# Multi-Electrostatic FPGA Placement Considering SLICEL-SLICEM Heterogeneity, Clock Feasibility, and Timing Optimization

Jing Mai, Jiarui Wang, Zhixiong Di *Member, IEEE*, Yibo Lin *Member, IEEE*

***Abstract*—When modern FPGA architecture becomes increasingly complicated, modern FPGA placement is a mixed optimization problem with multiple objectives, including wirelength, routability, timing closure, and clock feasibility. Typical FPGA devices nowadays consist of heterogeneous SLICEs like SLICEL and SLICEM. The resources of a SLICE can be configured to {LUT, FF, distributed RAM, SHIFT, CARRY}. Besides such heterogeneity, advanced FPGA architectures also bring complicated constraints like timing, clock routing, carry chain alignment, etc. The above heterogeneity and constraints impose increasing challenges to FPGA placement algorithms.**

**In this work, we propose a multi-electrostatic FPGA placer considering the aforementioned SLICEL-SLICEM heterogeneity under timing, clock routing and carry chain alignment constraints. We first propose an effective SLICEL-SLICEM heterogeneity model with a novel electrostatic-based density formulation. We also design a dynamically adjusted preconditioning and carry chain alignment technique to stabilize the optimization convergence. We then propose a timing-driven net weighting scheme to incorporate timing optimization. Finally, we put forward a nested Lagrangian relaxation-based placement framework to incorporate the optimization objectives of wirelength, routability, timing, and clock feasibility. Experimental results on both academic and industrial benchmarks demonstrate that our placer outperforms the state-of-the-art placers in quality and efficiency.**

## I. Introduction

Placement is a critical step in the FPGA design flow, with a great impact on routability and timing closure. In the literature, three types of FPGA placement have been investigated: 1) partitioning-based, 2) simulated annealing (SA), and 3) analytical approaches [1], [2]. Partitioning-based approaches such as [3] usually have good scalability, but often fail to achieve high-quality results. SA-based approaches like the widely-adopted academic tool `VPR` [3] can achieve good results on small designs, but suffer from poor scalability on large designs. Recent studies have shown that analytical approaches [4]–[15] can achieve the best trade-off between quality and runtime. Thus modern FPGA placers mainly adopt analytical approaches in academia and industry.

The preliminary version has been presented at the Design Automation Conference (DAC) in 2022. This work was supported in part by the National Science Foundation of China (Grant No. 62034007 and No. 62141404) and the 111 Project (B18001).

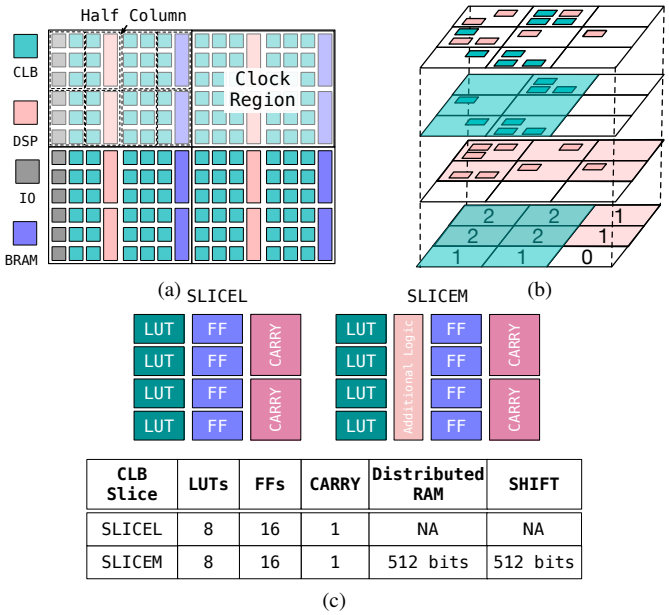
J. Mai and J. Wang are with School of Computer Science, Peking University, Beijing, China.

Y. Lin is with School of Integrated Circuits, Peking University, Beijing, China. Corresponding author: Yibo Lin (yibolin@pku.edu.cn).

Z. Di is with School of Information Science and Technology, Southwest Jiaotong University, Chengdu, China.

Modern FPGA placement has two major challenges: 1) the heterogeneity of FPGA architecture, 2) the various constraints (e.g., timing, clock routing, chain alignment, etc.) imposed by advanced circuit designs [2], [12], [16]–[18]. The heterogeneity of the FPGA architecture comes from the variety of instance types, the imbalance of resource distribution, and the asymmetric slice compatibility from the SLICEL-SLICEM heterogeneity [19]. The diversity of instance sizes and the inconsecutive site compatibility challenge the modern FPGA placement algorithms, which are mainly based on continuous optimization [2], [13]. Dealing with the inherent heterogeneity of SLICEL-SLICEM presents a considerable challenge for FPGA placement in practical scenarios. While some previous works have explored FPGA placement, a few previous works delved into SLICEL-SLICEM heterogeneity. [12] applies a progressive legalization strategy to LUTRAM, with an anchor set on the target location and pseudo nets added to link the anchor and LUTRAMs, allowing the instance to progressively move into the corresponding SLICEM. However, this method happens at the late placement stage and lacks a global perspective.

Furthermore, solving the highly heterogeneous FPGA placement problem while satisfying advanced constraints has become more challenging [2]. i) Wire-induced delays are becoming the primary source of overall circuit delay [20]–[22]. Timing-driven placement is required to meet aggressive timing constraints. However, the nonlinear and nonmonotonic wire delays impose unique challenges to timing optimization in FPGA placement. ii) Modern FPGA adopts complicated clock architectures to achieve low clocking skew and high performance [16], [17]. Such a clock architecture introduces complicated clock routing constraints, increasing the challenges in FPGA placement. Therefore, clock-aware FPGA placement is required to accommodate the needs of modern FPGA design flows. iii) Modern FPGA needs to align the cascaded instances like CARRY into an aligned chain at the placement stage to boost performance. This chain alignment requirement induces large placement blocks and tends to degrade the quality of the solution. CARRY chain positions in global placement have a high impact on the subsequent legalization steps and eventual quality of results. In global placement, the relative position legality of the CARRY chain is imperative, and subsequent lookahead legalization steps are performed to optimize further. CARRY chain placement algorithms are mainly classified into two categories in FPGA global placement. The first category is *hard macro*, which




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

(c)

Fig. 1: (a) An example of a simplified vertical FPGA architecture depiction showing a $2 \times 2$ clock region and half columns (dash lines) for *Xilinx UltraScale*. (b) An example of how the clock demand can be calculated. The different colors represent different clocks, and the numbers represent the clock demand of each CR. (c) An illustration of CLB slices classified into SLICEL and SLICEM with asymmetric compatibility. In a SLICEL, LUT blocks can be configured to be LUTs. A SLICEM can only be configured in *one* of the following modes: LUT, distributed RAM, or SHIFT. There is no mixing of LUTs, distributed RAMs, and SHIFTs in a CLB.

merges the CARRY chain and treats them as a whole during placement [12], [23], [24]. The second one is *macro shredding*, which replaces macros with many small and tightly interconnected fragments [25]–[27]. Based on the placement of the shredded netlist, the macro position can finally be inferred from their respective fragments, e.g., their center of gravity.

In this work, we propose a state-of-the-art placement framework considering SLICEL-SLCIEM heterogeneity and the co-optimation with wirelength, routability, clock feasibility, and timing optimization. We handle a comprehensive set of instance types, i.e, { LUT, FF, BRAM, distributed RAM, SHIFT, CARRY }, and cope with SLICEL-SLCIEM heterogeneity based on a multi-electrostatic system. We propose a uniform non-linear optimization paradigm taking wirelength, routability, clock feasibility, and timing optimization into consideration from the perspective of *nested Lagrangian method*. The main contributions of this work are summarized as follows.

- We adopt an effective SLICEL-SLICEM heterogeneity model based on the division and assembly of electrostatic-based density formulation.
- We develop a dynamically adjusted preconditioning and carry chain alignment technique to stabilize the optimization convergence and enable better final placement results.
- We cope with the time violation by an effective timing-criticality-based net weighting scheme, and incorporate the timing optimization into a continuous optimization algorithm.
- To achieve effective clock routing violation elimination, we adopt a instance-to-clock-region mapping considering the resource capacity of the clock regions and perturbation to the placement, and propose a quadratic clock penalty function in a continuous global placement engine with minor quality degradation.
- Putting the aforementioned techniques together, we put forward a nested Lagrangian relaxation framework incorporating the optimization objectives of wirelength, routability, timing, and clock feasibility.

Experiments on *ISPD 2017 contest benchmarks* demonstrate 14.2%, 11.7%, 9.6%, and 7.9% improvement in routed wirelength, compared to the recent cutting-edge FPGA placers [9]–[11], [15], respectively. Our placer also supports GPU acceleration and gains 1.45-6.58$\times$ speedup over the baselines. Further experiments on *industrial* benchmarks demonstrate that the proposed algorithms can achieve 16% better fMAX, 23.6% better WNS, 22.5% better TNS with about 2% routed wirelength degradation compared with the conference version.

The rest of the paper is organized as follows. Section II introduces the preliminary knowledge of the FPGA architecture and modern FPGA placement. Section III details the core placement algorithms. Section IV shows the experimental results, followed by the conclusion in Section V.

## II. Preliminaries

In this section, we primarily focus on the architecture of FPGAs and the methodology of multi-electrostatics-based FPGA placement.

### A. Device Architecture

In this work, we use the *Xilinx UltraScale* family [19], [30],e.g., the *UltraScale VU095* as a model FPGA design (illustrated in Fig. 1a). The ISPD 2016 and 2017 FPGA placement challenges employed a condensed version of this architecture with a limited number of instance types, including LUT, FF, BRAM, and DSP [16], [17].

*1) SLICEL-SLICEM heterogeneity:* Shift registers (SHIFT) and distributed RAMs are two additional LUT-like instances of the architecture in addition to regular LUTs. Slices in CLBs fall into two categories: SLICEL and SLICEM, whose architectures are depicted in Fig. 1c. Due to the modest differences in logic resources, SLICEL and SLICEM support various configurations. The SLICEL-SLICEM placment constraints can be summarized as follows:

- LUTs, FFs, and CARRYs can be placed in both SLICEL and SLICEM.
- SHIFT and distributed RAMs can only be placed in SLICEM, which sets them apart from other common instances such as DSPs and BRAMs.
- A SLICEM cannot be used as SHIFTs or distributed RAMs if it is configured as LUTs; vice versa.

TABLE I: Features of the published state-of-the-art FPGA placers.

| Placer | | RippleFPGA [28] | GPlace [7] | UTPlaceF [5] | elfPlace [13] | FTPlace [20] | GPlace 3.0 [29] | RippleFPGA Clock-Aware [11] | UTPlaceF 2.0&2.X [9], [10] | NTUfPlace [15] | Lin *et al.* [21] | Ours |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Clock Constraints | | × | × | × | × | × | × | ✓ | ✓ | ✓ | ✓ | ✓ |
| Resources Supported | LUT, FF, BRAM, DSP | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | CARRY, SHIFT, Distributed RAM | × | × | × | × | × | × | × | × | × | × | ✓ |
| Timing Optimization | | × | × | × | × | ✓ | × | × | × | × | ✓ | ✓ |
| GPU-Acceleration | | × | × | × | ✓ | × | × | × | × | × | × | ✓ |
| Algorithm Category | | Quadratic | Quadratic | Quadratic | Nolinear | Quadratic | Quadratic | Quadratic | Quadratic | Nonlinear | Nonlinear | Nonlinear |

*2) Carry Chain Alignment Constraints:* As a placement constraint for CARRY instances, we also need to take carry chain alignment into account. A carry chain is made up of several consecutive CARRY instances connected by cascaded wires from the lower bits to the upper bits, and the CARRY instances are arranged in CLB slices. According to the alignment constraint, each CARRY instance in a chain must be positioned in a single column and in subsequent slices in the correct sequence for the cascading wires.

*3) Clock Constraints:* The target FPGA device owns $5 \times 8$ rectangular-shaped clock regions (CRs) in a grid manner, as shown in Fig. 1a. [1]. Each CR is made up of columns of site resources and can be further horizontally subdivided into pairs of lower and upper half columns (HCs) of half-clock-region height. Except for a few corner cases, the width of each HC is the same as that of two site columns, as shown in Fig. 1a. All the clock sinks within a half column are driven by the same leaf clock tracks.

The clock routing architecture imposes two clock constraints on placement, i.e., the *clock region constraint* and the *half column constraint*, as shown in Fig. 1b. The clock region constraint limits each clock region's clock demand to a maximum of 24 clock nets, where the clock demand is the total number of clock nets whose bounding boxes intersect with the clock region. The half-column constraint limits the number of clock nets within the half-column to a maximum of 12.

### B. Multi-Electrostatics based FPGA Placement

As shown in Fig. 2, electrostatics-based placement models each instance as an electric particle in an electrostatic system. As firstly stated in the ASIC placement [31], *minimizing potential energy* can resolve density overflow in the layout. The principle is based on the fundamental physical insight that a balanced charge distribution in an electrostatic system contributes to low potential energy, so *minimizing potential energy can resolve density overflow and help spread instances in the layout*. We are also extending this approach to the use of multiple electrostatic fields, which will enable multiple types of resource to be handled in FPGA placement, such as LUTs, FFs, DSPs, and BRAMs. Fig. 3 [2]. illustrates a multi-electrostatic formulation of LUTs and DSPs. In order to reduce density overflow, we must minimize the total potential energy of multiple fields because low energy means a balanced distribution of instances. The issue can be summarized as follows.

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

where $\widetilde{W}(\cdot)$ is the wirelength objective, $\boldsymbol{x},\boldsymbol{y}$ are instance locations, $S$ denotes the field type set, and $\Phi_s(\cdot)$ is the electric potential energy for field type $s \in S$. Formally, we constrain the target potential energy of each field type to be zero, though the energy is usually non-negative. The constraints can be further relaxed to the objective and guide the instances to spread out. Practically, we stop the optimization when the energy is small enough; or equivalently, the density overflow is low enough. Notice that the formulation in Fig. 3 assumes that one instance occupies the resources of only one field, which cannot handle the complicated SLICEL-SLICEM heterogeneity shown in Section II-A1.

### C. Timing Optimization

Timing-driven placement imposes more concern about timing closure than the total wirelength objective in wirelength-driven placement. Worst negative slack (WNS) and total negative slack (TNS) are two widely adopted timing metrics. WNS is the maximum negative slack among all timing paths in the design, and TNS is the sum of all negative slacks of timing endpoints. Thus, WNS and TNS are used to evaluate the timing performance of a design from the worst and the global view respectively, and the smaller the WNS and TNS are, the worse the timing performance is. The timing-driven placement problem can be formulated as follows.

$$\min_{\boldsymbol{x},\boldsymbol{y}} \quad \mathcal{T}(\boldsymbol{x},\boldsymbol{y}), \tag{2a}$$

$$\text{s.t.} \quad \rho_s(\boldsymbol{x},\boldsymbol{y}) \le \hat{\rho}_s, \quad \forall s \in S, \tag{2b}$$

where $S$ is the instance type set, $\rho_s(\cdot)$ denotes the density for instance type $s \in S$, and $\hat{\rho}_s$ represents the target density

[1]Fig. 1a only contains part of the whole $5 \times 8$ CRs, but is sufficient for illustration.

[2]In this illustration, charges on the instances and the illegal sites are of the same polarity, and legal sites on the layout have zero charges for the respective field. To further make the total charge zero [31], [32], when we transform to the frequency domain using FFT we adopt the technique called *Direct-Current (DC) component removal*, which is also stated at Section IV-B in `ePlace-MS` [33], This is equivalent to subtracting the mean from the density $\rho(\cdot)$ in the space domain before performing FFT transformation, so that the overall charge is zero. Actually, the purpose of setting the overall charge to zero is to ensure that the electric potential, whose negative gradient is electric field strength, satisfies the *zero Neumann boundary condition* (where the normal derivative at a boundary is zero), which prevents the instances moving out of the boundary.

for instance type $s \in S$. The objective function $\mathcal{T}(\cdot)$ can be WNS, TNS, or the weighted sum of both. Improving TNS requires collaborative optimization of all timing paths, and is therefore suitable for the global placement stage. On the other hand, WNS is more suitable for the detailed placement stage, as it only considers the worst timing path. It is worth noting that directly solving Eq. (2) is very difficult, because the delay model generally has strong discrete and non-convex properties [34]. Therefore, we draw on the two intuitive elements of wirelength-driven placement and static timing analysis, i.e., net weights and slacks, to tackle this problem.

### D. Problem Formulation

TABLE I summarizes the characteristics of the published state-of-the-art FPGA placers. In recent years, modern FPGA placers mainly resort to quadratic programming-based approaches [4]–[12] and nonlinear optimization-based approaches [13]–[15] for the best trade-off between quality and efficiency. Among them, the current state-of-the-art quality is achieved by nonlinear approaches `elfPlace` [13] and `NTUfPlace` [15], whose instance density models are derived from a multi-electrostatics system and a hand-crafted bell-shaped field system. However, most existing FPGA placers only consider a simplified FPGA architecture, i.e., LUT, FF, DSP, and BRAM, ignoring the commonplace SLICEL-SLCIEM heterogeneity in real FPGA architectures [5]–[11], [13]–[15]. Among these placers, only a few placers utilize the parallelism that the GPU provides [13], and few consider timing and clock feasibility in practice [8], [9], [11], [14], [15].

In this work, we aim at optimizing wirelength, timing, and routability while cooperating with SLICEL-SLICEM heterogeneity, alignment feasibility, and clock constraints. We define the FPGA placement problem as follows.

**Problem 1** (FPGA Placement)**.** *Taking as input a netlist consisting of LUTs, FFs, DSPs, BRAMs, distributed RAMs, SHIFTs, and CARRYs, generate a plausible FPGA placement solution with optimized wirelength, timings, and routings, satisfying the requirements for alignment feasibility and meeting the clock constraints.*

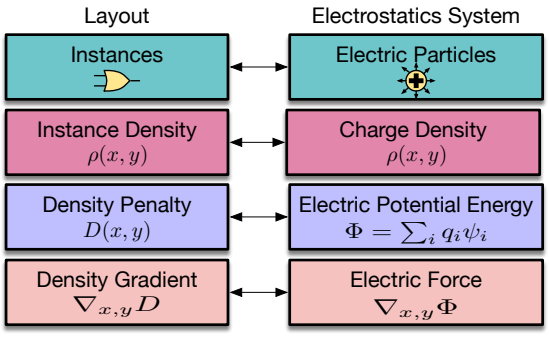



Fig. 2: Analogy between placement for a single resource type and an electrostatic system [33].

## III. Algorithms

We will detail the placement algorithm in this section.

### A. Overview of the Proposed Algorithm

As illustrated in Fig. 4, our method includes two fundamental stages: (1) nested global placement with timing awareness and clock feasibility, and (2) clock-aware legalization and detailed placement.

We cope with the SLICEL-SLICEM heterogeneity by defining the field type set as $S = \{$LUTL, LUTM-AL, FF, CARRY, DSP, BRAM$\}$ with a special field setup (Section III-B). With clock constraints, carry chain alignment feasibility, and timing optimization, we formulate the problem as Formulation (3).

$$\min_{\boldsymbol{x},\boldsymbol{y}} \quad \widetilde{\mathcal{T}}_{\boldsymbol{\omega}}(\boldsymbol{x},\boldsymbol{y}), \tag{3a}$$

$$\text{s.t.} \quad \Phi_s(\boldsymbol{x},\boldsymbol{y};\mathcal{A}^s) = 0, \quad \forall s \in S, \tag{3b}$$

$$\Gamma(\boldsymbol{x},\boldsymbol{y}) = 0, \tag{3c}$$

$$\textit{CARRY chain alignment constraints,} \tag{3d}$$

$$\textit{SLICEL-SLICEM constraints,} \tag{3e}$$

$$\textit{DSP and BRAM constraints,} \tag{3f}$$

$\widetilde{\mathcal{T}}_{\boldsymbol{\omega}}(\cdot)$ is the timing performance objective, where $\boldsymbol{\omega}$ measures the net criticality in the current timing graph (Section III-F). $\mathcal{A}^s$ denotes the instance areas in the field $s$, and $\Gamma(\cdot)$ is the clock penalty term (Section III-E). For brevity, in later discussions, we condense $\Phi_s(\boldsymbol{x},\boldsymbol{y};\mathcal{A}^s)$ to $\Phi_s$ for all $s \in S$, and denote $\boldsymbol{\Phi}$ as the potential energy vector, whose components are the potential energy for each field, i.e., $\Phi_s(\forall s \in S)$. Eq. (3d) (see Section II-A2), Eq. (3e) (see Section II-A1), and Eq. (3f) [16], [17] are introduced to ensure the legality of the placement. For brevity, we will not explicitly write them down in later discussions.

We relax the original problem (3) by leveraging the *augmented Lagrangian method (ALM)* [35] to formulate a better unconstrained subproblem,

$$\min_{\boldsymbol{x},\boldsymbol{y}} \quad \mathcal{L}(\boldsymbol{x},\boldsymbol{y};\boldsymbol{\lambda},\boldsymbol{\mathcal{A}},\eta,\boldsymbol{\omega}) = \widetilde{\mathcal{T}}_{\boldsymbol{\omega}}(\boldsymbol{x},\boldsymbol{y}) + \sum_{s\in S}\lambda_s \mathcal{D}_s + \eta\Gamma(\boldsymbol{x},\boldsymbol{y}), \tag{4a}$$

$$\mathcal{D}_s = \Phi_s + \frac{1}{2}\mathcal{C}_s\Phi_s^2, \quad \forall s \in S, \tag{4b}$$

The density multiplier vector is $\boldsymbol{\lambda} \in \mathbb{R}^{|S|}$, and the clock penalty multiplier is $\eta \in \mathbb{R}$. The purpose of the weighting coefficient vector $\boldsymbol{\mathcal{C}} \in \mathbb{R}^{|S|}$ is to achieve a balance between the first-order and second-order terms for density penalty. We follow the setup for $\boldsymbol{\lambda}$ and $\boldsymbol{\mathcal{C}}$ as [13]. To cope with multiple constraints, we rewrite the problem in a nested manner through the Lagrangian relaxation method,

$$\text{Timing Opt.:} \quad \mathcal{L}_1 = \max_{\boldsymbol{\omega}} \mathcal{L}_2(\boldsymbol{\omega}), \tag{5a}$$

$$\text{Clock Opt.:} \quad \mathcal{L}_2(\boldsymbol{\omega}) = \max_{\eta} \mathcal{L}_3(\eta,\boldsymbol{\omega}), \tag{5b}$$

$$\text{Routability Opt.:} \quad \mathcal{L}_3(\eta,\boldsymbol{\omega}) = \max_{\boldsymbol{\mathcal{A}}} \mathcal{L}_4(\boldsymbol{\mathcal{A}},\eta,\boldsymbol{\omega}), \tag{5c}$$

$$\text{Wirelength Opt.:} \quad \mathcal{L}_4(\boldsymbol{\mathcal{A}},\eta,\boldsymbol{\omega}) = \max_{\boldsymbol{\lambda}} \mathcal{L}_5(\boldsymbol{\lambda},\boldsymbol{\mathcal{A}},\eta,\boldsymbol{\omega}), \tag{5d}$$

$$\text{Subproblem:} \quad \mathcal{L}_5(\boldsymbol{\lambda},\boldsymbol{\mathcal{A}},\eta,\boldsymbol{\omega}) = \min_{\boldsymbol{x},\boldsymbol{y}} \mathcal{L}(\boldsymbol{x},\boldsymbol{y};\boldsymbol{\lambda},\boldsymbol{\mathcal{A}},\eta,\boldsymbol{\omega}), \tag{5e}$$

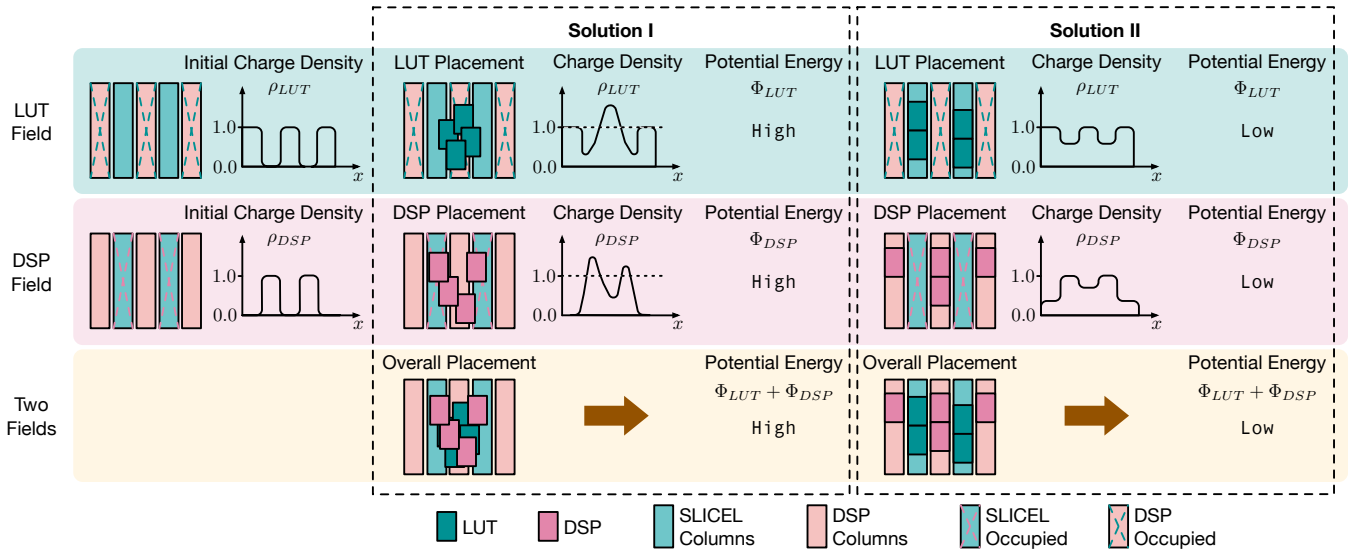



Fig. 3: An example of a multi-electrostatic formulation for LUT and DSP resources, which correspond to two electric fields. Unavailable columns are treated as occupied when calculating the initial charge density for each field. Take DSP as an example. Density overflow can occur if a DSP instance is not put in a DSP column or if there are overlaps between DSP instances, resulting in an uneven density distribution of the field and, finally, excessive electric potential energy. As a result, limiting energy in the layout can assist in the resolution of density overflow and spread cases. If we face density underflow ($< 1.0$), we can insert fillers with positive charges for each field to fill the vacant spaces, [13], [31]. As a result, only density overflow will provide considerable potential energy. We may handle them in the same way by including FF and BRAM fields.

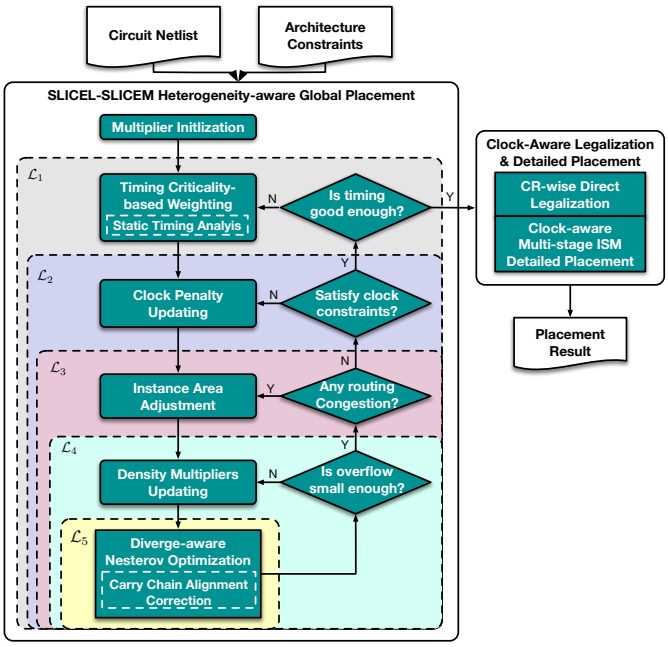



Fig. 4: The proposed Overall Flow.

where $\mathcal{L}_5$ denotes Eq. (4). To put it simply, a set of variables in the objective function is constrained to be the optimal solution of the extra optimization problem, and the variables of the exterior optimization problem are passed toward the sub-problem as fixed parameters. To illustrate, take the solving process of $\mathcal{L}_5$ as an example. The subproblem $\mathcal{L}_5$ aims at finding the optimal instance positions $\boldsymbol{x}$ and $\boldsymbol{y}$ given a set of fixed parameters $\boldsymbol{\lambda}$, $\boldsymbol{\mathcal{A}}$, $\eta$, and $\boldsymbol{\omega}$ from $\mathcal{L}_4$. When $\mathcal{L}_5$ comes to the convergence to its minimum solution, the $\mathcal{L}_4$ optimizer will improve the parameter $\boldsymbol{\lambda}$ so as to enlarge the magnitude of the density terms in the overall optimization objectives. Therefore, the $\mathcal{L}_5$ optimizer will try to find a better solution by moving the instances to a less dense region in a new iteration, and thus the density constraints will be forced to be gradually adequate. We also adopt the same procedure to solve the other subproblems.

Fig. 4 depicts the nested loops to solve the problem. $\mathcal{L}_1$ aims at improving the timing slacks via the timing-criticality-based weighting method. The stopping criterion of $\mathcal{L}_1$ is whether timing slacks can be improved (Section III-F) [3]. $\mathcal{L}_2$ (Section III-E) aims at excluding the cases where clock routing is illegal under an analytical formulation. We regard $\mathcal{L}_2$ as converged when there is no clock violation (Section II-A3). $\mathcal{L}_3$ develops the area inflation-based technique from [13] in order to optimize the routability, and the estimated routing congestion and pin density are the convergence criteria of $\mathcal{L}_3$. $\mathcal{L}_4$ is the core wirelength-driven placement problem. We empirically find that the density *overflow* is a good indicator of the density constraints. For $\mathcal{L}_5$, we always solve with a fixed number of iterations, e.g., one iteration in the experiments.

In each iteration, we resolve the carry chain alignment constraints through iterative support from *iterative carry chain*

[3]We consider timing is good enough when 1) The $\mathcal{L}_2$ sub-problem has converged, 2) there have been at least 100 $\mathcal{L}_5$ iterations since the last timing adjustment, and the WNS has not improved compared to the previous adjustment, 3) the density overflow for LUTs and FFs is less than 10%. Otherwise, If the number of $\mathcal{L}_5$ iterations is more than 100, we perform a timing criticality-based weighting and continue solving the $\mathcal{L}_2$ problem; else we opt to continue solve the $\mathcal{L}_2$ problem directly for further optimization.

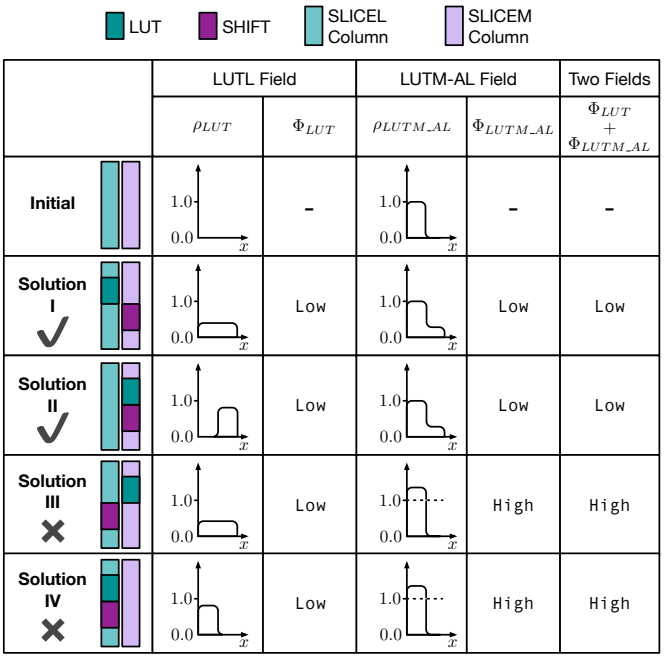



Fig. 5: As an example, here is a special electrostatic field setup that handles asymmetric slice compatibility as a result of SLICEL-SLICEM heterogeneity. LUTM-AL fields are prone to overflowing density due to the SHIFT instance placed on the SLICEL column in Solution III and IV. This results in high potential energy in the LUTM-AL field.

*alignment correction* (Section III-D). Following placement, we evolve SLICEM and clock-aware direct legalization and detailed placement algorithms that above clock feasibility constraints are met [8]. We do not include details on routability optimization, SLICEM-aware legalization, clock-aware legalization, or detailed placement for brevity.

### *B. Multi-Electrostatic Model for SLICEL-SLICEM Heterogeneity*

SLICEMs can operate in one of three modes: LUT, distributed RAM, or SHIFT, as discussed in Section II-A1. In the LUT slots of that SLICEM, only instances that match the mode can be placed. In order to address this constraint, we introduce two electrostatic fields, *LUTL* and *LUTM-AL*, into the multi-electrostatic placement model. In the field setup, it should be possible to prevent distributed RAM instances or SHIFT instances from being placed on SLICEL sites, however, it should be possible to place LUT instances both on SLICEL and SLICEM sites, as part of the field setup.

There is an example of how these two fields are set up in Fig. 5. LUTL models the LUT resources that are provided by both SLICEM and SLICEL, whereas LUTM-AL models the additional logic resources that are supplied by SLICEM but not by SLICEL. In contrast to a distributed RAM or SHIFT instance, a LUT instance only occupies resources within the LUTL field, while a distributed RAM or SHIFT instance occupies resources both in the LUTL and LUTM-AL fields.

Using an example of a LUT instance and a SHIFT instance as an example, this figure analyses four scenarios. As with SHIFT instances, distributed RAM instances work in exactly the same way. In order to indicate that a SLICEL does not contain any resources for LUTM-AL, we set the initial density for LUTM-AL in a SLICEL to 1, indicating the SLICEL is occupied; in contrast, the initial density for LUTL in a SLICEL is set to 0. We set the initial density of a SLICEM to zero for both LUTL and LUTM-AL due to the fact that it contains both LUTL and LUTM-AL resources. In Solution I, if the LUT instance is placed on a SLICEL and the SHIFT instance is placed on a SLICEM, there will be no overflow of density in either of the fields (a balanced density distribution can be achieved by inserting fillers [31]), which means that the potential energy will be minimized. In Solution II, the scenario is similar to the one in Solution I. Solution III and IV, on the other hand, where the SHIFT instance is placed on a SLICEL, produce a density overflow in the LUTM-AL field, which indicates that there is a high potential energy for this field. As long as the optimizer minimizes the potential energy, these solutions will be avoided. As a result, these two elaborate fields are able to accommodate LUT and distributed RAM/SHIFT to their compatible sites in an easy manner.

### *C. Divergence-aware Preconditioning*

It is important to precondition the gradient $\nabla\mathcal{L}^{(t)}$ for each iteration $t$ before it is fed to the optimizer, where $\mathcal{L}$ is the Lagrangian problem defined in Eq. (4). A gradient is discussed only in the direction of $x$, and a gradient in the direction of $y$ is the same. To make the Jacobi preconditioner $\mathcal{P}$ more efficient, we approximate the second-order derivatives of wirelength and density according to the following formula.

$$\mathcal{P}_i^W = \frac{\partial^2 \widetilde{\mathcal{T}}_\omega(\boldsymbol{x},\boldsymbol{y})}{\partial x_i^2} \sim \sum_{e\in E_i} \frac{w_e}{|e|-1}, \quad \forall i \in \mathcal{V}, \tag{6a}$$

$$\mathcal{P}_i^{(t)} \sim \max\left(1, \left[\mathcal{P}_i^W + \sum_{s\in S} \alpha_s^{(t)} \lambda_s^{(t)} \mathcal{A}_i^s\right]^{-1}\right), \tag{6b}$$

In the above equation, $\mathcal{V}$ denotes the set of instances, $E_i$ denotes the nets incident to instance $i \in \mathcal{V}$, $w_e \in \omega$ denotes the weight of net $e$, and $\mathcal{P}^W$ denotes the second-order derivative of the wirelength term. To optimize the model, we provide the optimizer with the preconditioned gradient $\hat{\nabla}\mathcal{L}^{(t)} = \nabla\mathcal{L}^{(t)} \odot \mathcal{P}^{(t)}$.

As shown in Fig. 6, after the loss surfaces have been preconditioned, they are now more isotropic and therefore can be optimized more rapidly. The intuition from the partial derivative itself is that we would expect that for $\mathcal{P}_i^W$ instances with more pins or pins incident to larger net weights, they would move slower than instances with fewer pins, and for instances with larger $\sum_{s\in S} \alpha_s^{(t)} \lambda_s^{(t)} \mathcal{A}_i^s$ would also move slower.

It has been observed that if the ratio of the gradient norms from the density term and the wirelength term becomes too large, the optimization can diverge easily [13], [36]. Thus, we introduce an additional weighting vector $\boldsymbol{\alpha} \in \mathbb{R}^{|S|}$, so that we can dynamically control the gradient norm ratio. It is illustrated in Fig. 6 that some instances are dominated

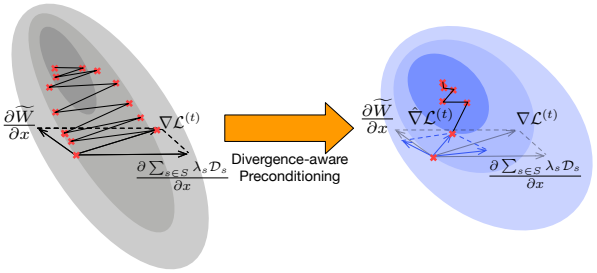



Fig. 6: It is shown in the figure how preconditioning takes place at iteration $t$. A preconditioning technique translates objection surfaces into a more isotropic form, and this leads to a reduction in iterations and a stabilization of the optimization process as a result.

by the density gradient, resulting in some instances moving too fast, and this causes the instances to diverge from each other. In most cases, this occurs during the second half of the placement iteration when the density term starts to compete with the wirelength term by increasing $\lambda$ to maintain its position in front of it. In order to stabilize the optimization process, we need a new preconditioner. For convenience, we define two auxiliary variables $\boldsymbol{\vartheta}^{(t)}$ and $\overline{\mathcal{P}}^{W}$. Taking the gradient norms of the density and wirelength terms to be equal, we can derive $\boldsymbol{\alpha}$ as follows.

$$\vartheta_s^{(t)} = \max\left(1, \frac{\nabla \mathcal{D}_s}{\sum_{i\in\mathcal{V}_s^r} |\partial \widetilde{\mathcal{T}}_\omega / \partial x_i|}\right), \quad \forall s \in S, \tag{7a}$$

$$\overline{\mathcal{P}}_s^W = \frac{\sum_{i\in\mathcal{V}_s^r} \mathcal{P}_i^W}{|\mathcal{V}_s^r|}, \quad \forall s \in S, \tag{7b}$$

$$\boldsymbol{\alpha}^{(t)} = \boldsymbol{\vartheta}^{(t)} \odot \overline{\boldsymbol{\mathcal{P}}}^{W}, \tag{7c}$$

$\mathcal{V}_s^r$ denotes the set of instances that have a demand in the field $s$. $\sum_{i\in\mathcal{V}_s^r} |\partial \widetilde{\mathcal{T}}_\omega / \partial x_i|$ 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. The detailed derivations have been omitted for brevity. Experiments will be conducted to further validate its effectiveness.

### D. Iterative Carry Chain Alignment Correction

we adopt the idea of *macro shredding* [25], [26] and propose a carry chain alignment technique that can better align the carry chains without affecting the effectiveness of the analytical global placement algorithm. We move the sequential CARRYs together at the end of each global placement iteration, align them based on their horizontal coordinates, and then move them into a column shape at the end of an iteration. The CARRY instances in a chain will move together during the global placement iterations, which eases the legalization step, since the chains will almost align once the global placement process is completed.

### E. Clock Network Planning Algorithm

There is a high degree of dissmoothness in the clock constraints in Section II-A3. The slightest movement within the clock region boundaries can result in an illegal clock configuration, which is detrimental to optimization. In order to simplify the clock planning process, we decompose it into two stages. Our first step is to find an *instance-to-clock-region mapping* in the first stage. This mapping ensures that all clock region constraints will be satisfied as long as all instances are located within the target clock region during the mapping. The second step involves moving all instances to their target clock regions by adding a penalty term to the placement objective, which is $\Gamma(\cdot)$. Following the global placement, the half-column constraints are then dealt with in the developed clock-aware direct legalization and detailed placement algorithm [8]. As we move forward, we will explain these two stages in more detail.

*1) Instance-to-Clock-Region Mapping Generation:* It is the goal of this step to generate mappings in a manner that ensures that clock constraints can be met with the minimum perturbation to the placement. As discussed in [10], we propose using *branch-and-bound method* to search through the solution space arising from different instance-to-clock-region mappings, and find a feasible solution with high quality within the solution space. As part of the second stage, we enlist the assignment that has the lowest cost and uses it as a base.

*2) The Clock Penalty for Placement:* Unlike the approach taken in [9], [11] that compels instances to move directly to their respective clock regions, we employ a bowl-shaped, smooth, and differentiable gravitational attraction term like [15], [37]. This term effectively draws instances toward their respective clock regions while adhering to their mapped locations. Let $lo_i^x$, $hi_i^x$, $lo_i^y$, and $hi_i^y$ be the left, right, bottom, and top boundary coordinates of the generated mapping result of instance $i$'. 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,

$$\Gamma(\boldsymbol{x}_i, \boldsymbol{y}_i)^x = \begin{cases} (\boldsymbol{x}_i - lo_i^x)^2, & \boldsymbol{x}_i < lo_i^x, \\ 0, & lo_i^x \le \boldsymbol{x}_i \le hi_i^x, \\ (\boldsymbol{x}_i - hi_i^x)^2, & hi_i^x < \boldsymbol{x}_i. \end{cases} \tag{8}$$

A visual representation of the clock penalty term can be found in Fig. 7. $\Gamma(\boldsymbol{x}, \boldsymbol{y})$ in Eq. (3) indicates that the sum of the clock penalty of all instances, i.e., $\Gamma(\boldsymbol{x}, \boldsymbol{y})$ is equal to $\sum_{i\in\mathcal{V}} \Gamma_i(x_i, y_i)$.

Initially, the clock penalty multiplier $\eta$ is set to a value of 0. As soon as we reset the clock penalty function $\Gamma(\cdot)$, we update $\eta$ with the relative ratio between the gradient norms of the wirelength and the clock penalty in order to maintain the stability of the clock penalty function.

$$\eta = \frac{\iota \|\nabla \widetilde{\mathcal{T}}_\omega\|_1}{\|\nabla \Gamma\|_1 + \varepsilon}. \tag{9}$$

As we observe, only 1% of the instances are out of their available clock regions right after they have been assigned the clock region, so most instances have no penalties as a

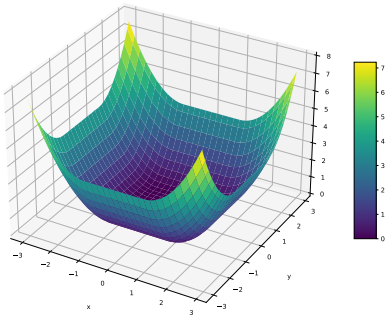


Fig. 7: The Visualization of clock penalty function $\Gamma_i(\cdot)$ for a single instance.

result of this. It is empirically established that $\iota$ is equal to $10^{-4}$ and $\varepsilon$ is equal to $10^{-2}$ as a method of balancing the gradient norm ratio.

### F. Timing-Criticality-based Weighting Method

The timing performance objective $\widetilde{\mathcal{T}}_{\boldsymbol{\omega}}(\cdot)$ (see Eq. (3)) consists of two components: i) the wirelength objective as a first-order approximation of WNS and TNS, and ii) the net criticality $\boldsymbol{\omega}$ that remedies the lack of timing information for first-order approximation.

$$\widetilde{\mathcal{T}}_{\boldsymbol{\omega}}(\boldsymbol{x},\boldsymbol{y}) = \sum_{e\in E} \omega_e \cdot \widetilde{W}(e) \tag{10}$$

$\omega_e \in \boldsymbol{\omega}$ measures the timing criticality of original nets, i.e., whether any timing path through nets violates timing constraints, what is the degree of the violation, and how much the path delay can be reduced. In this section we detail how $\widetilde{\mathcal{T}}(\cdot)$ functions.

*1) Static Timing Analysis:* Timing-driven placement leverages the static timing analysis (STA) to evaluate the timing criticality of nets. STA relies on (i) model of signal delays for nets and instances, and (ii) a timing analysis engine based on net and instance delays.

Delay models in FPGA are highly coupled with the pre-fabricated routing architecture and the behavior of the FPGA router. In our experiments, we adopt industrial routing architecture and timing data, and route the placed solutions by [38]. The delay model is obtained from the precise timing data and is the modeling of the FPGA routing mechanism. Specifically, on our target device, the FPGA routing mechanism uses a *multi-length routing segment mechanism* [39], with 1×, 2×, 4×, 8× routing segments, etc. The $k$-× value represents the length of the routing segment spanned across $k$ CLBs and is associated with a delay derived from the vendor's internal data. The actual timing delay is the sum of the signal propagation delays across routing segments of different lengths and the delays associated within the CLBs, as follows:

$$d_{s,t} = \sum_{k\in M} a_k d_k + \epsilon_s + \epsilon_t, \tag{11}$$

where $M$ is the collection of routing segments with various lengths; $a_k$ represents the number of $k$-× routing segments crossed by the signal along the path from the starting point $s$ to the endpoint $t$; $d_k$ is the delay associated with the $k$-× routing segment; $\epsilon_s$ and $\epsilon_d$ represent the delays associated with the CLBs at the starting point $s$ and the endpoint $t$, respectively. Note that the delay model and our proposed method are orthogonal, and our optimization strategy can be adapted to different delay models associated with different FPGA architectures.

Given the delay of the timing edges, the timing analysis engine determines which timing paths violate their timing constraints. A timing path $\pi$ is a directed acyclic path for particular source and sink pairs (primary I/Os and I/Os of store elements). The delay $t_\pi$ along a path $\pi$ is the sum of wire delays and cell delays, and every path comes with a timing constraint $c_\pi$ defined via the *actual arrival time (AAT)* and *required arrival time (RAT)* for every driver pin and primary output.

The slack $s_\pi$ of a path $\pi$ is defined as $s_\pi = c_\pi - t_\pi$. We mainly focus on the *setup time* constraint, which is defined as the difference between the AAT and RAT. A timing path $\pi$ violates the setup time constraints if the slack is negative. The slack of a timing edge is the smallest path slack among the paths containing this edge. To avoid enumerating all paths, we compute the slack from the actual arrival times and required arrival times at timing endpoints [40].

$$T_{ATT}(v_i) = \max_{v_j\in fanin(v_i)} T_{ATT}(v_j) + e_{j,i} \tag{12a}$$

$$T_{ATT}(v_i) = \min_{v_j\in fanout(v_i)} T_{RAT}(v_j) - e_{i,j} \tag{12b}$$

In Eq. (12), $v_i$ is an endpoint in the timing graph, and $e_{i,j}$ denotes the timing delays from endpoint $i$ to endpoint $j$, provided by the delay model. The slack of timing edge $s_{i,j}$ connecting the source endpoint $v_i$ and the sink endpoint $v_j$ is

$$s_{i,j} = T_{RAT}(v_j) - T_{ATT}(v_i) - e_{i,j}, \tag{13}$$

Fig. 8 illustrates the timing analysis process.

*2) Timing-driven Net Reweighting Scheme:* In the timing graph, nets have diverse effects on the timing slacks. Those nets with higher timing criticality should be more sensitive to the timing closure. To improve the timing in the analytical placement framework, we assign different net weights to different nets based on their timing criticality. Let $s_e$ and $s_{wns}$ denote the timing slacks of net $e$ and the worst negative slack, respectively. We define the timing criticality $c_e$ as follows,

$$c_e = \frac{\min(0, s_e)}{\min(0, s_{wns}) - T} \in [0, 1), \tag{14}$$

where $T$ is the clock period. When net $e$ is not on a path with timing violation, i.e. $s_e \geq 0$, the timing criticality $c_e$ remains zero. Otherwise, the timing criticality equals the ratio $\frac{|s_e|}{|s_{wns}|+T}$. The worse timing slack $|s_e|$ is, the larger timing criticality $c_e$ it will have. The largest timing criticality falls upon the nets on the most critical path.

After evaluating the timing criticality, we compute the net weight $w_e \in \boldsymbol{\omega}$ as

$$\beta_e = \alpha \cdot \max\left(1, \exp(c_e)\right) \tag{15a}$$

$$\omega'_e \leftarrow \omega_e \cdot \beta_e \tag{15b}$$

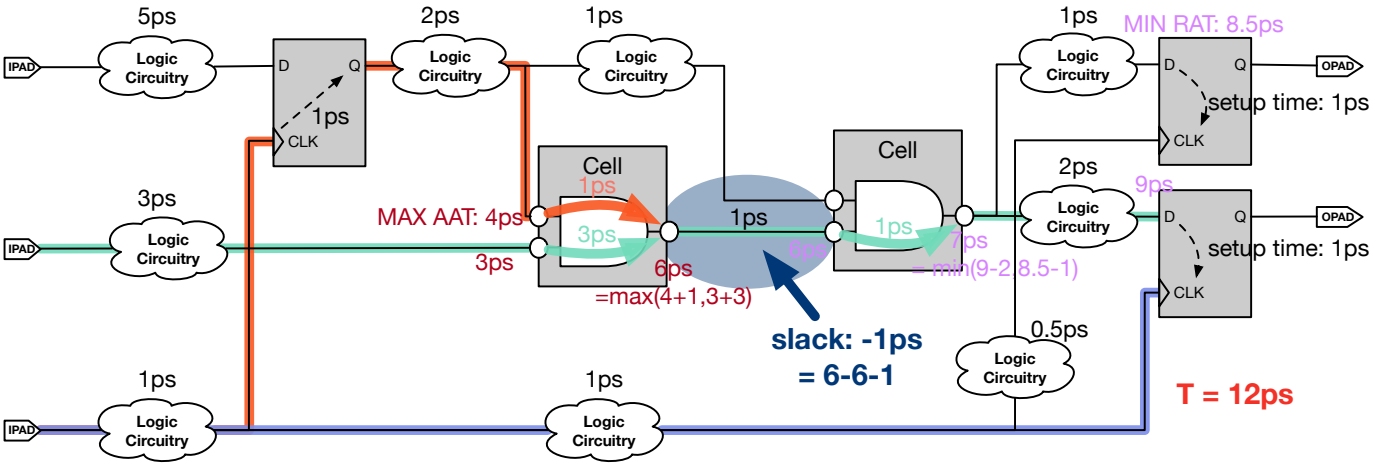



Fig. 8: An illustration for the static timing analysis on FPGA. Support the clock period is $T$ and the signal arrival time at primary input ports has minor skew. The required arrival time at the sink of a timing path is the clock period minus the clock delay at the capture FFs and the setup time of FFs (1ps in our example), and the required arrival time of a timing endpoint is determined by the required arrival time of its fanout endpoints and the fanout edge delays. The actual arrival time of a timing endpoint is the maximal accumulated delay from the primary inputs. The slack of a timing edge is the difference between the required arrival time at its sink, and the summation of the actual arrival time at its source and its edge delay.

where $\omega'_e$ is the updated net weight, $\beta_e$ denotes the reweighting magnitude for net $e$, and $\alpha$ is a hyper-parameter that controls the weighting magnitude [4].

*3) Effectiveness Analysis of the Reweighting Scheme:* It is suggested that our reweighting scheme can control the reweighting scale according to the severity of the target timing constraints. The reweighting magnitude $\beta_e$ of a net is determined by the largest magnitude of the timing paths that pass through it, i.e., $\beta_e = \max_{\pi \ni e} \beta_\pi$. We then gave a brief analysis of how the timing path $\pi$ affects the re-weighting scheme in nets.

For the paths with maximal delay $d_{wns}$, the intuition is that the reweighting method should make no effect, i.e., $\beta_{wns}$ equals 1, when it has no timing violations. Otherwise, the smaller $T - d_{wns}$ is, the larger the reweighting magnitude should be. Fig. 9a show the relationship between the magnitude of the reweighting $\beta_{wns}$ and the clock period $T$ for a certain path and its delay. This figure demonstrates that a path will only gain a reweighting magnitude greater than one when the target clock period $T$ is less than its path delay. The reweighting magnitude will grow faster with $T$ becoming smaller. Fig. 9b regards the path delay as a variable and shows how the reweighting magnitude $\beta_{wns}$ reacts under a certain clock period. The relationship curves align with our intuition that nets with a smaller timing delay always gain a smaller reweighting magnitude. No matter what the clock period is, the reweighting magnitude only takes effect on those paths with timing violations.

Nets on critical paths gain a larger weighting magnitude in this stage. In the subsequent analytical placement step, the increased net weights then help critical paths tilt more toward timing closure in the tradeoff between timing and area.

[4]In our experiments, $\alpha$ equals 1.

## IV. Experimental Results

We implemented our GPU-accelerated placer in C++ and Python along with the open-source machine learning framework Pytorch for fast gradient back propagation [41]. We conduct experiments on a Ubuntu 18.04 LTS platform that consists of an Intel(R) Xeon(R) Silver 4214 CPU @ 2.20GHz (24 cores), one NVIDIA TITAN GPU, and 251GB memory. We demonstrate the effectiveness and efficiency of our proposed algorithm on both academic benchmarks [17] and industrial benchmarks from the three most concerning aspects of routed wirelength (RWL), runtime (RT), and timing. The larger-scale ISPD2016 and ISPD2017 academic benchmarks can be used to evaluate the efficiency of our proposed algorithm. Meanwhile, industry benchmarks offer a wider variety of instances and routers with timing information, providing useful data to verify the effectiveness of our proposed algorithm. Currently, we only support one clock domain due to the limitation of the timing analysis engine, while the timing optimization strategy is general. In the future, we plan to extend the timing analysis engine to support multi-clock domains. Thanks again for your valuable comment and for your contribution to improving our research.

### A. Evaluation on Academic Benchmarks

*1) ISPD 2016 Academic Benchmark:* TABLE II summarizes the statistics of ISPD 2016 benchmarks as well as the the comparison with the state-of-the-art placers, It is worth noting that clock constraints were not included in ISPD 2016 benchmarks. Additionally, we also provide the CPU version's results `Ours (CPU)` to ensure a fair comparison from an algorithmic standpoint. We compare the routed wirelength reported by patched Xilinx Vivado v2015.4 and placement runtime with five state-of-the-art placers: `UTPlaceF` [5], `RippleFPGA` [28], `GPlace 3.0` [29], `UTPlaceF-NEP` [8], and `elfPlace (GPU)` [13].

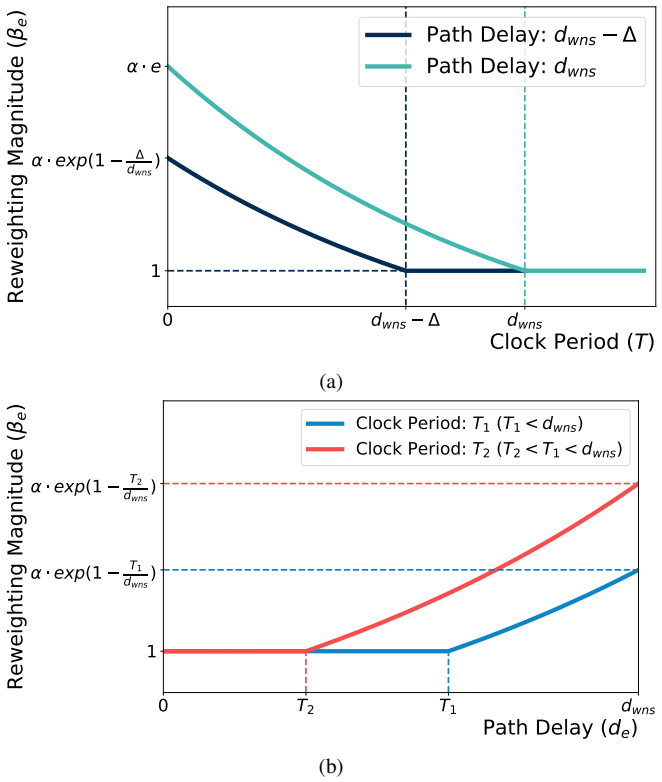



Fig. 9: (a) This figure illustrates the relationship between the reweighting magnitude $\beta_e$ and the clock period $T$ given a certain path and its delay. In particular, the path with delay $d_{wns}$ represents the most critical path. This figure also depicts another path with delay $d_{wns} - \Delta$, where $\Delta$ measures how much the delay is smaller the worst-case path $d_{wns}$. (b) This figure shows the relationship between the reweighting magnitude $\beta_e$ and the path delay $d_e$ given a clock period. This figure plots two curves for two certain clock periods $T_1$ and $T_2$ ($T_2 < T_1 < d_{wns}$).

We can see that our placer consistently achieves better routed wirelength than other placers, i.e., 13.8% smaller than `UTPlaceF`, 10.5% smaller than `RippleFPGA`, 9.1% smaller than `GPlace 3.0`, 7.3% smaller than `UTPlaceF-NEP`, 0.3% smaller than `elfPlace (GPU)` on average, respectively. Meanwhile, our placer is 8.21× faster than `UTPlaceF`, 2.43× faster than `RippleFPGA`, 6.04× faster than `GPlace 3.0` and 2.13× faster than `UTPlaceF-NEP`. Despite both algorithms utilizing GPU acceleration, our algorithm runs 10.5% slower than `elfPlace (GPU)`.

*2) ISPD 2017 Academic Benchmark:* The statistics of the ISPD 2017 academic benchmark are summarized in TABLE III. The number of instances varies from 400K to 900K with 32–58 clock nets. We do not evaluate the timing performance because on this benchmark we have no access to the timing information of the device. We compare the routed wirelength reported by patched Xilinx Vivado v2016.4 and placement runtime with four state-of-the-art placers, `UTPlaceF 2.0` [9], `RippleFPGA` [11], `UTPlaceF 2.X` [10], and `NTUfplace` [15]. All the results of these placers are from their original placers. We do not compare the results with `elfPlace` because its algorithm cannot handle clock constraints (see TABLE I).

The experimental results show that our placement algorithm consistently achieves better routed wirelengths than other placers. Specifically, our placer achieves 14.2% smaller routed wirelength 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. For some benchmarks, like `CLK-FPGA06`, which is a large design with about 925K cells and 58 clock nets, our routed wirelength is even 13.6%, 6.7%, 10.5% and 4.1% better than that of baseline placers, respectively. It needs to be mentioned that `UTPlaceF 2.X` is a follow-up work to `UTPlaceF 2.0`, which relaxes the clock region bounding box constraints to clock tree constraints. It allows for a larger solution space for clock routing feasibility and thus should yield better results. However, even in this unfair comparison, our routed wirelength is still 9.6% better than `UTPlaceF 2.X`, exhibiting the efficacy of our algorithm. Besides, our GPU-accelerated placer is the fastest one with 4.94×, 2.38×, 1.45×, and 6.58× speedup over other placers, respectively. These experiments demonstrate the effectiveness and efficiency of our proposed algorithms.

*3) CPU vs. GPU Acceleration:* In comparison to the CPU version, our GPU version achieved similar speedup ratios and obtained the same routed wirelength on both ISPD2016 and ISPD2017 benchmarks. Specifically, our algorithm achieved a 5.98× speedup on ISPD2016 benchmarks and a 5.19× speedup on ISPD2017 benchmarks. On the other hand, the CPU version only achieves a 1.37× speedup compared to `UTPlaceF` on ISPD2016 benchmarks and is slower than other CPU-based placers. Similarly, on ISPD2017 benchmarks, the CPU version was slower than other quadratic placement algorithms. The reason behind this is that non-convex optimization-based algorithms (see TABLE I) require greater computation. Nevertheless, compared to `NTUfplace`, which is also a non-convex optimization-based algorithm, our algorithm achieved a 1.27× speedup, indicating that our algorithm has advantage in both terms of speed and placement quality within the same algorithm category.

Overall, the experimental results suggest that while non-convex optimization-based placement algorithms tend to produce better results, they require greater computation amounts. However, non-convex optimization-based placement algorithms are highly amenable to parallel acceleration and can be drastically sped up through heterogeneous parallelization [32].

### B. Evaluation on Industry Benchmarks

We further evaluate our placer on industrial benchmarks which consist of a comprehensive instance set and an industrial FPGA architecture from real-world industry scenarios, including SHIFT, distributed RAM, and CARRY (see TABLE IV). Most previous FPGA placers [5], [7], [9]–[11], [13], [15], [21], [28], [29] cannot fully handle such an instance

TABLE II: Routed Wirelength ($\times 10^3$) and Runtime (Seconds) Comparison on ISPD 2016 Benchmarks.

| Design | #LUT/#FF/#BRAM/#DSP | UTPlaceF [5] | | RippleFPGA [28] | | GPlace 3.0 [29] | | UTPlaceF-NEP [8] | | elfPlace (GPU) [13] | | Ours (CPU) | | Ours (GPU) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | RWL | RT | RWL | RT | RWL | RT | RWL | RT | RWL | RT | RWL | RT | RWL | RT |
| FPGA01 | 50K/55K/0/0 | 357 | 162 | 350 | 32 | 356 | 83 | 340 | 58 | 317 | 30 | 322 | 386 | 322 | 34 |
| FPGA02 | 100K/66K/100/100 | 642 | 273 | 682 | 58 | 644 | 158 | 653 | 89 | 581 | 45 | 582 | 655 | 582 | 51 |
| FPGA03 | 250K/170K/600/500 | 3215 | 778 | 3251 | 209 | 3101 | 587 | 3139 | 336 | 2865 | 110 | 2869 | 690 | 2865 | 113 |
| FPGA04 | 250K/172K/600/500 | 5410 | 768 | 5492 | 286 | 5403 | 630 | 5331 | 349 | 4853 | 107 | 4878 | 660 | 4882 | 109 |
| FPGA05 | 250K/174K/600/500 | 9660 | 973 | 9909 | 334 | 10507 | 736 | 10045 | 381 | 9206 | 111 | 9198 | 807 | 9187 | 119 |
| FPGA06 | 350K/352K/1000/600 | 6488 | 1649 | 6145 | 518 | 5820 | 1189 | 5801 | 596 | 5699 | 201 | 5724 | 896 | 5726 | 230 |
| FPGA07 | 350K/355K/1000/600 | 10105 | 1647 | 9577 | 558 | 9509 | 1277 | 9356 | 597 | 8740 | 195 | 8633 | 844 | 8620 | 207 |
| FPGA08 | 500K/216K/600/500 | 7879 | 1642 | 8088 | 412 | 8126 | 1400 | 8298 | 273 | 7676 | 157 | 7412 | 838 | 7405 | 174 |
| FPGA09 | 500K/366K/1000/600 | 12369 | 2483 | 11376 | 662 | 11711 | 1848 | 11633 | 346 | 10650 | 214 | 10626 | 943 | 10622 | 251 |
| FPGA10 | 350K/600K/1000/600 | 8795 | 3043 | 6972 | 1002 | 6836 | 1794 | 6317 | 353 | 6068 | 228 | 5991 | 990 | 5984 | 256 |
| FPGA11 | 480K/363K/1000/400 | 10196 | 2044 | 10918 | 628 | 10260 | 1709 | 10476 | 309 | 10432 | 178 | 10409 | 930 | 10414 | 209 |
| FPGA12 | 500K/602K/600/500 | 7755 | 2934 | 7240 | 847 | 7224 | 2263 | 6835 | 418 | 6484 | 223 | 6548 | 1075 | 6546 | 286 |
| Ratio | | 1.138 | 8.210 | 1.105 | 2.430 | 1.091 | 6.040 | 1.073 | 2.128 | 1.003 | 0.895 | 1.000 | 5.975 | 1.000 | 1.000 |

* Results for other placers are from the most recent publication [13], collected from a Linux machine with Intel Xeon Gold 6230 CPU (2.10GHz and 20 physical cores) and 64 GiB RAM.

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

| Design | #LUT/#FF/#BRAM/#DSP | #Clock | UTPlaceF 2.0 [9] | | RippleFPGA [11] | | UTPlaceF 2.X [10] | | NTUfplace [15] | | Ours (CPU) | | Ours (GPU) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | RWL | RT | RWL | RT | RWL | RT | RWL | RT | RWL | RT | RWL | RT |
| CLK-FPGA01 | 211K/324K/164/75 | 32 | 2208 | 532 | 2011 | 288 | 2092 | 180 | 2039 | 698 | 1867 | 864 | 1868 | 136 |
| CLK-FPGA02 | 230K/280K/236/112 | 35 | 2279 | 513 | 2168 | 266 | 2194 | 179 | 2149 | 710 | 2013 | 782 | 2011 | 130 |
| CLK-FPGA03 | 410K/481K/850/395 | 57 | 5353 | 1039 | 5265 | 583 | 5109 | 343 | 4901 | 1704 | 4756 | 963 | 4755 | 215 |
| CLK-FPGA04 | 309K/372K/467/224 | 44 | 3698 | 711 | 3607 | 380 | 3600 | 242 | 3614 | 1148 | 3334 | 860 | 3338 | 162 |
| CLK-FPGA05 | 393K/469K/798/150 | 56 | 4692 | 939 | 4660 | 569 | 4556 | 323 | 4417 | 1540 | 4158 | 962 | 4154 | 208 |
| CLK-FPGA06 | 425K/511K/872/420 | 58 | 5589 | 1066 | 5737 | 591 | 5432 | 346 | 5122 | 2210 | 4920 | 988 | 4918 | 229 |
| CLK-FPGA07 | 254K/309K/313/149 | 38 | 2444 | 845 | 2326 | 304 | 2324 | 201 | 2320 | 795 | 2144 | 795 | 2145 | 141 |
| CLK-FPGA08 | 212K/257K/161/75 | 32 | 1886 | 529 | 1778 | 247 | 1807 | 169 | 1803 | 588 | 1648 | 672 | 1648 | 120 |
| CLK-FPGA09 | 231K/358K/236/112 | 35 | 2601 | 842 | 2530 | 327 | 2507 | 197 | 2436 | 717 | 2250 | 807 | 2248 | 144 |
| CLK-FPGA10 | 327K/506K/542/255 | 47 | 4464 | 974 | 4496 | 512 | 4229 | 286 | 4339 | 1597 | 3836 | 930 | 3839 | 200 |
| CLK-FPGA11 | 300K/468K/454/224 | 44 | 4183 | 1068 | 4190 | 455 | 3936 | 265 | 3964 | 1618 | 3629 | 897 | 3626 | 183 |
| CLK-FPGA12 | 277K/430K/389/187 | 41 | 3369 | 774 | 3388 | 409 | 3236 | 247 | 3179 | 849 | 2936 | 862 | 2938 | 168 |
| CLK-FPGA13 | 339K/405K/570/262 | 47 | 3816 | 1172 | 3833 | 441 | 3723 | 270 | 3680 | 985 | 3403 | 880 | 3404 | 181 |
| Ratio | | | 1.142 | 4.945 | 1.117 | 2.380 | 1.096 | 1.453 | 1.079 | 6.577 | 1.000 | 5.191 | 1.000 | 1.000 |

TABLE IV: Routed Wirelength ($\times 10^3$), Maximum Frequency (MHz), WNS($\times 10^3$ps), TNS ($\times 10^5$ps) and Runtime (Seconds) Comparison between Conference Version and Our Algorithm on Industry Benchmarks.

| Design | #LUT/#FF /#BRAM/#DSP | #Distributed RAM+#SHIFT | #CARRY | Clock Period (ns) | Conference Version [37] | | | | | Ours-dl (GPU) | | | | | Ours (GPU) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | RWL | fMAX | WNS | TNS | RT | RWL | fMAX | WNS | TNS | RT | RWL | fMAX | WNS | TNS | RT |
| IND01 | 17K/11K/0/13 | 9 | 2K | 5 | 90 | 111.84 | -3.94 | -2.42 | 45 | 93 | 115.14 | -3.69 | -3.85 | 72 | 90 | 148.13 | -1.75 | -1.32 | 70 |
| IND02 | 11K/10K/0/24 | 6 | 335 | 2 | 102 | 160.26 | -4.24 | -19.73 | 54 | 150 | 107.28 | -7.32 | -30.68 | 169 | 117 | 202.51 | -2.94 | -18.15 | 76 |
| IND03 | 109K/12K/0/0 | 0 | 0 | 3 | 1028 | 176.49 | -2.67 | -18.47 | 59 | 1064 | 169.49 | -2.90 | -10.90 | 99 | 1031 | 173.49 | -2.76 | -16.84 | 144 |
| IND04 | 29K/17K/0/16 | 218 | 1K | 5 | 279 | 96.16 | -5.40 | -27.92 | 83 | 332 | 101.60 | -4.84 | -16.00 | 68 | 283 | 87.86 | -6.38 | -21.11 | 88 |
| IND05 | 64K/191K/64/928 | 29K | 4K | 10 | 2305 | 49.25 | -10.31 | -3.01 | 97 | 2463 | 56.54 | -7.69 | -10.64 | 103 | 2312 | 68.69 | -4.56 | -2.35 | 218 |
| IND06 | 112K/65K/21/0 | 0 | 4K | 15 | 1585 | 38.48 | -10.99 | -106.38 | 84 | 1749 | 38.45 | -11.01 | -319.81 | 75 | 1585 | 46.51 | -6.50 | -51.92 | 193 |
| IND07 | 40K/156K/89/768 | 26K | 3K | 4 | 1498 | 97.07 | -6.30 | -20.59 | 83 | 1471 | 105.46 | -5.48 | -12.18 | 123 | 1505 | 99.61 | -6.04 | -21.03 | 265 |
| Ratio | | | | | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.13 | 0.99 | 1.04 | 1.63 | 1.52 | 1.02 | 1.16 | 0.76 | 0.78 | 2.03 |

TABLE V: Routed Wirelength($\times 10^3$), Maximum Frequency (MHz), WNS ($\times 10^3$ ps), TNS ($\times 10^5$ ps), and Runtime (Seconds) Comparison with Different Techniques on Industry Benchmarks.

| Design | w/o precond or chain align (GPU) | | | | | w/o precond (GPU) | | | | | w/o chain align (GPU) | | | | | Ours (GPU) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | RWL | fMAX | WNS | TNS | RT | RWL | fMAX | WNS | TNS | RT | RWL | fMAX | WNS | TNS | RT | RWL | fMAX | WNS | TNS | RT |
| IND01 | 103 | 117.77 | -3.491 | -2.849 | 53 | 94 | 124.36 | -3.041 | -1.826 | 66 | 108 | 131.91 | -2.581 | -143.009 | 74 | 90 | 148.13 | -1.751 | -1.323 | 70 |
| IND02 | 124 | 151.98 | -4.58 | -28.426 | 148 | 180 | 146.48 | -4.827 | -43.93 | 64 | 118 | 118.36 | -6.449 | -6029.39 | 73 | 117 | 202.51 | -2.938 | -18.148 | 76 |
| IND03 | 1021 | 164.47 | -3.08 | -17.301 | 123 | 1021 | 164.47 | -3.08 | -17.301 | 126 | 1030 | 178.44 | -2.604 | -1776.63 | 125 | 1031 | 173.49 | -2.764 | -16.836 | 144 |
| IND04 | 377 | 69.52 | -9.384 | -28.074 | 88 | 392 | 72.42 | -8.808 | -31.798 | 90 | 290 | 107.67 | -4.288 | -610.918 | 76 | 283 | 87.86 | -6.382 | -21.112 | 88 |
| IND05 | 2290 | 68.96 | -4.502 | -3.702 | 137 | diverge | diverge | diverge | diverge | diverge | 2260 | 67.25 | -4.87 | -238.678 | 153 | 2312 | 68.69 | -4.558 | -2.354 | 218 |
| IND06 | 1558 | 15.23 | -50.647 | -105.558 | 109 | 1576 | 37.81 | -11.45 | -113.733 | 107 | 1580 | 32.28 | -15.977 | -5320.1 | 114 | 1585 | 46.51 | -6.502 | -51.922 | 193 |
| IND07 | 1547 | 74.24 | -9.47 | -31.747 | 186 | 1393 | 82.34 | -8.145 | -20.816 | 155 | diverge | diverge | diverge | diverge | diverge | 1505 | 99.61 | -6.039 | -21.03 | 265 |
| Ratio | 1.076 | 0.766 | 2.355 | 1.599 | 0.922 | 1.147 | 0.829 | 1.497 | 1.586 | 0.801 | 1.034 | 0.900 | 1.468 | 1.298 | 0.837 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |

set with SLICEL-SLICEM heterogeneity. We also conducted experiments to evaluate the method of adding additional driver-to-load nets, which we named Ours-dl (GPU). Ours-dl (GPU) follows the same net weighting scheme defined by Eq. (15) in [21]. To better validate the effectiveness of our placer, we leverage a high-quality FPGA router [38] to evaluate the placement algorithms more precisely.

From TABLE IV, we can see the comparison between the conference version [37], Ours-dl (GPU) and our placer. Our placer can achieve 16% better fMAX, 23.6% better WNS and 22.5% better TNS, respectively, with minor routed wirelength degeneration, exhibiting the effectiveness of our placer. In some benchmarks, such as IND06, which is one of the most congestion benchmarks, our fMAX, WNS and TNS are 20.9%, 40.8% and 51.2% better than the baseline, respectively. These experiments demonstrate that our placer can effectively optimize timing even on congested benchmarks. The experiments also show that our placer requires more time to converge on industrial benchmarks. This is because optimizing timing requires additional iterations, which needs further optimization in the future. The difficulty in adjusting the priority of multiple objectives with varying wirelengths may be a contributing factor to these results. The experimental results also show that Ours-dl (GPU) has inferior performance compared to the conference version. The difficulty in adjusting the priority of multiple objectives with varying wirelengths may be a contributing factor to these results.

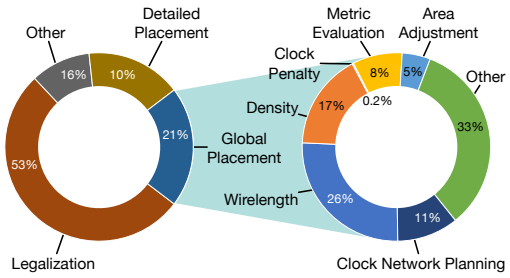



Fig. 10: Runtime breakdown on `CLK-FPGA13`. Similar distributions are observed on other benchmarks.

### C. Ablation Study for Optimization Techniques

To better understand the performance of our placer, we perform an ablation study and validate the effectiveness of our proposed methods by disabling optimization techniques as follows (see TABLE V).

- 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 [13], [36].
- Disable both techniques as the baseline.

We can see from TABLE V that both the iterative carry chain alignment correction technique and dynamically-adjusted precondition technique helps stabilizes the global placement convergence and enables better placement solution at convergence. Without either of these two techniques, the placer fails to converge on one design. The dynamically-adjusted preconditioner helps achieve 14.7% better routed wirelength, 17.1% fMAX, 49.7% better WNS, and 58.6% better TNS compared with the default preconditioner [13], [36]. Moreover, the iterative carry chain alignment correction helps optimize the routed wirelength, fMAX, WNS, and TNS by 3.4%, 10.0%, 46.8%, and 29.8% with minor runtime overhead. These experiments validate the effectiveness and efficacy of the proposed carry chain alignment and preconditioning technique.

### D. Runtime Breakdown

To analyze the time consumption of our placer, we further exhibit the runtime breakdown of our algorithm on one of the largest benchmarks as shown in Fig. 10. With the help of GPU acceleration, global placement is no longer the most time-consuming part. The core forward and backward propagation of the global placement, i.e., the computation of wirelength and electrostatic density, as well as their gradients, only take up 26% and 17% of the global placement runtime, respectively. With the great reduction in global placement runtime, legalization becomes the new runtime bottleneck, taking 53% of the total placement time. Meanwhile, other miscellaneous parts, including IOs, parsing, database establishment, etc., take up also the same time as that of global placement.

## V. Conclusion

In this paper, we present a heterogeneous FPGA placement algorithm that considers the heterogeneity of SLICELs and SLICEMs, as well as timing closure and clock feasibility. A new electrostatic formulation and a nested Lagrangian paradigm have been proposed to achieve uniform optimization of wirelength, routability, timing, and clock feasibility for heterogeneous instance types, including LUT, FF, BRAM, DSP, distributed RAM, SHIFT, and CARRY. Additionally, we propose a dynamically adjusted preconditioner, a timing-driven net-weighting scheme, and a smooth clock penalization technique in order to ensure that the placement is convergent to high-quality solutions. On the ISPD 2017 contest benchmark, experiments have revealed that our placer can achieve 14.2%, 11.7%, 9.6%, and 7.9% better routed wirelengths compared to the state-of-the-art placers, `UTPlaceF 2.0`, `RippleFPGA`, `UTPlaceF 2.X`, and `NTUfplace`, respectively, at 1.5-6× speedup leveraging GPU acceleration. We conducted experiments on industrial benchmarks to prove that our algorithms are capable of achieving 16% better fMAX, 23.6% better WNS and 22.5% better TNS with about 2% increase in routed wirelength.

## References


[1] S.-J. Lee and K. Raahemifar, "Fpga placement optimization methodology survey," in *Canadian Conference on Electrical and Computer Engineering (CCECE)*. IEEE, 2008, pp. 001 981–001 986.

[2] I. L. Markov, J. Hu, and M.-C. Kim, "Progress and challenges in VLSI placement research," *Proceedings of the IEEE*, vol. 103, no. 11, pp. 1985–2003, 2015.

[3] P. Maidee, C. Ababei, and K. Bazargan, "Timing-driven partitioning-based placement for island style fpgas," *IEEE TCAD*, vol. 24, no. 3, pp. 395–406, 2005.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.

[15] 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.

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

[17] 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.

[18] M.-C. Kim, N. Viswanathan, Z. Li, and C. Alpert, "ICCAD-2013 CAD contest in placement finishing and benchmark suite," in *Proc. ICCAD*, 2013, pp. 268–270.

[19] Ultrascale architecture clb slices. Accessed: 2023-10-02. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug574-ultrascale-clb.pdf

[20] T. Martin, D. Maarouf, Z. Abuowaimer, A. Alhyari, G. Gréwal, and S. Areibi, "A flat timing-driven placement flow for modern fpgas," in *Proc. DAC*. ACM, 2019, p. 4.

[21] Z. Lin, Y. Xie, G. Qian, J. Chen, S. Wang, J. Yu, and Y.-W. Chang, "Timing-driven placement for fpgas with heterogeneous architectures and clock constraints," in *Proc. DATE*. IEEE, 2021, pp. 1564–1569.

[22] A. Marquardt, V. Betz, and J. Rose, "Timing-driven placement for fpgas," in *Proc. FPGA*. ACM, 2000, pp. 203–213.

[23] C. M. Fuller, S. W. Gould, S. P. Hartman, E. E. Millham, and G. Yasar, "Field programmable gate arrays using semi-hard multicell macros," US Patent US5 761 078A, Jun., 1998.

[24] C. Lavin, M. Padilla, J. Lamprecht, P. Lundrigan, B. Nelson, and B. Hutchings, "HMFlow: Accelerating FPGA Compilation with Hard Macros for Rapid Prototyping," in *2011 IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines*. Salt Lake City, UT, USA: IEEE, May 2011, pp. 117–124.

[25] S. N. Adya and I. L. Markov, "Combinatorial techniques for mixed-size placement," *ACM Transactions on Design Automation of Electronic Systems*, vol. 10, no. 1, pp. 58–90, Jan. 2005.

[26] K. Doll, F. Johannes, and K. Antreich, "Iterative placement improvement by network flow methods," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 10, pp. 1189–1200, Oct./1994.

[27] P. Ochsendorf, "Timing-driven macro placement," Ph.D. dissertation, Universitäts-und Landesbibliothek Bonn, 2019.

[28] 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.

[29] 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.

[30] Ultrascale architecture clocking resources. Accessed: 2023-10-02. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug572-ultrascale-clocking.pdf

[31] 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.

[32] Y. Lin, S. Dhar, W. Li, H. Ren, B. Khailany, and D. Z. Pan, "DREAMPlace: Deep Learning Toolkit-Enabled GPU Acceleration for Modern VLSI Placement," in *Proc. DAC*, 2019, pp. 1–6.

[33] 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.

[34] A. B. Kahng, S. Mantik, and I. L. Markov, "Min-max placement for large-scale timing optimization," in *Proc. ISPD*, 2002, pp. 143–148.

[35] 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.

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

[37] J. Mai, Y. Meng, Z. Di, and Y. Lin, "Multi-electrostatic fpga placement considering slicel-slicem heterogeneity and clock feasibility," in *Proc. DAC*, 2022, pp. 649–654.

[38] J. Wang, J. Mai, Z. Di, and Y. Lin, "A robust fpga router with concurrent intra-clb rerouting," in *Proc. ASPDAC*, 2023, pp. 529–534.

[39] V. Betz and J. Rose, "FPGA routing architecture: Segmentation and buffering to optimize speed and density," in *Proc. FPGA*. Monterey California USA: ACM, Feb. 1999, pp. 59–68.

[40] T. Huang and M. D. F. Wong, "Opentimer: A high-performance timing analysis tool," in *Proc. ICCAD*. IEEE, 2015, pp. 895–902.

[41] 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.


**Jing Mai** received his B.S. degree in computer science from Peking University in 2021. He is currently a Ph.D. student in the School of Computer Science at Peking University. His research interests include machine learning-assisted EDA, MLSys, concurrency, and probabilistic modeling. He is currently working on FPGA placement and routing algorithm.

**Jiarui Wang** received his B.S. degree in Computer Science from Peking University in 2021. He is currently a Ph.D. student at Peking University. His research interest includes FPGA routing and clock tree synthesis.

**Zhixiong Di** (M'21) received the B.S. degree in microelectronics and solid electronics from Fuzhou University, Fuzhou, China, in 2007, and the Ph.D. degree in microelectronics and solid electronics from Xidian University, Xi'an, China, in 2014. He is currently an Associate Professor with the School of Information Science and Technology, Southwest Jiaotong University, Chengdu, China. His research interests include physical design algorithms and VLSI architecture design methodology.

**Yibo Lin** (S'16-M'19) received the B.S. degree in microelectronics from Shanghai Jiaotong University in 2013, and his Ph.D. degree from the Electrical and Computer Engineering Department of the University of Texas at Austin in 2018. He is current an assistant professor in the Computer Science Department associated with the Center for Energy-Efficient Computing and Applications at Peking University, China. His research interests include physical design, machine learning applications, GPU acceleration, and hardware security.