Timing-Driven Placement for FPGAs with Heterogeneous Architectures and Clock Constraints

Zhifeng Lin[1], Yanyue Xie[2], Gang Qian[3], Jianli Chen[1,3], Sifei Wang[3], Jun Yu[3] and Yao-Wen Chang[4]
[1]Center for Discrete Mathematics and Theoretical Computer Science, Fuzhou University, Fuzhou 350108, China
[2]Department of Electrical and Computer Engineering, Northeastern University, Boston, MA 02115, USA
[3]State Key Lab of ASIC & System, Fudan University, Shanghai 200433, China
[4]Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 10617, Taiwan
Email: chenjianli@fudan.edu.cn

***Abstract*—Modern FPGAs often contain heterogeneous architectures and clocking resources which must be considered to achieve desired solutions. As the design complexity keeps growing, placement has become critical for FPGA timing closure. In this paper, we present an analytical placement algorithm for heterogeneous FPGAs to optimize its worst slack and clock constraints simultaneously. First, a heterogeneity-aware and memory-friendly delay model is developed to accurately and rapidly assess each connection delay. Then, a two-stage clock region refinement method is presented to effectively resolve the clock and resource violations. Finally, we develop a novel timing-based co-optimization method to generate optimized placement without any clocking violations. Compared with the state-of-the-art placer based on the advanced commercial tool Xilinx Vivado 2019.1 with the Xilinx 7 Series FPGA architecture, our algorithm achieves the best worst slack and routed wirelength while satisfying all clock constraints.**

I. Introduction

Field-programmable gate arrays (FPGAs) are programmable platforms designed for user customization [6]. Compared with application-specific integrated circuits (ASICs), FPGAs achieve non-recurring costs, in-field programmability, and short turnaround time, and thus attract higher attention from circuit designers in recent years [2]. As the FPGA technologies keep improving, modern FPGAs often contain heterogeneous architectures and clock resources, which facilitate high-quality solutions for specific implementations, such as machine learning and data centers [11].

Placement plays an important role in the FPGA design flow because its performance highly affects the final wirelength optimization, timing closure, and power consumption [7]. The FPGA placement works can be generally categorized into the following three types: simulated annealing (SA), partition-based algorithms, and analytical formulations [8], [12]. Recent studies show that the analytical approaches achieve the best trade-off between quality and runtime, and are widely used in academia and industry [9].

With the continuous wire size shrinking in modern FPGAs, wiring-induced delays are becoming the main issue of the whole circuit delay. Traditional placers that address wirelength alone are no longer sufficient for timing closure, especially for the large-scale FPGAs. In this case, timing-driven placement is introduced to solve complex timing constraints while preserving desired performance. However, since the FPGA interconnect delays are highly non-linear and non-monotone with respect to geometric distance due to the significant delay with its programmable switches and segmented routing structures, the great difficulty in getting accurate delay information makes timing-driven placement much more challenging than ever [1].

Heterogeneous architecture is another major issue for FPGA placement. Different blocks such as CLBs, RAMs, and DSPs, must be placed on dedicated positions in an FPGA, which incurs non-convex objective and may lead to sub-optimal placements [10]. In addition, clock resources in modern FPGAs are of particular importance to achieve better performance. However, the resources also impose various layout constraints that affect the mapping of other logics, further complicating the already time-consuming placement process [5].

Existing works consider the crucial timing issue and the special architecture constraints separately. In [6], Lin *et al.* showed the importance of adopting a multilevel analytical framework to improve timing-driven placement scalability. In [9], Martin *et al.* optimized the timing cost by using a flat analytical model without explicit packing. In the work [3], Chen *et al.* presented a routing-architecture-aware cost function to minimize the inter-block delays and a complex block density model to deal with the heterogeneous components. Experimental results have shown that their proposed method achieves promising results on the Altera Stratix IV devices. However, all these placers overlook the critical clock constraints and may lead to timing failures.

In order to encourage placers to address large-scale FPGA designs with various clocking rules, the 2017 ISPD held a clock-aware FPGA placement contest [14]. The recent work [2] considered the contest problem, and developed an effective analytical placement algorithm. They designed a smooth quadratic region cost to reduce the overuse of clocking resources and achieved the best performance when compared with the top-3 winners of 2017 ISPD clock-aware FPGA placement contest. Unfortunately, due to the lack of timing optimization, the work may incur timing violations in their final placements.

In this paper, we propose a timing-driven FPGA placement flow to efficiently optimize timing and resolve clocking conflicts simultaneously. Our main contributions are summarized as follows:

- We propose a heterogeneous delay model to calculate each connection delay, and construct a memory-friendly look-up table to speed up delay access.
- We develop a two-stage clock region refinement method to solve clocking and resource violations while preserving good placement results.
- To achieve high-quality placement, we design a timing-based co-optimization scheme satisfying clocking constraints. We also derive a combinatorial legalization algorithm to remove all overlaps and further improve the solution quality.
- Compared with the commercial tool Xilinx Vivado 2019.1, experimental results show that our algorithm resolves all the timing violations and further improves the wirelength by 7.2% in 4.8% shorter runtime.

The remainder of this paper is organized as follows. Section II gives the preliminaries for heterogeneous FPGA placement. Section III presents our analytical placement algorithm and the optimization strategy. Experimental results are shown in Section IV. Finally, Section V concludes our work.

This work was partially supported by the National Key Research and Development Project under Grant 218YFB2202704, Natural Science Foundation of China under Grant 61977017, and by AnaGlobe, Maxeda, Synopsys, TSMC, and MOST of Taiwan under Grant Numbers 107-2221-E-002-161-MY3 and 108-2221-E-002-097-MY3.

978-3-9819263-5-4/DATE21/©2021 EDAA

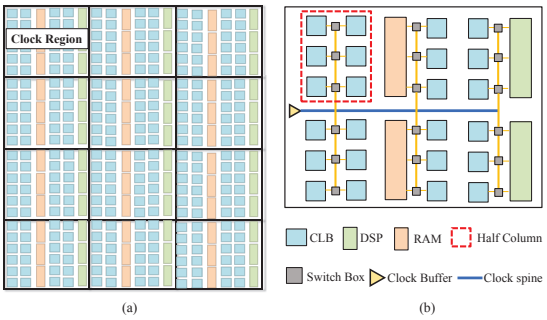

Fig. 1. An example FPGA architecture. (a) An FPGA contains CLBs, RAMs and DSPs and is divided into 4×3 clock regions. (b) An illustration of a clock region, which is partitioned into upper and lower half columns.

II. Preliminaries

In this section, we first introduce the heterogeneous FPGA architecture, and then give the problem formulation of heterogeneous FPGA placement.

A. Heterogeneous FPGA Architecture

In this paper, the target FPGA architectures are Xilinx 7 Series devices, which are optimized for low cost and high system performance, and are widely used in commercial applications [13]. The proposed techniques, however, are general and can easily apply to other heterogeneous FPGA architectures. Generally, there are different kinds of sites (CLB, RAM, and DSP) on the chip, and each block in a given netlist must be placed in sites of its own type.

The clocking architectures of FPGAs are similar to ASICs, in which clocks are routed from their clock sources to all of their loads through a pre-defined network, say a mesh-like one. In addition, the device is divided into a grid of clock regions, and each clock region is further partitioned into upper and lower half columns (as shown in Figure 1). A clock is in a clock region (half column) if the clock bounding box overlaps that clock region (half column). Specifically, each clock region can accommodate at most 12 clocks, and each half column can accommodate at most 6 clocks.

B. Problem Formulation

The timing-driven placement problem can be formulated as a hypergraph $G = (V, E)$, where $V = \{v_1, v_2, ..., v_m\}$ and $E = \{e_1, e_2, ..., e_n\}$ denote the blocks and nets in the netlist, respectively. Let x_i and y_i be the x and y coordinates of the center of block v_i. Given the FPGA architecture and a design netlist, our objective is to determine the positions of the blocks to optimize the routed wirelength subject to the following constraints:

- Every block is placed to its legal position with its own type of sites.
- The number of clocks inside each clock region is equal or less than 12, and the number of clocks inside each half column is less than 7.
- Each path in the netlist receives a non-negative slack, and the larger worst slack, the higher placement quality.

III. Our Placement Algorithm

Our algorithm for the timing-driven FPGA placement is summarized in Figure 2. It consists of three major steps: (1) timing modeling, (2) timing-driven global placement, and (3) timing-driven legalization and detailed placement.

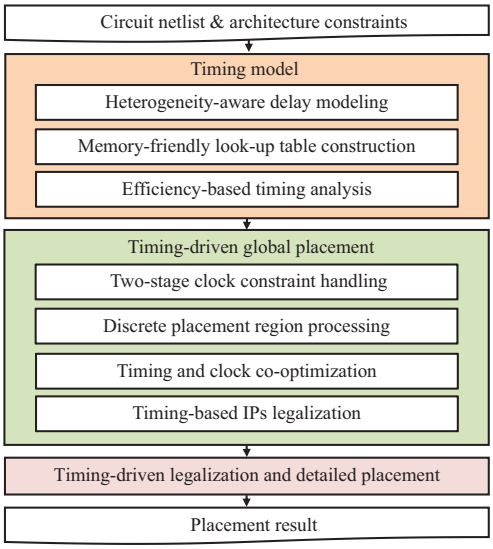

Fig. 2. Our analytical placement flow.

Timing modeling tries to capture the routing information of the corresponding net and provides a memory-friendly look-up table for rapid delay assessment. Then, by using the delay information provided by the timing modeling, global placement finds desired locations for heterogeneous blocks without clocking violations. Finally, timing-driven legalization and detailed placement remove all blocks overlaps and further improve the placement quality.

A. Timing Modeling

In this subsection, we first develop a novel delay model to provide accurate information for timing calculation. Then a memory-friendly delay look-up table is constructed to speed up delay access. Finally, we present an efficient timing analysis method to achieve desired performance.

1) Heterogeneity-Aware Delay Modeling: The delay model plays an important role in timing-driven FPGA placement. Existing delay models seldom consider the heterogeneity nature of an FPGA architecture, and thus may misguide block movement leading to poor solutions. To address this issue, we propose an accurate heterogeneity-aware timing model defined as follows:

$$D(i,j) = I(i,j) + P(i,j) + C(i,j), \tag{1}$$

where $I(i,j)$ is the internal delay of the edge connecting nodes i and j, $P(i,j)$ is the path delay, and $C(i,j)$ is the compensate delay. To be specific, path delay $P(i,j)$ can be further formulated as:

$$P(i,j) = B(i,j) + L(i,j), \tag{2}$$

$$L(i,j) = h \times WireDelayH + v \times WireDelayV, \tag{3}$$

where $B(i,j)$ is the base delay, $L(i,j)$ is the long delay, h, v, $WireDleayH$, and $WireDelayV$ are the number of long wires and long wire delays in the horizontal and vertical directions, respectively.

Figure 3 gives an overview of our delay modeling. The internal delay $I(i,j)$ computes the source pin delay from the output pin i to its related switch box sb_i and the sink pin delay from the switch box sb_j to the input pin j, respectively. In practice, internal delays for each type of blocks are easily found, as they remain the same for all locations in an FPGA. $P(i,j)$ gives the path delay between sb_i and sb_j which is heavily

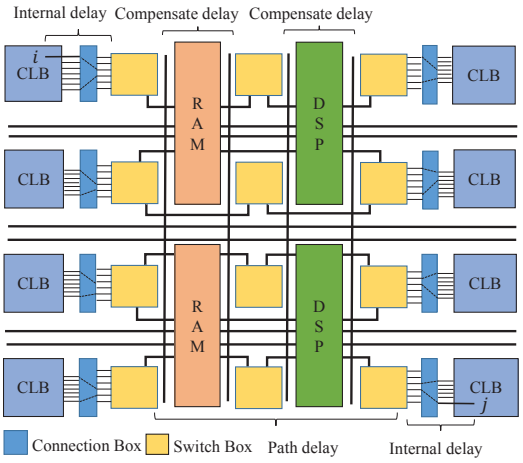

Fig. 3. Our proposed delay modeling.

affected by the distance (x, y) of the two terminals. For long distance connections that the sources are far away from the sinks, we transform their delays into short distance delays based on Equations (2) and (3). Finally, due to the different dimensions between heterogeneous blocks, the delay of a connection across different blocks could have large deviations. For example, IPs usually have larger dimensions compared with CLBs, and also requires longer delays. To remedy this problem, we use $C(i, j)$ to further compute the compensate delay based on the delay of CLBs. By splitting the connection delay into subtle components and considering the heterogeneous nature of FPGAs, we can accurately model the delay of each connection.

2) Memory-Friendly Look-Up Table Generation: After designing an accurate delay model, we need to construct a look-up table to rapidly assess each connection delay. Martin *et al.* [9] constructed an x and y based delay table for timing-driven placement, which records the delays of all possible x- and y-coordinate values in an FPGA. However, this approach is memory expensive and may produce inferior solutions. With our efficient delay model that long distance delays can be mapped into the short ones, in this paper, we construct a sufficiently small delay table $T_d(x, y)$ which only contains delays within short distance. Considering that blocks may move to specific positions with float coordinates during global placement, we further transform discrete $T_d(x, y)$ into the continuous delay table $T_c(x, y)$ using bilinear interpolation as follows:

$$T_c(x,y) = \begin{bmatrix} 1-\hat{x} & \hat{x} \end{bmatrix} \begin{bmatrix} T_d(x_l,y_b) & T_d(x_l,y_u)) \\ T_d(x_r,y_b) & T_d(x_r,y_u)) \end{bmatrix} \begin{bmatrix} 1-\hat{y} \\ \hat{y} \end{bmatrix}, \tag{4}$$

where $\hat{x} = x - x_l$, $\hat{y} = y - y_b$. x_l, x_r, y_b, y_u are the boundary coordinates of the bilinear interpolation.

In Equation (4), $T_c(x, y)$ is a continuous function, but it would not be differentiable in boundary intersections. Therefore, we need to smooth $T_c(x, y)$ such that the interpolation method can be integrated into our analytical framework. The most commonly used smoothing function is the sigmoid function which maps the input variable into the range 0 to 1. However, it may change the values of boundary intersections and thus degrade the delay accuracy. To remedy this issue, we propose a novel trigonometric function defined as:

$$\hat{S}(x) = \begin{cases} 0, & x \le 0, \\ \frac{1}{2}(\sin((x-\frac{1}{2})\pi)+1), & 0 \le x \le 1, \\ 1, & x \ge 1. \end{cases} \tag{5}$$

and then incorporate it with $T_c(x, y)$ to get a smoothed delay table $T_s(x, y)$:

$$T_s(x,y) = \begin{bmatrix} 1-\hat{S}(\hat{x}) & \hat{S}(\hat{x}) \end{bmatrix} \begin{bmatrix} T_d(x_l,y_b) & T_d(x_l,y_u) \\ T_d(x_r,y_b) & T_d(x_r,y_u) \end{bmatrix} \begin{bmatrix} 1-\hat{S}(\hat{y}) \\ \hat{S}(\hat{y}) \end{bmatrix}. \tag{6}$$

By using the constructed differentiable table $T_s(x, y)$, we can greatly speed up the delay access for any connections.

3) Efficiency-Based Timing Analysis: At the end of our timing modeling, we implement timing analysis to compute the delay of all of the paths in a given circuit. These delay values are then used to guide timing optimization during the placement process. Similar to [9], we first consider the circuit as a timing graph, in which each node denotes an input or output pin, and edges between these nodes store the corresponding physical delays.

To perform timing analysis, we traverse the timing graph from the sources to the sinks setting the arrival time, $T_{arrival}$, for all nodes with the following equation:

$$T_{arrival}(i) = \max_{j \in fanin(i)} (T_{arrival}(j) + D(j,i)), \tag{7}$$

where node i is the net sink, node j is the source, and $D(j, i)$ is the delay value of the connection joining node j to node i. Then, we implement a second traversal backward through the timing graph to compute the required time, $T_{required}$ at every node as follows:

$$T_{required}(i) = \min_{j \in fanout(i)} (T_{required}(j) - D(i,j)). \tag{8}$$

After that, the slack of a connection (i, j) driving node j, is given by:

$$Slack(i,j) = T_{required}(j) - T_{arrival}(i) - D(i,j). \tag{9}$$

Considering that placement is an iterative optimization process, a proper timing analysis can have significant effects on the whole design performance and closure efficiency. Existing path-based placers perform timing analysis to calculate path delays at every placement stage. Approaches of this type have more accurate timing view and control, but they are computationally complex because any node movement requires all paths going through that node to be re-analyzed [4]. The other net-based placers involve implementing a path-based timing analysis before placement, and paying higher attention to more critical connection. These placers achieve better runtime quality, but suffer from the poor global view of complete path delay. Hence, this paper implements an efficiency-based timing analysis by periodically performing a path-based timing analysis after a certain number of optimization iterations. This gives a better trade-off between delay accuracy and runtime.

B. Timing-driven Global Placement

The timing-driven global placement dominates the overall placement quality. In this subsection, we first present a two-stage clocking constraint handling method followed by a heterogeneous placement region solving algorithm. Then, a timing-based multi-objective co-optimization method is developed to generate an optimized placement while satisfying the clocking constraints. Finally, we propose a matching-based IP blocks legalization approach to remove all overlaps while preserving desired timing results.

1) Complex Clock Constraints Handling: A modern FPGA often contains a clocking architecture to achieve better skew and performance. As a result, without considering clocking resources during placement could lead to clocking routing failures. To deal with this problem, the work [11] proposed a clocking constraint legalization with a shrinking and expanding

Algorithm 1 Clocking Constraint Handling Algorithm

Require: A global placement result.

Ensure: The optimized placement without any clocking and resource conflict.

```
1:  while there are conflicting clock regions do
2:    cr ← the most conflicting clock region;
3:    while there are clock violations in cr do
4:      cn ← the most timing-critical clock net in cr;
5:      cho ← getOptShrCho;
6:      shrink cn with cho;
7:      while there are expanding choices for cn do
8:        cho ← getOptExpCho;
9:        expand cn with cho;
10:     end while
11:   end while
12: end while
```

scheme. However, the displacement target in this approach may not be efficient enough to guide the shrinking of a clock net bounding box, and may potentially hurt the routed wirelength of the resulting placement. Hence, we propose an effective clock region refinement method combined with a reasonable shrinking cost.

Given a global placement solution with conflicting clock regions, our algorithm solves those clocking violations while preserving good placement quality. First, we construct the clock domains for all clock nets, where a clock domain is denoted as the whole placement area covered by the corresponding clock signal bounding box. Then, as shown in Algorithm 1, an effective clocking constraint handling algorithm is developed to address the conflicting clock region. For every loop, we try to reduce the violations of the most conflicting clock region. In Lines 4 and 5, we focus on the most timing-critical clock net and use the *getOptShrCho* function to determine the best shrinking choice with the lowest cost. Here, the cost is calculated as the total block displacement over the number of the clocking conflict solved by this shrinking choice. Figure 4(b) gives an example of clocking conflict handling.

Then, we refine the clock domain towards the best direction. Chen *et al.* [2] proposed a density estimation approach to generate expanding choice. However, this method can not accurately compute the resource usage and may cause resource conflict in large scale circuits. In our algorithm, we calculate the number of available sites in the clock domain for every type of block. Then, similar to clocking conflict handing, we use the *getOptExpCho* function to generate the best expansion choices from four directions (top, bottom, left, right) that makes the clock bounding box largest. The expansion will be repeated as long as there is no clock conflict and resource violation. By continuing expanding the placement region of each block, our refinement strategy gives more flexibility for block movement, and leaves a positive impact on placement quality. Figure 4(c) gives an example of clock region refinement.

After generating each clock domain, the legal placement region for every cell is limited by the intersection of the domain of the clocks connected to the cell. Then, we relax this region constraints into our objective by adopting the quadratic region cost defined as follows [2]:

$$R(x_i, y_i) = R_H(x_i) R_V(y_i), \tag{10}$$

where

$$R_H(x_i) = \begin{cases} (x_i - x_{iL})^2, & \text{if } x_i < x_{iL}, \\ (x_i - x_{iR})^2, & \text{if } x_i > x_{iR}, \\ 0, & \text{otherwise.} \end{cases}$$

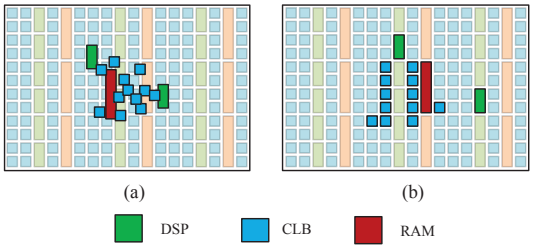

Fig. 5. Illustration of a large displacement caused by unreasonable global placement without heterogeneity consideration. (a) A global placement. (b) The legalization result.

$$R_V(y_i) = \begin{cases} (y_i - y_{iB})^2, & \text{if } y_i < y_{iB}, \\ (y_i - y_{iT})^2, & \text{if } y_i > y_{iT}, \\ 0, & \text{otherwise.} \end{cases}$$

In the above equations, $R_H(x_i)$ and $R_V(y_i)$ give the horizontal and vertical region costs respectively, and $x_{iL}(y_{iB})$ and $x_{iR}(y_{iT})$ are the respective left (bottom) and right (top) boundary coordinates of the clock domain for logic block v_i.

2) Discrete Placement Region Processing: In modern FPGA placement, a specific block must be placed in a corresponding site in an FPGA, which results in a discrete placement region and significantly affects block spreading. Therefore, global placement without considering FPGA heterogeneity may cause inferior solutions. Figure 5(a) shows a non-heterogeneity-aware global placement in which RAMs and DSPs are far away from their desired positions. Then, legalizing these blocks will incur large wirelength, as shown in Figure 5(b).

To address this problem, a complex block density model [3] is proposed to minimize the block displacement. However, the discrete target density in this approach may limit block spreading, resulting in poor placement quality. Therefore, we present a smooth density function defined as follows:

$$\begin{aligned} \hat{D}(\mathbf{x}, \mathbf{y}) = {} & \lambda^C \sum_b (\max(\hat{D}_b^C(\mathbf{x}, \mathbf{y}) - M_b^C, 0))^2 \\ & + \lambda^R \sum_b (\max(\hat{D}_b^R(\mathbf{x}, \mathbf{y}) - M_b^R, 0))^2 \\ & + \lambda^D \sum_b (\max(\hat{D}_b^D(\mathbf{x}, \mathbf{y}) - M_b^D, 0))^2, \end{aligned} \tag{11}$$

where λ^C, λ^R, and λ^D are the heterogeneous penalty factors for CLBs, RAMs, and DSPs, respectively. $\hat{D}_b^C(\mathbf{x}, \mathbf{y})$ ($\hat{D}_b^R(\mathbf{x}, \mathbf{y})$ and $\hat{D}_b^D(\mathbf{x}, \mathbf{y})$) is the total area of CLBs (RAMs and DSPs) in bin b, and M_b^C (M_b^R and M_b^D) is the maximum allowable area of CLBs (RAMs and DSPs) in bin b. By solving Equation (11), different types of blocks can be smoothly guided to near-legal locations in the placement process.

3) Clock and Timing Co-Optimization: To minimize the critical path delay to achieve timing closure, we need to incorporate the timing cost into our placement objective. By accumulating all the connection costs, we first give the total timing cost for a given circuit as follows:

$$\hat{T}(\mathbf{x}, \mathbf{y}) = \sum_{net\ e} \Big(\sum_{k \in e \setminus \{s\}} Crit(s,k) \cdot T_s(x_k - x_s, y_k - y_s) \Big), \tag{12}$$

where $T_s(x_k - x_s, y_k - y_s)$ is the accurate delay of the net connecting from the source s to one of its sinks k. In Equation (12), $Crit(s,k)$ is the connection criticality defined as follows:

$$Crit(s,k) = \left(1 - \frac{Slack(s,k)}{Dly_{max}}\right)^{\alpha}, \tag{13}$$

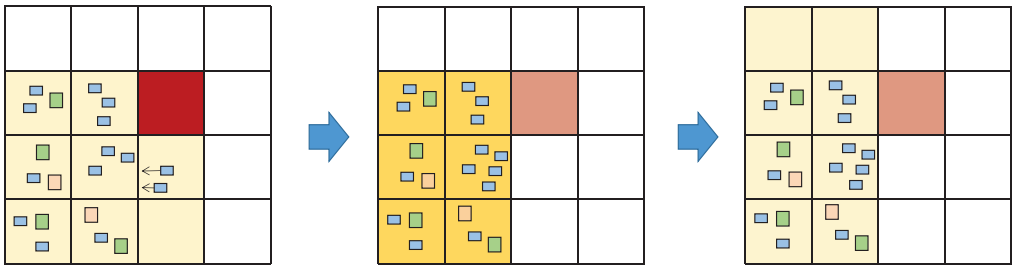

Fig. 4. Illustration for our complex clocking constraint handling. (a) A clock domain of the original global placement. (b) The clock domain after clocking conflict handing. (c) The clock domain after clock region refinement. In above figures, each grid is a clock region, the yellow grids give the domain of the clock net, and all blocks connect to the same clock net and the red grid in (a) is the conflicting clock region. In (b), as the clock domain shrinks, the number of clocks in the conflicting region is reduced, but the resulting domain is resource conflicting. After the clock region refinement in (c), the resource violation is handled without incurring any new clocking conflict.

where $Slack(s,k)$ is the slack calculated by our timing analysis, Dly_{max} is the maximum circuit delay, and α is a parameter that controls the relative importance of connections with different criticalities. Here, we heavily weight critical connections while giving fewer attentions to non-critical ones.

Then a timing-driven multi-objective co-optimization method is developed to improve placement quality while satisfying clocking constraints:

$$\min_{\mathbf{x},\mathbf{y}} \quad \lambda_1\hat{W}(\mathbf{x},\mathbf{y})+\lambda_2 R(\mathbf{x},\mathbf{y})+\lambda_3\hat{D}(\mathbf{x},\mathbf{y})+\lambda_4\hat{T}(\mathbf{x},\mathbf{y}), \quad (14)$$

where λ_1, λ_2, λ_3 and λ_4 are the weights, $\hat{W}(\mathbf{x},\mathbf{y})$ is the smoothed wirelength cost, $R(\mathbf{x},\mathbf{y})$, $\hat{D}(\mathbf{x},\mathbf{y})$, and $\hat{T}(\mathbf{x},\mathbf{y})$ are the aforementioned clock, density, and timing costs, respectively. During the process of iteratively finding the best positions of all placeable blocks, we constantly update the weights λ_1, λ_2, λ_3, and λ_4 based on the Gompertz curve $f(\lambda) = e^{-\sigma e^{-\upsilon k}}$, where σ controls the displacement, υ represents the growth rate, and k is the number of iterations.

4) Timing-Based IPs Legalization: Due to the higher connectivity and larger size of IPs (RAMs and DSPs), legalizing IPs often incurs much disturbance of placement quality compared to CLBs. Hence it is desirable to perform IPs legalization before CLBs. Inspired by the work [5], we adopt bipartite matching for our RAM/DSP legalization at the end of global placement. However, different from their delay-driven method, we further consider the net criticality and optimize the timing-based cost while satisfying clock constraints.

The edge cost for a specific block b and a legal location l is defined as follows:

$$cost(b,l) = \beta\hat{W}_b(l) + (1-\beta)\hat{T}_b(l), \quad (15)$$

where $\hat{W}_b(l)$ is the total wirelength when b is at l, β is the control parameter, and $\hat{T}_b(l)$ is the corresponding timing cost. Since clock domains have been constructed before the legalization, for a given block b, we restrict it to be legalized within its clock domain to avoid any new clocking conflicts. As a result, by solving the minimum weighted bipartite matching for each clock region, we can quickly achieve an optimal legalized solution while satisfying the clocking constraints.

C. Timing-driven Legalization and Detail Placement

After global placement, we have solved the region constraints, where the number of clocks inside a clock region is at most 12. In the timing-driven legalization stage, we aim to remove all overlaps with satisfied half column constraints and further improve timing quality. Similar to IPs legalization in Section III-B4, during each round of half column legalization, the most conflicting half column is selected and a clock with the smallest displacement and timing cost is moved from a half column. We repeat this process until there is no overflowing half column. Finally, the placement results are further improved by iteratively global swapping and local swapping during the detailed placement stage.

TABLE I
BENCHMARK STATISTIC

Benchmark	#LUTs	#FFs	#DSPs	#RAMs
cipher_core	167534	34526	278	156
pipelinedcipher	130351	28372	135	101
jpeg_perf	220428	65829	538	432
THEIA_jpeg	86341	19153	162	149
raptor_theia_jpeg	160187	38994	232	196
fft1D_512_1	210187	58994	284	187
fft1D_512	196849	42253	512	345
THEIA	221894	64737	526	418
xilinx_7x	155479	36678	193	134
diff_map	207453	62565	507	425
dma_top	198792	59832	496	383
pru_perf	186534	576641	354	262

IV. EXPERIMENTAL RESULTS

We implemented our timing- and clock-aware placement algorithm in the C++ programming language, and conducted experiments on the benchmarks of Xilinx for 7 Series devices. Table I gives the statistics of these FPGA benchmarks, where the numbers of LUTs, FFs, DSPs, and RAMs are denoted by "#LUTs", "#FFs", "#DSPs", and "#RAMs", respectively.

To study the performance of our algorithm, we first compare our clock- and timing-driven placement (named "Ours") with the leading clock-aware placer [2] (named "TCAD'20"). Then, the advanced commercial tool Xilinx Vivado 2019.1 with the default settings (named "Vivado 2019.1") is used to further evaluate the effectiveness of our timing-based optimization method. We reported all routed wirelength, and ran all the experiments on the same Linux workstation with Intel Xeon 3.5GHz CPUs and 512GB memory. Table II lists the comparisons of the placement quality among "Ours", "TCAD'20" and "Vivado 2019.1". In this table, "CPU(s)" gives the total placement and routing CPU time in second, and the last row shows the average normalized routed wirelength, and runtime ratios based on our results.

From Table II, our algorithm consistently gives the best worst slack and the shortest routed wirelength for every benchmark. Specifically, our algorithm achieves 8.9% smaller routed wire-

TABLE II
EXPERIMENTAL RESULTS

Benchmark	Worst slack (ns)			Routed Wirelength			CPU (sec)		
	TCAD'20	Vivado 2019.1	Ours	TCAD'20	Vivado 2019.1	Ours	TCAD'20	Vivado 2019.1	Ours
cipher_core	0.032	0.432	0.838	3409124	3584861	3294524	2732	2861	2643
pipelinedcipher	0.573	3.401	3.514	1896370	1902824	1824562	2762	2737	2533
jpeg_perf	-6.931	0.13	0.178	6944764	6598326	6084137	4837	4760	4662
THEIA_jpeg	-2.867	1.159	1.362	2199142	2219218	2046978	1511	1655	1574
raptor_theia_jpeg	-1.653	0.613	0.713	2746987	2510690	2389735	1964	1782	1763
fft1D_512_1	-2.316	1.801	1.895	4168143	4202573	3724039	4135	4156	3941
fft1D_512	-0.764	0.883	1.146	3360549	3389293	3032465	2013	2135	1975
THEIA	-5.579	0.374	0.458	6096747	5608882	5136783	3752	3669	3566
xilinx_7x	-1.968	1.242	1.557	3452352	3236301	3104452	1197	1205	1234
diff_map	-4.548	-0.436	0.123	5594835	5532148	5279632	5134	5356	5103
dma_top	-3.241	-0.142	0.053	5208741	5276214	5189423	4764	4852	4631
pru_perf	-2.983	-0.061	0.084	4314571	4301657	4147841	3541	3621	3309
Norm.				1.089	1.072	1.000	1.036	1.048	1.000

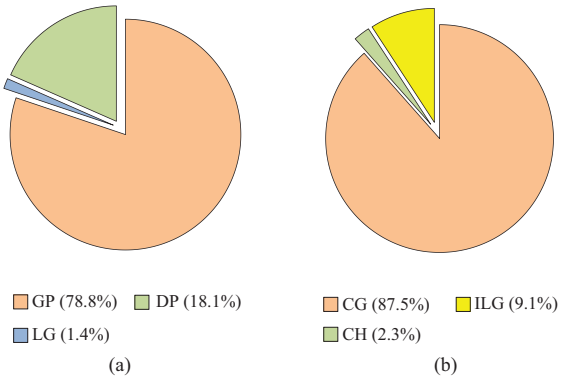

Fig. 6. Runtime breakdown of (a) the whole placement algorithm, where GP, LG, and DP denote Global Placement, Legalization, and detailed Placement, respectively, and (b) global placement alone, where CG, CH, and ILG denote conjugate gradient for nonlinear programming, clocking handing, and IPs legalization, respectively.

length, 3.6% shorter runtime and significant improvement of worst slack over "TCAD'20". The slack improvement is due to the consideration of timing constraints in our analytical global placement. For the advanced commercial tool Vivado 2019.1, it gives comparable wirelength and runtime results, but fails to close timing for three challenging testcases (diff_map, dma_top, and pru_perf). By leveraging our timing-driven optimization method, we resolve all the timing violations and further improve the placement quality by 7.2% smaller wirelength with 4.8% faster runtime. Overall, the experimental results show that our algorithm is effective and efficient for the addressed problem.

Finally, we report the runtime breakdown of our placement algorithm in Figure 6(a). On average, the majority (78.8%) of the total runtime is taken by global placement (GP), while legalization (LG) and detailed placement (DP) respectively consumes 1.4% and 18.1% of the total runtime. We further divide the runtime of global placement into three components, as shown in Figure 6(b), where the conjugate gradient (CG) for nonlinear programming takes 87.5% of the GP runtime, followed by 9.1% for IPs legalization (ILG), and only 2.3% is taken by the clocking handing (CH).

V. CONCLUSIONS

Modern FPGAs often contain heterogeneous architectures and clocking resources to achieve desired solutions, which bring up significant challenges in FPGA placement. Based on the developed accurate timing model, we have presented an effective timing-driven placement algorithm for heterogeneous FPGAs to optimize the worst slack and clocking constraints simultaneously. Compared with the state-of-the-art placer and the commercial tool Xilinx Vivado 2019.1, experimental results have shown that our algorithm achieves best quality for the addressed problem.

REFERENCES

[1] Y.-W. Chang, K. Zhu, and D. F. Wong. Timing-driven routing for symmetrical array-based FPGAs. *ACM Transactions on Design Automation of Electronic Systems*, 5(3):433–450, 2000.

[2] J. Chen, Z. Lin, Y.-C. Kuo, C.-C. Huang, Y.-W. Chang, S.-C. Chen, C.-H. Chiang, and S.-Y. Kuo. Clock-aware placement for large-scale heterogeneous FPGAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Early access, 2020.

[3] S.-Y. Chen and Y.-W. Chang. Routing-architecture-aware analytical placement for heterogeneous fpgas. In *Proceedings of the IEEE/ACM Design Automation Conference*, pages 1–6, 2015.

[4] S. Dhar, M. A. Iyer, S. Adya, L. Singhal, N. Rubanov, and D. Z. Pan. An effective timing-driven detailed placement algorithm for FPGAs. In *Proceedings of ACM on International Symposium on Physical Design*, pages 51–157, 2017.

[5] W. Li, Y. Lin, M. Li, S. Dhar, and D. Z. Pan. UTPlaceF 2.0: A high-performance clock-aware FPGA placement engine. *ACM Transactions on Design Automation of Electronic Systems*, 23(42):1–23, 2018.

[6] T.-H. Lin, P. Banerjee, and Y.-W. Chang. An efficient and effective analytical placer for FPGAs. In *Proceedings of the IEEE/ACM Design Automation Conference*, pages 1–6, 2013.

[7] J. Lu, P. Chen, C.-C. Chang, L. Sha, J. Dennis, H. Huang, C.-C. Teng, and C.-K. Cheng. eplace: Electrostatics based placement using nesterov's method. In *Proceedings of IEEE/ACM Design Automation Conference*, pages 1–6, 2014.

[8] A. Marquardt, V. Betz, and J. Rose. Timing-driven placement for FPGAs. In *Proceedings of the ACM international symposium on Field programmable gate arrays*, pages 203–213, 2000.

[9] T. Martin, D. Maarouf, Z. Abuowaimer, A. Alhyari, G. Grewal, and S. Areibi. A flat timing-driven placement flow for modern FPGAs. In *Proceedings of IEEE/ACM Design Automation Conference*, pages 1–6, 2019.

[10] C.-W. Pui, G. Chen, W.-K. Chow, K.-C. Lam, J. Kuang, P. Tu, H. Zhang, E. F. Young, and B. Yu. RippleFPGA: a routability-driven placement for large-scale heterogeneous FPGAs. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 1–8, 2016.

[11] C.-W. Pui, G. Chen, Y. Ma, E. F. Young, and B. Yu. Clock-aware ultrascale FPGA placement with machine learning routability prediction. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 929–936, 2017.

[12] C. C. Wang and G. G. Lemieux. Scalable and deterministic timing-driven parallel placement for FPGAs. In *Proceedings of ACM international symposium on Field programmable gate arrays*, pages 153–162, 2011.

[13] Xilinx, Inc. http://www.xilinx.com.

[14] S. Yang, C. Mulpuri, S. Reddy, M. Kalase, S. Dasasathyan, M. E. Dehkordi, M. Tom, and R. Aggarwal. Clock-aware FPGA placement contest. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 159–164, 2017.