Clock-Aware UltraScale FPGA Placement with Machine Learning Routability Prediction

(Invited Paper)

Chak-Wa Pui , Gengjie Chen , Yuzhe Ma , Evangeline F. Y. Young , and Bei Yu

Department of Computer Science and Engineering,
The Chinese University of Hong Kong, NT, Hong Kong
{cwpui,gjchen,yzma,fyyoung,byu}@cse.cuhk.edu.hk

***Abstract*—As the complexity and scale of circuits keep growing, clocking architectures of FPGAs have become more complex to meet the timing requirement. In this paper, to optimize wirelength and meanwhile meet emerging clocking architectural constraints, we propose several detailed placement techniques, i.e., two-step clock constraint legalization and chain move. After integrating these techniques into our FPGA placement framework, experimental results on ISPD 2017 benchmarks show that our proposed approach yields 2.3% shorter routed wirelength and the running time is 2× faster compared to the first place winner in the ISPD 2017 contest. Moreover, we explore the possibilities to use machine learning-based methods to predict routing congestion in UltraScale FPGAs. Experimental results on both ISPD 2016 and ISPD 2017 benchmarks show that our proposed congestion estimation model is a good approximation to the one obtained from Vivado and can lead to good placement results compared to the previous methods.**

I. Introduction

Field-programmable gate array (FPGA) is an integrated circuit designed to be reconfigurable by a customer after manufacturing. Compared to application specific integrated circuit (ASIC) and central processing unit (CPU) , FPGA is a good trade-off between performance and cost due to its faster time-to-market and simpler design cycle. With its unprecedentedly increasing logic density, FPGA has become more competitive with ASICs especially in application specific implementations, such as deep learning [1] and data center [2].

Most of the previous researches on FPGAs placement were conducted on the island-style FPGAs, which is a 2D array of configurable logic blocks (CLBs), I/O pads, routing channels, random access memory blocks (RAMs), digital signal processing blocks (DSPs), etc. The placement algorithms can be classified into three major categories: (1) simulated annealing-based approach, (2) partitioning-based approach, and (3) analytical approach. The widely used academic tool VPR [3] applies simulated annealing as its main tool to optimize objectives such as wirelength, timing, etc. Partitioning-based approaches like [4] shorten the running time by recursively partitioning a design and placing them hierarchically. However, these two kinds of placement methods cannot get a good balance between quality and running time. As the gap between FPGAs and ASICs is getting smaller in recent years, analytical approaches become more favorable due to its high efficiency and good quality in ASIC placement. In [5], SimPL [6] is applied to FPGA placement, which yields the potential of using analytical methods in FPGA placement. In [7], [8], NTUplace is used as the basic framework of the proposed analytical FPGA placer. Besides the placers mentioned above, there are also other analytical placers like LLP [9], StarPlace [10] and QPF [11].

These analytical placers mainly focus on how to migrate the traditional ASIC placement methods to FPGA placement and are usually applying on the CLB level rather than the lookup tables (LUTs) and flip-flops (FFs) level. Most of them resolve the timing [12] or routability [13], [14] issues during the packing step which packs LUTs and FFs into CLBs. However, forbidding the LUTs and FFs to move out of a CLB during placement may yield worse quality than those that do not. Moreover, when the architecture becomes more complex such as Xilinx UltraScale, the approaches like above may not work well. Recently, several placement tools [15]–[17] are proposed for Xilinx UltraScale FPGAs. But they can only handle single clock design, which is not practical today. Hence, a flat analytical clock-aware placement for modern heterogeneous FPGAs is needed.

Routability is always an issue in placement for both FPGAs and ASICs. There are lots of previous works focusing on reducing routing congestion in FPGA placement. To reduce congestion, some placers handle it during packing while others may do congestion-driven global placement. Most of these methods need to estimate the congestion level in order to reduce congestion. In [18], the congestion map is built according to the net bounding box (BB). The routing congestion of a site is measured by the number of BBs overlapping with it. In [15], both HPWL and BBs are used to measure congestion. Some other works like [16] use ASIC global router to estimate the routing congestion. Recently, there are several works [19]–[21] for ASIC designs that use machine learning to predict design-rule-check (DRC) violations. Since DRC violation is highly related to routing congestion, these works yield the possibilities to use machine learning to predict routing congestion in FPGAs.

In this paper, we propose several placement techniques for UltraScale FPGAs to meet the challenges of clock constraints, routability, wirelength. We integrate all the techniques into our previous framework RippleFPGA [15] to evaluate the performance. The major contributions of this paper are summarized as follows:

- A two-step displacement-driven legalization is introduced to remove all clock constraint violations. It includes two stages which are clock region planning and half column legalization. In clock region planning, shrinking and expanding will try to remove all the clock region constraint violations. Under some assumptions, it can guarantee to give a global placement result that satisfies the clock region constraint. A greedy algorithm is then performed to remove all the violations of the half column constraint. In the experimental results, this legalization method is shown to be efficient and effective.
- Chain move is proposed to put a cell into a desired site efficiently. It can be modified to optimize different objectives, e.g. displacement, HPWL, timing, etc. In legalization, it is used to reduce the total and maximum displacement. In detailed placement, it is used to further optimize the HPWL by moving the cells to their optimal regions.
- We study the performance of different routability prediction methods in FPGAs. Several features, machine learning models

978-1-5386-3093-8/17/$31.00 ©2017 IEEE

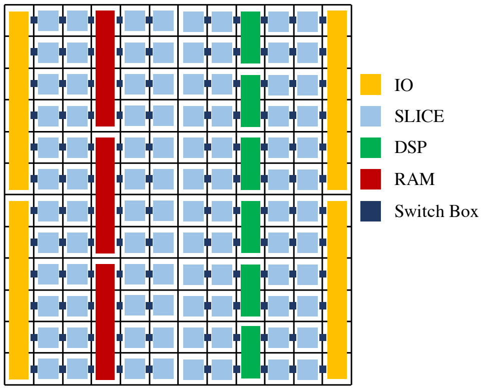


Fig. 1: An illustration of Xilinx UltraScale architecture.

and training methods are discussed. According to the experimental results, a linear regression method is used to estimate the routing congestion. This method can closely approximate the routing congestion estimation generated by Vivado and guide the placement to generate good results.
- All the above techniques are incorporated into our FPGA placer RippleFPGA [15]. The experimental results shows that we can produce better results in terms of routed wirelength and running time compared with the winning teams of the ISPD 2017 contest.

The remainder of this paper is organized as follows. Section II gives an introduction to our target FPGA architecture and the problem formulation. Section III-V introduces details of our proposed techniques including two-step clock constraint legalization, chain move, and machine learning-based congestion estimation. Section VI gives an overview of our placement framework and how the above techniques are integrated into the framework. Section VII shows the experimental results, followed by conclusion in Section VIII.

II. Preliminaries

A. Target Architecture

In this paper, the target FPGA architecture is Xilinx UltraScale VU095 [22], whose layout is shown in Fig. 1. There are four kinds of sites (SLICE, RAM, DSP and IO) on the chip and they are connected by wires through switch boxes. The given netlist of an FPGA design consists of five types of cells, which are LUTs, FFs, RAMs, DSPs and IOs. Each type of cells can only be put into sites of its own type and a site can accommodate multiple cells. Due to the internal wires inside a SLICE, routing the nets between two connected LUT and FF in the same site may use less routing resources. According to [23], when placing LUTs and FFs into SLICEs, extra legalization rules need to be considered besides the cell number constraints. For example, the total number of input signals of LUT$(2i)$ and LUT$(2i+1)$ should be less than 7, the set/reset signal of the FFs in the same half of a SLICE should be the same, etc.

The clocking architecture of Xilinx UltraScale is similar to ASICs. It can route clocks from their clock sources all the way to all of their loads through a mesh-like routing structure, which gives big flexibility to how the clocks are routed. The chip is divided into a 5×8 grid of clock regions such that each clock region consists of about 30×60 sites and a clock is in a clock region if the bounding box of its loads overlaps with the clock region. Each clock region is further divided into a grid of half columns such that each half column consists of 2×30 sites and a clock is in a half column if one of its loads is inside the half column. To utilize such a routing architecture shown in Fig. 2, the numbers of clocks inside each half column and clock region are limited to 12 and 24 respectively.

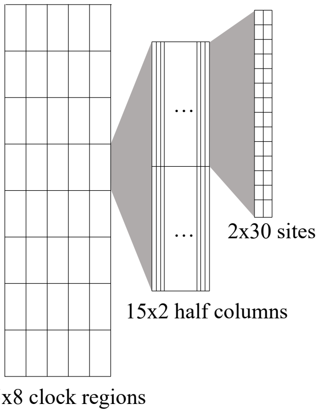


Fig. 2: An illustration of the routing architecture.

B. Problem Formulation

Given the FPGA architecture and a design net-list, we need to determine the positions of the cells on the FPGA to minimize the routed wirelength subject to the following constraints:

- Every cell is assigned to a position on the FPGA that satisfies the placement constraints [23];
- The number of clocks inside each half column is less than 13 and a clock is in a half column if one of its loads is inside that half column;
- The number of clocks inside each clock region is less than 25 and a clock is in a clock region if the bounding box of its loads overlaps with that clock region.

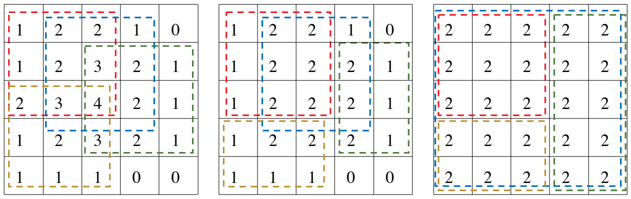
III. Two-Step Clock Constraint Legalization

As mentioned in Section II, a clock region or half column is clock overflowed if the number of clocks inside exceeds the limit. In this section, overflow refers to clock overflow. In order to remove all of the overflows, a two-step clock constraint legalization is performed, it is consisted of two steps : clock region planning and half column legalization. In clock region planning, violations of the clock region constraint in global placement will be removed and a bounding box (BB) is assigned to each clock to limit the cell movement. After clock region planning, each cell should remain in the intersection of the BBs of the clocks that it is connected to. In stage two, a displacement-driven half column legalization is performed in which the overflow of half column will be reduced gradually. After half column legalization, all the subsequent moves cannot violate the half column constraint.

A. Clock Region Planning

Our clock region planning can be divided into two stages: shrinking and expanding. In the following, one unit in the shrinking and expanding stages is at most the height or width of a clock region (60 or 30 sites).

1) Shrinking Stage: A clock region is overflowed if the number of clock BBs overlapping with it is more than 24. In the shrinking stage, given a global placement result with overflowed clock regions, we need to minimize the disturbance to the placement while removing those violations. Before introducing the details, we first need to prove Theorem 1.



1	2	2	1	0
1	2	3	2	1
2	3	4	2	1
1	2	3	2	1
1	1	1	0	0

(a) The BBs of clocks in the original global placement.

1	2	2	1	0
1	2	2	2	1
1	2	2	2	1
1	2	2	2	1
1	1	1	0	0

(b) The BBs of clocks after shrinking.

2	2	2	2	2
2	2	2	2	2
2	2	2	2	2
2	2	2	2	2
2	2	2	2	2

(c) The BBs of clocks after expanding.

Fig. 3: Assuming the maximum number of clocks in a clock region is 2, an example of clock region planning.

Theorem 1. *Suppose each cell is only connected to one clock. Given a grid g of clock regions R and an overflowed region r_i, if there exists at least one neighboring regions whose overflow is smaller, there always exists a direction such that shrinking the bounding box of a clock in r_i to that direction by one unit will reduce the overflow of r_i by one and not increase the overflow of any other clock regions.*

Proof. It is clear that shrinking the bounding box of a clock will not increase the overflow of any other clock regions if each cell is only connected to one clock. Let C be the set of clocks whose bounding boxes overlap with r_i but do not overlap with at least one of the neighboring regions of r_i. If $C \neq \emptyset$, we can shrink the bounding box of a clock $c_i \in C$ in a direction by 1 unit and reduce the overflow of r_i by 1. If $C = \emptyset$, all the clocks in r_i will be in all its neighboring regions. But this contradicts with the fact that the overflow of its neighboring regions is smaller. □

As our experiment shows, only cells containing FFs will have more than one clocks since their clock enable (CE) and set/reset (SR) signals may also connect to the clock signals. However, the BBs of these clock signals usually cover the whole placement and thus moving these cells will not increase the overflow of other clock regions. To minimize the displacement, we will only shrink a clock BB by 1 unit at a time and continue to shrink them until there is no overflow as shown in Algorithm 1. In every round, we will try to reduce the overflow of the most overflowed but shrinkable clock region. In line 5, `GetBestDir` is used to determine the direction to shrink the BB of clock clk_j such that the displacement is smallest. To be specific, `Shrink` is invoked for four times to get the displacement of shrinking the BB of clk_j to each direction respectively. An example of how we shrink the BB of a clock in LEFT direction is shown in lines 13–24 and the other three directions can be deduced accordingly. After getting the clock net and direction that incur the smallest displacement, we will do the corresponding shrinking before starting the next round. When the shrinking stage finishes, the global placement result will satisfy the clock region constraint and there will be no further violation as long as the cells of each clock stay inside its BB. An example of the shrinking stage is given in Fig. 3(b).

2) Expanding Stage: Since a cell's movement is limited by the intersection of the BBs of the clocks connected to the cell, if the intersection area is too small, there will be very limited flexibility in the later placement stages, giving results of poor quality. Therefore, in the expanding stage, the BB of each clock will be expanded as long as there is no violation of the clock region constraint. Similar to the shrinking stage, we will only increase either the width or the height of a BB by 1 unit in each round. Starting with the BB with the highest cell density calculated by Equation (1), we will try all the four directions and choose the one that makes the area of the resulting BB largest while incurring no violation.

Algorithm 1 Shrinking Stage

```
Require: A global placement result.
 1: while there are overflowed clock regions do
 2:     $rgn_i \leftarrow$ the most overflowed clock region;
 3:     $(minDisp, bestDir, net) \leftarrow (\inf, \mathrm{NULL}, \mathrm{NULL})$;
 4:     for all $clk_j$ in $rgn_i$ do
 5:         $(dir, disp) \leftarrow$GetBestDir;
 6:         if $disp \neq \inf$ and $minDisp > disp$ then
 7:             $(minDisp, bestDir, net) \leftarrow (disp, dir, clk_j)$;
 8:         end if
 9:     end for
10:     Shrink($net$, $rgn_i$, $bestDir$);
11:     Update clock region overflow info;
12: end while

13: function Shrink($clk_j$, $rgn_i$, LEFT)
14:     $b_{clk} \leftarrow$ BB of $clk_j$, $b_{rgn} \leftarrow$ box of $rgn_i$;
15:     if $b_{rgn}.lx \leq b_{clk}.hx \leq b_{rgn}.hx$ then
16:         for all cell $c$ connected to $clk_j$ in $rgn_i$ do
17:             if $b_{rgn}.lx \leq c.x \leq b_{clk}.hx$ then
18:                 $c.x \leftarrow b_{rgn}.lx$;
19:             end if
20:         end for
21:         return true;
22:     end if
23:     return false;
24: end function
```

$$\text{cell density}_{clk_i} = \frac{\text{total area of cells connected to } clk_i}{\text{area of the BB of } clk_i}. \quad (1)$$

If the BB of the clock with the highest cell density cannot be expanded, we will try the clock with the second highest cell density and so on until a BB is selected to be expanded. If no BB is expanded in one round, the whole expanding stage is completed. An example of the expanding stage is given in Fig. 3(c).

B. Half Column Legalization

During each round of half column legalization, the most overflowed half column is selected and a clock whose removal will induce the smallest cell displacement will be moved away from the overflowed half column. When a clock is moved away from a half column, each load will be moved to its nearest site in another half column. The move is legal if it does not incur more overflow in the half column that the target site belongs to. After a clock is moved away from the most overflowed half column, another round of move will be invoked. This process will continue until there is no overflowed half column.

IV. Chain Move

Given a partially placed design, the complex legalization rules of modern FPGAs make it much harder to move a cell to a desired location. Inspired by [24]–[26], chain move is proposed to help moving cells into their desired positions, e.g. the sites inside their optimal region or the sites that will only incur small disturbance to the global placement result.

A. General Chain Move Algorithm

Given a cell and an objective, a sequence of cell moves is found such that all the cells are placed to one of its candidate sites and the objective value of the resulting placement is better than that of the original placement. This sequence of cell moves is found by a DFS-based algorithm. First a set of candidate sites S is found depending on the objective. We then try if the current cell to be moved c_j can be directly put into a site $s \in S$. If so, the search ends. Otherwise, we will try to find a cell c_k in a site $s \in S$ such that c_j can be put into s if c_k is moved away. In our implementation, we limit the number of trials of each cell to be 35. If such a cell c_k can be found, we let it to be the next cell to be moved and continue our search. The number of moved cell in the chain is limited to 5 in our implementation.. Moreover, we will randomly decide the number of trials in each site, which helps to find the next cell to be moved more quickly since it may take more trials in some sites while others may take less depending on the cells inside. An example of chain move is shown in Fig. 4.

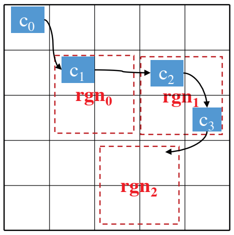


Fig. 4: An example of chain move. The sites in rgn_0, rgn_1, rgn_1 and rgn_2 are the candidate sites of c_0, c_1, c_2 and c_3 respectively.

Unlike [24]–[26] which all optimize HPWL, our chain move can be easily adapted to optimize different objectives by selecting different sets of cells. Similar to [25], [26], our chain move in detailed placement will move cells to their optimal regions. Compared to [26], ours is much more flexible because the last position need not be the position of the first cell in the chain, and we can guarantee that all the placed cells are still legal after chain move while only cell density is maintained in [25]. Unlike [24], we limit the breadth and depth of the search to maintain a good trade-off between quality and running time. Since the legality checking is more complicated than the overlap checking in [24], determining whether a cell can be put into a site is much more time consuming, which makes it necessary to limit the number of trials in a search.

B. Chain Move-based Legalization

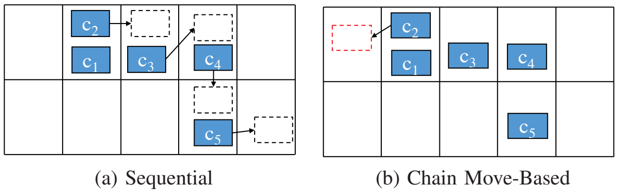
For BLEs, a chain move-based displacement-driven legalization shown in Algorithm 2 is used to place them in legal positions. Candidate sites at displacement d are first obtained (line 4). We then try to find a legal site for the cell by attempting the candidate sites in ascending order of the HPWL incurred. If such a site is found and d is small, we continue to legalize the next cell. If d is large, chain move will be invoked in the maximum displacement optimization mode (line 8). If no site of displacement d can be found, chain move will be invoked in the displacement optimization mode (line 13).

As discussed in Section IV-A, different objectives can be optimized by selecting different sets of cells. In the displacement optimization mode, the set S of candidate sites for a cell c_j are selected such that $s \in S$ iff,

$$dist(s, c_j) \begin{cases} = d, & \text{if } c_j \text{ is the first cell in the chain,} \\ \leq dist(s_j, c_j), & \text{otherwise,} \end{cases}$$

Algorithm 2 Chain Move-Based Legalization

```
Require: A global placement result.
 1: for all BLE $c_i$ do
 2:     $d \leftarrow 0$;
 3:     while $c_i$ is not placed do
 4:         $S \leftarrow$ sites whose distance to $c_i$ is $d$;
 5:         Sort $S$ in ascending order of HPWL;
 6:         if $c_i$ can be put into $s \in S$ then
 7:             if $d \geq 2$ then
 8:                 Optimize max disp. using chain move;
 9:             else
10:                 place $c_i$ in $s$;
11:             end if
12:         else
13:             Optimize disp., try to place $c_i$ by chain move;
14:         end if
15:         $d \leftarrow d + 1$;
16:     end while
17: end for
```



(a) Sequential

(b) Chain Move-Based

Fig. 5: Comparison of the total displacement between the result obtained by sequential legalization and chain move-based legalization.

where s_j is the site where c_j is currently placed and $dist(s_j, c_j)$ is the distance between s_j and c_j. In the maximum displacement optimization mode, the set S of candidate sites for a cell c_j are selected such that $s \in S$ iff,

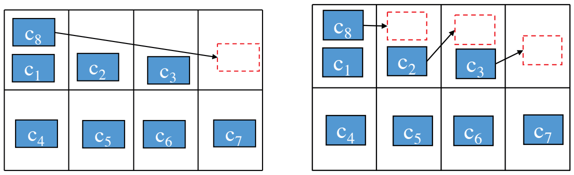
$$dist(s, c_j) + \sum_{c_i \in C \setminus c_j} dist(s_i^{ST}, c_i) \leq \sum_{c_i \in C} dist(s_i, c_i)$$

and the distance between s and c_j is less than d ($dist(s, c_j) < d$), where C is the set of cells in the chain and s_i^{ST} is the target site of c_i selected by chain move.

Compared with the traditional sequential legalization, our algorithm can reduce quality degradation due to some bad decisions made by legalizing cells in the early stage. Two examples are given in Fig. 5 and Fig. 6 where cell c_i is legalized before cell c_j whenever $i < j$. In Fig. 5, due to the bad decision made by placing c_2, the displacement of legalizing these cells is 4. However, in chain move-based legalization, chain move will be invoked in the displacement optimization mode when placing c_3 and the total displacement of legalizing these cells is reduce to 1. In Fig. 6, chain move will slight shift c_3 and c_4 in the maximum displacement optimization mode and reduce the maximum displacement from 3 to 1 while not increasing the total displacement.

C. Chain Move-based Detailed Placement

During detailed placement, there are two kinds of moves for BLEs outside their optimal regions. To be specific, for such a BLE, we will first try to place it in its optimal region. If it is not possible, we will try those sites that are closer to its optimal region than its current position. If a BLE cannot be moved by the above two operations, the chain move discussed in Section IV-A will be invoked in the detailed placement mode. The objective is to move the BLE to its



(a) Sequential (b) Chain Move-Based

Fig. 6: Comparison of the maximum displacement between the result obtained by sequential legalization and chain move-based legalization.

optimal region by a set of cell moves such that all the moved cells are in their optimal regions in the resulting placement. To achieve this, the candidate sites S for a cell c_j are the sites inside its optimal region.

V. Machine Learning-based Congestion Estimation

To estimate the routing congestion more precisely during placement, a congestion map is needed. In this section, we will study how to leverage machine learning to build a congestion model. To better understand how different machine learning methods perform on the congestion estimation, we will compare and discuss the performance of a few learning models. Moreover, we will discuss the motivations and benefits of using machine learning-based congestion estimation on FPGAs placement.

A. Congestion Models

Since more than 95% of the routing resources in FPGA placement are consumed by connections between SLICEs, our congestion model will only consider the congestion levels of sites whose types are SLICE. First, the FPGA is divided into global routing cells (gcells) in a similar way as in [15] such that a gcell corresponds to a switch box and a SLICE is mapped to only one gcell. For each site, the values of its features are equal to those of the gcell covering it and a machine learning-based model is used to predict the routing congestion of each site. In our estimation models, the predicted congestion value represents the percentage of routing resources used in that site.

Intuitively, the number of internal pins within a gcell and the number of BBs covering a gcell have significant impact on routability. Therefore, we extract three features from each gcell, which are defined as follows,

$$x_1 = \sum_{m \in N_i} \frac{w_m \cdot \text{HPWL}_m}{\#\text{gcell}_m}, \tag{2}$$

$$x_2 = \sum_{m \in N_i} \#\text{pins of net } m, \tag{3}$$

$$x_3 = \sum_{m \in N_i} \frac{p_{m,i}}{\#\text{pins of net } m} \times \frac{w_m \cdot \text{HPWL}_m}{\#\text{gcell}_m}, \tag{4}$$

where w_m is a weight proportional to the number of pins of net m, N_i is the set of nets connected to the gcell covering site s_i, $p_{m,i}$ is the number of pins of net m inside the gcell covering site s_i and $\#\text{gcell}_m$ is the number of gcells covered by net m. The first two features x_1 and x_2 capture the pin and overlapping BB information respectively while x_3 captures their combined effects.

In the following, three machine learning models will be introduced: (1) local linear model, (2) hierarchical hybrid model, (3) global linear model.

TABLE I: Average r^2 score, MPE (%) comparison between different training methods and congestion estimation models.

Model*	Unified		Independent	
	Avg. r^2	Avg. MPE	Avg. r^2	Avg. MPE
Local Linear Model	0.891	0.161	0.878	0.176
Hierarchical Hybrid Model	-	-	0.833	0.163
Global Linear Model	0.943	0.115	0.933	0.128

* The model trained by the unified method is trained and tested by the data from all designs while the models trained by the independent method are only trained and tested by the data from their corresponding designs.

1) Local Linear Model: In our local linear model, only the information of the site itself will be used to predict its congestion. The model is represented as,

$$y_{\text{llm}} = f_{\text{llm}}(\mathbf{x}) = \sum_{i=1}^{3} \theta_i x_i. \tag{5}$$

2) Hierarchical Hybrid Model: Our hierarchical hybrid model consists of two stages which consider local and global congestion information respectively. In the first stage, a linear model defined by Equation (5) is used to estimate the local congestion, denoted as y_{hm1}. In the second stage, a non-linear model is used to capture the congestion information of nearby sites, whose result is denoted as y_{hm2}. To be specific, for each site, we use the the support vector machine (SVM) as our machine learning model with the y_{hm1} value of the site and its neighboring eight sites as features. Finally, we use y_{hm2} as the estimation result of each site, which captures both local and neighborhood congestion information.

3) Global Linear Model: In this section, the non-linear model in the second stage of the hierarchical hybrid model is substituted by a linear model. Since a two-stage linear model is equivalent to a single stage linear model, we use a single linear model to represent it. To be specific, for each site, 27 features are extracted, which are the x_1, x_2 and x_3 values of the site itself and its neighboring eight sites, which can be formulated as

$$y_{\text{glm}} = f_{\text{glm}}(\mathbf{x}) = \sum_{i=1}^{27} \theta_i x_i. \tag{6}$$

B. Training and Regression Results

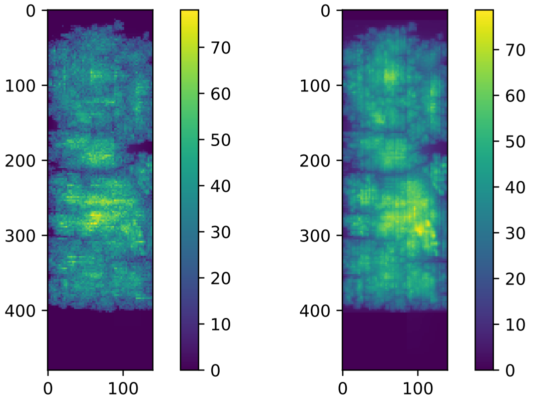
To train the models, we extract the feature vectors from the placement and obtain the congestion estimation values from Vivado. We generate these data from the placement results of our placer for ISPD 2016 and ISPD 2017 benchmarks. Noted that, since there are empty sites in the placement, we only use a site as test data if it satisfies at least one of the two following conditions,

- the site is not empty in the placement,
- the congestion value of the site is not equal to zero in the reference congestion map.

We then divide the data from each design into training set (70%) and testing set (30%), which are used to build the models and test the reliability of the models respectively.

There are two different options in model training, one is to train a unified model for all designs while the other is to train independent models for different designs. The benefit of a unified model is that if gets more training data and can be used to predict the congestion of an unknown design. The benefit of using different models for different designs is that it can better capture the characteristics of different designs and the training time is much shorter especially for those high timing complexity models like SVM.

TABLE I shows the r^2 score, mean percentage error (MPE) of our proposed estimators trained by different methods. The result



(a) (b)

Fig. 7: Congestion Maps of CLK-FPGA13 in ISPD2017 contest: (a) routing congestion predicted by Vivado; (b) our estimation.

demonstrates how accurately our proposed model can predict the congestion. Compared to the global linear model, the hierarchical hybrid model does not perform as good and its results are very close to the local model. The most likely underlying reason is that the congestion estimation produced by Vivado has a linear relationship with our features. Compared with the local linear model, the global linear model performs better since it captures both local and global congestion information. Hence, we will use the global linear model as our congestion estimation model. Note that the output of the regression model is close to the estimated value of Vivado according to the mean percentage error. Using the global linear model, we can predict congestion quite accurately as illustrated in Fig. 7.

As shown in TABLE I, we have studied the performance of the two training methods. Since the training time complexity of the hierarchical hybrid model is much higher than that of the linear model and its performance is not as good, we only compare the performance of linear models trained by unified method and independent method. As one can see, the performance of the unified method is slightly better than that of the independent method. The underlying reason is that most of the designs are similar to each other in terms of technology, constraints, etc. and thus can be represented by one model. The advantage of the independent training method over the unified training method will be more obvious if the characteristics of each design are very different, because the unified method may not be able to capture the differences between designs with only one model. However, when the differences between the designs' characteristics are small, the unified model actually gets more data for training compared to the independent model, which explains why the average performance of the unified method is better than that of the independent method in our dataset. Another drawback of having multiple independent models is that each model may be over-fitted (due to the lack of data) and thus cannot be used to predict the congestion levels of an unknown design. One solution is ensembling all the individual models. To be specific, the prediction of a given feature vector $\mathbf{x}$ is obtained by combining the output of all the existing models, which can be represented by

$$y = \sum_{i=1}^{N} w_i f_{\mathrm{glm},i}(\mathbf{x}), \tag{7}$$

where $f_{\mathrm{glm},i}$ is the model trained with design i and w_i is the weight of model i which is set to $\frac{1}{N}$ in our implementation. We also test the performance of the ensemble method and compare it with the method of training one single model using the data across multiple designs. We randomly select 12 designs for training and use the rest for testing. The result is shown in TABLE II. According to the results of our experiments, both methods can generalize well to other designs.

TABLE II: Average r^2 score, MPE (%) comparison between unified model and ensemble.

Model[1]	Unified+[2]		Ensemble	
	Avg. r^2	Avg. MPE	Avg. r^2	Avg. MPE
Global Linear Model	0.914	0.172	0.926	0.163

[1] "Unified+" represents the global linear model trained by our unified method and "Ensemble" represents the model defined in Equation (7).

[2] The difference between "Unified+" and "Unified" in Table I is that the training set of "Unified+" is 12 randomly selected designs while that of "Unified" is all of the designs.

C. *Comparison*

As mentioned in Section I, previous works on FPGAs placement use different methods to estimate routing congestions. However, these methods all have their shortcomings. Probabilistic models like [15], [18] may have good correlation with the relative congestion level but it is hard to use their absolute values to determine whether a site is congested or not. Given the routing capacity of a design, global routers can perform better since they "actually" do the routing instead of using a probabilistic model only. But due to IP issues, it is difficult to obtain information of the routing architecture or the routing capacity of the FPGAs.

Similar to other EDA tools in ASICs, design tools of FPGAs like Vivado will also give users an estimation of the routing congestion. If we assume that the estimation from the tool itself can guide the placement well, it is worthwhile to build a congestion model that can approximate well the congestion estimation of the tool. Another benefit of approximating the estimation of the tool is that it can give the users a good sense of how congested a site is. The last but not least benefit of using machine learning is that it saves a lot of effort in parameters tuning which is a big shortcoming of the previous methods. All of the parameters needed to predict the congestion levels are generated automatically by machine learning, however, probabilistic models need to find the correspondence between the real congestion level and the predicted value while the capacity values of the global routers also need to be carefully chosen.

VI. RippleFPGA

In this section, we summarize our framework and introduce how the proposed techniques are incorporated into it. As shown in Fig. 8, the framework is based on RippleFPGA [15] and we mainly refine the detail placement stage to improve the quality and to meet the new clock constraints.

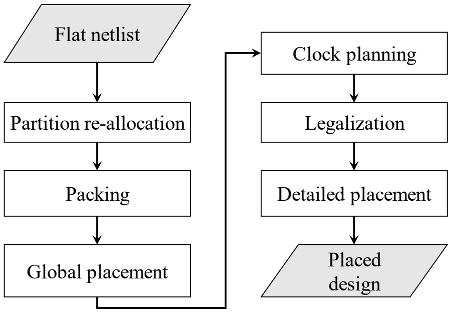


Fig. 8: The overall flow of our proposed placer.

TABLE III: Routed wirelength and running time (s) comparison with the ISPD 2017 contest winners.

Design	RippleFPGA				1st Place				2nd Place				3rd Place			
	WL	ratio	Time	ratio	WL	ratio	Time	ratio	WL	ratio	Time	ratio	WL	ratio	Time	ratio
CLK-FPGA01	2011452	1	288	1	2208170	1.098	530	1.840	2209328	1.098	2686	9.326	2268532	1.128	231	0.802
CLK-FPGA02	2167861	1	266	1	2279171	1.051	521	1.959	2273729	1.049	2788	10.481	2504444	1.155	218	0.820
CLK-FPGA03	5265206	1	583	1	5353071	1.017	1038	1.780	6229292	1.183	3740	6.415	5803110	1.102	447	0.767
CLK-FPGA04	3606567	1	380	1	3697950	1.025	725	1.908	3817377	1.058	2850	7.500	4085670	1.133	309	0.813
CLK-FPGA05	4660136	1	569	1	4692356	1.007	943	1.657	4995177	1.072	3164	5.561	5180916	1.112	444	0.780
CLK-FPGA06	5736998	1	591	1	5588507	0.974	1075	1.819	5605573	0.977	3570	6.041	6216898	1.084	471	0.797
CLK-FPGA07	2325787	1	304	1	2444359	1.051	585	1.924	2504544	1.077	3698	12.164	2676088	1.151	247	0.813
CLK-FPGA08	1778292	1	247	1	1885632	1.060	482	1.951	1989632	1.119	2504	10.138	2057117	1.157	198	0.802
CLK-FPGA09	2530105	1	327	1	2601161	1.028	600	1.835	2583442	1.021	3158	9.657	2813538	1.112	269	0.823
CLK-FPGA10	4495500	1	512	1	4464341	0.993	868	1.695	4770168	1.061	2971	5.803	4839765	1.077	414	0.809
CLK-FPGA11	4189622	1	455	1	4182726	0.998	768	1.688	4207699	1.004	2535	5.571	4777177	1.140	369	0.811
CLK-FPGA12	3387586	1	409	1	3368698	0.994	744	1.819	3376930	0.997	3007	7.352	3739517	1.104	338	0.826
CLK-FPGA13	3833106	1	441	1	3815718	0.995	822	1.864	3920965	1.023	3155	7.154	4320345	1.127	359	0.814
Average		1.000		1.000		1.023		1.826		1.057		7.936		1.122		0.806

In the target architecture, there is a highly unbalanced routing supply in the horizontal and vertical directions. Hence, after an initial global placement, the input netlist will be partitioned and the partitions will be reallocated according to the aspect ratio of the chip to reduce congestion in the first stage of our flow.

After partition reallocation, LUTs and FFs are packed into basic logic elements (BLEs) to reduce the inter-connections between sites in routing. After packing, the netlist is modified such that LUTs and FFs are replaced by BLEs.

An upper bound and lower bound based global placement is then performed on the modified netlist, where the wirelength is optimized. Our global placement engine consists of two stages where HPWL and routing congestion are optimized respectively. In the second stage, a congestion-driven global placement step will be performed to reduce the routing congestion if there is any. We use the machine learning method in Section V to predict the routing congestion instead of the original probabilistic model.

After global placement, the clock region planning introduced in Section III-A will be performed to ensure no violation of the clock region constraint.

In legalization, the global placement result is first legalized regardless of the half column constraint. The half column legalization introduced in Section III-B will then be performed to remove all the violations of the half column constraint. Finally, detailed placement at the BLE and site levels will be performed to further improve wirelength. Chain move is applied in both legalization and detailed placement to optimize displacement and HPWL respectively as mentioned in Section IV.

VII. Experimental Results

To evaluate our proposed method, the algorithms are implemented in C++. The experiments were performed on a 64-bit Linux workstation with Intel Xeon 3.7GHz CPU and 16GB memory, using the benchmarks provided by the ISPD 2016 Routability-Driven FPGA Placement Contest [23] and the ISPD 2017 Clock-Aware FPGA Placement Contest [22].

In our experiments, we use the architecture of the commercial device family, Xilinx UltraScale [27], for our comparative study. The device's aspect ratio and average spacing between blocks were determined based on the Xilinx UltraScale VU095 architecture, the latest 20nm FPGA chip. For fair comparison, all the placers were executed with a single thread. We set Vivado to be in the same configurations as in the ISPD 2016 and ISPD 2017 contest, which uses two threads in routing and limits the running time to 12 hours.

TABLE IV: Comparison of HPWL and running time (s) before and after applying the two-step clock constraint legalization (CCL).

Design	w/ CCL				w/o CCL			
	HPWL	ratio	Time	ratio	HPWL	ratio	Time	ratio
CLK-FPGA01	1582915	1	288	1	1582916.5	1.000	276	0.958
CLK-FPGA02	1577050.5	1	266	1	1577174.5	1.000	254	0.955
CLK-FPGA03	4059161.5	1	583	1	4060707.5	1.000	558	0.957
CLK-FPGA04	2716961	1	380	1	2717721.5	1.000	367	0.966
CLK-FPGA05	3532758.5	1	569	1	3533406.5	1.000	534	0.938
CLK-FPGA06	4485497.5	1	591	1	4486400.5	1.000	572	0.968
CLK-FPGA07	1708920	1	304	1	1708954	1.000	293	0.964
CLK-FPGA08	1355308	1	247	1	1354246.5	0.999	244	0.988
CLK-FPGA09	1946224.5	1	327	1	1945947.5	1.000	313	0.957
CLK-FPGA10	3505733	1	512	1	3506731.5	1.000	499	0.975
CLK-FPGA11	3270337.5	1	455	1	3270688.5	1.000	440	0.967
CLK-FPGA12	2592324	1	409	1	2593720.5	1.001	395	0.966
CLK-FPGA13	2927103	1	441	1	2926785.5	1.000	420	0.952
Average		1.000		1.000		1.000		0.962

TABLE III shows the routed wirelength and running time compared to the winners of the ISPD 2017 contest. Columns “WL” and “Time” list the wirelength and the computational time in seconds, respectively. Our proposed placement can generate legal and routable solutions for all the benchmarks. Compared to the 1st place winner of the ISPD 2017 contest, our average routed wirelength is about 2.3% better and our running time is 2× faster than theirs. Due to the efficient clock-aware placement framework, we can solve the largest design in the benchmarks which contains about 1 million cells in just 10 minutes.

TABLE IV shows that the running time and HPWL generated by our framework compared with that obtained by neglecting the clock constraints. Since a placement violating the constraints cannot be routed by Vivado, we can only compare the HPWL instead of the routed wirelength. Note that the only difference between these two frameworks is that the steps in Section III are applied in the first framework and all cell moves are not constrained by the clock related constraints. The results show that the HPWLs of these two frameworks are very close, which means that our proposed framework can generate a placement satisfying the clock constraints with very little overhead.

As discussed in Section V, we will use the unified trained global linear model as our congestion prediction model. According to the prediction from our model, our congestion-driven global placement is never invoked to reduce congestion in ISPD 2017 benchmarks since the designs are not congested. Hence, we use the ISPD 2016 benchmarks to test the quality of our congestion estimation model. TABLE V shows a comparison of routed wirelength generated by our framework before and after replacing the congestion estimation model in [15] with the proposed one. Our proposed congestion model can generate legal and routable solutions for all the benchmarks in

TABLE V: Routed wirelength comparison between different routing congestion estimation models.

Design	[15]		ML-based congestion model	
	WL	ratio	WL	ratio
FPGA01	350060	1	350802	1.002
FPGA02	635044	1	634700	0.999
FPGA03	3251264	1	3251721	1.000
FPGA04	5492214	1	5411107	0.985
FPGA05	9909270	1	9911182	1.000
FPGA06	6144522	1	6143973	1.000
FPGA07	9593240	1	9520252	0.992
FPGA08	8087931	1	8036647	0.994
FPGA09	12062928	1	12123865	1.005
FPGA10	6972278	1	7020054	1.007
FPGA11	10918250	1	10462601	0.958
FPGA12	7239553	1	7605996	1.051
Average		1		0.999

ISPD 2016 benchmarks. Compared to the original model, our model can lead to results of similar average routed wirelength. Although the result obtained by the machine learning approach is just marginally better than that by the probabilistic method, the machine learning model can predict the percentage of routing resources used "directly" while it's needed to find the correspondence between the predicted value and the real routing usage for the probabilistic method. We also need to set appropriate edge capacity values for the global router-based methods to generate good results.

VIII. Conclusion

Several previous works on routability-driven analytical placer [15]–[17] have been proposed for the Xilinx UltraScale FPGAs. However, they can only place design with single clock which is not practical nowadays. This work is based on the framework of a flat analytical placer for heterogeneous FPGAs [15]. In this paper, to meet the new requirements in UltraScale FPGA placement, we have proposed several techniques, i.e., a two-step clock constraint legalization, chain move, and machine learning-based congestion estimation. The experimental results show that our algorithm can achieve better quality and running time compared to the ISPD 2017 contest winners. Moreover, our machine learning-based congestion model can approximate the routing estimation of Vivado well and guide the placement to generate good results in ISPD 2016 benchmarks. Future work will include how to better spread different types of cells in the global placement phrase and to use actual routing congestion as golden results in machine learning, etc.

References

[1] A. Ling and J. Anderson, "The role of FPGAs in deep learning," in *ACM Symposium on FPGAs*, 2017, pp. 3–3.

[2] G. A. Constantinides, "FPGAs in the cloud," in *ACM Symposium on FPGAs*, 2017, pp. 167–167.

[3] J. Luu, J. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, T. Liu, N. Ahmed *et al.*, "VTR 7.0: Next generation architecture and CAD system for FPGAs," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 7, no. 2, p. 6, 2014.

[4] P. Maidee, C. Ababei, and K. Bazargan, "Timing-driven partitioning-based placement for island style FPGAs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 24, no. 3, pp. 395–406, 2005.

[5] M. Gort and J. H. Anderson, "Analytical placement for heterogeneous FPGAs," in *IEEE International Conference on Field Programmable Logic and Applications (FPL)*, 2012, pp. 143–150.

[6] M.-C. Kim, D.-J. Lee, and I. L. Markov, "SimPL: An effective placement algorithm," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 31, no. 1, pp. 50–60, 2012.

[7] T.-H. Lin, P. Banerjee, and Y.-W. Chang, "An efficient and effective analytical placer for FPGAs," in *ACM/IEEE Design Automation Conference (DAC)*, 2013, pp. 10:1–10:6.

[8] Y.-C. Chen, S.-Y. Chen, and Y.-W. Chang, "Efficient and effective packing and analytical placement for large-scale heterogeneous FPGAs," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2014, pp. 647–654.

[9] D. Xie, J. Xu, and J. Lai, "A new FPGA placement algorithm for heterogeneous resources," in *IEEE International Conference on ASIC (ASICON)*, 2009, pp. 742–746.

[10] M. Xu, G. Gréwal, and S. Areibi, "StarPlace: A new analytic method for FPGA placement," *Integration, the VLSI Journal*, vol. 44, no. 3, pp. 192–204, 2011.

[11] Y. Xu and M. A. S. Khalid, "QPF: efficient quadratic placement for FPGAs," in *IEEE International Conference on Field Programmable Logic and Applications (FPL)*, 2005, pp. 555–558.

[12] D. T. Chen, K. Vorwerk, and A. Kennings, "Improving timing-driven FPGA packing with physical information," in *IEEE International Conference on Field Programmable Logic and Applications (FPL)*, 2007, pp. 117–123.

[13] M. Tom, D. Leong, and G. Lemieux, "Un/DoPack: re-clustering of large system-on-chip designs with interconnect variation for low-cost FPGAs," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2006, pp. 680–687.

[14] W. Feng, J. Greene, K. Vorwerk, V. Pevzner, and A. Kundu, "Rent's rule based FPGA packing for routability optimization," in *ACM Symposium on FPGAs*, 2014, pp. 31–34.

[15] C.-W. Pui, G. Chen, W.-K. Chow, K.-C. 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 *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2016, pp. 67:1–67:8.

[16] W. Li, S. Dhar, and D. Z. Pan, "UTPlaceF: A routability-driven FPGA placer with physical and congestion aware packing," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2016, pp. 66:1–66:7.

[17] R. Pattison, Z. Abuowaimer, S. Areibi, G. Gréwal, and A. Vannelli, "GPlace: A congestion-aware placement tool for ultrascale FPGAs," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2016, pp. 68:1–68:7.

[18] Y. Zhuo, H. Li, and S. P. Mohanty, "A congestion driven placement algorithm for fpga synthesis," in *IEEE International Conference on Field Programmable Logic and Applications (FPL)*, 2006, pp. 1–4.

[19] W.-T. J. Chan, P.-H. Ho, A. B. Kahng, and P. Saxena, "Routability optimization for industrial designs at sub-14nm process nodes using machine learning," in *ACM International Symposium on Physical Design (ISPD)*, 2017, pp. 15–21.

[20] Z. Qi, Y. Cai, and Q. Zhou, "Accurate prediction of detailed routing congestion using supervised data learning," in *IEEE International Conference on Computer Design (ICCD)*, 2014, pp. 97–103.

[21] W. T. J. Chan, Y. Du, A. B. Kahng, S. Nath, and K. Samadi, "Beol stack-aware routability prediction from placement using data mining techniques," in *IEEE International Conference on Computer Design (ICCD)*, 2016, pp. 41–48.

[22] S. Yang, C. Mulpuri, S. Reddy, M. Kalase, S. Dasasathyan, M. E. Dehkordi, M. Tom, and R. Aggarwal, "Clock-aware FPGA placement contest," in *ACM International Symposium on Physical Design (ISPD)*, 2017, pp. 159–164.

[23] S. Yang, A. Gayasen, C. Mulpuri, S. Reddy, and R. Aggarwal, "Routability-driven FPGA placement contest," in *ACM International Symposium on Physical Design (ISPD)*, 2016, pp. 139–143.

[24] Y. Lin, B. Yu, X. Xu, J.-R. Gao, N. Viswanathan, W.-H. Liu, Z. Li, C. J. Alpert, and D. Z. Pan, "MrDP: Multiple-row detailed placement of heterogeneous-sized cells for advanced nodes," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2016, pp. 7:1–7:8.

[25] T. Lin, C. Chu, J. R. Shinnerl, I. Bustany, and I. Nedelchev, "POLAR: A high performance mixed-size wirelengh-driven placer with density constraints," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 34, no. 3, pp. 447–459, 2015.

[26] S. Goto, "An efficient algorithm for the two-dimensional placement problem in electrical circuit layout," *IEEE Transactions on Circuits and Systems I*, vol. 28, no. 1, pp. 12–18, 1981.

[27] "Xilinx UltraScale Architecture," https://www.xilinx.com/products/technology/ultrascale.html.