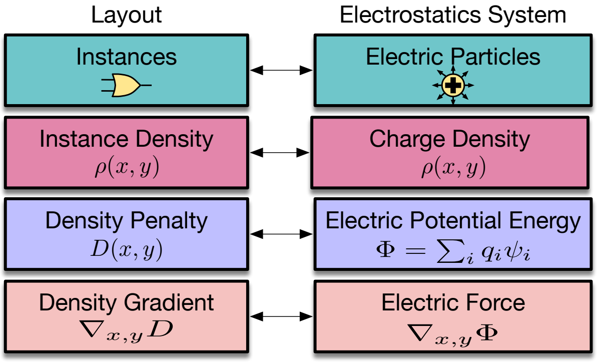


Figure 2: Analogy between placement of a single resource type and an electrostatic system [20].

3 ALGORITHMS

In this section, we describe the details of our placement algorithm.

3.1 Overview of the Proposed Algorithm

Our framework consists of two major phases: (1) clock-driven global placement, (2) clock-aware legalization and detailed placement, as shown in Figure 4.

We define the field type set as S = {LUTL, LUTM-AL, FF, CARRY, DSP, BRAM} with special field setup to handle SLICEL-SLICEM heterogeneity (Section 3.2). With clock constraints and carry chain alignment constraints, we formulate the problem as Formulation (2):

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{y}} \quad & \widetilde{W}(\boldsymbol{x},\boldsymbol{y}), & \text{(2a)} \\ \text{s.t.} \quad & \Phi_s(\boldsymbol{x},\boldsymbol{y};\mathcal{A}^s) = 0, \quad \forall s \in S, & \text{(2b)} \\ & \Gamma(\boldsymbol{x},\boldsymbol{y}) = 0, & \text{(2c)} \\ & \textit{Carry chain alignment constraint}, & \text{(2d)} \end{aligned}$$

[1] Due to the page limit, Figure 1(a) only contains part of the whole 5 × 8 CRs, but is sufficient for illustration.

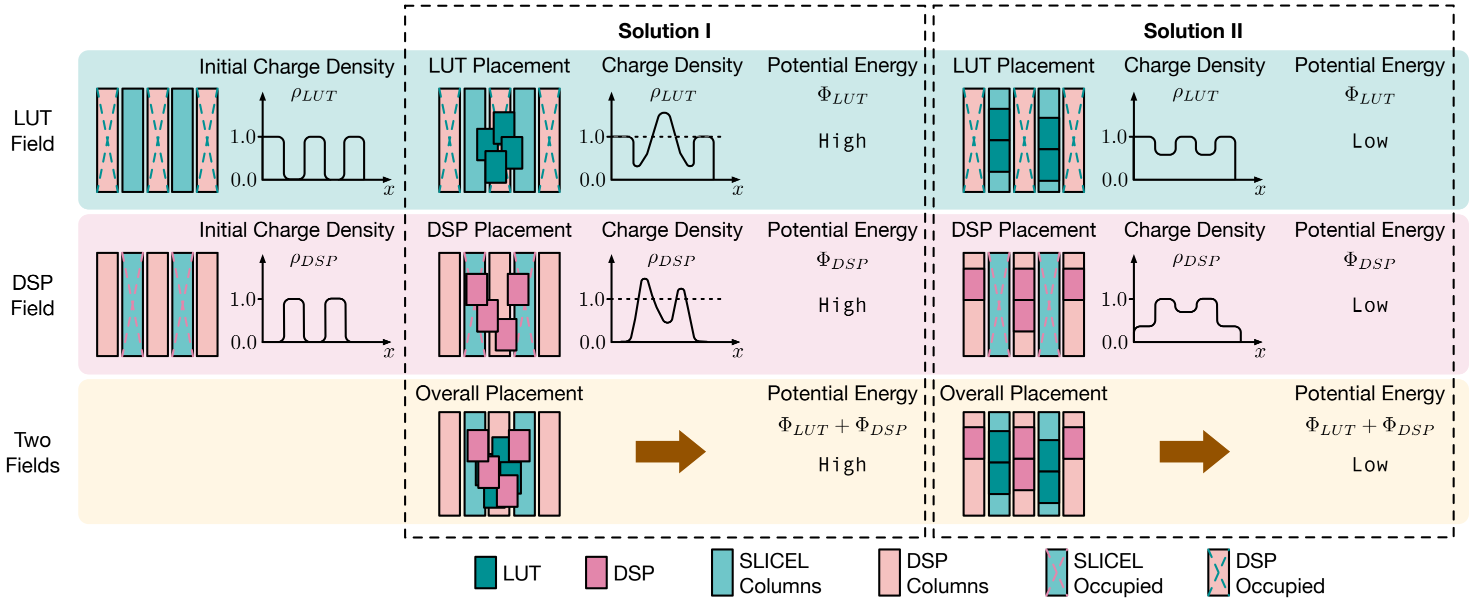


Figure 3: Example of the multi-electrostatic formulation for LUT and DSP resources, corresponding to two electric fields, respectively. For each field, unavailable columns are considered as occupied when computing the initial charge density. Take LUT as an example. If a LUT instance is not placed at a SLICEL column or there are overlaps between LUT instances, it can cause density overflow and lead to imbalanced density distribution of the field, eventually resulting in high electric potential energy. Thus, minimizing the energy can help resolve density overflow and spread instances in the layout. Note that if we encounter density underflow (< 1.0), we can insert fillers with positive charges for each field to fill the empty spaces to avoid imbalanced density distribution [10, 19]. As a result, only density overflow will cause high potential energy. FF and BRAM can be handled in the same way by adding two more fields.

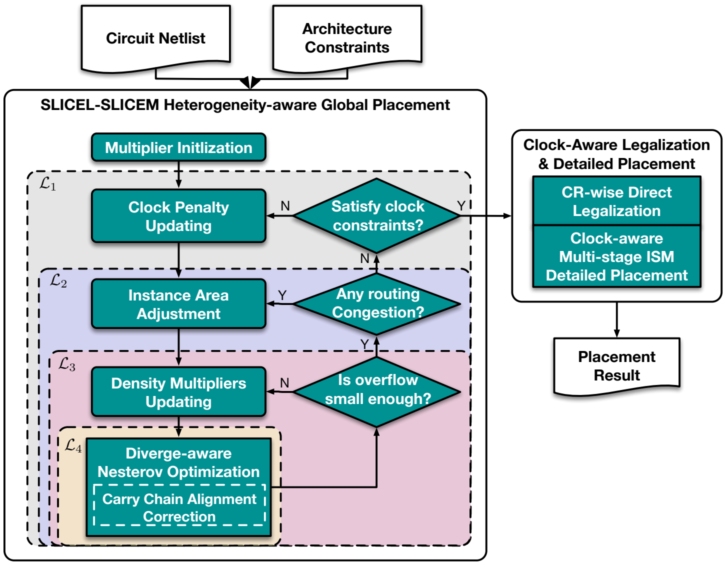

Circuit Netlist
Architecture Constraints
SLICEL-SLICEM Heterogeneity-aware Global Placement
Multiplier Initlization
$\mathcal{L}_1$
Clock Penalty Updating
N
Satisfy clock constraints?
Y
$\mathcal{L}_2$
Instance Area Adjustment
Y
Any routing Congestion?
N
$\mathcal{L}_3$
Density Multipliers Updating
N
Is overflow small enough?
Y
$\mathcal{L}_4$
Diverge-aware Nesterov Optimization
Carry Chain Alignment Correction
Clock-Aware Legalization & Detailed Placement
CR-wise Direct Legalization
Clock-aware Multi-stage ISM Detailed Placement
Placement Result

Figure 4: The proposed Overall Flow.

where $\mathcal{A}^s$ denotes all the instance areas in the field s, and $\Gamma(\cdot)$ denotes the clock penalty term (defined in Section 3.5). For brevity, we simplify $\Phi_s(\boldsymbol{x}, \boldsymbol{y}; \mathcal{A}^s)$ to Φ_s for all $s \in S$, and the potential energy vector whose elements are $\Phi_s(\forall s \in S)$ is denoted by $\boldsymbol{\Phi}$ in later discussions.

We leverage the *augmented Lagrangian method (ALM)* [21] to formulate a better unconstrained subproblem,

$$\min_{\boldsymbol{x},\boldsymbol{y}} \quad \mathcal{L}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\lambda}, \boldsymbol{\mathcal{A}}, \eta) = \widetilde{W}(\boldsymbol{x}, \boldsymbol{y}) + \sum_{s \in S} \lambda_s \mathcal{D}_s + \eta \Gamma(\boldsymbol{x}, \boldsymbol{y}), \tag{3a}$$

$$\mathcal{D}_s = \Phi_s + \frac{1}{2} \mathfrak{C}_s \Phi_s^2, \quad \forall s \in S, \tag{3b}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^{|S|}$ is the density multiplier vector, and $\eta \in \mathbb{R}$ is the clock penalty multiplier. The weighting coefficient vector $\boldsymbol{\mathfrak{C}} \in \mathbb{R}^{|S|}$ balances between first-order and second-order density penalty terms for each field type $s \in S$. Both $\boldsymbol{\lambda}$ and $\boldsymbol{\mathfrak{C}}$ follow the settings in [10]. As we have multiple constraints, we leverage Lagrangian relaxation and break the problem into nested optimization problems,

$$\text{Clock Opt.:} \quad \mathcal{L}_1 = \max_{\eta} \mathcal{L}_2(\eta), \tag{4a}$$

$$\text{Routability Opt.:} \quad \mathcal{L}_2(\eta) = \max_{\boldsymbol{\mathcal{A}}} \mathcal{L}_3(\boldsymbol{\mathcal{A}}, \eta), \tag{4b}$$

$$\text{Wirelength Opt.:} \quad \mathcal{L}_3(\boldsymbol{\mathcal{A}}, \eta) = \max_{\boldsymbol{\lambda}} \mathcal{L}_4(\boldsymbol{\lambda}, \boldsymbol{\mathcal{A}}, \eta), \tag{4c}$$

$$\text{Subproblem:} \quad \mathcal{L}_4(\boldsymbol{\lambda}, \boldsymbol{\mathcal{A}}, \eta) = \min_{\boldsymbol{x},\boldsymbol{y}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\lambda}, \boldsymbol{\mathcal{A}}, \eta), \tag{4d}$$

where $\mathcal{L}_4$ denotes Eq. (3). Each problem passes its variables to the subproblem as fixed hyperparameters. Take the process of solving $\mathcal{L}_3$ as an example. Every time $\mathcal{L}_4$ converges to its minimum solution given a fixed $\boldsymbol{\lambda}$, $\boldsymbol{\mathcal{A}}$ and η, the $\mathcal{L}_3$ optimizer will increase $\boldsymbol{\lambda}$ to emphasize more on the density term, forcing the density constraints to be gradually satisfied. Similar strategies apply to the $\mathcal{L}_1$ and $\mathcal{L}_2$ optimizers. Figure 4 plots the nested loops for solving the problem.

$\mathcal{L}_1$ (Section 3.5) aims at eliminating the clock violations in a analytical manner. The stopping criteria of $\mathcal{L}_1$ is whether the clock constraints are satisfied (Section 2.1.3). $\mathcal{L}_2$ aims at optimizing the routability by the area inflation-based technique from [10], and the stopping criteria of $\mathcal{L}_2$ is determined by routing congestion estimation and pin density. $\mathcal{L}_3$ solves the core wirelength-driven placement problem. The stopping criteria for $\mathcal{L}_3$ is determined by the density *overflow* of all. [2] For $\mathcal{L}_4$, we always solve with a fixed number of iterations, e.g., 1 iteration in the experiments.

The carry chain alignment constraint is resolved through *iterative carry chain alignment correction* in each iteration (Section 3.4). After the placement phase, we develop clock-aware direct legalization and detailed placement algorithms based on [5] to honor the aforementioned clock feasibility and SLICEL-SLICEM heterogeneity. Due to page limit, we omit the details on routability optimization, clock-aware legalization and detailed placement.

3.2 Multi-Electrostatic Model for SLICEL-SLICEM Heterogeneity

As discussed in Section 2.1.1, SLICEMs can be configured in one of the three modes: LUT, distributed RAM, and SHIFT. Only instances corresponding to the mode can be placed in the LUT slots of that SLICEM. To deal with this constraint, we introduce two electrostatic fields *LUTL* and *LUTM-AL*

[2] In our experiments, the target is empirically set to 10% overflow for LUTs and FFs.

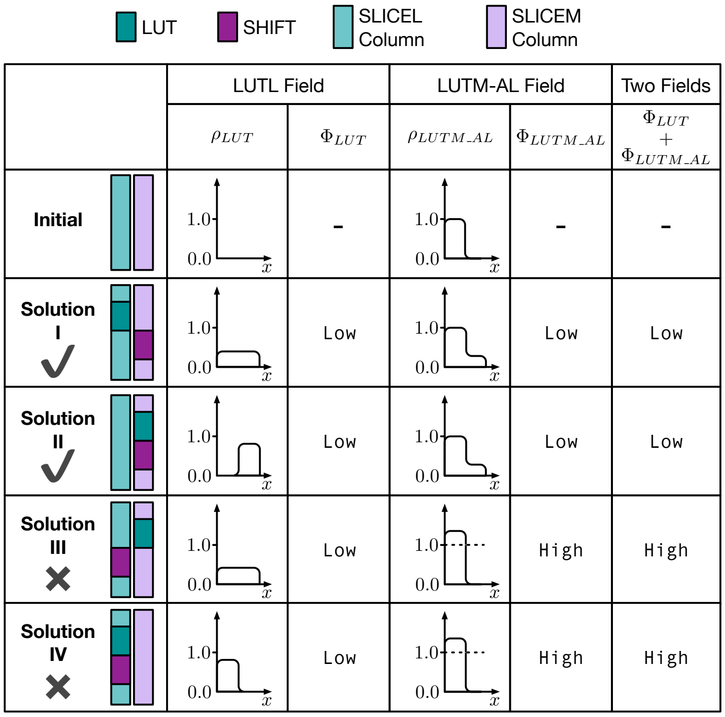


Figure 5: An example of special electrostatic field setup to handle asymmetric slice compatibility from SLICEL-SLICEM heterogeneity. Solutions III and IV are illegal because the SHIFT instance is placed on the SLICEL column, which can cause density overflow in the LUTM-AL field, eventually resulting in high potential energy.

in the multi-electrostatic placement model. The field setup should avoid a distributed RAM or SHIFT instance to be placed in SLICEL sites, but allow a LUT instance to be placed in both SLICEL and SLICEM sites.

The setup of these two fields is illustrated in Figure 5. LUTL models the LUT resources supplied in SLICEL and SLICEM, while LUTM-AL models the additional logic resources supplied in SLICEM, but not in SLICEL. A LUT instance only occupies resources in the LUTL field, while a distributed RAM or SHIFT instance occupies resources in both the LUTL and LUTM-AL fields.

The figure analyzes four scenarios taking the combinations of a LUT instance and a SHIFT instance as an example. Distributed RAM instances work in the same way as SHIFT instances do. As a SLICEL does not contain resources for LUTM-AL, we set its initial density to 1, indicating it is occupied; the initial density for LUTL is set to 0. As a SLICEM contains both resources for LUTL and LUTM-AL, we set its initial density to 0. In Solution I, where the LUT instance is placed on a SLICEL and the SHIFT instance is placed on a SLICEM, there is no density overflow for both fields (balanced density distribution can be achieved by inserting fillers [19]), which means the potential energy is at the minimum. The scenario in Solution II is similar. However, in Solution III and IV, where the SHIFT instance is placed on a SLICEL, the density overflow in the LUTM-AL field occurs, indicating high potential energy for this field. These solutions will be avoided when the optimizer minimizes the potential energy. In this way, these two elaborate fields can accommodate LUT and distributed RAM/SHIFT to their compatible sites naturally.

3.3 Divergence-aware Preconditioning

The gradient $\nabla\mathcal{L}^{(t)}$ at iteration t will be preconditioned before it is fed to the optimizer, where $\mathcal{L}$ is the Lagrangian problem defined in Eq. (3). We only discuss the gradient on x direction and that on y direction is the same. For the sake of efficiency, we adapt the Jacobi preconditioner $\mathcal{P}$ by approximating the second-order derivatives of wirelength and density as

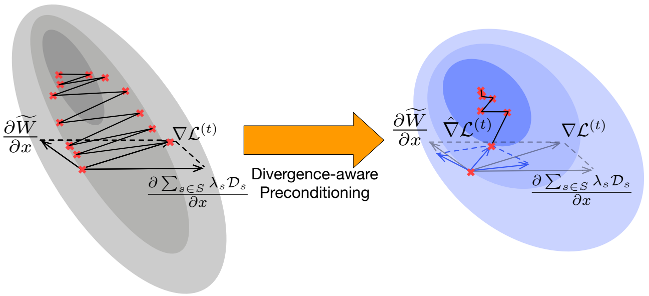

$\frac{\partial \widetilde{W}}{\partial x}$
$\nabla\mathcal{L}^{(t)}$
$\frac{\partial \sum_{s\in S}\lambda_s\mathcal{D}_s}{\partial x}$
Divergence-aware Preconditioning
$\frac{\partial \widetilde{W}}{\partial x}$
$\hat{\nabla}\mathcal{L}^{(t)}$
$\nabla\mathcal{L}^{(t)}$
$\frac{\partial \sum_{s\in S}\lambda_s\mathcal{D}_s}{\partial x}$

Figure 6: Illustration of preconditioning at iteration t. The preconditioning technique equivalently transforms the objection surfaces into a more isotropic shape, and thus helps reduce the iterations and stablize the optimization process.

follows,

$$\mathcal{P}_i^W = \frac{\partial^2 \widetilde{W}(\boldsymbol{x},\boldsymbol{y})}{\partial x_i^2} \sim \sum_{e\in E_i}\frac{w_e}{|e|-1}, \quad \forall i \in \mathcal{V}, \tag{5a}$$

$$\mathcal{P}_i^{(t)} \sim \max\left(1, \left[\mathcal{P}_i^W + \sum_{s\in S}\alpha_s^{(t)}\lambda_s^{(t)}\mathcal{A}_i^s\right]^{-1}\right), \tag{5b}$$

where $\mathcal{V}$ denotes the instance set, E_i denotes the nets incident to instance $i \in \mathcal{V}$, w_e is the weight of net e, and $\mathcal{P}^W$ denotes the second-order derivative of the wirelength term. We give the preconditioned gradient $\hat{\nabla}\mathcal{L}^{(t)} = \nabla\mathcal{L}^{(t)} \odot \mathcal{P}^{(t)}$ to the optimizer. As illustrated in Figure 6, after preconditioning the loss surfaces are thus being more isotropic and can speed up optimization. The intuition from the partial derivative itself is, for $\mathcal{P}_i^W$ that instances with more pins and pins incident to larger net weights should move slower, and for $\sum_{s\in S}\alpha_s^{(t)}\lambda_s^{(t)}\mathcal{A}_i^s$ that larger instances should move slower.

Different from [10, 22], we introduce an additional weighting vector $\boldsymbol{\alpha} \in \mathbb{R}^{|S|}$ to dynamically control the ratio of the gradient norms from the density term and the wirelength term, because we observe that the optimization can easily diverge if the ratio goes too large. Figure 6 illustrates the situation when some instances are dominated by the density gradient, resulting in these instances moving too fast and cause divergence. This usually happens at the second half of the placement iteration when the density term starts competing with the wirelength term by increasing λ. Thus, we need the preconditioner to stabilize the optimization. For convenience, we define two auxiliary variables $\boldsymbol{\vartheta}^{(t)}$ and $\overline{\boldsymbol{P}}^W$. With some approximation, we can derive $\boldsymbol{\alpha}$ as following by assuming that the gradient norms from the density term and the wirelength term are in the same scale.

$$\vartheta_s^{(t)} = \max\left(1, \frac{\nabla\mathcal{D}_s}{\sum_{i\in\mathcal{V}_s^r}|\partial\widetilde{W}/\partial x_i|}\right), \quad \forall s \in S, \tag{6a}$$

$$\overline{\mathcal{P}}_s^W = \frac{\sum_{i\in\mathcal{V}_s^r}\mathcal{P}_i^W}{|\mathcal{V}_s^r|}, \quad \forall s \in S, \tag{6b}$$

$$\boldsymbol{\alpha}^{(t)} = \boldsymbol{\vartheta}^{(t)} \odot \overline{\boldsymbol{\mathcal{P}}}^W, \tag{6c}$$

where $\mathcal{V}_s^r$ denotes the set of instances that have demands in field s. $\sum_{i\in\mathcal{V}_s^r}|\partial\widetilde{W}/\partial x_i|$ is the wirelength gradient norm summation of $\mathcal{V}_s^r$. The weighting vector $\boldsymbol{\vartheta}^{(t)} \in \mathbb{R}^{|S|}$ measures the gradient norm ratio between the density term and the wirelength term, and $\overline{\boldsymbol{\mathcal{P}}}^W$ denotes the average wirelength preconditioner for each field type. Due to the page limit, we omit the detailed derivations. We will further validate its effectiveness in the experiments.

3.4 Iterative Carry Chain Alignment Correction

To better align the carry chains without ruining the effectiveness of the analytical global placement algorithm, we propose a carry chain alignment technique. At the end of each global placement iteration, we move the sequential CARRYs together and align them into a column shape based

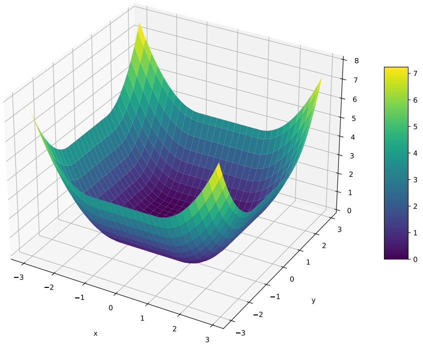


Figure 7: The Visualization of clock penalty function $\Gamma_i(\cdot)$ for a single instance.

on the horizontal coordinates of their center gravity. This means that the CARRY instances in a chain will move together during the global placement iterations, which eases the legalization step, as chains are almost aligned after global placement.

3.5 Clock Network Planning Algorithm

The clock constraints in Section 2.1.3 are highly unsmooth. Minor movements across the clock region boundaries can cause illegal clock configuration, which is unfriendly to placement optimization. Therefore, we decompose clock planning into two stages.

In the first stage, we find an *instance-to-clock-region mapping*. The mapping ensures the satisfaction of all clock region constraints, as long as all instances are located inside their target clock regions. In the second stage, we move all instances to their target clock regions by adding a penalty term $\Gamma(\cdot)$ to the placement objective. After global placement, the half-column constraints are then being handled in the developed clock-aware direct legalization and detailed placement [5]. Next, we explain these two stages.

3.5.1 Instance-to-Clock-Region Mapping Generation. The target of this step is to generate mappings such that clock constraints can be satisfied with minimum perturbation to the placement. We develop the *branch-and-bound method* in [7] to search through the solution space of different instance-to-clock-region mappings and find a feasible solution with high quality. We take the assignment with the smallest cost and use it for the second stage.

3.5.2 The Clock Penalty for Placement. Unlike [6, 8] that force the instances to directly move to their clock regions, we introduce a bowl-like, smooth, and differentiable "gravitational" attraction term, drawing the instances to their mapped clock regions. For instance i, let lo_i^x, hi_i^x, lo_i^y and hi_i^y be the left, right, bottom and top boundary coordinates of its generated mapping result. We define the penalty term for instance i as $\Gamma_i(\boldsymbol{x}_i, \boldsymbol{y}_i) = \Gamma_i(\boldsymbol{x}_i, \boldsymbol{y}_i)^x + \Gamma_i(\boldsymbol{x}_i, \boldsymbol{y}_i)^y$, where $\Gamma_i(\boldsymbol{x}_i, \boldsymbol{y}_i)^x$ is defined as,

$$\Gamma(\boldsymbol{x}_i, \boldsymbol{y}_i)^x = \begin{cases} (\boldsymbol{x}_i - lo_i^x)^2, \boldsymbol{x}_i < lo_i^x, \\ 0, lo_i^x \le \boldsymbol{x}_i \le hi_i^x, \\ (\boldsymbol{x}_i - hi_i^x)^2, hi_i^x < \boldsymbol{x}_i. \end{cases} \tag{7}$$

A visualization of the clock penalty term is shown in Figure 7. The $\Gamma(\boldsymbol{x}, \boldsymbol{y})$ in Eq. (2) is the summation of the clock penalty of all instances, i.e., $\Gamma(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i \in \mathcal{V}} \Gamma_i(x_i, y_i)$.

The clock penalty multiplier η is initialized to 0. Each time we reset the clock penalty function $\Gamma(\cdot)$, we update η with the relative ratio between the gradient norms of the wirelength and the clock penalty to maintain stability,

$$\eta = \frac{\iota \|\nabla \widetilde{W}\|_1}{\|\nabla \Gamma\|_1 + \varepsilon}. \tag{8}$$

We observe that only 1% of instances are out of their available clock regions right after the clock region assignment, so most instances have zero penalties. To balance the gradient norm ratio, ι is empirically set to 10^{-4}, and ε is set to 10^{-2} to avoid numerical explosion.

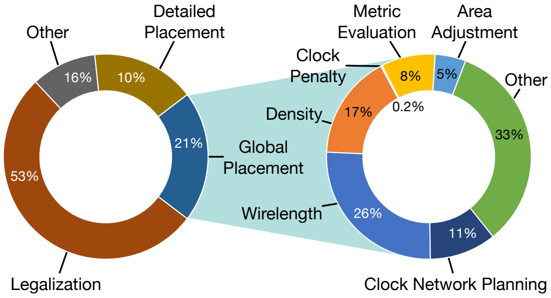


Figure 8: Runtime breakdown on CLK-FPGA13. Similar distributions are observed on other benchmarks.

4 EXPERIMENTAL RESULTS

We implemented our algorithm in C++ and Python with PyTorch for GPU-accelerated gradient computation like [23]. We conducted experiments on a Linux machine with an Intel Xeon Silver 4214 CPU (2.20 GHz and 24 cores), 251 GB RAM, and one NVIDIA TITAN RTX GPU. We tested our work on both the ISPD 2017 clock-aware FPGA placement contest benchmarks [16] and industrial benchmarks.

Table 2 summarizes the statistics of ISPD 2017 benchmarks as well as the comparison with the state-of-the-art placers, `UTPlaceF 2.0` [6], `RippleFPGA` [8], `UTPlaceF 2.X` [7], and `NTUfplace` [12]. All results of the other placers are from their original publications. It needs to be mentioned that we do not compare with `elfPlace` because its algorithm cannot handle clock constraints (see Table 1). We compare the routed wirelength by the patched Xilinx Vivado v2016.4 for ISPD 2017 and the runtime of different placers.

We can see that our placer consistently achieves better routed wirelength than other placers, i.e., 14.2% smaller than `UTPlaceF 2.0`, 11.7% smaller than `RippleFPGA`, 9.6% smaller than `UTPlaceF 2.X`, and 7.9% smaller than `NTUfplace` on average, respectively. Meanwhile, our placer is the fastest one with GPU acceleration, specifically, 4.94× faster than `UTPlaceF 2.0`, 2.38× faster than `RippleFPGA`, 1.45× faster than `UTPlaceF 2.X`, and 6.58× faster than `NTUfplace`.

We further validate our placer on industrial benchmarks, which contain a full set of heterogeneous instances including distributed RAM, SHIFT, and CARRY (see Table 3). We cannot evaluate the routed wirelength either due to the incompatibility with the Vivado patch for ISPD 2017 benchmarks. Thus, we evaluate half-perimeter wirelength (HPWL) and validate the effectiveness of our placer by disabling specific optimization techniques as following.

- Disable the iterative carry chain alignment correction in global placement and only align chains once in legalization.
- Disable the dynamic preconditioner and use the default preconditioner in [10, 22].
- Disable both techniques as the baseline.

We can see from Table 3 that the dynamically-adjusted preconditioner (Section 3.3) stabilizes the global placement iterations and enables better solution quality at convergence, while the default preconditioner [10, 22] diverges on one design. The proposed preconditioner also enables faster convergence, reducing the runtime by more than 80%. Moreover, the iterative carry chain alignment correction helps achieve 6.6% better wirelength with minor runtime overhead. This experiment validates the efficiency and effectiveness of the proposed preconditioning and carry chain alignment techniques.

We further report the runtime breakdown of our algorithm in Figure 8. With GPU acceleration, unlike most placers, global placement is no longer the runtime bottleneck. The core steps of global placement, i.e., the calculation of wirelength and electrostatic density, as well as their gradients, take only 43% of the global placement runtime. Legalization is the most time consuming part, taking 53% of the total runtime. The clock constraint related parts are relatively fast. Miscellaneous parts, such as disk IO, parsing, and data initialization, take up almost the same time as global placement, which need further optimization in the future.

Table 2: Routed Wirelength ($\times 10^3$) and Runtime (Seconds) Comparison on ISPD 2017 Benchmarks.

Design	#LUT/#FF/#BRAM/#DSP	#Clock	UTPlaceF 2.0 [6]		RippleFPGA [8]		UTPlaceF 2.X [7]		NTUfplace [12]		Ours (GPU)	
			WL	RT	WL	RT	WL	RT	WL	RT	WL	RT
CLK-FGPA01	211K/324K/164/75	32	2208	532	2011	288	2092	180	2039	698	1868	136
CLK-FGPA02	230K/280K/236/112	35	2279	513	2168	266	2194	179	2149	710	2011	130
CLK-FGPA03	410K/481K/850/395	57	5353	1039	5265	583	5109	343	4901	1704	4755	215
CLK-FGPA04	309K/372K/467/224	44	3698	711	3607	380	3600	242	3614	1148	3338	162
CLK-FGPA05	393K/469K/798/150	56	4692	939	4660	569	4556	323	4417	1540	4154	208
CLK-FGPA06	425K/511K/872/420	58	5589	1066	5737	591	5432	346	5122	2210	4918	229
CLK-FGPA07	254K/309K/313/149	38	2444	845	2326	304	2324	201	2320	795	2145	141
CLK-FGPA08	212K/257K/161/75	32	1886	529	1778	247	1807	169	1803	588	1648	120
CLK-FGPA09	231K/358K/236/112	35	2601	842	2530	327	2507	197	2436	717	2248	144
CLK-FGPA10	327K/506K/542/255	47	4464	974	4496	512	4229	286	4339	1597	3839	200
CLK-FGPA11	300K/468K/454/224	44	4183	1068	4190	455	3936	265	3964	1618	3626	183
CLK-FGPA12	277K/430K/389/187	41	3369	774	3388	409	3236	247	3179	849	2938	168
CLK-FGPA13	339K/405K/570/262	47	3816	1172	3833	441	3723	270	3680	985	3404	181
Ratio			1.142	4.943	1.117	2.379	1.096	1.453	1.079	6.575	1.000	1.000

Table 3: HPWL ($\times 10^3$) and Runtime (Seconds) Comparison with Different Techniques on Industry Benchmarks.

Design	#LUT/#FF/#BRAM/#DSP	#Distributed RAM+#SHIFT	#CARRY	w/o precond or chain align (GPU)		w/o precond (GPU)		w/o chain align (GPU)		Ours (GPU)	
				WL	RT	WL	RT	WL	RT	WL	RT
IND01	17K/11K/0/13	9	2K	87	123	76	131	76	44	65	45
IND02	11K/10K/0/24	6	335	86	75	78	80	83	54	77	54
IND03	109K/12K/0/0	0	0	611	55	600	56	602	58	597	59
IND04	29K/17K/0/16	218	1K	200	181	186	156	174	84	165	83
IND05	64K/191K/64/928	29K	4K	1228	173	1152	187	1040	100	998	97
IND06	112K/65K/21/0	0	4K	967	166	926	203	861	82	792	84
IND07	40K/156K/89/768	26K	3K	diverge	diverge	diverge	diverge	761	81	745	83
Ratio				1.193	1.841	1.107	1.935	1.066	0.996	1.000	1.000

5 CONCLUSION

In this paper, we propose a heterogeneous FPGA placement algorithm considering SLICEL-SLICEM heterogeneity and clock feasibility. Based on a new electrostatic formulation, we design a uniform optimization paradigm for wirelength, routability, and clock feasibility supporting heterogeneous instance types, including LUT, FF, BRAM, DSP, distributed RAM, SHIFT, and CARRY. We further propose a dynamically-adjusted preconditioner and a smooth clock penalization technique to ensure the placement converging to high-quality solutions. Experiments on ISPD 2017 contest benchmarks demonstrate that our placer can achieve 14.2%, 11.7%, 9.6%, and 7.9% better routed wirelength than the state-of-the-art placers like UTPlaceF 2.0, RippleFPGA, UTPlaceF 2.X, and NTUfplace, respectively, with 1.5-6× speedup leveraging GPU acceleration. Experiments on industrial benchmarks demonstrate that our techniques can improve the wirelength by 19.3%, 10.7%, and 6.6%.

ACKNOWLEDGE

This work was supported in part by the National Science Foundation of China (Grant No. 62034007 and No. 62141404) and the 111 Project (B18001).

REFERENCES

[1] S. Chen and Y. Chang, “Routing-architecture-aware analytical placement for heterogeneous fpgas,” in *Proc. DAC*. ACM, 2015, pp. 27:1–27:6.

[2] W. Li, S. Dhar, and D. Z. Pan, “Utplacef: A routability-driven FPGA placer with physical and congestion aware packing,” *IEEE TCAD*, vol. 37, no. 4, pp. 869–882, 2018.

[3] C. Pui, G. Chen, W. Chow, K. Lam, J. Kuang, P. Tu, H. Zhang, E. F. Y. Young, and B. Yu, “Ripplefpga: a routability-driven placement for large-scale heterogeneous fpgas,” in *Proc. ICCAD*, 2016, p. 67.

[4] R. Pattison, Z. Abuowaimer, S. Areibi, G. Gréwal, and A. Vannelli, “Gplace: a congestion-aware placement tool for ultrascale fpgas,” in *Proc. ICCAD*, 2016, p. 68.

[5] W. Li and D. Z. Pan, “A new paradigm for FPGA placement without explicit packing,” *IEEE TCAD*, vol. 38, no. 11, pp. 2113–2126, 2019.

[6] W. Li, Y. Lin, M. Li, S. Dhar, and D. Z. Pan, “Utplacef 2.0: A high-performance clock-aware FPGA placement engine,” *ACM TODAES*, vol. 23, no. 4, pp. 42:1–42:23, 2018.

[7] W. Li, M. E. Dehkordi, S. Yang, and D. Z. Pan, “Simultaneous placement and clock tree construction for modern fpgas,” in *Proc. FPGA*, 2019, pp. 132–141.

[8] C. Pui, G. Chen, Y. Ma, E. F. Y. Young, and B. Yu, “Clock-aware ultrascale FPGA placement with machine learning routability prediction: (invited paper),” in *Proc. ICCAD*. IEEE, 2017, pp. 929–936.

[9] T. Liang, G. Chen, J. Zhao, L. Feng, S. Sinha, and W. Zhang, “Amf-placer: High-performance analytical mixed-size placer for fpga,” in *Proc. ICCAD*, 2021, pp. 1–6.

[10] Y. Meng, W. Li, Y. Lin, and D. Z. Pan, “elfplace: Electrostatics-based placement for large-scale heterogeneous fpgas,” *IEEE TCAD*, 2021.

[11] Y. Kuo, C. Huang, S. Chen, C. Chiang, Y. Chang, and S. Kuo, “Clock-aware placement for large-scale heterogeneous fpgas,” in *Proc. ICCAD*, 2017, pp. 519–526.

[12] J. Chen, Z. Lin, Y. Kuo, C. Huang, Y. Chang, S. Chen, C. Chiang, and S. Kuo, “Clock-aware placement for large-scale heterogeneous fpgas,” *IEEE TCAD*, vol. 39, no. 12, pp. 5042–5055, 2020.

[13] G. Chen, C.-W. Pui, W.-K. Chow, K.-C. Lam, J. Kuang, E. F. Young, and B. Yu, “RippleFPGA: Routability-driven simultaneous packing and placement for modern FPGAs,” *IEEE TCAD*, vol. 37, no. 10, pp. 2022–2035, 2018.

[14] Z. Abuowaimer, D. Maarouf, T. Martin, J. Foxcroft, G. Gréwal, S. Areibi, and A. Vannelli, “GPlace3.0: Routability-driven analytic placer for UltraScale FPGA architectures,” *ACM TODAES*, vol. 23, no. 5, pp. 66:1–66:33, 2018.

[15] Ultrascale architecture clb slices. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug574-ultrascale-clb.pdf

[16] S. Yang, C. Mulpuri, S. Reddy, M. Kalase, S. Dasasathyan, M. E. Dehkordi, M. Tom, and R. Aggarwal, “Clock-aware FPGA placement contest,” in *Proc. ISPD*, 2017, pp. 159–164.

[17] Ultrascale architecture clocking resources. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug572-ultrascale-clocking.pdf

[18] S. Yang, A. Gayasen, C. Mulpuri, S. Reddy, and R. Aggarwal, “Routability-driven FPGA placement contest,” in *Proc. ISPD*, 2016, pp. 139–143.

[19] J. Lu, P. Chen, C.-C. Chang, L. Sha, D. J.-H. Huang, C.-C. Teng, and C.-K. Cheng, “ePlace: Electrostatics-based placement using fast fourier transform and nesterov’s method,” *ACM TODAES*, vol. 20, no. 2, p. 17, 2015.

[20] J. Lu, H. Zhuang, P. Chen, H. Chang, C.-C. Chang, Y.-C. Wong, L. Sha, D. Huang, Y. Luo, C.-C. Teng *et al.*, “ePlace-MS: Electrostatics-based placement for mixed-size circuits,” *IEEE TCAD*, vol. 34, no. 5, pp. 685–698, 2015.

[21] R. Andreani, E. G. Birgin, J. M. Martínez, and M. L. Schuverdt, “On augmented lagrangian methods with general lower-level constraints,” *SIAM Journal on Optimization*, vol. 18, no. 4, pp. 1286–1309, 2008.

[22] C.-K. Cheng, A. B. Kahng, I. Kang, and L. Wang, “Replace: Advancing solution quality and routability validation in global placement,” *IEEE TCAD*, 2018.

[23] Y. Lin, Z. Jiang, J. Gu, W. Li, S. Dhar, H. Ren, B. Khailany, and D. Z. Pan, “DREAMPlace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement,” *IEEE TCAD*, June 2020.