FPGA Placement and Routing

Shih-Chun Chen[1] and Yao-Wen Chang[1,2]
[1]Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 106, Taiwan
[2]Department of Electrical Engineering, National Taiwan University, Taipei 106, Taiwan
scchen@eda.ee.ntu.edu.tw; ywchang@ntu.edu.tw

***Abstract*—FPGAs have emerged as a popular style for modern circuit designs, due mainly to their non-recurring costs, in-field reprogrammability, short turn-around time, etc. A modern FPGA consists of an array of heterogeneous logic components, surrounded by routing resources and bounded by I/O cells. Compared to an ASIC, an FPGA has more limited logic and routing resources, diverse architectures, strict design constraints, etc.; as a result, FPGA placement and routing problems become much more challenging. With growing complexity, diverse design objectives, high heterogeneity, and evolving technologies, further, modern FPGA placement and routing bring up many emerging research opportunities. In this paper, we introduce basic architectures of FPGAs, describe the placement and routing problems for FPGAs, and explain key techniques to solve the problems (including three major placement paradigms: partitioning, simulated annealing, and analytical placement; two routing paradigms: sequential and concurrent routing, and simultaneous placement and routing). Finally, we provide some future research directions for FPGA placement and routing.**

I. Introduction

With their reprogrammability at the design field, *field-programmable gate arrays* (*FPGAs*) have emerged as an unparalleled solution to the cost and time-to-market challenges in modern circuit designs. Compared with *application-specific integrated circuits* (ASICs), FPGAs have key advantages such as non-recurring costs (low risk), in-field reprogrammability (easy design change), short turnaround time (quick time to market), etc. As a result, FPGAs have become one of the most popular styles for circuit designs and the fastest growing segments in semiconductor industry since its first introduction in mid-1980s.

A. FPGA Architecture

A classical FPGA, as illustrated in Figure 1, contains three major parts: *logic modules* (called *configurable logic blocks*, CLBs for short), *routing resources*, and *input/output (I/O) cells* [3, 4, 25, 41]. In a classical island-style FPGA, a two-dimensional array of logic modules is surrounded by general routing resources bounded by I/O cells. The logic modules consist of combinational and sequential circuits to implement desired logic functions (see Figures 1(b) and (c)). The routing resources are composed of pre-fabricated *wire segments* and *programmable switches*, where the wiring among logic modules and I/O cells is user-programmable. A junction of a horizontal channel and a vertical one is referred to as a *switch box* (SB for short), serving to connect wire segments, which requires using programmable switches inside a switch box. A circuit is realized in an FPGA by partitioning and packing its logic into individual logic modules and then interconnecting the modules by wire segments with appropriately programmed switches [4].

As shown in Figure 1(b), a CLB mainly consists of *flip-flops* (FFs), *lookup tables* (LUTs), and *multiplexers* (MUXs). The FF is used to store the state information of a circuit. An LUT is usually a segment of static random-access memory (SRAM), where the programming data defining the logic function are loaded into the SRAM at power-up; see Figure 1(c) for an example LUT with two inputs, where the squares marked as "M" are the locations of the memories. MUXs are used to select circuitry to implement desired logic functions. A designer can change the stored bits of SRAMs in the field by a programming device to reconfigure a design. As a result, an FPGA enjoys a significant advantage of easy design change.

This work was partially supported by AnaGlobe, IBM, MediaTek, TSMC, and MOST of Taiwan under Grant MOST 103-2221-E-002-259-MY3, MOST 104-2221-E-002-132-MY3, MOST 105-2221-E-002-190-MY3, MOST 106-2911-I-002-511, and MOST 106-2221-E-002-203-MY3.

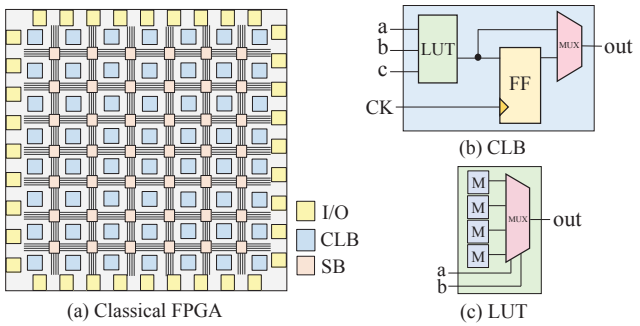


Fig. 1: A classical FPGA architecture. (a) An island-style FPGA architecture with three major parts: logic modules (CLBs), routing resources with switch boxes (SB), and I/O cells. (b) An example CLB structure. (c) An example LUT structure.

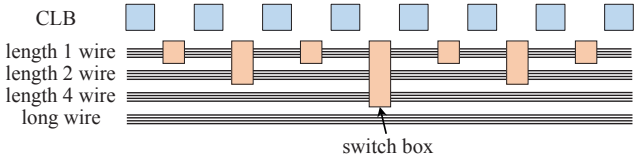


Fig. 2: An example segmented routing architecture with length-1, length-2, length-4, and long wires, where its wiring delay depends mainly on the number of switches used.

A routing channel of an FPGA often contains various types of interconnects, distinguished by their relative segment lengths; for example, length-1 wires, length-2 wires, length-4 wires, and long wires, as illustrated in Figure 2. The length-1 wires form a grid of horizontal and vertical wires that intersect at switch boxes, the length-2 wires contain a grid of segments twice as long as the length-1 wires, and so on. The long wires are a grid of segments that run the entire vertical or horizontal channel. Further, the horizontal and vertical wires of various lengths can be interconnected with programmable switches inside switch boxes shown in the figure. A programmable switch typically has much larger resistance and capacitance than a metal wire, incurring a much larger signal delay. As a result, the wiring delay in an FPGA is not linearly proportional to the geometric distance of the wiring, but depends mainly on the number of segments (switches) used for the wiring [8, 50].

As technology advances, the circuit complexity grows rapidly and the design requirements diversify significantly. To address the

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

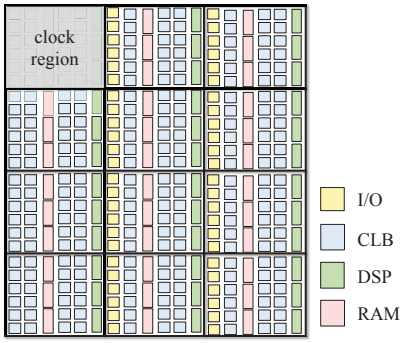


Fig. 3: A column-based heterogeneous FPGA with clock regions [22].

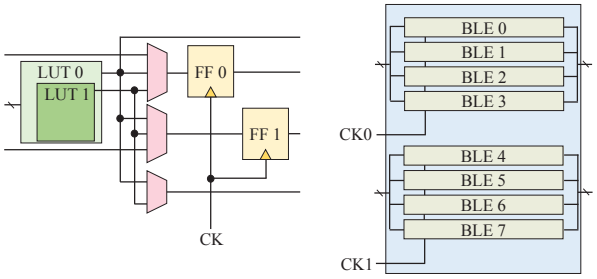


Fig. 4: Typical CLB structure in a modern FPGA [22].

emerging circuit design needs, a modern FPGA typically consists of large-scale, heterogeneous logic components. Figure 3 illustrates an example architecture of a modern FPGA. Different from the classical FPGA architecture, the I/O cells of a modern FPGA are often located at some middle columns, rather than along the boundaries of an FPGA chip; further, the architecture of a CLB becomes more hierarchical and sophisticated, as an example shown in Figure 4, which consists of eight *basic logic elements* (BLEs) and two clock signals. A BLE is a logic element comprising LUTs and FFs. Besides CLBs and I/Os, *random access memories* (RAMs), *digital signal processors* (DSPs), and some *intellectual properties* (IPs) are embedded. Also, diverse clock routing resources are included in a modern FPGA, applicable to multiple, yet limited clock sources in a given region, named as a *clock region* (see the regions enclosed by black bounding boxes in Figure 3). With recent development in new memory technologies, *non-volatile memory* (NVM) has been introduced to FPGA, where NVM is area-friendly, non-volatile memory, but suffers from significant aging effects and long write cycles.

B. FPGA Design Process and Metrics

The FPGA design process consists of three major stages (see Figure 5) [4]:

1) **System design:** An FPGA-based design begins with system design, including defining formal specifications of a system and designing system architectures and logic functions to realize the system. In this stage, a designed circuit is optimized by a logic synthesis tool and represented by Boolean expressions.
2) **Physical design:** Physical design converts an optimized circuit into a geometric representation called a *layout*. This process is composed of technology mapping, placement, and routing. A technology mapper maps an optimized circuit into a set of LUTs, FFs, and logic gates. Then, LUTs, FFs, and logic gates are packed into logic components, including CLBs, DSPs, and/or RAMs (also I/Os). These components are then placed at desired sites in a given FPGA by a placer. Finally, a router assigns wire segments and selects programmable switches to construct the required connections among the logic components.
3) **Customization:** Upon successful routing completion, the output configuration is fed into a programming unit to customize the FPGA chip.

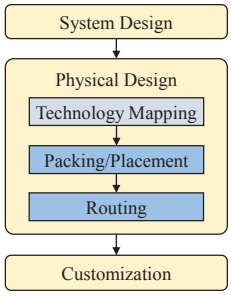


Fig. 5: FPGA design flow.

Compared to an ASIC, an FPGA does not need a fabrication process to realize a circuit, but a modern FPGA has more restricted logic and routing resources, diverse architectures, stringent design constraints, etc. For example, the prefabrication of routing resources and programmable switches causes severe resource competitions and thus imposes more constraints on physical design, especially the technology-dependent steps, *placement (including packing) and routing*. Further, the diversity of FPGA architectures prevents any single algorithm from being applied to all kinds of architectures. As a result, the layout design (especially, placement and routing) of an FPGA becomes much more challenging.

The placement and routing are defined as follows:

1) **Placement:** Given an FPGA chip with a heterogeneous logic architecture and a netlist with logic type specifications, the placement problem is to assign all the placeable modules to their legal locations (sites) such that some predefined objective function (e.g., total wirelength) is optimized.
2) **Routing:** Given a placed circuit and a netlist, the routing problem is to assign each net to wire segments and switches such that all nets are connected and some predefined objective function (e.g., total wirelength and segments) is optimized.

The cost metrics for an FPGA-based design might not be the same as those for a classical ASIC one. Wirelength is often the first and most basic metric for consideration because smaller wirelength typically leads to less wire resource consumption and smaller delay. As pointed out in the previous works [8, 50], however, the delay of a connection in an FPGA might not be linearly proportional to the geometric distance of this connection, due mainly to the FPGA segmented routing structure. As a result, FPGA timing depends more on the number of segments (switches) used because of the much bigger resistance and capacitance of a programmable switch. Congestion has also emerged as an important cost metric for modern FPGA designs because of the increasing design complexity, heterogeneous logic structures, limited routing resources (segmentations and switches). Clocking resources in a specific region of an FPGA are often limited, which imposes an additional constraint for FPGA designs. As the design complexity increases, runtime scalability is becoming an important limiting

factor for FPGA designs. In addition, power and signal integrity are often significant design requirements to be considered. With the introduction of new technologies such as NVM-based FPGAs, the aging effect and the long write cycle of an NVM FPGA have brought up new challenges to the FPGA placement and routing problems. For the NVM technology, it is more desirable to place a circuit component in a young site or in a site keeping the memory state unchanged.

The remainder of this paper is organized as follows. Sections II and III present key FPGA placement and routing algorithms, respectively. Section IV describes existing algorithms for simultaneous placement and routing. Section V presents some important future directions. Finally, Section VI concludes this article.

II. Placement

A great variety of algorithms were proposed to solve the FPGA placement problem. These algorithms can be classified into three major categories: *partitioning-based method*, *simulated annealing*, and *analytical placement*.

A. Partitioning-Based Placement

The partitioning-based placement can be realized as recursively calling the partitioning process by picking a region containing some circuit modules, dividing the region into a set of subregions, and assigning each module to one of the subregions to optimize some predefined metric (e.g., wirelength and cut size). The partitioning process is recursively called until the number of modules in each region is smaller than a threshold such that their placement can be solved by enumeration or other methods.

The work [30] gives one example of the partitioning-based placer. Besides the basic partitioning process, this placer adopts a terminal alignment heuristic and uses the average usage of wire segments as a parameter of the delay cost. The heuristic tries to align connected circuit modules to the same row or column to minimize the switch usage and wirelength.

The partitioning-based placer is often efficient and has good scalability for handling large-scale designs, but its quality is often limited because of the lack of global information in the top/coarse level and the lack of flexibility in the bottom/fine one, especially when a design with large whitespaces.

B. Simulated Annealing

Simulated annealing (SA for short) is an optimization method which provides a probability-based mechanism for "uphill" moves (i.e., a state/solution with a higher cost) to escape from being trapped in a local minimum, where the probability depends on the magnitude of the "uphill" move and the total search time. Figure 6 illustrates an SA search process. Given an initial temperature and an initial state $\mathbf{X}_0$, the cost function $f(\mathbf{X})$ which maps $\mathbf{X}$ to a real number, an annealing process iteratively finds a neighboring state $\mathbf{X}'$ to replace the previous state $\mathbf{X}$ if $f(\mathbf{X}')$ is smaller than $f(\mathbf{X})$, or replace the previous state with a probability depending on the temperature and the cost difference between $f(\mathbf{X}')$ and $f(\mathbf{X})$. After repeating such a process for a predefined number of neighboring states for each iteration, the temperature is lowered iteration by iteration. The process stops when the temperature is sufficiently low. Because SA allows a state of a higher cost to replace its previous state (i.e., an "uphill" move), SA can escape from a local minimum; together with a proper cooling schedule for exploring desired states, SA often can find a high-quality solution.

For an SA-based placer, a solution is often given by the assignment of physical locations for all the modules, and its solution space is a collection of all the feasible assignments. By changing the location(s) of one module or more, we can identify a neighboring state (a new placement) which is evaluated by a predefined cost function to determine whether this neighboring state is kept. With a proper annealing schedule that determines the initial temperature, the ratio for temperature reduction per iteration, and the temperature threshold for termination, a desired placement solution can often be achieved.

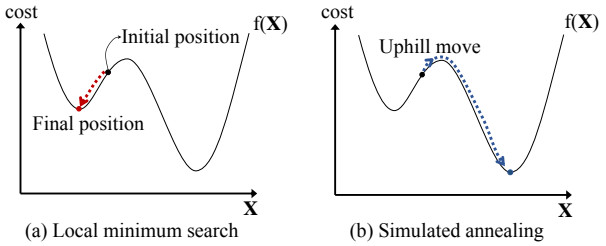


Fig. 6: Comparison of a greedy search technique and the SA search one. (a) A greedy search technique might lead to a local minimum. (b) The SA search technique can escape from a local minimum to reach the global optimal state.

The well-known *Versatile Place and Route* (VPR) is a classical SA-based FPGA placer [2]. Its cost function is defined as

$$\sum_{n \in nets} q(n) \times \left(\frac{w_x(n)}{c_{av,x}(n)} + \frac{w_y(n)}{c_{av,y}(n)} \right),$$

where $q(n)$ is the function to reflect a wire demand for a multiple-pin net, $w(n)$ is the wirelength estimation using the bounding box of the net n, and $c_{av}(n)$ is the average channel capacity in the bounding box of the net n (the subscripts x and y denote the horizontal and vertical components, respectively). This cost function considers wirelength and congestion at the same time. For the annealing schedule, it adjusts the cooling factors such that the temperature decreases slowly when the acceptance probability is near 0.44 and fast when the probability is far from 0.44. VPR adopts a local search technique. When finding a neighboring state, it only accepts the state with modules changed in a limited range, which is initially set to the full chip and decreases as the temperature reduces. VPR has been shown to be of high quality, so many variants of VPR are developed to deal with other cost metrics.

SA is typically general and robust, very suitable for handling a design with multiple objectives. By moving a defective logic element to a spare space, the work [1] presented a fault-tolerant FPGA placer; by analyzing the impact of a clock architecture on FPGA clocking, the work [23] developed a cost function to reduce the delays of an FPGA design; by enhancing the accuracy of the net capacitance model, the work [42] reduced the power consumption of an FPGA design; by using static timing analysis, FPGA timing can be improved and variation-aware placement can be obtained [29]. Recently, the work [48] adopted an aging cost to avoid using old memory on an NVM-based FPGA; to reduce the power density caused by a shrunk feature size, the work [28] presented a hotspot-driven FPGA placement with a heat cost.

However, SA is often time-consuming. As the design complexity increases, the runtime of an SA-based FPGA placement might be prohibitively long. Although some existing work proposes to use parallel algorithms to reduce the runtime [14], more and more researchers migrate to the analytical formulation for FPGA placement.

C. Analytical Placement

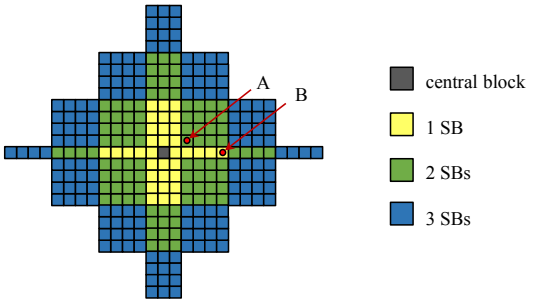
Modern FPGA placement generally consists of three major stages: global placement, legalization, and detailed placement. Global placement computes the best position for each module to minimize some cost metric (e.g., wirelength), ignoring the module non-overlapping constraint. Legalization then removes module overlaps and places the modules into their corresponding sites to minimize the displacement from global placement. Detailed placement further refines the solution quality. Among the three stages, global placement plays the most pivotal role in determining the quality and efficiency of the placement process. Recently, a significant paradigm shift for FPGA global placement (and even legalization) is moving from simulated annealing to analytical formulation [46]. Analytical placement computes the desired locations of modules under given constraints with a mathematical formulation. Recent works have shown analytical formulations to be the most effective for most FPGA placement problems [10, 22, 26–28, 37].

Global placement is often modeled as a wirelength optimization problem, with overlap reduction constraints. Thus, the key issues of global placement lie in the analytical models of wirelength, the techniques of overlap removal, and the integration and optimization of objective functions.

- **Wirelength model**: *Half-perimeter wirelength* (HPWL) of a net is the most popular wirelength model for placement [43]. However, a mathematical formulation requires its objective function and constraints to be differentiable, but HPWL is not differentiable everywhere. Thus, differentiable quadratic wirelength models and non-quadratic wirelength models were proposed. A quadratic model [43] uses the squared Euclidean wirelength to approximate HPWL, and non-quadratic models such as the weighted-average [19, 21] and log-sum-exponential [35] models were proposed to approximate HPWL. Quadratic models are intrinsically faster but less accurate, while non-quadratic models are more accurate but slower.
- **Overlap reduction**: To spread the concentrated overlap modules (due to wirelength minimization), various overlap reduction techniques were proposed. For FPGA placement, density guidance [11, 22, 38] and rough legalization [10, 26] are two common techniques. Density guidance moves modules along the descent direction of its density function, while rough legalization applies legalization to force modules to move to legal sites with minimized displacement.
- **Integration**: During global placement, we often need to handle the simultaneous optimization for multiple objectives. As a result, it is desirable for an analytical formulation to integrate these objectives for effective co-optimization. The most popular *penalty method* first integrates two objective functions W and D (say, wirelength and density, respectively) in the form of $W + \lambda D$, starts with minimizing W by assigning a small initial value for the penalty multiplier λ, and then gradually increases the value of λ to minimize D until some desired balance between W and D is achieved [9, 10, 22, 27, 38]. It should be noted that both objective functions W and D (or more) must be differentiable for this method.

Different from an ASIC, the heterogeneous logic structure and the segmented routing structure make the layout design for FPGAs very unique and much more difficult.

- **Heterogeneous Logic Structure:** The heterogeneous logic structure with CLBs, RAMs, and DSPs makes the FPGA placement region discrete. A logic module in a heterogeneous FPGA can only be placed on a site of its own type. So a placement without considering the heterogeneity could cause a large displacement during legalization. To handle this issue, the work [10, 11] presented *complex module models* with a dedicated density model for each type of modules. The work [22] then refined the complex module model to further consider the *narrow channel effect* between two different types of sites and target area congestion by a *coordinate transformation technique* and a *smoothed target area method*, respectively. The work [26] legalized heterogeneous logic modules by adding anchor points to legalized sites, and the works [22, 38] applied a density guiding method by adjusting the target density to make the density penalty on the legal site smaller.
- **Segmented Routing Structure:** Due to the segmented routing structure, the number of segments (switches), instead of wirelength, used by a net is the most dominating factor in determining routing delay in an FPGA [8, 13, 50]. Traditional measure of routing delay based on the geometric distance of a signal is thus not accurate for FPGAs. As shown in Figure 7, the delay for a signal net with fewer switches is smaller, even though its length is longer. To cope with this issue, the work [10] applied a sigmoid function [20] to smooth its delay model for timing optimization in an FPGA.

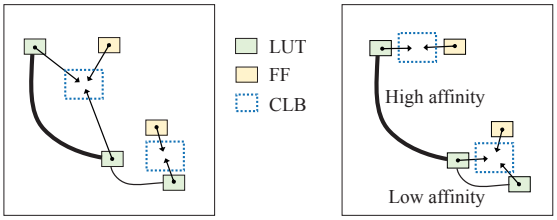


Module Position	Distance	HPWL	# of SBs	Delay (ps)
A	(2,1)	3	2	405
B	(5,0)	5	1	309

Fig. 7: The inter-module delay [10]. The number of segments (switches), instead of wirelength, used by a net is the most dominating factor in determining routing delay in an FPGA.

To achieve better placement results, a packing process to pack logic elements into CLBs is often incorporated into placement as a preprocessor. The general packing algorithm only considers the *affinity* of a logic module (i.e., the number of common inputs in a logic module). However, ignoring physical information may result in poor wirelength. The ideal position of a module with high affinity may be far away. As shown in Figure 8, LUTs and FFs are packed into a CLB considering their physical locations, not only their affinity. By honoring global placement results, physical-aware packing can reduce the total displacement effectively.

Here, we detail two state-of-the-art analytical placers. UTplaceF [26] is a quadratic placer, which consists of the following five major steps: (1) Flat initial placement (FIP): A quadratic programming is used to obtain an initial placement for LUTs, FFs (not CLBs), RAMs, I/Os, and DSPs. It applies legalization to remove module overlaps and adds anchor points to legalized sites to handle the heterogeneity. (2) Packing: Logic elements are packed into CLBs considering both affinity and physical information. (3) CLB-level global placement: An FIP-like process is employed for logic modules at the CLB level. (4) Legalization: A min-cost bipartite matching algorithm which matches CLBs to their sites



(a) Packing with affinity (b) Packing w/ physical information

Fig. 8: Physical-aware packing. The thickness of a connection between two LUTs represents the degree of affinity, and the length of an arrow represents the displacement after packing.

TABLE I: Comparisons of the three placement methods.

	Partitioning	SA	Analytical
Runtime	Fast	Slow	Fast
Quality	Worst	Good	Best
FPGA architecture awareness in existing works	Basic architecture with CLBs and uniform wire segments	Basic, Fault tolerance [1], Library-based one [31]	Basic, Segmented wire [17, 27], Heterogeneity [10], Physical-aware packing [26, 37, 38], Clocking [22]
Metrics in existing works	Basic metrics on wirelength, congestion, and timing [30]	Basic [2], Power [42], Heat [28], Aging [48]	Basic, Skew [49]

with minimized displacement is applied. (5) Detailed placement: An independent-set matching algorithm which matches independent CLBs to empty sites with minimized wirelength.

The work [22] additionally considers clocking resources and consists of the following three major steps: (1) Initial placement: A non-quadratic analytical engine is used to place logic modules. This analytical engine adopts the weighted-average wirelength model, uses a density function to spread modules, integrates objective functions with the penalty method, and applies a heterogeneous density map to guide modules moving to handle the heterogeneity. (2) Clock-aware packing: It sets up fence regions for all FFs, and packs the FFs considering not only the original constraints but also the number of clock signals. (3) Clock-aware placement: It uses fence regions to prevent CLB-level modules from moving out of their given clock regions, and then performs legalization and detailed placement to obtain the final placement results.

Analytical placement has been shown to be the most effective, with good scalability. As a result, it is the most popular not only for academic placers but also commercial tools [18, 37].

D. Comparisons

Table I summarizes the key strengths and weaknesses of the three types of placement algorithms. The partitioning-based placement is robust and has good scalability, but its quality is typically limited. The SA-based placement often can achieve good quality and apply to various metrics and constraints, but its scalability is often poor. The analytical placement is the most effective and also general for handling various objectives and constraints.

III. Routing

Routing is a complex combinatorial optimization problem, especially for modern FPGAs with limited, discrete routing structures. To handle the FPGA routing problems, researchers often model an FPGA as a weighted graph such that the graph topology can

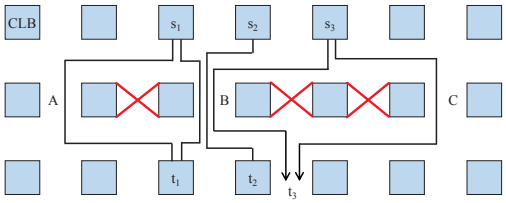


Fig. 9: A red cross represents a congested channel.

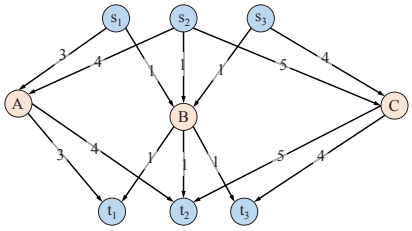


Fig. 10: The graph model of Figure 9. Each blue node denotes a CLB, s_1, s_2, and s_3 are three sources, and t_1, t_2, t_3 are three sinks. Each orange node represents a competing routing resource (i.e., the wire segments in channel A, B, or C).

represent the architecture of the FPGA. The weight on an edge is proportional to the congestion on the corresponding resources. Then, a graph-search technique is applied to find desired connections by dynamically updating the weights based on available resources.

Figure 9 illustrates such a graph modeling, where a node models a source, a sink, or some competing routing resource, and an edge denotes a connection. The competing resource can be a routing channel or a switch box. By setting the costs of edges and nodes properly, we can optimize predefined cost metrics. In Figure 9, the net i connects to the source s_i and the sink t_i, and Figure 10 gives the graph model of the circuit in Figure 9. Nodes A, B, and C are introduced in Figure 10 to represent the corresponding competing routing resources (channels or switch boxes), and an edge weight gives an approximate length of a connection.

Based on the net routing order, routing approaches can be classified into *sequential routing* where nets are routed one by one, and *concurrent routing* where all nets are routed simultaneously. We detail the two routing approaches by their most popular example algorithms in the following subsections.

A. Sequential Approach

A sequential router typically consists of two major steps: path searching and congestion removal. During path searching, we explore all feasible paths for each net and identify the most desired one. After path searching, some routing regions might be congested, and thus rip-up and rerouting is used to eliminate congestion. Assume that the capacity for each channel in Figures 9 and 10 is one, congestion violation could occur at node B (channel B) where more than one net tries to route through this node (channel or switch box). If such a congestion violation does occur, rip-up and rerouting is often employed to resolve the problem.

PathFinder [32], a negotiation-based routing algorithm, is a well-known FPGA sequential router. It is composed of a signal router and a global router which calls the signal router to route a net. The signal router uses the breadth-first search to find the shortest path measured by delay and congestion. The global router iteratively refines a route until one feasible routing solution is found. In each iteration, the

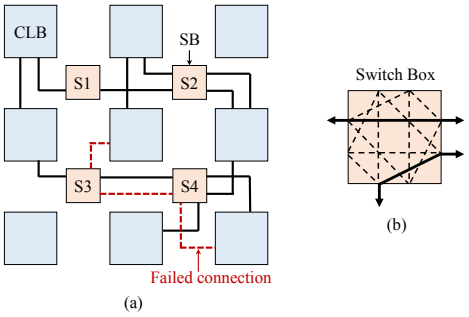


Fig. 11: An unroutable instance. Assume that the capacity of each routing channel is two. (a) A routing configuration satisfying the channel capacity of two, but is still unroutable because of the infeasible connection topology inside the switch box $S4$. (b) The internal connections of the switch box $S4$ (a Xilink XC4000-like switch box).

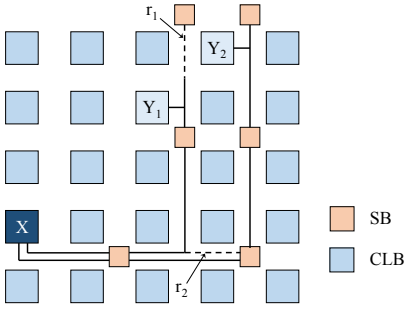


Fig. 12: The impact of placement on routing in terms of wire wastage and delay.

router rip-ups and reroutes all nets, not only the unroutable nets, in order to release the routing resources occupied by the routed nets. To determine the order of nets routing through the congested regions (e.g., node B in Figure 10) first, the following congestion cost function was proposed to spread congested nodes (wires) iteratively. At the i-th iteration, the cost for a node n is given by

$$\left(d_n + h_n^{(i)}\right) \cdot p_n,$$

where d_n is the base cost for delay, $h_n^{(i)}$ is the congestion at n in the history (i.e., *the history cost*), and p_n is the number of nets presently using n (i.e., *the congestion cost*). Here, $h_n^{(i)} = h_n^{(i-1)} + 1$ if n has an overflow; otherwise, $h_n^{(i)} = h_n^{(i-1)}$. With p_n, congested nets can negotiate with each other for using a routing resource (a channel or a switch box); with $h_n^{(i)}$, non-congested nets can release a resource for congested ones. Take Figure 10 for example. In the beginning, all the three nets occupy node B because their resulting delays are smallest. After a few iterations, the congestion cost and history cost associated with node B would increase because of increasing congestion; as a result, node B is no longer the best choice for nets 1 and 3, and thus both nets might choose other routes, removing the congestion.

However, PathFinder does not consider some impacts of the FPGA architecture. For example, the internal architecture of a switch box decides what can route through the box, the traditional measure of routing capacity along a channel is no longer accurate because the internal routing resource of a switch box is more restricted than that of a channel. Figure 11 shows an unroutable instance for an FPGA design. In Figure 11, the restricted connection topology inside a switch box makes the configuration unroutable, even though all the routes satisfy the channel capacity of two.

The works [6, 7] presented a mechanism to compute the capacity of a switch box off-line, developed a congestion metric based on the switch-box capacity, and employed the metric to guide FPGA routing. Existing works also extended PathFinder to handle various considerations, such as hold time violations and delay simultaneously [16, 44] and signal integrity [45]. To further speed up the process for sequential routing, some previous works resorted to parallelism [33, 39].

B. Concurrent Approach

A concurrent router routes all nets simultaneously. Existing concurrent routers are mainly based on the formulations of network flow [24], linear assignment [8, 50], and Boolean satisfiability (SAT) [15]. To cope with the high complexity of routing multiple nets simultaneously, the hierarchical framework with some aforementioned formulation is often adopted [8, 50].

The network-flow-based algorithm typically models a routing resource constraint as a capacity, a demand of routing resources as a flow, and routing wirelength/delay as a cost, and then finds a minimum-cost flow to identify a set of routes with minimized wirelength/delay [24].

The works [8, 50] considered the utilization of routing resources with a global viewpoint and identified desired routing paths with minimized costs based on linear assignment. To speed up the processing, a hierarchical top-down approach was adopted to assign routing paths to nets level by level.

The work [15] presented two routing-architecture-independent formulations for FPGA routing based on the SAT formulation which provides an exhaustive scheme to search for a desired routing solution. The SAT-based routing encodes an FPGA routing problem into a set of Boolean constraints such that any variable assignment satisfying the constraints gives a feasible routing solution; that is, the FPGA routing problem is converted into an equivalent SAT problem which is solved by a SAT solver. The exhaustive search behavior of a SAT solver can prove whether a feasible routing solution exists, a key advantage over other classical routing algorithms, yet at the cost of lower scalability.

Concurrent routing can have a global view for the FPGA routing problem and remedy the long-standing, net-ordering problem with sequential routing; however, its scalability is often limited, due to its higher complexity. For modern large-scale FPGA routing, as a result, sequential routing is still more popular for industrial applications.

IV. Simultaneous Placement and Routing

Placement and routing are highly dependent. Routing quality highly depends on placement configurations, and placement should address the constraints induced from the routing architecture and resources. Ideally, placement and routing should be performed simultaneously, but it is often too hard because of the high complexity even for placement or routing alone. Existing works applied partitioning [40], SA [34], and graph-based [5] algorithms for simultaneous FPGA placement and routing.

The graph-based work [5] presented an architecture-driven metric for simultaneous placement and global routing for FPGA designs. The metric considers the available segments and their lengths to optimize the cost for placement and global routing. Figure 12 illustrates the impact of placement on routing in terms of wire wastage and delay for an FPGA design, where X is the source node, Y_1 and Y_2 are two candidate positions for the sink, and r_1 and

r_2 denote two wasted wires. A longer unused wire incurs a larger capacitance (and thus also a larger delay). So placing the sink at Y_2 would lead to a smaller delay than that at Y_1, even if their wire segment usages are the same. Based on this observation, the work [5] developed an architecture-driven metric for simultaneous FPGA placement and global routing. The placer first modeled the chip architecture into a graph with multiple routing layers modeling wire segments of different lengths. An architecture-driven cost metric was derived based on the configurations of wire segments and congestions, where the delay cost was measured mainly by the number of segments (switches) used, penalized with wasted wires. A graph-clustering based algorithm was used to demonstrate the superiority of the architecture-driven cost over a classical one based on geometric distance.

V. Future Research Directions

Although recent works have significantly advanced FPGA placement and routing, many emerging challenges arise from growing complexity, diverse objectives, high heterogeneity, and evolving technologies. In this section, we present some potential research directions for modern FPGA placement and routing.

A. Wirelength Model

As shown in Figures 2, 3, and 7 and mentioned earlier, nonuniform wire segments make delay non-linear and discrete to their geometric distance, and the heterogenous logic structure incurs discrete displacements. To minimize routing delay, we need to place modules to the positions with fewer wire segments (switches) and logic columns of proper types. However, it is often difficult to compute the number of required wire segments and select the best column for a position, due mainly to the nonuniform wire segment lengths and discrete logic columns. The work [10] used inter-module delay information to develop a routing-architecture-aware cost model to consider the information with wire segments and logic columns, then applied interpolation to transform the model into a continuous function, and finally employed a quadratic sigmoid function to smooth the function; see Figure 13 for an illustration. The recent work [22] then refined the cost model to further consider narrow channel effects and target area congestion. So a question arises: Is there any other more effective and efficient model to fully address the heterogeneity with modern FPGAs.

B. Mixed-Size Placement

IP reuse and library-based implementations with presynthesized modules become popular in modern FPGA designs, which can significantly shorten the design cycle. They both lead to logic modules similar to the macros in classical ASIC designs. With more restricted logic and routing resources, further, the alignments of macro-like modules to their corresponding sites make this FPGA placement problem much more difficult than that for ASICs. See Figure 14 for an example. It is obvious that this FPGA "mixed-size" placement brings up many more challenges than its counterpart in ASICs.

C. Parallel Placement for Heterogeneous FPGAs

Existing works for parallel FPGA placement consists of two types: (1) SA-based parallel placement [14] partitioned a problem into subproblems and solved the subproblems simultaneously. (2) Analytical placement used parallelism [12] to accelerate the updates of module positions. However, both parallel algorithms do not consider the heterogeneity of FPGAs, which may cause significant difficulties for the parallelism.

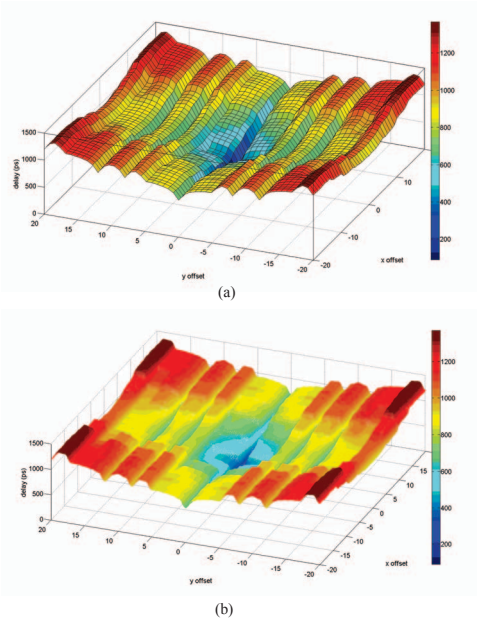


Fig. 13: The routing-architecture-aware cost model which considers the segmented routing structure and the heterogenous logic structure. (a) The interpolated delay function. (b) The smoothed delay function [10].

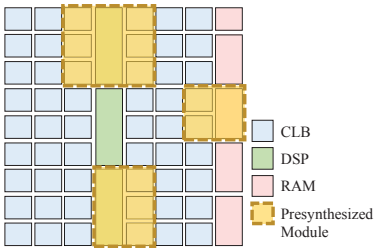


Fig. 14: Library-based placement with big macros.

D. Placement with New Clock Architectures

Modern heterogeneous FPGAs often adopt new clock architectures, where an FPGA chip is divided into a set of clock regions (see Figure 3), and a clock region is further divided into half columns, illustrated in Figure 15. The outermost rectangle gives a clock region, which consists of two upper half columns and two lower ones, and the modules colored red, blue, and green are connected to different clock signals. If only two types of clock are allowed to be placed in one half column, then the upper-right half column violates the clocking constraint. It is thus desirable to consider such a clocking constraint for FPGA placement.

Another research direction is clock-aware timing-driven placement. Considering the locations of clock buffers and networks, located between the upper and lower half columns, an FPGA placer can leverage this feature to resolve some timing violations.

E. NVM-based FPGA Placement

Although more area-efficient and power-saving, NVM-based FPGAs have longer write time and shorter device endurance; as a result, it is desirable to consider resource reuses in a series of implementations to resolve the two significant problems. The recent

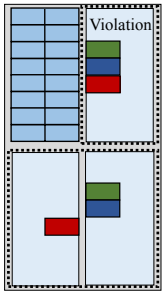


Fig. 15: Half-column legalization. The outermost rectangle gives a clock region, containing two upper half columns and two lower ones. If only two types of clock are allowed in one half column and the green, blue, and red modules are triggered by three different clocks, then the upper-right half column violates the clocking constraint.

work [48] studied the aging effects of logic/memory hybrid modules during placement, while the work [47] handled the reuses of switch boxes during routing. There is still no existing work dealing with the reuses of both logic and routing resources during placement and routing.

F. Simultaneous Analytical Placement and Routing

Due to the tight mutual dependency of placement and routing with the special modern FPGA architectures, it should be promising to explore such dependency for simultaneous placement and routing, especially under an analytical framework like the work [36] because such a framework has been shown to be the most effective for modern FPGA designs.

G. Machine Learning for FPGA Placement and Routing

Machine learning (ML) is an emerging generic algorithm that can provide a desired prediction and solution for many complex design problems. During placement, it is often difficult to predict the subsequent routing behaviors (e.g., routed wirelength, congestion, timing). ML could provide some insightful prediction and evaluation to facilitate the FPGA placement and routing processes.

VI. Conclusions

We have introduced the basic architectures of FPGAs, described popular placement and routing algorithms for FPGAs, and provided key future research directions for FPGA placement and routing. With the increasing design complexity, diverse design requirements, high heterogeneity, and new technologies, we expect more emerging challenges for FPGA placement and routing are yet to come in the near future.

References

[1] A. Agarwal, J. Cong, and B. Tagiku, "Fault tolerant placement and defect reconfiguration for nano-FPGAs," *Proc. ICCAD*, pp. 714–721, Nov. 2008.

[2] V. Betz and J. Rose, "VPR: a new packing, placement and routing tool for FPGA research," *Proc. FPL*, pp. 213–222, Sept. 1997.

[3] S. D. Brown, R. J. Francis, J. Rose, and Z. G. Vranesic, *Field-Programmable Gate Arrays*, Kluwer Academic Publishers, 1992.

[4] Y.-W. Chang, *Routing architecture and CAD for field-programmable gate arrays*, Ph.D. Dissertation, The University of Texas at Austin, Aug. 1996.

[5] Y.-W. Chang and Y.-T. Chang, "An architecture-rriven metric for simultaneous placement and global routing for FPGAs," *Proc. DAC*, pp. 567–572, June 2000.

[6] Y.-W. Chang, S. Thakur, K. Zhu, and D. F. Wong, "A new global routing algorithm for FPGAs," *Proc. ICCAD*, pp. 380–385, Nov. 1994.

[7] Y.-W. Chang, D. F. Wong, and C. K. Wong, "FPGA global routing based on a new congestion metric," *Proc. ICCD*, pp. 372–378, Oct. 1995.

[8] Y.-W. Chang, K. Zhu, and D. F. Wong, "Timing-driven routing for symmetrical-array-based FPGAs," *ACM TODAES*, vol. 5, no. 3, pp. 433–450, July 2000.

[9] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang, "NTUplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints," *IEEE TCAD*, vol. 27, no. 7, pp. 1228–1240, July 2008.

[10] S.-Y. Chen and Y.-W. Chang, "Routing-architecture-aware analytical placement for heterogeneous FPGAs," *Proc. DAC*, June 2015.

[11] Y.-C. Chen and Y.-W. Chang, "Efficient and effective packing and analytical placement for large-scale heterogeneous FPGAs," *Proc. ICCAD*, Nov. 2014.

[12] J. Cong and Y. Zou, "Parallel multi-level analytical global placement on graphics processing units," *Proc. ICCAD*, pp. 681–688, Nov. 2009.

[13] B. Fallah and J. Rose, "Timing-driven routing segment assignment in FPGAs," *Proc. Canadian Conf. VLSI*, 1992.

[14] C. Fobel, G. W. Grewal, and D. Stacey, "A scalable, serially-equivalent, high-quality parallel placement methodology suitable for modern multicore and GPU architectures," *Proc. FPL*, Sept. 2014.

[15] H. Fraisse, A. Joshi, D. Gaitonde, and A. Kaviani, "Boolean satisfiability-based routing and its application to Xilinx ultraScale clock network," *Proc. FPGA*, pp. 74–79, Feb. 2016.

[16] R. Fung, V. Betz, and W. Chow, "Simultaneous short-path and long-path timing optimization for FPGAs," *Proc. ICCAD*, pp. 838–845, Nov. 2004.

[17] P. Gopalakrishnan, X. Li, and L. T. Pileggi:, "Architecture-aware FPGA placement using metric embedding," *Proc. DAC*, pp. 460–465, June 2006.

[18] M. Gort and J. H. Anderson, "Analytical placement for heterogeneous FPGAs," *Proc. FPL*, pp. 143–150, Aug. 2012.

[19] M.-K. Hsu, V. Balabanov, and Y.-W. Chang, "TSV-aware analytical placement for 3D IC designs based on a novel weighted-average wirelength model," *IEEE TCAD*, vol. 32, no. 4, pp. 497–509, Apr. 2013.

[20] M.-K. Hsu and Y.-W. Chang, "Unified analytical global placement for large-scale mixed-size circuit designs," *IEEE TCAD*, pp. 1366-1378, vol. 31, no. 9, Sept. 2012.

[21] M.-K. Hsu, Y.-W. Chang, and V. Balabanov, "TSV-aware analytical placement for 3D IC designs," *Proc. DAC*, pp. 664–669, June 2011.

[22] Y.-C. Kuo, C.-C. Huang, S.-C. Chen, C.-H. Chiang, Y.-W. Chang, and S.-Y. Kuo, "Clock-aware placement for large-scale heterogeneous FPGAs," *Proc. ICCAD*, Nov. 2017.

[23] J. Lamoureux and S. J. E. Wilton, "Clock-aware placement for FPGAs," *Proc. FPL*, pp. 124–131, Aug. 2007.

[24] S. Lee, H. Xiang, D. F. Wong, and R. Y. Sun, "Wire type assignment for FPGA routing," *Proc. FPGA*, pp. 61–67, Feb. 2003.

[25] G. Lemieux and D. Lewis, *Design of Interconnectuon Networks for Programmable Logic*, Kluwer Academic Publishers, 2004.

[26] W. Li, S. Dhar, and D. Z. Pan, "UTPlaceF: a routability-driven FPGA placer with physical and congestion aware packing," *Proc. ICCAD*, Nov. 2016.

[27] T.-H. Lin, P. Banerjee, and Y.-W. Chang, "An efficient and effective analytical placer for FPGAs," *Proc. DAC*, June 2013.

[28] W. Lu, Y. Hu, J. Ye, and X. Li, "TeSHoP: A temperature sensing based hotspot-driven placement technique for FPGAs," *Proc. FPL*, Aug. 2016.

[29] G. Lucas, C. Dong, and D. Chen, "Variation-aware placement for FPGAs with multi-cycle statistical timing analysis," *Proc. FPGA*, pp. 177–180, Feb. 2010.

[30] P. Maidee, C. Ababei, and K. Bazargan, "Fast timing-driven partitioning-based placement for island style FPGAs," *Proc. DAC*, pp. 598–603, June 2003.

[31] F. Mao, W. Zhang, B. He, and S. Lam, "Dynamic module partitioning for library based placement on heterogeneous FPGAs," *Proc. FCCM*, pp. 194, Apr. 2017.

[32] L. McMurchie and C. Ebeling, "PathFinder: a negotiation-based performance-driven router for FPGAs," *Proc. FPGA*, pp. 111–117, Feb. 1995.

[33] Y. O. M. Moctar and P. Brisk, "Parallel FPGA routing based on the operator formulation," *Proc. DAC*, June 2014.

[34] S. Nag and R. A. Rutenbar, "Performance-driven simultaneous place and route for row-based FPGAs," *Proc. DAC*, pp. 301–307, June 1994.

[35] W. C. Naylor, R. Donelly, and L. Sha, "Non-linear optimization system and method for wire length and delay optimization for an automatic electric circuit placer," US Patent 6,301,693, 2001.

[36] H.-C. Ou, H.-C. C. Chien, and Y.-W. Chang, "Simultaneous analog placement and routing with current flow and current density considerations," *Proc. DAC*, June 2013.

[37] R. Pattison, Z. Abuowaimer, S. Areibi, G. Grewal, and A. Vannelli, "GPlace: a congestion-aware placement tool for ultrascale FPGAs," *Proc. ICCAD*, Nov. 2016.

[38] C.-W. Pui, G. Chen, W.-K. Chow, K.-C. Lam, J. Kuang, P. Tu, H. Zhang, E. F. Y. Young, and B. Yu, "RippleFPGA: a routability-driven placement for large-scale heterogeneous FPGAs," *Proc. ICCAD*, Nov. 2016.

[39] M. Shen and G. Luo, "Accelerate FPGA routing with parallel recursive partitioning," *Proc. ICCAD*, pp. 118–125, Nov. 2015.

[40] N. Togawa, M. Sato, and T. Ohtsuki, "A simultaneous placement and global routing algorithm with path length constraints for transport-processing FPGAs," *Proc. ASP-DAC*, pp. 569–578, Jan. 1997.

[41] S. Trimberger, *Field-Programmable Gate Array Technology*, Kluwer Academic Publishers, 1996.

[42] K. Vorwerk, M. Raman, J. Dunoyer, Y.-C. Hsu, A. Kundu, and A. A. Kennings, "A technique for minimizing power during FPGA placement," *Proc. FPL*, pp. 233–238, Sept. 2008.

[43] L.-T. Wang, Y.-W. Chang, and K.-T. Cheng, *Electronic Design Automation: Synthesis, Verification, and Test*, Morgan Kaufmann, 2009.

[44] E. Wegley and Q. Zhang, "Application of specific delay window routing for timing optimization in FPGA designs," *Proc. FPGA*, pp. 37–45, Feb. 2015.

[45] S. J. E. Wilton, "A crosstalk-Aware timing-driven router for FPGAs," *Proc. FPGA*, pp. 21–28, Feb. 2001.

[46] Xilinx, "Xilinx unveils the Vivado design suite for the next decade of 'all programmable' devices," June 2012.

[47] Y. Xue, P. Cronin, C. Yang, and J. Hu, "Fine-tuning CLB placement to speed up reconfigurations in NVM-based FPGAs," *Proc. FPL*, Sept. 2015.

[48] Y. Xue, C. Yang, and J. Hu, "Age-aware logic and memory co-placement for RRAM-FPGAs," *Proc. DAC*, June 2017.

[49] C.-Y. Yeh and M. Marek-Sadowska, "Skew-programmable clock design for FPGA and skew-aware placement," *Proc. FPGA*, pp. 33–40, Feb. 2005.

[50] K. Zhu, Y.-W. Chang, and D. F. Wong, "Timing-driven routing for symmetrical-array-based FPGAs," *Proc. ICCD*, pp. 628–633, Oct. 1998.