This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2017.2778058, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

RippleFPGA: Routability-Driven Simultaneous Packing and Placement for Modern FPGAs

Gengjie Chen, Chak-Wa Pui, Wing-Kai Chow, Ka-Chun Lam, Jian Kuang, Evangeline F. Y. Young, and Bei Yu

***Abstract*—As a good trade-off between CPU and ASIC, FPGA is becoming more widely used in both industry and academia. The increasing complexity and scale of modern FPGA, however, impose great challenges on the FPGA placement and packing problem. In this paper, we propose RippleFPGA to solve the packing and placement simultaneously through a set of novel techniques, such as (i) smooth stair-step flow, (ii) implicit packing similar to ASIC legalization, and (iii) two-level detailed placement. To enable the flow, a generic, efficient and false-alarm-free legality checking method is also proposed. Besides, due to the insufficiency of ASIC-like congestion alleviation methods, some FPGA-routing-architecture-aware optimization techniques are proposed to improve the routability. When evaluated by ISPD 2016 Contest benchmarks, RippleFPGA has 5.1% better routed wirelength and 5.5× speedup compared to all the state-of-the-art FPGA placers.**

I. Introduction

Field-programmable gate array (FPGA) is an integrated circuit (IC) designed to be reconfigurable by users after manufacturing. As technology scaling slows down, the central processing unit (CPU), which supports high-level programming languages, is experiencing difficulties in boosting performance and power efficiency. Meanwhile, application specific IC (ASIC) is becoming extremely expensive to design and manufacture. As a good trade-off between CPU and ASIC, FPGA is growing in many areas such as communications, industrial control, aerospace, consumer electronics, and artificial intelligence [1]–[3].

In order to meet the needs of the applications, FPGA is also evolving rapidly and over time becoming bigger, faster and more like ASICs (e.g., the number of logic cells in a modern FPGA reaches 5.5 million [4]). The complexity and size together with the FPGA-specific constraints, impose many challenges on the FPGA packing and placement [5], which consume roughly half of the compilation time [6].

A typical FPGA CAD flow is shown in Fig. 1. After logic synthesis, the netlist consists of lookup tables (LUTs), flip-flops (FFs), digital signal processors (DSPs), random access memories (RAMs) and I/O pads. During *packing*, LUTs and FFs are grouped together into *basic logic elements (BLEs)* and then further clustered into *configurable logic blocks (CLBs)*. After packing, *placement* legally maps all the CLB/DSP/RAM/IO blocks onto the FPGA and optimizes some metrics (e.g., wirelength and routability). Note that besides half-perimeter wirelength (HPWL), routability is also an important objective in placement. For one thing, an unroutable placement, even with perfect HPWL, is useless. For another, a difficult-to-route placement not only requires very long routing time but also incurs significant detour and thus increase in routed wirelength, leading to timing and power problems.

The work described in this paper was partially supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CUHK14206015). The preliminary version has been presented at the International Conference on Computer-Aided Design (ICCAD) in 2016. The authors are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, N.T., Hong Kong.

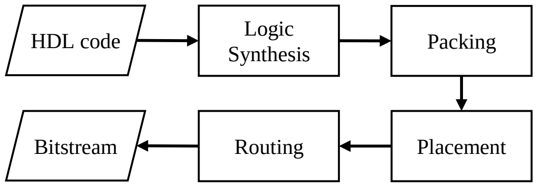


Fig. 1: The typical FPGA CAD flow.

A. Previous Work

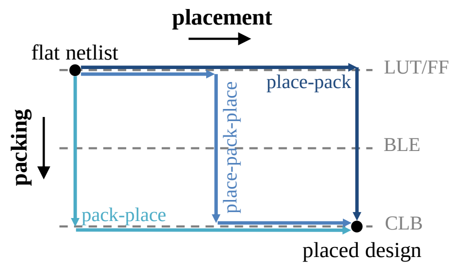
1) Packing for FPGAs: Previous approaches for FPGA packing can be loosely categorized into: (i) seed-based approach, (ii) clustering-based approach, and (iii) partitioning-based approach. We follow the convection in ASIC that clustering refers to a bottom-up process that groups several cells together, and partitioning means a top-down process that breaks the whole circuit into several sub-circuits [7].

Seed-based approach iteratively selects a cell as a seed, and then keeps merging an unmerged cell (with highest attraction) into the seed until the capacity limit is reached. It was extensively used in academia. By setting the attraction function differently, VPack [8], T-VPack [9], RPack [10], iRAC [11], MO-Pack [12] AAPack [13] and ALMPack [14] achieve various objectives. The seed-based approach is an efficient greedy heuristic but lacks a global view. It also tends to pack too densely by exhausting the capacity of a CLB [15].

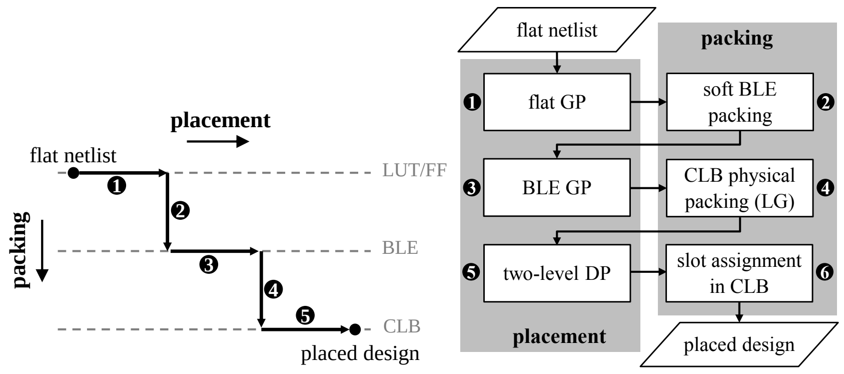
For clustering, there are mainly two types of methods used in ASIC placement. (1) One is a greedy heuristics that visits cells in an arbitrary order and find a neighbor with the highest attraction for merging, such as edge coarsening (EC) [16], [17] and FirstChoice (FC) [18]. (2) Another approach is priority-queue-based, which chooses the best pair among all to combine iteratively and update the queue correspondingly, such as edge separability based clustering (ESC) [19], BestChoice (BC) [20] and SafeChoice (SC) [21]. Among them, BC, which prefers stronger connection but smaller area, is the most popular with application in many ASIC placers (e.g., FastPlace [22]). For FPGA packing, BC is adopted in HDPack [23], UTPlaceF [24] and [25]. Note that a direct application of BC to packing will result in a large number of clusters with half utilization, i.e., a loose packing. A hybrid method (e.g., combined with seed-based approach) is usually needed for avoiding the loose packing.

Netlist partitioning (refer to [26] for a survey) is a necessary process for partitioning-based placers in ASIC. PPFF [27] applies this placement paradigm to FPGA, where packing is done implicitly within placement. Besides, an explicit way of packing is recursive partitioning followed by adjusting illegal CLBs, e.g., the work [28] and PPack [29]. Since netlist partitioning is time-consuming, the packing approaches based on it are slow. As another way of partitioning-based packing, GPlace-flat [30] employs recursive geometric bi-partitioning on a flat placement. To handle the complex design rules, they ignore the precious LUT-FF connections when moving cells globally between two partitions. In general, the legality checking in partitioning-based packing is more complicated than that in bottom-up approaches (i.e., seed-based and clustering-based ones).

0278-0070 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.



(a) Three types of flows in previous work.



(b) Our proposed flow of simultaneous packing and placement.

Fig. 2: Different FPGA packing and placement flows.

2) Placement for FPGAs: Same as ASIC, placement in FPGA can be classified into three categories: (i) simulated annealing, (ii) partitioning-based approaches, and (iii) analytical approaches. The most famous academic FPGA CAD tool VPR/VTR [31], [32] is a representative of simulated-annealing-based methods. Partitioning-based FPGA placers like the aforementioned PPFF recursively partition a design and place it hierarchically.

In ASIC placement, simulated annealing and partitioning-based approaches are outperformed by analytical ones (e.g., [33]–[35]), as the number of placement instances substantially increases nowadays [36]. The situation is the same in FPGA placement. Not only the industrial placers migrate to analytical approaches [37], but also many academic analytical FPGA placers have appeared and shown very good result. They include quadratic placers (e.g., QPF [38], StarPlace [39], HeAP [40], UTPlaceF [24], GPlace [30]), nonlinear optimization placers (e.g., [25], [41], [42]) and others (e.g., CAPRI [43] using a formulation of binary quadratic assignment).

3) Packing and placement cooperation: In the convectional FPGA CAD flow (Fig. 1), the separation of netlist-based packing and CLB-level placement reduces the complexity of each subproblem. But the drawback is two-fold. (1) Without prior placement, efficient packing algorithms suffer from the limitation that only local connectivity information can be considered [19], [21]. Moreover, to incorporate other metrics (e.g., timing, power, routability), the netlist-level estimation is very inaccurate [44]. (2) After packing, placement is greatly confined to a fixed circuit structure [45], [46]. The wirelength optimization is thus restricted and suboptimal. To consider other objectives, which may require BLE-level movement, the restriction is highly undesirable.

Various methods are proposed to overcome the drawback. One way is to perform bottom-up packing according to the information obtained from an initial placement, similar to physical clustering in the ASIC world (e.g., physical SC [21], FastPlace [22]). To handle congestion better, Un/DoPack [47] redoes the packing and placement based on the trial pack-place result. Bringing this idea from simulated annealing to analytical placement, UTPlaceF [24] and GPlace-pack [30] pack after a flat initial placement. With the same idea, HDPack [23] uses the physical information to improve timing. As another way, GPlace-flat [30] conducts implicit packing after flat placement. A pictorial comparison of the three types of flows, i.e., (i) the conventional one (pack-place), (ii) packing with physical information (place-pack-place), and (iii) flat placement followed by legalization (place-pack), are shown in Fig. 2(a). Moreover, in detailed placement, significantly larger solution space can be explored by breaking the packed result and allowing BLE move [45], [46].

4) Routability in FPGA placement: The general approach for improving placement routability, which inflates/depopulates cells according to a congestion estimation at g-cell level, has been successfully applied in both ASIC and FPGA fields [24], [30], [33]. Besides, being programmable, the routing architecture in FPGA is quite different from that in ASIC [1]. CAPRI [43] starts considering the segmented routing of FPGA by graph embedding, but the objective is timing and congestion actually becomes worse in some cases. Authors in [42] propose a smooth function to approximate the discrete routing cost under their nonlinear placement framework. As the major weakness, both works are not scalable for large designs due to the complicated problem formulation.

B. Motivations

The key issues in previous work about FPGA packing and placement that motivate our work are as follows.

1) The artificial separation of packing and placement stages may be undesirable. However, this is a chicken-and-egg problem: without placement information, a good packing solution is difficult to obtain; without packing, there is no way to do a legal placement. Aware of this, some previous works start to blur the boundary between the two. It is of particular interest whether stronger integration between packing and placement is possible.
2) The bottom-up packing (including seed-based and clustering-based one) is efficient and friendly for considering design rules, in contrast to the partitioning-based one. But even with physical information, the process is still based on the pairwise attraction, which concentrates on local situation. Besides, it requires careful engineering to control the packing density. Many methods need to iteratively update the packing parameters until number of packed CLBs is not overflowed [24], [30]. On the other hand, the partitioning-based approach has more global view and controls the packing density implicitly, but is time-consuming and has difficulty in legality checking. Therefore, it is highly desirable to combine the strengths of the both together.
3) As FPGA technology advances, more flexible configuration within CLB is allowed to improve the performance [48]. This, however, makes the legality checking in CLB much more complicated. It is a major motivation for a separated packing stage [14], [23]. Therefore, an efficient and effective legality checking for packing, which can enable many high-level techniques, is in need.
4) To better handle routability, it is also necessary to consider the routing issues special to FPGAs in an effective and efficient way.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2017.2778058, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

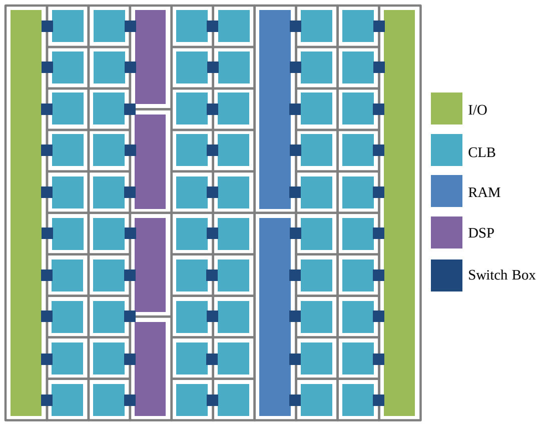


Fig. 3: Illustration of Xilinx UltraScale architecture.

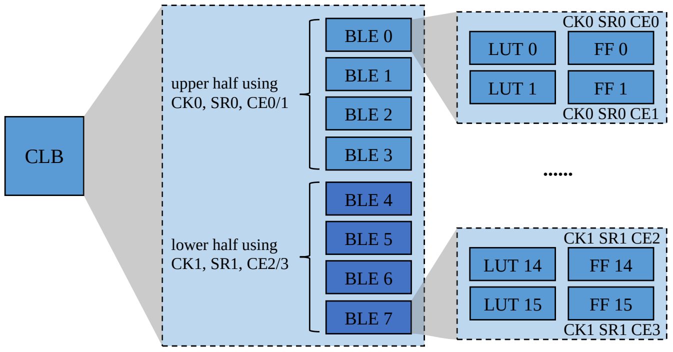


Fig. 4: Illustration of CLB and BLE in Xilinx UltraScale.

C. Contributions

In this paper, we propose a routability-driven simultaneous packing and placement engine called RippleFPGA for modern FPGAs. Our major contributions are summarized as follows.

1) We propose a stair-step framework that interleaves the packing and placement stages (briefly illustrated by Fig. 2(b)). It not only makes the optimization flow more smooth and integrated, but also enables fast feedback of accurate estimation (e.g., routability) from a final state to an intermediate state.
2) We design an implicit CLB packing scheme, which is similar to ASIC legalization. By approximating the analytical global placement directly, it can implicitly control the packing density and directly reflect objectives considered in placement.
3) We propose a generic, efficient and false-alarm-free CLB legality checking and slot assignment methods for the modern FPGA with complex design rules, which enable the implicit CLB packing and the BLE move in detailed placement.
4) Under this framework, we propose some routing-architecture-aware techniques including partition allocation, CLB slot assignment and alignment optimization, together with the ASIC-like congestion alleviation methods in global and detailed placement stages.

The remainder of this paper is organized as follows. Section II gives an introduction to our target FPGA architecture as well as the problem formulation. Section III provides an overview of our flow. Section IV and V then introduce packing and placement algorithms in detail respectively. Section VI describes our speedup techniques. Section VII shows the experimental results, and we finally conclude in Section VIII.

II. Preliminaries

A. Target Architecture

Our target architecture is Xilinx Ultrascale VU095 [5], a representative of modern FPGAs. Its layout is illustrated in Fig. 3. There are a large number of CLB/RAM/DSP/IO blocks of various sizes. Each block can only be placed into sites of its own type.

The internal structure of CLB is shown by Fig. 4. Each CLB can contain at most eight BLEs, which are divided into two *halves*. Within a BLE, there are at most two LUTs and two FFs under the following constraints.

1) For the two LUTs, if both are used, the total number of distinct inputs should be no more than five. If only one LUT is occupied, there is no restriction on it. Note that the input of a single LUT ranges from two to six.
2) For the two FFs, their clock enable (CE) signals can differ, but the clock (CK) and set/reset (SR) signals need to be identical.
3) CK, SR and the two CEs for the four BLEs in a half should be the same.

The four FFs that are required to use the same CE will be referred as an *FF group*. Note that there are thus totally four FF groups in a CLB, with two in each half.

For routing, all the external connections of blocks go to the dedicated switch boxes first, which are then connected by pre-defined routing segments. Inside a CLB block, two types of nets can be physically achieved without going through the switch box: (i) inputs shared by the two LUTs in the same BLE require no wire, and (ii) an LUT drives an FF within the same BLE (referred as *LUT-FF connection* in this paper), which is implemented by internal exclusive wires. In terms of routing demand as well as delay, the preference of connections should be: (i) requiring no switch box, (ii) assigning pins to the same switch box, and (iii) assigning pins to different switch boxes. This preference is the major consideration of packing.

Note that the architecture of Xilinx Ultrascale VU095 is very typical in modern FPGAs. Our methodologists are also generic, which can thus be easily adapted to other commercial families or FPGAs from other vendors.

B. Problem Formulation

The wirelength and routability evaluation of a placed design can be combined into a single metric: the routed wirelength. Note that to have routed wirelength, the placement should be routable first. The problem formulation is thus as follows.

Problem 1. *(Routability-Driven FPGA Placement) Given a netlist of LUTs, FFs, DSPs, RAMs and IOs, and the target FPGA architecture, decide the sites and slots of cells to minimize the routed wirelength.*

III. Flow Overview

In ASIC placement, there are typically three stages: (i) *global placement* (GP), (ii) *legalization* (LG), and (iii) *detailed placement* (DP) [33]. GP gives the location of each cell across the chip to optimize wirelength as well as others under the cell density constraint. LG aligns cells to the placement sites without overlapping. Finally, DP further improves the solution by relocating cells.

RippleFPGA integrates packing and placement by a stair-step flow, which is divided six stages (Fig. 2(b)):

❶ **Flat GP**: An initial GP is conducted on the flat netlist to minimize HPWL. At an early point of this stage, netlist partitions are globally allocated to fit the unbalanced horizontal and vertical routing resources.

0278-0070 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2017.2778058, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

❷ **Soft BLE packing**. LUTs and FFs are packed into BLEs. The BLEs generated here are *soft* since in later stages they can not only be merged with each other but also be split. Only LUTs and FFs with strong mutual connection and of small distance apart are packed together.

❸ **BLE GP**. The second GP is on the coarsened BLE-level netlist. Some later iterations are congestion-driven. In the middle of this GP, DSPs and RAMs are legalized and fixed in their locations due to their large sizes and high connectivity.

❹ **Implicit CLB packing (LG)**. Soft BLEs are packed into CLB implicitly, based on an efficient legality checker and guided by Tetris-like LG [49].

❺ **Two-level DP**. In DP, block-level and BLE-level move are conducted to further improve HPWL as well as routability.

❻ **Slot assignment in CLB**. Lastly, LUT/FF slots within CLBs are assigned to maximize the switch-box-free connection.

This smooth star-step framework helps us to achieve superior results over others (e.g, UTPlaceF [24] and GPlace-flat [30], representatives of place-pack-place and place-pack flows).

IV. Packing Algorithms

This section focuses on packing methodologies (i.e., Stages ❷ ❹ ❻), while placement algorithms will be introduced in the next section.

A. Max-Weight-Matching-Based BLE Packing

In Stage ❷, LUTs and FFs are packed into soft BLEs according to both the physical information obtained from the initial GP and the connectivity. Only LUTs and FFs with strong mutual connection and small distance are packed together. A simple BLE structure, where each BLE contains only one LUT and one FF, is assumed in many previous work. In UltraScale, the two LUTs and two FFs with sophisticated constraints in a BLE pose great challenges to packing.

In general, our BLE packing consists of two steps. First, FFs driven by an LUT are considered to be merged with the LUT. Second, two LUTs (and their attached FFs) are considered to be merged together. Besides saving spaces, the other objective of BLE packing is to increase the local switch-box-free connections (i.e., the shared LUT inputs and the LUT-FF connections), which is significant for routability and wirelength. For the simplicity of illustration, an LUT or FF may also be regarded as a single-cell BLE from now on.

1) LUT-FF connecting: LUT-FF connection is precious as mentioned in Section II-A. In this step, we try to merge an LUT with the FFs driven by it. FFs with the Manhattan distance from the LUT exceeding a threshold d_{max} (which is empirically set to 20 switch-box units) will be ignored, to avoid a huge disturbance to wirelength. An LUT may drive multiple FFs. In this case, FFs are sorted by the distance from the LUT and the first two legal FFs are selected.

2) LUT pairing: After the first step, each BLE contains at most one LUT. The single-LUT BLEs are considered to be merged together, in order to save spaces and increase input sharing. We construct a weighted graph $G_1(V_1, E_1; w_1)$, where a vertex $v \in V_1$ represents a single-LUT BLE and an edge $(u, v) \in E_1$ represents a candidate pairing. A pairing is a candidate if and only if: (1) the two LUTs have no conflict. (2) the attached FFs have no conflict. (3) the distance between the two LUTs is smaller than d_{max}. (4) the number of shared inputs between the two LUTs is not smaller than a threshold (which is empirically set to 2).

Furthermore, we let the weight of an edge (u, v) be the number of inputs that u and v share. A maximum weight matching problem on G_1 is then solved, where each matching indicates that the two corresponding BLEs should be merged into one. Note that the edge weight here does not incorporate physical distance according to two considerations. (1) LUT pairs with large distance are already forbidden, so ignoring distance leads to negligible wirelength loss. (2) Within a BLE, shared inputs between two LUTs directly imply precious switch-box-free connections, while for LUTs with close GP locations, it makes no difference by assigning them (i) to the same BLE, (ii) to different BLEs but the same CLB, or (iii) even to different CLBs sharing the same switch box.

In short, three types of soft BLEs are obtained from the above BLE packing: (i) BLEs containing both LUTs and FFs, (ii) BLEs containing LUTs only, (iii) BLEs containing FFs only. The packing result here is a guidance instead of a restriction for the following stages. The LUTs and FFs that merged into the same soft BLE are very likely to stay in the same BLE in the end, but separation is possible if needed. Besides, soft BLEs can also merge with each other to save space, without reducing the switch-box-free connections.

B. Max-Cardinality-Matching-Based CLB Legality Checking

In the design flow, we need to repeatedly query whether several given soft BLEs can be legally placed into a CLB. Only with an efficient legality checker, the implicit CLB packing guided by Tetris-like LG (Stage ❹) and the BLE move in DP (Stage ❺) are possible. In most scenarios, checking is done incrementally, which is formally defined as follows.

Problem 2. *(Incremental Legality Checking). Given a set of soft BLEs which can be legally placed into a CLB, and another soft BLEs, decide whether they can be legally placed into a CLB together.*

In [50], we use a dedicated finite state machine with many tedious case discussion to check legality and assign LUT/FF slots. There, two (or even three or four) soft BLEs can be merged into one, but LUT-FF connections existing in the input BLEs are not allowed to be broken. To avoid the number of states growing exponentially, some merging between BLEs is decided greedily, which in some cases leads to false alarms (e.g., results sensitive to order of BLEs added).

In this paper, a scheme that separates the legality checking and the slot assignment is proposed. The improvement is in four aspects. (1) The checking is now optimal without false alarm. (2) By deferring the slot assignment, the legality checking, which is frequently invoked, is more efficient. (3) Without tedious discussion of the complicated design rules, the scheme is generic and also easy to implement. (4) By allowing breaking LUT-FF connections, the success rate and the number of shared LUT inputs are increased. Note that even though LUT-FF connections can be broken, the actual number of breaking is tiny.

The legality is checked incrementally by Algorithm 1. By ignoring LUT-FF connections temporarily, LUTs and FFs inside the new BLE are checked individually (lines 3–13). When trying adding an LUT (lines 15–29), optimal solution will be found by max-cardinality graph matching (lines 27-28) after the greedy method (lines 16–26) fails. In the LUT pair graph for matching (line 27), each vertex is an LUT while edges represent valid LUT pairs. To add an FF (lines 30–38), FF groups with the same CK, SR and CE will be exhausted first to save space. In our implementation, some redundant checking (e.g., FG_0 and FG_1 have the same CK and SR) is avoided by two-level loops, but here the idea is shown by a single loop.

Suppose there are n_l LUTs and n_f FFs in a CLB, then the time complexity of Algorithm 1 is $\mathcal{O}(n_l^{2.5} + n_f)$. To be more specific, backup and recover (lines 2 and 10) take $\mathcal{O}(n_l + n_f)$; the matching in `AddLUT` needs $\mathcal{O}(n_l^{2.5})$ [51]; `AddFF` is $\mathcal{O}(1)$ because of checking up to four FF groups. The legality checking in [50] is also polynomial

0278-0070 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

Algorithm 1 Incremental Legality Checking

```
Require: a CLB $c$ containing several BLEs, another BLE $b$;
Ensure: whether $b$ can be added;
 1: Local variables: set of single LUTs $LS$, set of LUT pairs $LP$, FF group $FG_i$ ($i = 0, 1, 2, 3$);
 2: Backup $LS, LP, FG_i$;
 3: for LUT/FF $v$ in $b$ do
 4:     if $v$ is an LUT then
 5:         $succ \leftarrow$ AddLUT($v$);
 6:     else
 7:         $succ \leftarrow$ AddFF($v$);
 8:     end if
 9:     if !$succ$ then
10:         Recover $LS, LP, FG_i$;
11:         return false;
12:     end if
13: end for
14: return true;

15: function AddLUT($v$)
16:     for $u \in LS$ do;
17:         if $(u, v)$ is a valid LUT pair then;
18:             $LS \leftarrow LS \backslash \{u\}$;
19:             $LP \leftarrow LP \cup \{(u, v)\}$;
20:             return true;
21:         end if
22:     end for
23:     $LS \leftarrow LS \cup \{v\}$;
24:     if $|LS| + |LP| \leq 8$ then
25:         return true;
26:     end if
27:     Solve max-cardinality matching on LUT pair graph;
28:     return $|LS| + |LP| \leq 8$;
29: end function

30: function AddFF($v$)
31:     for $i := 0$ to 3 do
32:         if $|FG_i| = 0$ or ($v$ shares CK, SR & CE with $FG_i$ and $|FG_i| < 4|$) then
33:             $FG_i \leftarrow FG_i \cup \{v\}$;
34:             return true;
35:         end if
36:     end for
37:     return false;
38: end function
```

to n_l and n_f, but the constant here is smaller by very simple checking, which will be evidenced by the experiment in Section VII.

C. Refined-Tetris-Based BLE Legalization

In Stage ❹, CLB packing is implicitly conducted under a LG framework. LG needs to consider the legality of a move and to minimize the disturbance to GP simultaneously. Unlike DSP/RAMs, a CLB site can contain multiple BLEs under the complicated legalization rules.

Based on the legality checking method described in Section IV-B, our BLE LG algorithm (Algorithm 2) refines Tetris, which legalizes cells sequentialy without affecting previously legalized cells. Apart from the displacement, which is the only objective of Tetris, HPWL is also captured in the LG of RippleFPGA. Candidate sites under the restriction of displacement are first obtained (line 6). A legal site is then found by attempting the candidate sites in increasing HPWL order (lines 10–15). If all attempts fail in this round, candidate sites with larger displacement are tried. To encourage HPWL optimization, the number of candidates sites in each round is a relatively large number. As this algorithm is highly flexible in choosing the secondary object (HPWL in ours), it can also be adapted for optimizing other objectives like timing, power, etc.

If treating the bounded n_l and n_f as constants, the time complexity of Algorithm 2 depends on the final displacement d_{max}.

Algorithm 2 Refined-Tetris-Based BLE LG

```
Require: BLEs $B$ with their GP locations, min number of candidate sites in a round $n_{cand}$;
Ensure: The CLB site that each BLE belongs to;
 1: for $b \in B$ do
 2:     Displacement $d \leftarrow 0$;
 3:     while $b$ is not placed do
 4:         Candidate sites $S \leftarrow \emptyset$;
 5:         while $|S| < n_{cand}$ do
 6:             Add CLB sites with displacement $d$ to $S$;
 7:             $d \leftarrow d + 1$;
 8:         end while
 9:         Sort $S$ by ascending HPWL;
10:         for $s \in S$ do
11:             if $b$ can be assigned to $s$ (by Algorithm 1) then
12:                 Assign $b$ to $s$;
13:                 Break;
14:             end if
15:         end for
16:     end while
17: end for
```

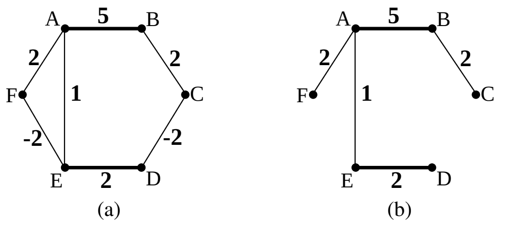


Fig. 5: Max-weight matching on (a) the original graph G_2 and (b) the graph without non-positive edges G_2'.

In each round with displacement d, $|S| = \mathcal{O}(d)$ and the runtime is $\mathcal{O}(d \log d)$ due to the sorting (line 9). The total time is thus $\mathcal{O}(\sum_{d=0}^{d_{max}} d \log d) = \mathcal{O}(d_{max}^2 \log d_{max})$. Note that d_{max} is mostly very small in practice.

D. ILP-Based Slot Assignment within CLB

After repeatedly checking legality and adding BLE during LG and DP stages, the CLB that each BLE belongs to is known, but the slot for each LUT/FF still needs to be determined. In our flow, the slots are assigned in Stage ❻ after all cell movements are settled. Besides legality, the other target in slot assignment is to maximize the shared LUT inputs and the LUT-FF connections, which is similar to BLE packing. In general, slots are assigned by two steps. First, the LUT pairs are modified to consider the above two connections instead of legality only. Second, slots for the LUT pairs and FFs are determined to maximize the LUT-FF connections.

1) LUT Pairing: Different from BLE packing, LUT pairing in slot assignment is decided firstly, since legality is a hard constraint now. Not only shared LUT inputs but also LUT-FF connections are considered in LUT pairing. We construct a weighted graph $G_2(V_2, E_2; w_2)$ again, where a vertex $v \in V_2$ represents an LUT and an edge $(u, v) \in E_2$ represents a legal LUT pair. The edge weight $w_2(u, v)$ between LUTs u and v is set as follows:

$$w_2(u, v) = NSI(u, v) + (PNF(u, v) - PNF(u) - PNF(v)), \quad (1)$$

where NSI is the number of shared inputs and PNF is the potential number of FFs for an LUT (pair). PNF can be zero, one or two, which is the maximum number of FFs that (i) are in this CLB, (ii) are logically driven by this LUT (pair), and (iii) have no type conflict (if PNF is two).

After constructing G_2, general maximum-weight matching algorithm does not work because of negative-weight edges and constraint

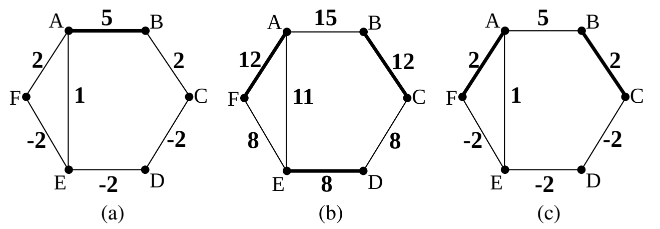


Fig. 6: Max-weight matching on (a) the original graph G_2 and (b) the graph with penalized edges. (c) Max-weight matching with at least two edges on G_2.

C on cardinality. The cardinality constraint is due to the limited capacity of CLB (eight BLEs at most), which is the major difficulty of the problem. By removing the non-positive edges and the cardinality constraint first, the maximum-weight matching can be easily solved. If the result is legal, it is optimal by Theorem 1. Fig. 5 shows an example. Suppose at least two matchings are required (i.e., $C = 2$). By ignoring non-positive edges CD and EF in G_2 and the cardinality constraint, it becomes easy to obtain max-weight matching $M = \{AB, DE\}$ on the resulted graph G_2'. Since $|M| = 2 \geq C$, M is the optimal constrained matching on G_2.

Theorem 1. *For a graph $G_2 = (V_2, E_2; w_2)$, and a graph induced by its positive edges $G_2' = G_2[\{e|e \in E, w_2(e) > 0\}]$, if the maximum-weight matching M on G_2' satisfies $|M| \geq C$, then M is also the maximum-weight matching on G_2 under constraint $|M| \geq C$.*

Proof. A negative edge of G_2 only decreases the matching weight and thus cannot occur in the maximum-weight matching. Therefore, G_2 and G_2' have the same maximum-weight matching. Since the cardinality constraint only makes solution space smaller, optimal matching on G_2 without constraint is the upper bound of matching on G_2 with constraint. □

If M violates the cardinality constraint, we add a constant P to each edge weight as penalty, by the idea of Lagrangian relaxation. Since it will be computationally impractical to iteratively update P, P is set large enough. Note that $P = \sum_{e \in E_2} |w_2(e)|$ is sufficient for guaranteeing the legality (i.e. maximum cardinality) because of Theorem 2. Then, pairs with negative weight are split in descending order until the cardinality constraint is not violated. An example ($C = 2$) is in Fig. 6. The max-weight matching without constraint on G_2 is $M = \{AB\}$, which is illegal. By adding sufficient penalty to edges, the max-weight matching on Fig. 6(b) is $\{AF, BC, DE\}$. The solution is legal now but can be improved by excluding negative edge DE, as Fig. 6(c) shows.

Theorem 2. *For a graph $G_2 = (V_2, E_2; w_2)$, define $G_2'' = (V_2, E_2; w_2'')$, where $w_2''(e) = w_2(e) + P$ with $P = \sum_{e \in E_2} |w_2(e)|$. The maximum-weight matching M'' of G_2'' is also a maximum-cardinality matching of G_2''.*

Proof. Suppose it is false and the maximum-cardinality matching is M^* with $|M^*| > |M''|$. Then $w_2''(M^*) - w_2''(M'') = (|M^*| - |M''|) \cdot P + w_2(M^*) - w_2(M'') \geq P + w_2(M^*) - w_2(M'') > P - \sum_{e \in E_2} |w_2(e)| = 0$. Therefore, M'' is not a maximum-weight matching, which is a contradiction. □

2) LUT-FF Connecting: Now, there are some LUTs (or LUT pairs) and FFs. For *candidate* LUT-FF connections, the preferred action is to assign the corresponding LUTs and FFs to the same BLE (i.e., to *achieve* the connections). However, different candidates may conflict with each other. The problem is formally stated as follows.

Problem 3. *(LUT-FF Connecting in CLB). Given some LUT pairs and FFs which can be legally placed into a CLB, decide the specific legal slots for them to maximize the achieved LUT-FF connection.*

The problem can be solved by integer linear programming (ILP). Suppose the set of LUT pairs is L and the set of FFs is F. Binary variables $x_{l,i}$ represents whether LUT pair $l \in L$ is assigned to the i-th half, $y_{f,i}$ represents whether FF $f \in F$ is assigned to the i-th FF group. Let $C = \{(l, f)\}$ be the set of candidate LUT-FF connections where LUT pair l drives FF f. The decision variable $z_{l,f}$ indicates whether $(l, f) \in C$ is achieved. There are then the following legality constraints.

- Each LUT pair or FF has one and only one assignment:

$$\sum_{i=0}^{1} x_{l,i} = 1, \qquad \forall l \in L, \tag{2a}$$

$$\sum_{i=0}^{3} y_{f,j} = 1, \qquad \forall f \in F. \tag{2b}$$

- Each half or FF group has limited capacity:

$$\sum_{l \in L} x_{l,i} \leq 4, \qquad \forall i \in \{0, 1\}, \tag{3a}$$

$$\sum_{f \in F} y_{f,j} \leq 4, \qquad \forall j \in \{0, 1, 2, 3\}. \tag{3b}$$

- FFs with different CK, SR and CE are conflicted:

$$y_{f,j} + y_{f',j} \leq 1, \quad \forall (f, f') \text{ with different CK/SR/CE}, \; \forall j \in \{0, 1, 2, 3\}, \tag{4a}$$

$$y_{f,j} + y_{f',j'} \leq 1, \quad \forall (f, f') \text{ with different CK/SR}, \; \forall (j, j') \in \{(0, 1), (2, 3)\}. \tag{4b}$$

- $x_{l,i}$ and $y_{f,j}$ is binded if the LUT-FF connection is achieved. That is, $z_{l,f} \rightarrow (x_{l,0} \wedge (y_{f,0} \vee y_{f,1})) \vee (x_{l,1} \wedge (y_{f,2} \vee y_{f,3}))$. By introducing auxiliary binary variables $s_{l,f,0}$ and $s_{l,f,1}$, it can be expressed as:

$$s_{l,f,i} \leq x_{l,i}, \qquad \forall i \in \{0, 1\}, \tag{5a}$$

$$s_{l,f,i} \leq y_{f,2i} + y_{f,2i+1}, \qquad \forall i \in \{0, 1\}, \tag{5b}$$

$$z_{l,f} \leq s_{l,f,0} + s_{l,f,1}, \qquad \forall (l, f) \in C. \tag{5c}$$

- Due to the limited BLE capacity, from an FF group to an LUT pair, there can be only one achieved connection:

$$z_{l,f} + z_{l,f'} + y_{f,j} + y_{f',j} \leq 4, \forall (l, f), (l, f') \in C, \; \forall j \in \{0, 1, 2, 3\}. \tag{6}$$

The ILP formulation is thus:

$$\max \sum_{(l,f) \in C} z_{l,f}, \tag{7a}$$

$$\text{s.t. } x_{l,i}, y_{f,j}, z_{l,f}, s_{l,f,i} \text{ are binary variables}, \; (2) - (6). \tag{7b}$$

This formulation can be improved by two modifications.

First, there are numerous constraints (between every pair of conflicted FFs) in (4). For example, Assuming two CK/SR types, two CE types in each CK/SR type, and four FFs under each CE type, the total number of constraints is $(4 \times 4) \times \binom{4}{2} \times 4 + (8 \times 8) \times \binom{2}{2} \times 4 = 640$. To avoid such extensive enumeration, type representatives can be used (in this subsection, a type refers to the same configuration of CK SR and CE hereafter). Suppose K is the set of FF types. Let F_k represent the set of FFs of type $k \in K$, and the *type representative*

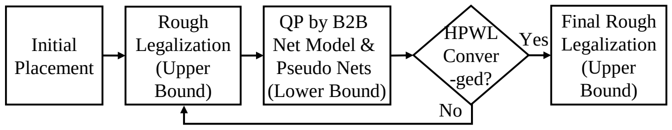

Initial Placement
Rough Legalization (Upper Bound)
QP by B2B Net Model & Pseudo Nets (Lower Bound)
HPWL Conver -ged?
Yes
No
Final Rough Legalization (Upper Bound)

Fig. 7: Overall GP flow.

$t_{k,j}$ indicate whether F_k occupies j-th FF group. Then, (4) can be replaced by:

$$t_{k,j} \geq y_{f,j}, \qquad \forall f \in F_k, \forall k \in K, \forall j \in \{0,1,2,3\}, \tag{8a}$$

$$\sum_k t_{k,j} \leq 1, \qquad \forall j \in \{1,2,3,4\}, \tag{8b}$$

$$t_{k,j} + t_{k',j'} \leq 1, \qquad \forall(k,k') \text{ with different CK/SR}, \quad \forall(j,j') \in \{(0,1),(2,3)\}. \tag{8c}$$

Under the same assumption, the number of constraints is reduced from 640 to $16 \times 4 + 4 + (2 \times 2) \times 4 = 84$.

Second, the LUT pairs and FFs without internal LUT-FF connection, which are the majority in L and F, can be removed from the ILP formulation to reduce the variable number significantly. The variables for such unconnected LUT pairs can be deleted directly since theirs halves can be arbitrarily determined later without affecting both constraints and objectives. However, postponed assignment of unconnected FFs may fail due to the FF conflict constraints. To guarantee sufficient legal slots, type representative $t_{k,j}$ helps again:

$$\sum_{j=0}^{3} t_{k,j} \geq \lceil |F_k|/4 \rceil, \quad \forall k \in K. \tag{9}$$

In summary, with the help of type representatives, a much more efficient ILP formulation equivalent to (7) is:

$$\max \sum_{(l,f)\in C} z_{l,f}, \tag{10a}$$

$$\text{s.t.} \quad x_{l,i}, y_{f,j}, z_{l,f}, s_{l,f,i}, t_{k,j} \text{ are binary}, \quad (2), (3), (5), (6), (8) \text{ and } (9). \tag{10b}$$

with unconnected LUT pairs and FFs removed from sets L, F, F_k in (2), (3), and (8). It can be efficiently solved by ILP solver.

In practice, many cases may be easy to optimize. Therefore, a greedy heuristics is attempted for each case first. Essentially, LUT pairs are sequentially committed in the order of number of candidate LUT-FF connections. For each LUT pair, the achieved LUT-FF connection is maximized as long as it is legal. Then, an upper bound of the total achieved LUT-FF connection is calculated by $\sum_{l\in L} PNF(l)$. If the heuristics can accomplish the upper bound, there is no need to invoke ILP, which saves runtime. In experiment results, we will show that usually only single-digit number percent of cases actually call ILP, which makes the slot assignment more efficient in general.

V. Placement Algorithms

There are two GP stages in our flow, a flat one (Stage ❶) and a BLE one (Stage ❸), where two routibility optimization techniques, including partition allocation and cell inflation/shrinkage, are applied. After LG, the congestion-aware two-level DP (Stage ❺) further improves the solution.

A. Heterogeneous Global Placement Engine

Our GP engine in Stages ❶ ❸ is based on Ripple [33], which invokes the lower bound and upper bound computations alternatively. The overall flow of GP is shown in Fig. 7. In the lower bound phase, the wirelength minimization is formulated as a quadratic programing (QP), where the "Bound2Bound" (B2B) [52] net model is used to capture the HPWL objective. The minimized wirelength, however, leads to many cell overlaps. In the upper bound phase, the placement is roughly legalized by spreading the cells. To be more specific, the chip is divided into bins and cells are iteratively spread until the cell density of each bin is within a threshold. In the next lower bound phase, pseudo pins and pseudo nets are added for movable cells. A pseudo pin is placed at the cell location in the last upper bound phase, while a pseduo net connects a pseudo pin and the corresponding cell. By iteratively calling the lower and upper bound computations and gradually increasing the pseudo net weights, we can obtain a converged GP result with few overlaps and minimized wirelength.

Recall that each site in the heterogeneous FPGA architecture is dedicated for a type of cells, which should be taken care of. Otherwise, due to type mismatch, LG may cause significant displacement even if GP has very few overlaps. In our GP, fence constraints are used to avoid placing cells into illegal sites. That is, when spreading cells of a type in the upper bound phase, the maximum cell density of unmatched sites is set to zero.

Another issue in GP is how to set the areas of LUT/FFs (in flat GP) and BLEs (in BLE GP). For DSPs and RAMs, the area can be set straightforwardly, since each DSP/RAM site can hold one and only one DSP/RAM block. Without CLB packing, the number of LUTs and FFs that a CLB site can contain is unknown. Furthermore, LUTs and FFs share the CLB sites and cannot be separated by fence constraints. Their areas are thus set by the following adaptive estimation. The estimated number of FFs that a CLB can contain c_{ff} is set to 9, which is the average of maximum (16 if there is totally no conflict) and minimum (2 if FF CK/SR types are all different). Similarly, the estimated number of LUTs c_{lut} that a CLB can contain is 12, the average of 16 and 8. Here, a six-input LUT is counted twice. For a design with n_{ff} FFs and n_{lut} LUTs, the number of CLBs needed is thus estimated to be $\max\{n_{lut}/c_{lut}, n_{ff}/c_{ff}\}$. The base LUT/FF area a_{lf} is then adaptively calculated by:

$$a_{lf} = \frac{\max\{n_{lut}/c_{lut}, n_{ff}/c_{ff}\}}{n_{lut} + n_{ff}}. \tag{11}$$

The actual LUT/FF area is therefore:

$$a'_{lf} = a_{lf} \cdot scale, \tag{12}$$

where $scale$ is a scale factor tuned for a trade-off among HPWL, routability and LG difficulty, which is design independent. In our implementation, the default value is 1.4. If the design shows extremely large cut ratio during netlist partitioning, which implies routing difficulty, $scale$ will be set larger. Besides, the area of a soft BLE is just the summation of those of its member LUT/FFs.

B. Routing-Architecture-Aware Partition Allocation

At an early point of Stage ❶, netlist partitions are globally allocated to fit the routing supply imbalance.

Analytical ASIC placers typically do not need netlist partitioning, where the netlist minimization relies on the mathematical optimization on the flat netlist. However, the Xilinx UltraScale FPGA is unbalanced in the routing supply of the horizontal and vertical directions. It has 168×480 sites and 82×480 switch boxes on the chip (roughly two horizontally-neighbored sites share a switch box, as Fig. 3 shows). Despite the chip area is actually square, the distance measured by number of switch boxes, which implies timing as well as routing resource in FPGA, is significantly unbalanced between the width and height. Meanwhile, the number of wire

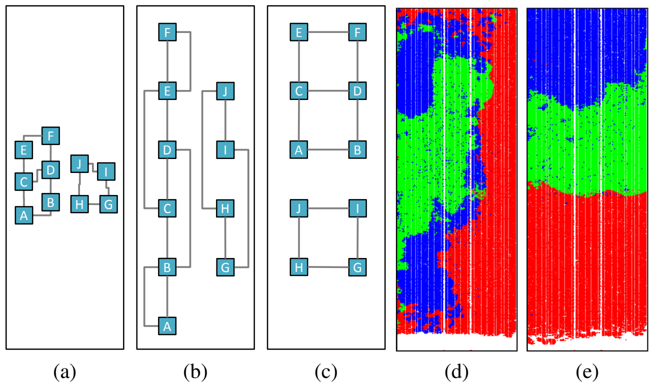


Fig. 8: Routability problem due to the unbalanced horizontal and vertical routing supply: (a) placement with small wirelength but illegal cell density; (b) spreading cells by strictly maintaining cell order; (c) spreading cells with partition allocation. The final placement results of a design (d) without initial partition allocation and (e) with allocation.

segments between two horizontally-aligned switch boxes (another type of routing resource) is nearly the same as the corresponding vertical one.

The imbalance in routing supply leads to more serious congestion in the vertical direction. Regarding our GP method, the lack of free space in the horizontal direction leads to high cost in wirelength and congestion during cell spreading, as Fig. 8(a) and Fig. 8(b) show. Furthermore, when using the cell inflation technique to alleviate the identified congestion, much horizontal demand is transformed to vertical one during the vertical cell spreading. It thus exacerbates the vertical congestion, which is already more serious.

As a result, RippleFPGA allocates the netlist partitions vertically at an early point of the flat initial GP. In this way, the routing demand fits the unbalanced routing supply and avoids the congestion, as Fig. 8(c) shows. Note that different from partitioning-based GP, where a partition of the netlist is restricted to a geometric region, our partition allocation only generates a better initial solution to guide the later stages without posing any constraint. Moreover, we put high requirement on partitioning and only manipulate global partitions, which is controlled by a minimum cut ratio c_{min} and a minimum partition size n_{min} respectively.

The partitions are allocated after a few (e.g., two) iterations of GP, which provides initial cell locations. There are generally two steps. First, a recursive hypergragh bipartition is conducted to identify the obvious sub-circuits in the netlist. Second, the identified partitions are relocated according to their locations in GP and their relationship in the bipartition hierarchy.

1) Recursive Netlist Bipartition: A direct k-way partitioning usually performs better than recursive bipartitioning in terms of cut size. However, the number of partitions needed is unknown in our application scenario, since only partitioning with small cut size is desirable. Therefore, the scheme of recursive netlist bipartition, which stops when the cut size becomes too large, is adopted. Also, the area of a partition should be sufficiently large to span the chip horizontally with a reasonable aspect ratio, because partitions need to be vertically allocated later. The detailed method is shown in Algorithm 3. The netlist G_{nl} is partitioned recursively until either the size of the sub-circuit too small (line 2) or the cut size is too large (line 7). Note that the cut constraint (determined by ratio c_{min}) is a local relative value measuring how obvious the partitioning is, while the constraint n_{min} on the sub-circuit size is a global absolute value.

Algorithm 3 Recursive Netlist Bipartition

```
Require: Netlist $G_{nl} = (V_{nl}, E_{nl})$, min cut ratio $c_{min}$, min partition size $n_{min}$, max imbalance $\delta_{max}$;
Ensure: Recursive bipartition of $G_{nl}$;
 1: function Bipart($G_{nl}$)
 2:     if $|V| \leq n_{min}$ then
 3:         return ;
 4:     end if
 5:     Obtain sub-partitions $G_{nl1} = (V_{nl1}, E_{nl1})$ and $G_{nl2} = (V_{nl2}, E_{nl2})$ of $G_{nl}$ by multi-level hypergraph bipartition under max imbalance constraint of $\delta_{max}$;
 6:     if cut size $> c_{min} \cdot |E_{nl}|$ then
 7:         Abandon $G_{nl1}$ and $G_{nl2}$;
 8:     else
 9:         Bipart($G_{nl1}$);
10:         Bipart($G_{nl2}$);
11:     end if
12: end function
```

Algorithm 4 Partition Allocation

```
Require: Netlist $G_{nl} = (V_{nl}, E_{nl})$, cell locations $(x_i, y_i)$, chip width $W$;
Ensure: updated cell locations $(x'_i, y'_i)$;
 1: Bipart($G_{nl}$);
 2: Increase cell area if the cut size is large;
 3: $A, y_{avg} \leftarrow$ ObtainInfo($G_{nl}$);
 4: Bottom of the target region $y_l \leftarrow y_{avg} - \frac{A}{2W}$;
 5: RelocPart($G_{nl}, y_l$);

 6: function ObtainInfo($G_{nl}$)
 7:     if $G_{nl}$ has no sub-partitions then
 8:         $A \leftarrow$ total cell area of $V$;
 9:         $y_{avg} \leftarrow$ average y-coordinate of $V$;
10:     else
11:         $A_1, y_{avg1} \leftarrow$ ObtainInfo($G_{nl1}$);
12:         $A_2, y_{avg2} \leftarrow$ ObtainInfo($G_{nl2}$);
13:         $A \leftarrow A_1 + A_2$;
14:         $y_{avg} \leftarrow \frac{|V_1| \times y_{avg1} + |V_2| \times y_{avg2}}{(|V_1| + |V_2|)}$;
15:     end if
16:     return $A, y_{avg}$;
17: end function

18: function RelocPart($G_{nl}, y_l$)
19:     if $G_{nl}$ has no sub-partition then
20:         Height of target region $h \leftarrow A/W$;
21:         Sort $V_{nl}$ by $y_i$;
22:         $y'_i \leftarrow \frac{i}{|V|} \cdot h + y_l$ for $v_i \in V$;
23:         Sort $V_{nl}$ by $x_i$;
24:         $x'_i \leftarrow \frac{i}{|V|} \cdot W$ for $v_i \in V$;
25:     else if $y_{avg1} < y_{avg2}$ then
26:         RelocPart($G_{nl1}, y_l$);
27:         RelocPart($G_{nl2}, y_l + \frac{A_1}{W}$);
28:     else
29:         RelocPart($G_2, y_l$);
30:         RelocPart($G_1, y_l + \frac{A_2}{W}$);
31:     end if
32: end function
```

Besides, if the cut size obtained in the partitioning is huge, which indicates the design is difficult to route, the cell area used in GP is increased correspondingly (line 2 in Algorithm 4).

2) Partition Relocation: In order to minimize the disturbance to the wirelength in initial rough GP, sub-circuits are vertically aligned by the method as shown in Algorithm 4. To respect the connections between partition, sub-partitions of a same parent partition should be placed next to each other; the relative order in vertical direction is maintained if possible, since it implies their connections to IO (lines 25–31). Within each sub-circuit, the cells are moved and spread to a designated bounding box while keeping their relative order of in x and y directions (lines 21–24). The height of the bounding box

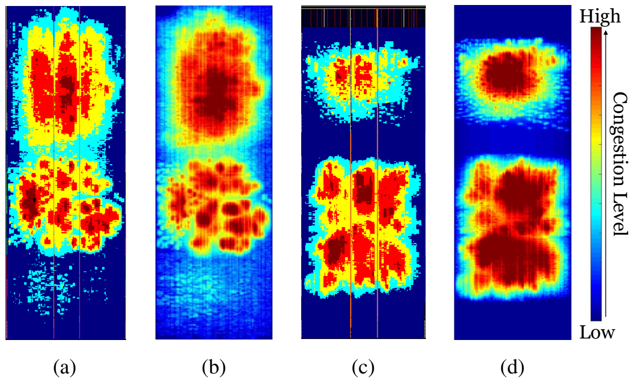


Fig. 9: Congestion estimation on a design by (a) Vivado and (b) ours, and on another design by (c) Vivado and (d) ours.

is determined by total cell area (line 20) and the width is the chip width. Before moving cells, the information needed is calculated in a post-order traversal of the bipartition tree (lines 6–17).

The impact of our partition allocation is shown by Fig. 8. Without the initial partition allocation, sub-circuits are messed up and a sub-circuit may distribute vertically along the chip, which causes bad routability (Fig. 8(d)). With guidance, the messing is avoided (Fig. 8(e)).

C. Congestion-Driven Global Placement

In some later iterations of Stage ❸, cells are inflated and shrunk to improve routibility and wirelength according to congestion. Here, a congestion estimation is first needed. Routing result generated by the actual FPGA routing tool, which will also be the final evaluation, is the most accurate choice. It, however, requires much runtime. Meanwhile, the industrial global router is integrated in the commercial tool and cannot be individually invoked. Therefore, our own congestion estimation is needed.

In [53], the congestion map is built according to the net bounding box. It estimates routing congestion by the number of nets that may consume a site, i.e, the total number of bounding boxes covering a site.

However, two nets with the same bounding box may differ substantially in routing demand, due to the highly diverse number of pins. To take this into account, a bounding box should be weighted properly. We will first divide the FPGA chip into global routing cells (g-cells) such each g-cell represents a switch box. For a g-cell, the cell pins that it covers are mapped as its pins. After obtaining a new netlist on g-cells, we can calculate the weighted bounding box overlaps of the i-th g-cell by:

$$cong_i = \sum_{m \in N_i} \frac{NW(|m|) \cdot HPWL(m)}{|G_m|}, \tag{13}$$

where NW is the net weight related to the pin number [54], N_i is the set of nets intersecting with g-cell i, and G_m is the set of g-cells covered by net m. As Fig. 9 shows, our model can estimate routing congestion quite accurately comparing with that reported by Vivado.

Based on the congestion estimation, we focus on routing congestion when maintaining previous good HPWL in some later GP iterations. More BLEs are placed to regions with low congestion to save wirelength, while congested regions should be sparser. Therefore, in each iteration, the sizes of BLEs placed in congested and uncongested regions are inflated and shrunk respectively. In implementation, we inflate the BLEs in the congested (10% most congested if there are more) g-cells, and shrink those in the uncongested (30% least congested if there are more) g-cells. The rate of inflating and shrinking is empirically set to 20%. After changing the areas of the BLEs, a short GP with high pseudo net weight is executed to adjust the positions of the BLEs and alleviate routing congestion. Since shrinking BLEs may result in placing too many BLEs in a region and thus cause large displacement during LG. LG is applied at the beginning of each iteration, and cells with large displacement are not allowed to shrink. By the procedure above, routing congestion can be distributed more evenly across the chip.

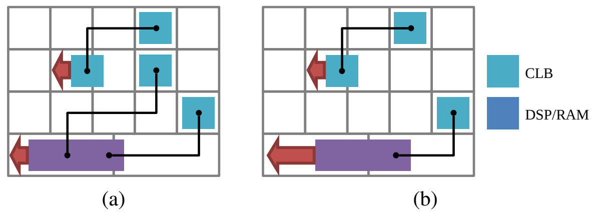


Fig. 10: Larger disturbance to GP when legalizing DSP/RAMs due to their (a) higher connectivity, and (b) larger area.

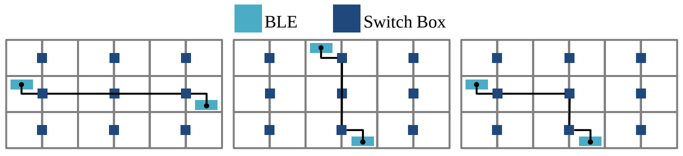
D. Bipartite-Matching-Based DSP/RAM Legalization

Inspired by [42], RippleFPGA legalizes DSP/RAMs before BLEs, because legalizing DSP/RAMs incurs much larger disturbance to GP than BLEs. As Fig. 10(a) shows, with the same displacement, DSP/RAMs will increase the wirelength much more than BLEs due to their high connectivity. Moreover, since DSP/RAMs are much bigger, legalizing them tends to induce more displacement, as Fig. 10(b) shows. However, instead of two consecutive batches of DSP/RAM LG and BLE LG used in [42], we put DSP/RAM LG even earlier and in the middle of BLE GP (Stage ❸). In this way, after legalizing and fixing DSP/RAMs, BLEs can move during a few GP iterations to repair the wirelength degradation.

Further inspired by [24], bipartite matching is adopted in our DSP/RAM LG. Nevertheless, different from their displacement-driven window-by-window LG, our LG optimizes HPWL (with displacement constraint) in the full chip scale.

In the bipartite graph $G_3(X_3, Y_3, E_3)$ constructed, X_3 are blocks to be legalized and Y_3 are all legal sites available. For each block $x \in X_3$, a number of (e.g., 10) sites nearest to it are candidates, the cost of which is the HPWL when x is assigned to y with all other blocks staying at GP locations. Different from the bipartite complete graph used in [24], we limit the number candidate sites for each block with three motivations. (1) The displacement in LG should be limited in order to increase the fidelity of HPWL estimation, which assumes all other blocks are not moved. In this way, the final HPWL improvement may still not correspond to the edges by exactly the same value, but the error is under control. (2) Limiting displacement avoids significant disturbance to GP result. (3) The runtime is improved by having significantly fewer edges.

After constructing the bipartite graph G_3, its min-cost matching, which can be solved by the min-cost network flow, is a LG solution with minimized HPWL cost. However, there is no theoretical guarantee that all blocks can be assigned, even though illegal cases were not observed by us. To guarantee a legal solution, edges corresponding to the assignment in a trial LG (by Tetris) are also added into E_3. Note that it will be different from Tetris itself since escape only happens in overflowed regions.



(a) Horizontal alignment (b) Vertical alignment (c) No alignment

Fig. 11: Three cases with the same switch-box HPWL but different alignment.

E. Congestion-Aware Two-Level Detailed Placement

In Stage ❺, multiple DP techniques are applied in both block (i.e., CLB, DSP and RAM) and BLE levels to improve wirelength and routability. In general, two types of moves are used: (i) DSP/RAM bipartite matching, and (ii) CLB/BLE global move/swap towards the optimal region [55]. Both are performed in each round and many rounds are carried out until no improvement.

DSP/RAM bipartite matching DP borrows the idea of DSP/RAM LG in Section V-D with two modification. First, the initial block positions are already legal positions instead of those generated by GP. Therefore, the edges added by trial LG to guarantee a legal solution are no longer needed, since the nearest sites of a block definitely include the site that is solely occupied by itself. Second, to avoid ruining the cell density optimized for routability, candidate sites of a block cannot simultaneously (i) have congestion level higher than the current site and (ii) be in the congested regions. The first condition restricts the movement worsening congestion. There is, however, no need to restrict movement in uncongested regions, which is the purpose of the second condition.

In global move/swap, we move a CLB to an empty site within its optimal region if there is. Otherwise, it is swapped with a CLB in its optimal region if the HPWL can be improved. For a BLE, all CLB sites in its optimal region are attempted by the incremental legality checking in Section IV-B. Besides, same as bipartite matching DP, moves that are not beneficial to routability are forbidden in global move/swap.

The above DP moves can be purely HPWL-driven under the congestion constraint. But the discrete nature of the switch-box-based FPGA routing is not captured by HPWL. The situation is demonstrated by Fig. 11. In all three cases, the HPWL (measured by the grid of switch boxes) is exactly the same (i.e., two). However, when the two pins are aligned horizontally or vertically, the connection probably will be routed via two switch boxes and a two-hop wire segment (Fig. 11(a) and Fig. 11(b)). Without alignment, at least three switch boxes and two wire segments are required (Fig. 11(c)). Therefore, an alignment score is calculated for each candidate site, besides the HPWL score. Basically, a unit score is added for each alignment between the cell to move and its connected cell. Due to the preciousness of routing resources in vertical direction as Section V-B mentions, horizontal alignment is encouraged more by a larger unit score. In our implementation, the score for horizontal alignment is set to 2, while the vertical one is set to 1. By incorporating the alignment score into DP objective, the placement thus has better routability for the switch-box-based routing.

VI. Speedup Techniques

We have made many optimizations for runtime compared to [50]. For instance, the slot assignment is separated from the legality checking and deferred to the end of the flow, as mentioned in Section IV-B. In this section we discuss two additional techniques applied in our placer.

TABLE I: Statistics About ISPD 2016 Contest benchmarks

Design	# LUT	# FF	# RAM	# DSP	# Net	# Control Set
FPGA-1	50K	55K	0	0	105K	12
FPGA-2	100K	66K	100	100	168K	121
FPGA-3	250K	170K	600	500	429K	1281
FPGA-4	250K	172K	600	500	430K	1281
FPGA-5	250K	174K	600	500	433K	1281
FPGA-6	350K	352K	1000	600	713K	2541
FPGA-7	350K	355K	1000	600	716K	2541
FPGA-8	500K	216K	600	500	725K	1281
FPGA-9	500K	366K	1000	600	877K	2541
FPGA-10	350K	600K	1000	600	961K	2541
FPGA-11	480K	363K	1000	400	851K	2091
FPGA-12	500K	600K	600	500	1111K	1281

In the quadratic programming of GP, we change the representation of a large sparse matrix from the list of list (LIL) to the compressed column storage (CCS) [56]. In LIL, an element can be easily queried, modified, inserted and removed. But these operations are of no need during the intensive computation of quadratic programming. By ignoring the efficiency of the single-element query and update, CCS stores a matrix in three compact arrays and need much less storage. As a result, the critical low-level matrix operations (e.g., matrix-vector multiplication) are significantly accelerated, due to the reduced memory bandwidth demand and cache miss rate.

In the optimal region calculation for DP, quickselect [57] replaces sorting for finding the medians of x and y coordinates of pins. As the optimal regions of all BLEs and CLBs need to be obtained in each DP round (for global move/swap), the runtime of optimal region calculation is originally a bottleneck in DP. The best sorting algorithms can achieve $\mathcal{O}(n \log n)$ complexity both in worst case and in average. Despite $\mathcal{O}(n^2)$ in the worst case, quickselect has much better average performance of $\mathcal{O}(n)$ with small constant.

VII. Experimental Results

To evaluate our proposed method, the algorithms are implemented in C++. Patoh [58], lpsolve [59] and boost graph library [60] are used for solving hypergraph partitioning, ILP and min-cost network flow respectively.

Experiments were performed on a 64-bit Linux workstation with Intel Xeon 3.4 GHz CPU and 32 GB memory. Even though RippleFPGA can be multi-threading, only single thread is used here for a fair comparison with the previous placers. Benchmarks are from ISPD 2016 Routability-Driven FPGA Placement Contest [5], the statistics of which is shown in TABLE I, where a control set means a kind of configuration of CK, SR and CE. The routing evaluation is conducted by Xilinx Vivado.

A. Effectiveness of Our Techniques

TABLE II shows the strength our new legality checking and slot assignment scheme compared to [50]. First, the matching-base legality checking not only is optimal but also increases flexibility, which is evidenced by the 6 % higher success rate in BLE LG (corresponding to Algorithm 2 line 11). Second, without false alarm, some scenarios (e.g., removing a BLE out of a CLB) need no legality checking now, which further reduces times of invoking incremental legality checking. Third, deferring slot assignment to the end itself also benefits the runtime. In general, both wirelength and runtime are improved. Besides, note that only a small ratio of CLBs are difficult instances and require solving by ILP, making the whole slot assignment very efficient.

Fig. 12 illustrates the runtime breakdown of RippleFPGA, where we can see that GP dominates. Actually, before the implementation

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2017.2778058, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

TABLE II: Impact of Matching-Based Legality Checking and ILP-Based Slot Assignment

Design	RippleFPGA w/ Legality Checking in [50]					RippleFPGA						
	Wirelength Impact		Runtime Impact			Wirelength Impact		Runtime Impact				
	% LG Success	Routed Wirelength	Legality Checking # Call	Legality Checking Runtime	Total Runtime	% LG Success	Routed Wirelength	Legality Checking # Call	Legality Checking Runtime	Slot Assignment % ILP	Slot Assignment Runtime	Total Runtime
FPGA-1	32.16%	**350749**	2716687	2	34	**43.40%**	352628	**438418**	**0.3**	3.99%	1	**31**
FPGA-2	43.39%	645618	3739596	3	61	**46.11%**	**645400**	**563878**	**0.5**	3.39%	1	**58**
FPGA-3	10.31%	3333579	13553726	10	212	**10.57%**	**3262106**	**4709863**	**3**	1.33%	2	**203**
FPGA-4	**9.37%**	5555391	14386077	10	237	9.35%	**5509661**	**5500975**	**4**	0.54%	2	**227**
FPGA-5	23.18%	**9908559**	13121108	11	284	**23.59%**	9968955	**4255253**	**3**	0.66%	2	**273**
FPGA-6	2.72%	6270243	37533293	21	447	**2.77%**	**6180104**	**21860787**	**12**	0.89%	3	**437**
FPGA-7	2.33%	9672488	36735440	19	**488**	**2.40%**	**9639639**	**26722085**	**16**	1.49%	4	499
FPGA-8	39.59%	8170991	14590118	13	439	**39.83%**	**8156951**	**3521360**	**3**	0.27%	3	**421**
FPGA-9	4.04%	12472932	36144354	23	625	**4.54%**	**12305192**	**18795015**	**11**	1.79%	6	**594**
FPGA-10	†	†	†	†	†	**1.80%**	**7139694**	**43532711**	**21**	3.78%	13	**673**
FPGA-11	4.06%	11147130	41969816	28	571	**4.25%**	**11022815**	**21019965**	**12**	1.74%	5	**554**
FPGA-12	†	†	†	†	†	**4.86%**	**7363451**	**19381135**	**11**	5.04%	14	**665**
Avg. Ratio	0.943	1.006	3.256	3.604	1.039	**1.000**	**1.000**	**1.000**	**1.000**	-	-	**1.000**

†Cannot legalize all BLEs

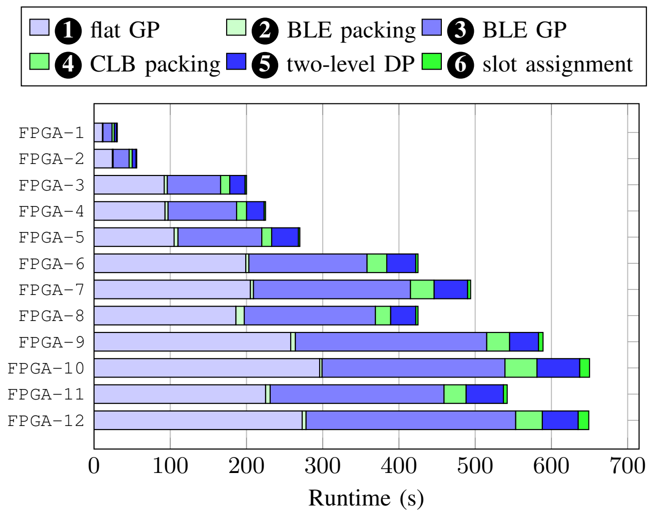


Fig. 12: Runtime breakdown of RippleFPGA.

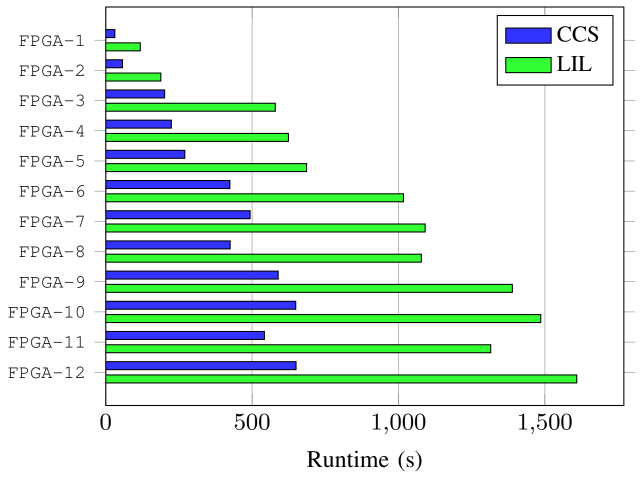


Fig. 13: Runtime impact of sparse matrix representation: compressed column storage (CCS) is more efficient than list of list (LIL).

TABLE III: Routed Wirelength Impact of Partition Allocation (PA) and Congestion-Driven GP (CGP)

Design	Baseline	PA	CGP	PA+CGP
FPGA-1	356314	353516	356081	**352628**
FPGA-2	669133	649051	655740	**645400**
FPGA-3	3532052	3277033	3533351	**3262106**
FPGA-4	5556177	**5508595**	5531058	5509661
FPGA-5	*	*	10120061	**9968955**
FPGA-6	6549187	6238933	6521672	**6180104**
FPGA-7	9723248	**9500233**	9933324	9639639
FPGA-8	8423217	**8122288**	8409769	8156951
FPGA-9	12050941	**12044246**	12104565	12305192
FPGA-10	7820378	7308750	7682063	**7139694**
FPGA-11	11172550	**10672421**	11528235	11022815
FPGA-12	8464954	*	8105645	**7363451**
Avg. Ratio	1.043	0.997	1.038	**1.000**

*Unroutable placement

TABLE IV: Routed Wirelength Impact of Bipartite-Matching-Based DSP/RAM LG and DP

Design	Neither	LG	DP	Both
FPGA-1	352628	352628	352628	**352628**
FPGA-2	656531	645384	645877	**645400**
FPGA-3	3429419	3268983	3278954	**3262106**
FPGA-4	5665941	5514118	5526963	**5509661**
FPGA-5	10087236	10009207	**9944421**	9968955
FPGA-6	6279386	6191009	6206049	**6180104**
FPGA-7	9760486	9643013	9673528	**9639639**
FPGA-8	8206043	8159135	**8151310**	8156951
FPGA-9	12386444	12303073	12326051	**12305192**
FPGA-10	7214541	7141872	7142227	**7139694**
FPGA-11	11084104	11021053	11035423	**11022815**
FPGA-12	7429943	7362895	7386437	**7363451**
Avg. Ratio	1.015	1.001	1.002	**1.000**

optimization on GP engine, both GP runtime percentage and total runtime were significantly larger. For example, the previous sparse matrix storage scheme of list of list (LIL) is much more timing-consuming than compressed column storage (CCS), as Fig. 13 shows.

TABLE III shows the effectiveness of two routability optimization techniques, routing-architecture-aware partition allocation (PA) and congestion-driven GP (CGP). PA improves the routed wirelength for all benchmarks with only one exception (PA+CGP v.s. CGP for FPGA-9) by resolving the problem of unbalanced routing supply. Even for FPGA-9, there is still improvement by using PA only, compared with the baseline. For CGP, it makes all designs routable. Even though the current CGP worsens wirelength on some designs, it probably would not be the case if the global router of Vivado can be accessed for congestion estimation.

The effectiveness of bipartite-matching-based method for DSP/RAM LG and DP is illustrated by TABLE IV. In the baseline

0278-0070 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2017.2778058, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

TABLE V: Routed Wirelength Comparison with State-of-the-Art FPGA Placers on ISPD 2016 Benchmarks

Design	1st Place		2nd Place		3rd Place		[50]		[24]		RippleFPGA	
	Wirelength	Ratio	Wirelength	Ratio	Wirelength	Ratio	Wirelength	Ratio	Wirelength	Ratio	Wirelength	Ratio
FPGA-1	†	†	379932	1.077	581975	1.650	362563	1.028	384709	1.091	**352628**	**1.000**
FPGA-2	677877	1.050	679878	1.053	1046859	1.622	677563	1.050	652690	1.011	**645400**	**1.000**
FPGA-3	3223042	0.988	3660659	1.122	5029157	1.542	3617466	1.109	**3181331**	**0.975**	3262106	1.000
FPGA-4	5628519	1.022	6497023	1.179	7247233	1.315	6037293	1.096	**5504083**	**0.999**	5509661	1.000
FPGA-5	10264769	1.030	*	*	*	*	10455204	1.049	10068879	1.010	**9968955**	**1.000**
FPGA-6	6330179	1.024	7008525	1.134	6822707	1.104	6960037	1.126	6411247	1.037	**6180104**	**1.000**
FPGA-7	10236827	1.062	10415871	1.081	10973376	1.138	10248020	1.063	10040562	1.042	**9639639**	**1.000**
FPGA-8	8384338	1.028	8986361	1.102	12299898	1.508	8874454	1.088	**8113483**	**0.995**	8156951	1.000
FPGA-9	*	*	13908997	1.130	*	*	12954350	1.053	13616625	1.107	**12305192**	**1.000**
FPGA-10	†	†	†	†	*	*	8564363	1.200	8866049	1.242	**7139694**	**1.000**
FPGA-11	11091383	1.006	11713479	1.063	*	*	11226088	1.018	**10834629**	**0.983**	11022815	1.000
FPGA-12	9021768	1.225	†	†	*	*	8928528	1.213	8246410	1.120	**7363451**	**1.000**
Avg.		1.048		1.105		1.411		1.091		1.051		**1.000**

*Unroutable placement †Placement error

TABLE VI: Runtime (Seconds) Comparison with State-of-the-Art FPGA Placers on ISPD 2016 Benchmarks

Design	1st Place		2nd Place		3rd Place		[50]		[24]		RippleFPGA	
	Runtime	Ratio	Runtime	Ratio	Runtime	Ratio	Runtime	Ratio	Runtime	Ratio	Runtime	Ratio
FPGA-1	†	†	118	3.862	97	3.175	74	2.422	215	7.036	**31**	**1.000**
FPGA-2	435	7.689	208	3.677	191	3.376	167	2.952	399	7.053	**57**	**1.000**
FPGA-3	1527	7.600	1159	5.768	862	4.290	1037	5.161	1555	7.739	**201**	**1.000**
FPGA-4	1257	5.608	1449	6.464	889	3.966	621	2.770	1289	5.751	**224**	**1.000**
FPGA-5	1266	4.691	*	*	*	*	1012	3.750	1237	4.584	**270**	**1.000**
FPGA-6	2920	6.879	4166	9.815	8613	20.291	2772	6.531	2827	6.660	**424**	**1.000**
FPGA-7	2703	5.481	4572	9.271	9196	18.647	2170	4.400	2588	5.248	**493**	**1.000**
FPGA-8	2645	6.229	2942	6.929	2741	6.456	1426	3.358	2705	6.371	**425**	**1.000**
FPGA-9	*	*	5833	9.901	*	*	2683	4.554	3407	5.783	**589**	**1.000**
FPGA-10	†	†	†	†	*	*	5555	8.555	4091	6.300	**649**	**1.000**
FPGA-11	3227	5.953	7331	13.524	*	*	3636	6.708	3267	6.027	**542**	**1.000**
FPGA-12	4539	6.986	†	†	*	*	9748	15.004	4625	7.119	**650**	**1.000**
Avg.		6.346		7.690		8.600		5.514		6.306		**1.000**

*Unroutable placement †Placement error

[50], DSP/RAM blocks are manipulated by Tetris-like LG and global move/swap in DP. It can be seen that the bipartite-matching-based LG and DP help the routed wirelength by 1.5%.

B. *Comparison with State-of-the-Art FPGA Placers*

TABLE V and TABLE VI show our routed wirelength and runtime compared with [24], [50] as well as the winners of ISPD 2016 Contest. The result of contest winners comes from the contest organizer. When we test the binary of the second place on our machine, exactly the same wirelength and similar runtime result can be reproduced, in spite of the different machine configurations.

For the routed wirelength, RippleFPGA not only generates legal and routable solutions for all benchmarks but also has the best average wirelength. To be more specific, our wirelength is in average 5.1% better than the second best [24]. Note that among all, `FPGA-10` is the most difficult to pack/legalize, on which, all winners of the contest failed to produce legal and routable placement. Actually, `FPGA-10` has the largest number of FFs and control sets (TABLE I), which also results in the lowest success rate in LG for RippleFPGA (TABLE II). On this particular design, RippleFPGA outperforms other placers the most (24.2% better than the second [24]). That is, our approach is much better on difficult designs. This demonstrates the strength of the stair-step flow and implicit CLB packing, which smoothly pack soft BLEs into CLBs.

For the runtime, as the fastest on all benchmarks, RippleFPGA outperforms the second fastest placer with a 4.76× speedup. For the largest design (`FPGA-12`), the runtime is reduced from hours to minutes, which is highly desirable to the field programming need of FPGA. The runtime superiority shows the effectiveness of our speedup techniques illustrated by Section VI.

VIII. Conclusion

Facing the increasing complexity and scale of modern FPGAs, the proposed RippleFPGA integrates FPGA packing and placement together through a set of novel techniques, such as a smooth stair-step flow, implicit CLB packing, and two-level DP. The stair-step flow consists of six stages: (i) flat GP, (ii) soft BLE packing, (iii) BLE GP, (iv) implicit CLB packing, (v) two-level DP, and (vi) slot assignment in CLB. To improve routability, both ASIC-like congestion alleviation methods and FPGA-routing-architecture-aware optimization techniques are applied. The experimental results show that RippleFPGA achieves the best routed wirelength and runtime compared to all the state-of-the-art academic placers.

References

[1] R. Aggarwal, "FPGA place & route challenges," in *ACM International Symposium on Physical Design (ISPD)*, 2014, pp. 45–46.

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

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

[4] Virtex UltraScale product table. [Online]. Available: http://www.xilinx.com/products/silicon-devices/fpga/virtex-ultrascale.html#productTable

[5] 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.

[6] H. Bian, A. C. Ling, A. Choong, and J. Zhu, "Towards scalable placement for FPGAs," in *ACM Symposium on FPGAs*, 2010, pp. 147–156.

[7] C. J. Alpert, D. P. Mehta, and S. S. Sapatnekar, *Handbook of Algorithms for Physical Design Automation*. CRC press, 2008.

[8] V. Betz and J. Rose, "Cluster-based logic blocks for FPGAs: Area-efficiency vs. input sharing and size," in *IEEE Custom Integrated Circuits Conference (CICC)*. IEEE, 1997, pp. 551–554.

0278-0070 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2017.2778058, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

[9] A. S. Marquardt, V. Betz, and J. Rose, "Using cluster-based logic blocks and timing-driven packing to improve FPGA speed and density," in *ACM Symposium on FPGAs*, 1999, pp. 37–46.

[10] E. Bozorgzadeh, S. Ogrenci-Memik, and M. Sarrafzadeh, "Rpack: routability-driven packing for cluster-based FPGAs," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2001, pp. 629–634.

[11] A. Singh, G. Parthasarathy, and M. Marek-Sadowska, "Efficient circuit clustering for area and power reduction in FPGAs," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 7, no. 4, pp. 643–663, 2002.

[12] S. T. Rajavel and A. Akoglu, "MO-pack: Many-objective clustering for FPGA CAD," in *ACM/IEEE Design Automation Conference (DAC)*, 2011, pp. 818–823.

[13] J. Luu, J. Rose, and J. Anderson, "Towards interconnect-adaptive packing for FPGAs," in *ACM Symposium on FPGAs*, 2014, pp. 21–30.

[14] S.-K. Wu, P.-Y. Hsu, and W.-K. Mak, "A novel wirelength-driven packing algorithm for FPGAs with adaptive logic modules," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2014, pp. 501–506.

[15] K. E. Murray, S. Whitty, S. Liu, J. Luu, and V. Betz, "Timing-driven Titan: Enabling large benchmarks and exploring the gap between academic and commercial CAD," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 8, no. 2, pp. 10:1–10:18, 2015.

[16] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: applications in vlsi domain," *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 7, no. 1, pp. 69–79, 1999.

[17] C. J. Alpert, J.-H. Huang, and A. B. Kahng, "Multilevel circuit partitioning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 17, no. 8, pp. 655–667, 1998.

[18] G. Karypis and V. Kumar, "Multilevel k-way hypergraph partitioning," *VLSI Design*, vol. 11, no. 3, pp. 285–300, 2000.

[19] J. Cong and S. K. Lim, "Edge separability-based circuit clustering with application to multilevel circuit partitioning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 23, no. 3, pp. 346–357, 2004.

[20] C. Alpert, A. Kahng, G.-J. Nam, S. Reda, and P. Villarrubia, "A semi-persistent clustering technique for vlsi circuit placement," in *ACM International Symposium on Physical Design (ISPD)*, 2005, pp. 200–207.

[21] J. Z. Yan, C. Chu, and W.-K. Mak, "SafeChoice: A novel approach to hypergraph clustering for wirelength-driven placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 30, no. 7, pp. 1020–1033, 2011.

[22] N. Viswanathan, M. Pan, and C. Chu, "FastPlace 3.0: A fast multilevel quadratic placement algorithm with placement congestion control," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2007, pp. 135–140.

[23] 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)*. IEEE, 2007, pp. 117–123.

[24] 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.

[25] 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.

[26] C. J. Alpert and A. B. Kahng, "Recent directions in netlist partitioning: a survey," *Integration, the VLSI Journal*, vol. 19, no. 1, pp. 1–81, 1995.

[27] 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.

[28] Z. Marrakchi, H. Mrabet, and H. Mehrez, "Hierarchical FPGA clustering based on multilevel partitioning approach to improve routability and reduce power dissipation," in *Reconfigurable Computing and FPGAs*. IEEE, 2005, pp. 25:1–25:4.

[29] W. Feng, "K-way partitioning based packing for fpga logic blocks without input bandwidth constraint," in *IEEE International Conference on Field-Programmable Technology (FPT)*. IEEE, 2012, pp. 8–15.

[30] 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.

[31] V. Betz and J. Rose, "VPR: A new packing, placement and routing tool for FPGA research," in *IEEE International Conference on Field Programmable Logic and Applications (FPL)*. Springer, 1997, pp. 213–222.

[32] 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, pp. 6:1–6:30, 2014.

[33] X. He, T. Huang, L. Xiao, H. Tian, and E. F. Y. Young, "Ripple: A robust and effective routability-driven placer," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 32, no. 10, pp. 1546–1556, 2013.

[34] 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.

[35] 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 Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 34, no. 5, pp. 685–698, 2015.

[36] 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.

[37] T. Ahmed, P. D. Kundarewich, and J. H. Anderson, "Packing techniques for Virtex-5 FPGAs," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 2, no. 3, pp. 18:1–18:24, 2009.

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

[39] 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.

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

[41] 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.

[42] S.-Y. Chen and Y.-W. Chang, "Routing-architecture-aware analytical placement for heterogeneous FPGAs," in *ACM/IEEE Design Automation Conference (DAC)*, 2015, pp. 27:1–27:6.

[43] P. Gopalakrishnan, X. Li, and L. Pileggi, "Architecture-aware FPGA placement using metric embedding," in *ACM/IEEE Design Automation Conference (DAC)*, 2006, pp. 460–465.

[44] V. Manohararajah, G. R. Chiu, D. P. Singh, and S. D. Brown, "Difficulty of predicting interconnect delay in a timing driven FPGA CAD flow," in *ACM Workshop on System Level Interconnect Prediction (SLIP)*. ACM, 2006, pp. 3–8.

[45] G. Chen and J. Cong, "Simultaneous timing driven clustering and placement for FPGAs," in *IEEE International Conference on Field Programmable Logic and Applications (FPL)*. Springer, 2004, pp. 158–167.

[46] ——, "Simultaneous placement with clustering and duplication," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 11, no. 3, pp. 740–772, 2006.

[47] 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.

[48] M. Hutton, J. Schleicher, D. Lewis, B. Pedersen, R. Yuan, S. Kaptanoglu, G. Baeckler, B. Ratchev, K. Padalia, M. Bourgeault *et al.*, "Improving FPGA performance and area using an adaptive logic module," in *IEEE International Conference on Field Programmable Logic and Applications (FPL)*. Springer, 2004, pp. 135–144.

[49] D. Hill, "Method and system for high speed detailed placement of cells within an integrated circuit design," Apr. 9 2002, US Patent 6370673.

[50] 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 *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2016, pp. 67:1–67:8.

[51] S. Micali and V. V. Vazirani, "An O(V1/2E) algorithm for finding maximum matching in general graphs," in *IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE, 1980, pp. 17–27.

0278-0070 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

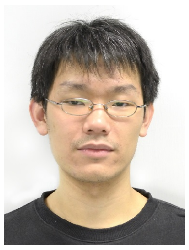
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2017.2778058, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

[52] P. Spindler, U. Schlichtmann, and F. M. Johannes, "Kraftwerk2: A fast force-directed quadratic placement approach using an accurate net model," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 27, no. 8, pp. 1398–1411, 2008.
[53] 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.
[54] C.-L. E. Cheng, "Risa: Accurate and efficient placement routability modeling," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 1994, pp. 690–695.
[55] M. Pan, N. Viswanathan, and C. Chu, "An efficient and effective detailed placement algorithm," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2005, pp. 48–55.
[56] I. S. Duff, R. G. Grimes, and J. G. Lewis, "Sparse matrix test problems," *ACM Transactions on Mathematical Software (TOMS)*, vol. 15, no. 1, pp. 1–14, 1989.
[57] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.
[58] Patoh. [Online]. Available: http://bmi.osu.edu/umit/software.html
[59] lpsolve. [Online]. Available: http://lpsolve.sourceforge.net/5.5/
[60] Boost graph library. [Online]. Available: http://www.boost.org/doc/libs/1_63_0/libs/graph/doc/

Gengjie Chen received the B.Sc. degree from the Department of Electronic and Communication Engineering, Sun Yat-sen University, Guangzhou, China, in 2015. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong. His research interests include combinatorial optimization and physical design. He received the best paper award in ICCAD 2017, Hong Kong Ph.D. Fellowship since 2015, and four ICCAD/ISPD contest awards.

Chak-Wa Pui received his B.Sc. degree in Computer Science and Technology from Shanghai Jiao Tong University, Shanghai, China, in 2015. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, under the supervision of Prof. Evangeline F.Y. Young. His research interests include physical design for both ASICs and FPGAs, Boolean matching in logic synthesis, and machine learning in physical design. He received the best paper award nomination in DAC 2016 and three ISPD/ICCAD contest awards in 2016, 2017.

Wing-Kai Chow Wing-Kai Chow received the Bachelor degree from The Hong Kong Polytechnic University in 2009. He has worked as a research assistant in the same university in 2010 and in The Chinese University of Hong Kong in 2010-2012. In 2012, he received the Master degree in Computer Science from The Chinese University of Hong Kong. He started his PhD in The Chinese University of Hong Kong till now. In 2016, he joined Cadence Design System, Inc, Austin, TX, as a Lead Software Engineer. His research interests include design automation of VLSI, especially placement and routing.



Ka-Chun Lam received the B.E. and M.Phil. degrees in computer science and engineering from The Chinese University of Hong Kong, Shatin, Hong Kong, in 2011 and 2014.

Jian Kuang received the B.E. degree from Sun Yat-sen University, Guangzhou, China, in 2012 and the Ph.D. degree in Computer Science and Engineering from The Chinese University of Hong Kong, Hong Kong, in 2016. He is now with Cadence Design Systems, San Jose, CA, USA. His current research interests include VLSI computer-aided design, physical design automation, and design for manufacturability.

Evangeline F.Y. Young received the B.Sc. and M.Phil. degrees in computer science from The Chinese University of Hong Kong (CUHK), Hong Kong, and the Ph.D. degree from The University of Texas at Austin, Austin, TX, USA, in 1999. She is currently a Professor with the Department of Computer Science and Engineering, CUHK. She is actively working on floorplanning, placement, routing, design for manufacturability, and algorithmic designs. Her current research interests include algorithms and computer-aided design of VLSI circuits.

Bei Yu (S'11–M'14) received the Ph.D. degree from The University of Texas at Austin in 2014. He is currently an Assistant Professor in the Department of Computer Science and Engineering, The Chinese University of Hong Kong. He has served in the editorial boards of Integration, the VLSI Journal and IET Cyber-Physical Systems: Theory & Applications. He received four Best Paper Awards at International Symposium on Physical Design 2017, SPIE Advanced Lithography Conference 2016, International Conference on Computer Aided Design 2013. and Asia and South Pacific Design Automation Conference 2012, and four ICCAD/ISPD contest awards.

0278-0070 (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.