FLORA: FLoorplan Optimizer for Reconfigurable Areas in FPGAs

BIRUK B. SEYOUM, ALESSANDRO BIONDI, and GIORGIO C. BUTTAZZO,
Scuola Superiore Sant'Anna

Floorplanning is a mandatory step in the design of hardware accelerators for FPGA platforms, especially when adopting dynamic partial reconfiguration (DPR). This paper presents FLORA, an automated floorplanner based on optimization via Mixed-Integer Linear Programming (MILP). The floorplanning problem is solved by means of a novel fine-grained modeling strategy of FPGA resources. Furthermore, differently from other proposals, our approach takes into account several realistic Partial Reconfiguration (PR) floorplanning constraints on FPGAs. FLORA was compared against state-of-the-art floorplanners by means of benchmark suites, showing that it is capable of providing better performance in terms of resource consumption, maximum inter-region, wire-length, and running time required to produce the solutions. Finally, FLORA was utilized to generate placements for a partially-reconfigurable video processing engine that was implemented on a Xilinx Zynq-7020.

CCS Concepts: • **Computer systems organization** → *Reconfigurable computing*; *Heterogeneous (hybrid) systems*; • **Hardware** → *Reconfigurable logic applications*;

Additional Key Words and Phrases: FPGA, Heterogenous SoC, Floorplanning, partial-reconfiguration, dynamic reconfiguration, MILP

ACM Reference format:
Biruk. B. Seyoum, Alessandro Biondi, and Giorgio C. Buttazzo. 2019. FLORA: FLoorplan Optimizer for Reconfigurable Areas in FPGAs. *ACM Trans. Embed. Comput. Syst.* 18, 5s, Article 73 (October 2019), 20 pages.
https://doi.org/10.1145/3358202

1 INTRODUCTION

System-on-chips (SoC) based on *field-programmable gate arrays* (FPGA) are promising platforms to address the needs of modern cyber-physical systems, offering the possibility to deploy high-performance and energy-efficient hardware accelerators onto the FPGA fabric. In addition, modern FPGAs come with a very interesting feature, named *dynamic partial reconfiguration* (DPR). With DPR, it is possible to dynamically reconfigure a portion of the FPGA area while the modules programmed on the rest of the area continue to operate. This feature opens a new dimension in resource management for FPGAs and, similarly to multitasking for classical processors, it allows *virtualizing* the available area by interleaving the configuration of multiple hardware modules that would not fit in the available area under a static allocation [3].

This article appears as part of the ESWEEK-TECS special issue and was presented in the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2019.
Authors' addresses: B. B. Seyoum, A. Biondi, and G. C. Buttazzo, Scuola Superiore Sant'Anna, Via G. Moruzzi, 1, Pisa, Pisa, 56124; emails: {b.seyoum, al.biondi, g.buttazzo}@santannapisa.it.

1539-9087/2019/10-ART73 $15.00
https://doi.org/10.1145/3358202

However, to properly exploit the power of FPGA-based SoCs, today's technologies still require a considerable expertise with hardware design and, with respect to software development, involve complex design flows in which some steps must be *manually* performed. Despite FPGA vendors, such as Xilinx, are pushing for improving the programmability of FPGAs, e.g., by means of high-level synthesis (HLS) and tools such as SoC Accel, there are still some design steps that do not dispose of proper automated tools to perform the corresponding tasks.

In particular, consider the design flow under DPR. The first stage corresponds to the synthesis of the behavioral description of hardware modules (written in hardware description languages such as Verilog or VHDL, or obtained via HLS), which are converted into gate-level net-lists. The second stage consists of *floor-planning*, i.e., the geometrical placement within the FPGA fabric of regions in which the net-lists will be implemented. Finally, the last stage requires the implementation (and the corresponding routing) of the net-lists within the regions selected in the previous stage.

Unfortunately, the floorplanning requires the manual intervention of the designer, which has to be experienced with FPGAs. The floorplanning for partial reconfiguration (PR) is even more complex than the standard floorplanning for static areas, as it involves generating placements that must also adhere to an additional set of non-trivial PR-related constraints [13]. Also, the decisions made during floor-planning may severely impact resource utilization and performance. To achieve a fully-automated design flow for FPGAs, hence enhancing their programmability, automated solutions to perform the floor-planning are required.

Contribution. This paper aims at filling this gap in the design flow for partial reconfiguration by proposing FLORA, an automated floor-planner based on optimization via Mixed-Integer Linear Programming (MILP). Differently from previous proposals, the proposed approach adopts a novel fine-grained modeling of FPGA resources, taking into account several realistic technological constraints mandated by commercial FPGA design tools. FLORA has been tested on platforms by Xilinx using a synthetic benchmark suite and its performance has been compared with other two state-of-the-art floorplanners.

Paper Structure. The remainder of this paper is organized as follows. Section 2 provides an analysis of the state-of-the-art based on a categorization of previously-proposed floorplanners for PR. Section 3 presents the adopted FPGA model with some background analysis. Section 4 provides a systematic definition of the floorplanning problem. The proposed MILP formulation of the floorplanning problem, the definitions of the constraints, and the objective function are presented in Section 5. Section 6 reports the experimental results. Finally, Section 7 concludes the paper and discusses possible future work.

2 RELATED WORK

Over the years, several authors proposed different FPGA floor-planners for partial reconfiguration. Earlier works [12, 16] focused on floor-planning by considering only a single type of resource (mostly CLBs), while later works [1, 6, 15] started to consider different types of resources (CLBs, BRAMs, DSPs) with a *uniform* layout on the FPGA fabric. Both of these approaches considered a simplified model of the device and therefore they would not be suitable for real-world FPGA families, which consist of *heterogeneous resources with a non-uniform distribution*. Furthermore, with the new generations of FPGA families, the requirements for DPR became more complex. This section only focuses on reviewing the works that targeted heterogeneous resources distributed in a non-uniform manner.

The approaches proposed in the literature can be differentiated by those that apply a *direct* representation of the FPGA fabric, and those that apply an *indirect* representation.

Direct representation. Few works adopted a direct representation of the solution space for their floor-planning algorithm [10, 14]. Among the solutions that do not use heuristics and optimization, Vipin et al. [14] proposed an iterative algorithm defining each tile of resources as a kernel. Their approach uses primary information about the types and locations of each kernel, and prioritizes regions based on the type and number of their resource requirement. Other authors [10] achieved improvements on the quality of the solution obtained by this approach.

Specifically, Rabozzi et al. [10] proposed two algorithms, (HO) and (O), based on a MILP formulation. The first algorithm (HO) improves the quality of sub-optimal solutions of other heuristic approaches, such as [5]. The second algorithm (O) directly encodes the floor-planning problem as a MILP formulation. Similar to [14], the authors adopted a strategy for reducing the minimum reconfigurable unit into tiles. (HO) is dependent on the solution of other approaches while (O) is capable of exploring the whole solution space. As the problem instance gets harder (higher utilization and higher number of reconfigurable regions), (O) was reported [10] as being very slow to even find a feasible solution and it had to be initialized with a sub-optimal solution of other algorithms. A key characteristic of [10] is that the authors partitioned the FPGA fabric into abstract rectangles called portions. This reduced the solution space to be explored at the expense of reducing the precision of the formulation.

The solution proposed in this paper strongly differs from the one in [10] as **(i)** it uses a more fine-grained modeling strategy of the FPGA fabric and **(ii)** it supports more realistic (and updated with respect to the today's FPGA technology) constraints mandated by synthesis tools for partial reconfiguration. Furthermore, as it will be detailed in Section 6, our solution significantly improves on the performance of [10].

Indirect representation. Indirect representations of the FPGA fabric were proposed by using different methods, such as slicing trees, sequence pairs [5], and binary trees, while other authors constructed different types of higher-level abstract structures to characterize the distribution of resources on the FPGA area [8, 9, 11]. In a follow up work [9, 11], Rabozzi et al. proposed an improvement to their earlier work, using again an MILP optimization but this time defining an abstract structure called *conflict graph* to describe the feasible placements for all reconfigurable regions and the conflicts between them. This approach improved the results achieved by (HO) and (O). The same authors [9] tried to improve their work by employing a similar concept of conflict graph and replacing the MILP optimization with genetic algorithms. Nguyen et al. [8] proposed a floorplanner where a biparitioning heuristic is used to find a feasible placement. Similarly to [9, 11], their algorithm first iteratively lists all the possible placements for each region on the FPGA fabric and constructs a graph based on these placements.

Unfortunately, the approaches in [8, 9, 11] suffer of a crucial shortcoming when used in floor-planning for partial reconfiguration: *the resources required by each reconfigurable region must be known in advance to generate the indirect representation* (e.g., the conflict graph in [9, 11]) before performing the actual optimization. This requirement may not be suitable for all cases in which the amount of resources within each reconfigurable region must be computed at the stage of optimization. For instance, if floorplanning optimization is jointly considered with a partitioning phase to assign reconfigurable modules to reconfigurable regions, such as in [4], the resource requirement of the regions *is not known a-priori* as it depends on the reconfigurable modules they host. That is, each reconfigurable region must dispose of enough resources to host all hardware modules assigned to it, which are unknown before starting the optimization. Note that a two-stage approach that first partitions the modules into a set of reconfigurable regions, and then performs the floorplanning, may lead to sub-optimal solutions.

For this reason, this work focuses on a direct representation of the FPGA fabric.

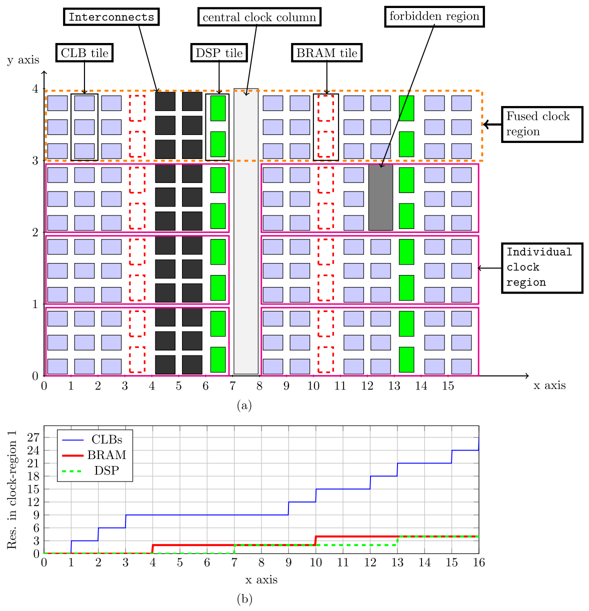


Fig. 1. (a) Resource layout of a sample FPGA and (b) its resource fingerprint.

3 BACKGROUND AND MODELING

This section briefly describes the general architecture of FPGAs, together with a set of technological constraints related to floor-planning for partial reconfiguration, and the corresponding model adopted in this paper. This work is based on the 7-series Virtex and Zynq FPGA families from Xilinx, but can easily be adapted for older FPGA families by disabling some of the constraints related to modern FPGA families

3.1 FPGA Architecture and Technological Constraints

FPGAs are characterized by heterogeneous resources distributed in a non-uniform manner across the fabric. The configurable fabric of Xilinx FPGAs is divided into quadrants named *clock regions*. As it is illustrated in Figure 1(a), each clock region includes columns of different configurable resources, such as CLBs, BRAMs, or DSPs.

A single column in a clock region is named *tile* and contains resources of the same type. The number of resources in a tile varies depending on the FPGA family. For example, in the Virtex 7 family FPGAs, a CLB tile contains 50 CLBs, while BRAM and DSP tiles contain 10 BRAMs and 20 DSPs each, respectively.

Xilinx FPGAs, in particular 7-series devices, also contain specific tiles denoted as *interconnect tiles*, which serve to realize routing. These tiles are placed back-to-back in groups of two as shown in Figure 1(a) (black boxes). Finally, the configurable fabric is also characterized by a central clock column and by *forbidden regions* (such as clock buffers).

The problem of floorplanning for partial reconfiguration consists in geometrically placing *reconfigurable regions* (RR) within the total area available on the fabric. Each RR can host multiple hardware modules (one at a time). Note that the placement of RRs cannot be arbitrary, as Xilinx tools pose the following constraints:

- RRs must be rectangular;
- the vertical boundaries of each RRs cannot be placed between pairs of back-to-back interconnect tiles;
- forbidden regions cannot be included in RRs; and
- the horizontal boundaries of each RR shall be aligned to the boundary of a tile (i.e., the height of RRs spans a full clock-region) to improve the design performance.[1]

The solution proposed in this paper is able to hande all these constraints.

Note that, as shown in Figure 1(a), the central clock column of Xilinx FPGAs divides the FPGA into left and right regions. Since clock regions do not pose particular constraints for placing the vertical boundaries of RRs, it is convenient to fuse each pair of horizontally adjacent clock regions into a single one, which is denoted as fused clock region in Figure 1(a). To simplify terminology, from now on fused clock regions will be simply denoted by clock regions.

3.2 Model

The FPGA area is identified with a discrete Cartesian coordinate system, placing the origin at the bottom-left corner. Each unit on the x-axis denotes a line that separates columns of resources (CLB, BRAM, DSP, interconnects, central clock column), while each unit on the y-axis represents a line that separates clock regions (see Figure 1(a)). Note that a different granularity is adopted between the axes. The FPGA area is hence said to be W columns wide and H clock regions high. The area includes N_{int} pairs of back-to-back interconnect columns, and the x coordinate of the line separating the z-th pair is denoted by I_z, with $z = 1, \ldots, N_{\text{int}}$. For instance, in Figure 1(a), $N_{\text{int}} = 1$ and $I_1 = 5$.

This work considers the problem of floor-planning a set $\mathcal{R} = \{R_1, \ldots, R_{N_r}\}$ of N_r RRs. Each reconfigurable region R_i is characterized by a vector $\mathbf{c_i}$ of *resource requirements*, where $c_{i,t}$ denotes the amount of resources required by R_i for each type $t \in \{\text{CLB, BRAM, DSP}\}$. Clearly, the amount of resources required by each RR must be enough to host any hardware module that can be programmed into it (i.e., $\mathbf{c_i}$ must reflect a component-wise maximum of the resource requirements of each module).

Once a floor plan is obtained, each reconfigurable region R_i can be characterized by a tuple $r_i = (x_i, y_i, w_i, h_i)$ where x_i and y_i represent its bottom-left coordinates, and w_i and h_i represent its width and height, respectively. A valid floor plan must hence guarantee that the following inequalities hold for each reconfigurable region $R_i \in \mathcal{R}$:

$$x_i + w_i \leq W \ \wedge \ y_i + h_i \leq H. \tag{1}$$

Forbidden regions are encapsulated within minimal-bounding rectangles, which are identified with the same tuple used for RRs: the parameters specifying the rectangle associated with the k-th forbidden region are denoted by $\delta_k = (x_k, y_k, w_k, h_k)$.

RRs may be connected (e.g., to realize direct communication channels): $Q_{i,k}$ denotes the number of wires connecting R_i to R_k, where $Q_{i,k} = 0$ if the two RRs are not connected.

FPGA resource finger-printing. This paper adopts a new modeling strategy to cope with the heterogeneous distribution of FPGA resources over the x-axis. As illustrated in Figure 1(b), the distribution of resources can be modeled with piece-wise functions $f_t(x)$, where $f_t(x^*)$ denotes

[1]This constraint is driven by combining a less restrictive Xilinx PR constraint and a good design practice. The constraint states that two RRs cannot be stacked on top of each other in the same clock region (i.e., they cannot share columnar resources in the same clock region). Aligning horizontal boundaries of RRs to clock regions is a good design practice in that doing so allows the designer to take advantage of the fabric's native reset-after-reconfiguration feature to improve synchronization and avoid implementing a custom reset-after-reconfiguration logic for the RRs.

the amount of resources of type t included in a clock region within the range $[0, x^*]$. For instance, in Figure 1, $f_t(9) = 12$ when t = CLB since the range $[0, 9]$ in the clock region includes the tiles of the first, second, third, and ninth columns, each composed of three CLBs.

Using the function $f_t(x)$, it is possible to analytically define $\eta_{i,t}$, which represents the total amount of type $t \in$ {CLB, BRAM, DSP} of resource within a reconfigurable region R_i, i.e.,

$$\eta_{i,t} = h_i \cdot (f_t(x_i + w_i) - f_t(x_i)). \tag{2}$$

Note that these functions are defined by assuming the same distribution of resources across all FPGA rows, i.e., by neglecting the existence of forbidden regions: this is because a valid floor plan cannot comprise RRs that include forbidden regions, and hence there is no need to explicitly model their (negative) contribution to the resources available in an area. In other words, the impact of forbidden regions is *binary*: either a placement is valid, and they do not have to be taken into account, or it is not, and hence it has no meaning to compute the resources within an area.

4 PROBLEM DEFINITION

This paper proposes an approach based on Mixed-Integer Linear Programming (MILP) to compute a floor plan for a set $\{R_1, \dots, R_{N_r}\}$ of N_r RRs under the modeling strategies reported in Section 3.2. A valid floor plan is characterized by the following properties:

- the geometrical placement satisfies the constraints reported in Section 3.1;
- for each reconfigurable region R_i, and for each resource type $t \in$ {CLB, BRAM, DSP}, it holds that R_i encompasses at least $c_{i,t}$ units of resources of type t; and
- RRs do not overlap.

The floor-planning algorithm takes as input **(i)** a description of the resource distribution on the FPGA area, i.e., its *layout*, from which it is possible to obtain functions $f_t(x)$, and the set of forbidden regions; **(ii)** the set of RRs with their resource requirements $\mathbf{c_i}$; and **(iii)** the interconnections between regions characterized by the number of wires that implement each connection. The logical and computational components of the FPGA that are required to implement the communication between RRs and the static region (such as AXI interconnects) are considered as part of the static region. The algorithm outputs the coordinates and the sizes of each RR.

4.1 Evaluation Metrics

The floor-planning problem can have multiple valid solutions—i.e., as long as a valid geometrical placement is found for each RR, the floor-planning is considered correct. However, not all valid floor-plans lead to the same performance. This work considers the following two metrics to evaluate the solutions produced by the proposed algorithm.

Wasted resources (WR): The wasted resources inside a reconfigurable region R_i refer to the amount of extra resources contained in the area in which R_i is geometrically placed beyond the amount of resources required by the RR. Since different numbers of resources are available for each type, and since the types differ in function, each wasted resource is associated to a type-specific weight. Formally, let v_t be the cost associated to wasting a resource of type t. Then, the WR metric ω is defined as

$$\omega = \sum_{i=1}^{N_r} \sum_{t \in \{\text{CLB}, \text{BRAM}, \text{DSP}\}} v_t \cdot (\eta_{i,t} - c_{i,t}), \tag{3}$$

where $\eta_{i,t}$ denotes the number of resources of type t included in R_i and $c_{i,t}$ denotes the amount of resources of type t required by R_i.

This metric accounts for the weighted number of wasted resources within each region R_i ($i = 1, \ldots, N_r$). A natural choice for the weights v_t corresponds to the case in which the number of wasted resources are normalized to the total amount of resources T_t available for each type t, i.e., $v_t = 1/T_t$. Note that, besides favoring the extensibility of a floor-plan, lowering the number of wasted resources may reduce the reconfiguration time. Indeed, the smaller the RR, the lower the size of the corresponding bitstream. As the reconfiguration process consists in copying a bitstream into the FPGA configuration memory, smaller bitstreams correspond to shorter reconfiguration times.

Maximum inter-region wire-length (MIW): The wire-length between two reconfigurable regions R_i and R_k is defined as the product of (i) the Manhattan distance between the centroids of R_i and R_k, and (ii) the total number $Q_{i,k}$ of wires between them.

Let S be a set that contains all the interconnections between RRs in the design. The elements of this set are tuples of the form $(R_i, R_k, Q_{i,k})$ where $R_i \in \mathcal{R}$, $R_k \in \mathcal{R}$, and $Q_{i,k} \geq 0$ is the number of wires between R_i and R_k that realize the corresponding interconnection. To support the presentation of the MIW metric, $P_{i,x}$ and $P_{i,y}$ are defined as the centroids of a region R_i in the x and y axes, respectively. The centroids can be simply computed as $P_{i,x} = x_i + w_i/2$ and $P_{i,y} = y_i + h_i/2$.

Finally, the inter-region wire-length between regions R_i and R_k can mathematically be defined as the product of the Manhattan distance between the centroids of the RRs and the number of wires between them. The total inter-region wire-length, Ω, is then obtained as the sum of the wire-length between all interconnected regions, that is

$$\Omega = \sum_{(R_i, R_k, Q_{i,k}) \in S} Q_{i,k} \cdot \mid (P_{i,x} - P_{k,x}) \mid + \mid (P_{i,y} - P_{k,y}) \mid \tag{4}$$

The MIW metric can be extended to account for connections between a RR and a portion of the static region (e.g., in which an AXI interconnect is placed) by considering the centroid of the latter.

These metrics (WR and MIW) can either be singularly considered or combined in a single performance index (e.g., each weighted by a scalar factor). In both the cases, they can be used to define an *objective function* to be minimized at the stage of optimization: this is addressed in Section 5.5. Although this paper focuses on these two metrics, the proposed solution is prone to be extended for supporting other metrics such as the aspect ratio of RRs and the wire-length for the connections between RRs and I/O ports.

5 MILP FORMULATION

This section presents the proposed MILP formulation to perform floorplanning. First (Section 5.1), the main optimization variables are introduced and then the constraints are presented. The constraints are split into (i) *structural* constraints (Section 5.2), (ii) constraints related to *resource availability* (Section 5.3), and (iii) constraints to *avoid splitting interconnects* (Section 5.4). Finally, the *objective function* to be optimized is presented (Section 5.5).

5.1 Optimization Variables

The following binary and real variables are defined to formulate the floorplanning problem as a MILP.

For each reconfigurable region $R_i \in \mathcal{R}$, we define the following variables:

- $\{x_i, y_i, w_i, h_i\} \in \mathbb{R}_{\geq 0}$: bottom-left coordinates and width-height values of R_i, respectively;
- $\beta_{i,j} \in \{0, 1\}$: a binary variable such that $\beta_{i,j} = 1$ if the j-th clock region is included in R_i, $\beta_{i,j} = 0$ otherwise;

For each pair of reconfigurable regions $(R_i, R_k) \in \mathcal{R} \times \mathcal{R}$, with $R_i \neq R_k$ we define:

- a binary variable $\gamma_{i,k} \in \{0, 1\}$ such that $\gamma_{i,k} = 1$ if and only if the bottom-left corner of R_i is placed on the left of the bottom-left corner of R_k (or they are aligned), i.e., $x_i \leq x_k$.

For each reconfigurable region $R_i \in \mathcal{R}$ and for each FPGA resource type $t \in \{\text{CLB, BRAM, DSP}\}$, we define:

- $\eta_{i,t} \in \mathbb{R}_{\geq 0}$, which specifies the number of resources of type t included in R_i.

Note that, even though most of the optimization variables are defined as real variables, their integrality will be enforced by the following constraints. This choice has been made to limit the number of integer variables, hence aiming at minimizing the branching of the MILP solver. Some of the following constraints make use of a large numerical constant M to represent *infinity*, which is formally defined as $M = f_{\text{CLB}}(W) \cdot H + 1$.

5.2 Structural Constraints

The constraints reported in this section ensure the structural integrity of the RRs.

First, we enforce that each RR must be at least be one column wide and one clock region high (positive area), and that its rightmost x coordinate and its top y coordinate must not exceed the boundaries of the fabric.

Constraint 1. $\forall R_i \in \mathcal{R}$,

$$\begin{gathered} w_i \geq 1, \;\; h_i \geq 1 \\ x_i + w_i \leq W, \;\; y_i + h_i \leq H \end{gathered} \tag{5}$$

Second, another constraint is provided to enforce the *contiguity* of the RRs on the y axis with respect to variables $\beta_{i,j}$. Indeed, if a RR includes the j-th and the $(j + 2)$-th clock region, then it must also include the $(j + 1)$-th one.

Constraint 2. $\forall R_i \in \mathcal{R}, \forall j = 1, \ldots, H - 2$,

$$\beta_{i,j+1} \geq \beta_{i,j} + \beta_{i,j+2} - 1 \tag{6}$$

Proof. If R_i includes the j-th and the $(j + 2)$-th clock regions, then $\beta_{i,j+2} = \beta_{i,j}$. Hence, Equation (6) can be rewritten as $\beta_{i,j+1} \geq 1 + 1 - 1 = 1$, correctly enforcing the desired property. If any of the terms $\beta_{i,j+2}$ and $\beta_{i,j}$ is zero, then the constraint can be rewritten as $\beta_{i,j+1} \geq -1$ or $\beta_{i,j+1} \geq 0$, not enforcing any constraint. □

The height of a RR can then be obtained by enforcing the following simple constraint.

Constraint 3. $\forall R_i \in \mathcal{R}, h_i = \sum_{j=1}^{H} \beta_{i,j}$

Finally, it is of paramount importance to encode a constraint to enforce that RRs must not overlap each other.

Constraint 4. $\forall (R_i, R_k) \in \mathcal{R} \times \mathcal{R}, R_i \neq R_k, \forall j = 1, \ldots, H$,

$$x_k \geq x_i + w_i - (3 - \gamma_{i,k} - \beta_{ij} - \beta_{k,j}) \cdot M \tag{7}$$

Proof. Consider two reconfigurable regions R_i and R_k. Without loss of generality, assume that R_i is the one with the leftmost x coordinate for its bottom-left corner (on any of the two if they have the same coordinate), i.e., $x_i \leq x_k$. In this case, it holds that $\gamma_{i,k} = 1$. Note that, if the two RRs overlap, then they must share a clock region, say the j-th one. Hence, it must also hold that

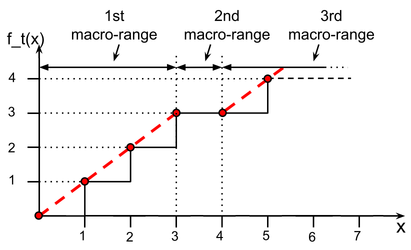


Fig. 2. Example of compression of functions $f_t(t)$ for the purpose of linearization.

$\beta_{i,j} = \beta_{k,j} = 1$ and it follows that the term $(3 - \gamma_{i,k} - \beta_{ij} - \beta_{k,j})$ is zero. Consequently, the constraint degenerates to $x_k \geq x_i + w_i$, correctly enforcing that the two regions do not overlap (the left vertical boundary of R_k is after, or overlapped with, the right vertical boundary of R_i). In all other cases, the term $(3 - \gamma_{i,k} - \beta_{i,j} - \beta_{k,j})$ is positive and hence the right-hand side of the constraint degenerates to a negative number: as a result, no constraint is enforced. □

Since forbidden regions are also modeled as rectangles, an analogous constraint can be enforced to avoid overlapping RRs with forbidden regions.

Furthermore, a simple if-then constraint is provided to enforce the definition of variables $\gamma_{i,k}$.

5.3 Resource Constraints

This section presents the constraints that need to be enforced in order to satisfy the resource requirements of each RR by leveraging the resource finger-printing model presented in Section 3.2. Unfortunately, it is not directly possible to encode the resource requirements as linear constraints due to the following two reasons. First, note that Equation (2) is not linear and hence it cannot be directly encoded in a MILP. Second, even if treating each piece of functions $f_t(x)$ individually, a multiplication by the optimization variable h_i would be anyway required to obtain the total amount of resources available in an area, hence originating quadratic constraints.

These issues are solved by linearizing Equation (2) with the help of a limited set of auxiliary variables and a pre-processing of functions $f_t(x)$. As a first step, for each reconfigurable region R_i, for each clock region $j = 1, \ldots, H$, and for each type of resource t, an auxiliary variable $\mu_{i,j,t}$ is defined as

$$\mu_{i,j,t} = \beta_{i,j} \cdot (f_t(x_i + w_i) - f_t(x_i)). \tag{8}$$

In this way, the number of resources of type t available in R_i can be simply expressed by summing the resource contribution provided by each clock region, i.e., $\eta_{i,t} = \sum_{j=1}^{H} \mu_{i,j,t}$, and the resource requirement can be finally enforced as follows:

Constraint 5. $\forall R_i \in \mathcal{R}, \forall j = 1, \ldots, H, \qquad \eta_{i,t} \geq c_{i,t}$

To encode such a constraint in MILPs, both $f_t(x)$ and Equation (8) must be linearized. To this end, *before starting the optimization*, each function $f_t(x)$ is *compressed* by identifying macro-ranges that can be expressed by a linear function, rather than a sequence of steps—see Figure 2. This strategy allows for a significant reduction of the number of variables and constraints of the MILP formulation.

Let N_{mr} be the total number of macro-ranges identified in $f_t(x)$, and MR_k be the k^{th} macro-range. A pair of auxiliary variables Z_{2k} and Z_{2k-1} is defined for each macro-range MR_k such that, given a coordinate x, it falls within MR_k if and only if the corresponding variables comply to the following two requirements:

REQ1 $Z_{2k} = Z_{2k-1} = 1$; and

REQ2 among all other pairs said variables related *to other macro-ranges*, only one variable of the pair is set = 1.

5.3.1 *Example.* We first explain the compression and linearization of $f_t(x)$ with an intuitive example and then provide the generic mathematical formulation. Consider the case of function $f_t(x)$ illustrated in Figure 2 (continuous line), where three macro-ranges have been identified: $[0, 3)$, $[3, 4)$, and $[4, W)$. Suppose that auxiliary variables Z_1, Z_2 correspond to the first macro-range, Z_3, Z_4 to the second one, and Z_5, Z_6 to the third one. By observing the linear bounds placed above the first and third macro-ranges (vertical dashed lines), it is possible to rewrite $f_t(x)$ as

$$f_t(x) = \begin{cases} x & 0 \le x < 4, \\ 3 & 4 \le x < 5, \\ x - 1 & 5 \le x < W. \end{cases} \tag{9}$$

Hence, function $f_t(x)$ can be expressed by means of the following three constraints:

$$\begin{gathered} f_t(x) \ge x - M(2 - Z_1 - Z_2) \wedge x \ge f_t(x) - M(2 - Z_1 - Z_2) \\ f_t(x) \ge 3 - M(2 - Z_3 - Z_4) \wedge 3 \ge f_t(x) - M(2 - Z_3 - Z_4) \\ f_t(x) \ge (x - 1) - M(2 - Z_5 - Z_6) \wedge (x - 1) \ge f_t(x) - M(2 - Z_5 - Z_6) \end{gathered} \tag{10}$$

The intuition behind these constraints is that, once a coordinate x falls inside a macro-range, only the corresponding pair of auxiliary variables will be set—hence correctly enforcing the corresponding constraint, while all the other constraints will take no effect. For instance, if x falls in the second macro-range, then $Z_3 = Z_4 = 1$, the second constraint is enforced, and the other two are disabled as one between Z_3 and Z_4 is zero, and one between Z_5 and Z_6 is zero. Consequently, it results that $f_t(x) \ge 3 \wedge 3 \ge f_t(x)$, which implies $f_t(x) = 3$.

5.3.2 *Computing the Macro-ranges.* With the above example in place, the linearization of functions $f_t(x)$ can now be formalized. Each macro-range MR_k is defined by a tuple $(\lambda_k, \Lambda_k, \theta_k, \alpha_k)$ where:

- $\lambda_k \in \mathbb{R}$ and $\Lambda_k \in \mathbb{R}$ are *constant* values that represent the slope and the intercept of the linear function that describes MR_k (linear functions are considered in the form $\lambda_k \cdot x + \Lambda_k$);
- $\theta_k \in \mathbb{R}$ and $\alpha_k \in \mathbb{R}$ are *constant* values that represent the left and right vertical bounds of MR_k, respectively, i.e., $\theta_k \le x < \alpha_k$.

Algorithm 1 is proposed to automatically compress functions $f_t(x)$, and is meant to be executed for each resource type $t \in$ [CLB, BRAM, DSP] before setting up the MILP. The algorithm outputs the macro-ranges described by the corresponding tuples).

The algorithm explores the x axis of the FPGA area (from 0 up to the maximum width W) and tries to identify contiguous steps in function $f_t(x)$ that can be merged into the same macro-range. This is accomplished by computing the slope of the line passing between the two points that would delimit a macro-range (line 7), and moving the right-hand-side point of the macro-range until the slope does not change (line 12). Then, the intercept of the line passing between the two points is also computed (line 15), and the tuple of the corresponding macro-range is saved (line 17). From this point on, the algorithm starts looking for the next macro-range. This procedure is repeated until the maximum value on the x axis is reached.

5.3.3 *Deriving the Constraints.* Given a coordinate x and a function $f_t(x)$ with the corresponding set of macro-ranges $\{MR_1, \ldots, MR_{N_{mr}}\}$, requirements REQ1 and REQ2 mentioned above can be enforced with the the following constraint.

ALGORITHM 1: pseudo-code for compressing functions $f_t(x)$

```
$x_1$ = 0;
$x_2$ = 1;
$k$ = 1;

do {
    // Compute the slope
    $\lambda_k = (f_t(x_2) - f_t(x_1))/(x_2 - x_1)$;

    do {
        $x_2 += 1$;
        $\lambda'_k = (f_t(x_2) - f_t(x_1))/(x_2 - x_1)$;
    }while($\lambda_k == \lambda'_k$)

    // Compute the intercept
    $\theta_k = f_t(x_1) - \lambda_k \cdot x_1$;

    <add macro-range> $MR_k := (\lambda_k, \theta_k, x_1, x_2 - 1)$
    $x_1 = x_2 - 1$;
    $k += 1$;
}while($x_2 != W$);
```

Constraint 6. $\forall k = 1, \ldots, N_{mr}$

$$\begin{aligned} M \cdot Z_{2k-1} &\geq x - \theta_k + \epsilon \\ M \cdot Z_{2k} &\geq \alpha_k - x \\ \sum_{k=1}^{N_{mr}} Z_{2k-1} + Z_{2k} &= N_{mr} + 1 \end{aligned} \tag{11}$$

where $\epsilon > 0$ is an arbitrarily small positive number.

Proof. By definition of constants θ_k and α_k, when x falls in the kth macro-range it holds $\theta_k \leq x < \alpha_k$. Hence, both the right-hand-sides of the first two inequalities in Equation (11) are positive and Z_{2k} and Z_{2k-1} are forced to be set to 1. This satisfies REQ1. Note that there are $k-1$ macro-ranges on the left of MR_k and $N_{mr} - k$ macro-ranges on the right of MR_k. For the former ones, i.e., MR_j with $j < k$, it holds that x is always greater than their left boundary (θ_j): hence, variables Z_{2j-1} are forced to 1 by the first inequality in Equation (11). Meanwhile, the corresponding variables Z_{2j} remain unconstrained (the right-hand-side of the second inequality is negative, hence Z_{2j} can take both 1 or 0). Following the same reasoning, the contrary holds for all the macro-ranges on the right of MR_k, i.e., for each macro range M_j with $j > k$ variable Z_{2j} is constrained to 1 while variable Z_{2j-1} remains unconstrained.

Now, note that there are $k-1$ variables forced to 1 for macro-range on the left of MR_k, and $N_{mr} - k$ variables forced to 0 for the ones on the right. As shown at the beginning of the proof, when x falls in the kth the two corresponding variables are forced to 1. Therefore, there are $(k-1) + (N_{mr} - k) + 2 = N_{mr} + 1$ variables forced to 1, while the others are unconstrained. The last inequality in Equation (11) forces such variables that would be unconstrained to be zero, hence matching REQ2. □

Note that the constraints in Equation (10) follow a specific pattern: **(i)** they enforce upper- and lower-bounds for $f_t(x)$ in a macro-range MR_k, and **(ii)** provide a term $M \cdot (2 - Z_{2k-1} - Z_{2k})$ to disable the constraint when the coordinate x does not fall in MR_k. Following this rationale, the constraints in Equation (10) can be generalized for each macro-range MR_k as follows:

$$\begin{aligned} f_t(x) \geq \lambda_k \cdot x + \Lambda_k - M \cdot (2 - Z_{2k-1} - Z_{2k}) \\ \wedge \\ \lambda_k \cdot x + \Lambda_k \geq f_t(x) - M \cdot (2 - Z_{2k-1} - Z_{2k}). \end{aligned} \tag{12}$$

Therefore, every time a term of the form $f_t(x)$ has to be involved in a constraint, it is sufficient to create an auxiliary variable and enforce the two corresponding auxiliary constraints reported in Equation (12) (the auxiliary variable should simply replace $f_t(x)$ in the constraints). Thanks to this result, we are finally ready to express Equation (8) by means of linear constraints.

Constraint 7. $\forall R_i \in \mathcal{R}, \forall j = 1, \ldots, H, \forall t \in$*[CLB, BRAM, DSP]*

$$\begin{aligned} \mu_{i,j,t} &\geq 0 \\ \mu_{i,j,t} &\leq \beta_{i,j} \cdot M \\ \mu_{i,j,t} &\leq f_t(x_i + w_i) - f_t(x_i) \\ \mu_{i,j,t} &\geq f_t(x_i + w_i) - f_t(x_i) - (1 - \beta_{i,j}) \cdot M \end{aligned} \tag{13}$$

Proof. The objective is to show that Equation (13) and Equation (8) are equivalent. There are two cases: $\beta_{i,j} = 1$ and $\beta_{i,j} = 0$. When $\beta_{i,j} = 1$, the first two constraints in Equation (13) become $\mu_{i,j,t} \geq 0$ and $\mu_{i,j,t} \leq M$ and hence have no effect. Furthermore, the last two constraints in Equation (13) become $\mu_{i,j,t} \leq f_t(x_i + w_i) - f_t(x_i)$ and $\mu_{i,j,t} \geq f_t(x_i + w_i) - f_t(x_i)$, which essentially reduce to a single constraint $\mu_{i,j,t} = f_t(x_i + w_i) - f_t(x_i)$. Note that the combination of the four constraints results in Equation (8). When $\beta_{i,j} = 0$, the first two constraints in Equation (13) become $\mu_{i,j,t} \geq 0$ and $\mu_{i,j,t} \leq 0$, which reduce to a single constraint $\mu_{i,j,t} = 0$. The last two constraints in Equation (13) become $\mu_{i,j,t} \leq f_t(x_i + w_i) - f_t(x_i)$ and $\mu_{i,j,t} \geq -M$, which still allow forcing $\mu_{i,j,t}$ to zero. In this case, note that also Equation (8) gives $\mu_{i,j,t} = 0$. Hence the constraints follow. □

5.4 Avoid Splitting Interconnects

As stated in Section 3.1, RR vertical boundaries cannot be placed in the middle of back-to-back interconnect columns. Formally, it must be enforced that, for each reconfigurable region $R_i \in \mathcal{R}$, and for each interconnect $z = 1, \ldots, N_{\text{int}}$, it holds $x_i \neq I_z \wedge x_i + w_i \neq I_z$. Since these two conditions can be enforced in the same manner, it is sufficient to show how to deal with the first one ($x_i \neq I_z$).

An auxiliary binary variable $\sigma_{i,z}$ is first defined such that $\sigma_{i,z} = 1$ if $x_i > I_z$ (the left vertical boundary of R_i is after the central column) and $\sigma_{i,z} = 0$ if $x_i < I_z$ (the left vertical boundary of R_i is before the central column). When $x_i = I_z$, $\sigma_{i,z}$ can be either 0 or 1. This definition can be enforced with the following auxiliary constraints:

$\forall R_i \in \mathcal{R}, \forall z = 1, \ldots, N_{\text{int}}$,

$$x_i \geq I_z - M \cdot (1 - \sigma_{i,z}) \wedge x_i \leq I_z + M \cdot \sigma_{i,z}. \tag{14}$$

Note that the second inequality of the above constraint forces $\sigma_{i,z} = 1$ when $x_i > I_z$, while the first one forces $\sigma_{i,z} = 0$ when $x_i < I_z$.

The main constraint to enforce $x_i \neq I_z$ can be finally presented.

Constraint 8. $\forall R_i \in \mathcal{R}, \forall z = 1, \ldots, N_{int}$,

$$\begin{aligned} x_i - I_z &\leq -\epsilon + M \cdot \sigma_{i,z} \\ x_i - I_z &\geq \epsilon - (1 - \sigma_{i,z}) \cdot M \end{aligned} \tag{15}$$

where $\epsilon > 0$ is an arbitrarily small positive number.

Proof. Remember that M is a numerical constant used to represent infinity. There are three cases. If $x_i > I_z$, then $\sigma_{i,z} = 1$ and the two constraints become $x_i \leq I_z - \epsilon + \infty$ and $x_i \geq I_z + \epsilon$, which are both true. If $x_i < I_z$, then $\sigma_{i,z} = 0$ and the two constraints become $x_i \leq I_z - \epsilon$ and $x_i \geq I_z + \epsilon - \infty$, which are both true. Finally, when $x_i = I_z$ then σ becomes either 0 or 1 (i.e., the auxiliary constraint of Equation (14) does not constrain σ to a single value). Hence, under this condition, both constraints cannot be true at the same time. Overall, the case $x_i = I_z$ is not allowed by the constraint, while all other cases are allowed. □

5.5 Objective Function

The proposed MILP formulation can be integrated with several objective functions. As stated in Section 4.1, in this work we selected to minimize a linear combination of the MIW and WR metrics. Hence, the objective function to be minimized can be expressed as

$$min\,\{a \cdot \Omega/\Omega_{max} + b \cdot \omega/\omega_{max}\}, \tag{16}$$

where Ω and ω respectively represent the total MIW and WR as defined in Equations (3) and (4), while Ω_{max} and ω_{max} are the maximum values for MIW and WR, respectively, which are used for normalization purposes. Finally, $a \in [0, 1]$ and $b \in [0, 1]$ are two tunable real weights to balance the contribution of the two metrics to the objective function. Specific values for these parameters will be used in the experimental evaluations discussed in the next section.

6 EXPERIMENTAL RESULTS

The proposed solution, denoted by FLORA, has been implemented in C++ leveraging the Gurobi solver v.7.0.2 as MILP optimization engine. A visualization tool was also developed to graphically analyze the floorplanning generated by the solver, which served for debugging purposes.

Three experimental sessions have been conducted. In the first one, FLORA has extensively been compared against (i) the [O] algorithm from [10] and (ii) the floorplanner based on a genetic algorithm (GA) proposed in [9], for a state-of-the-art test suite. It bears repeating that the solution proposed in [9] works upon an indirect representation of the FPGA area (see Section 2), and hence the comparison against FLORA is not perfectly fair in the sense that the approach of [9] is conceived for more stringent assumptions (the resource requirements of the RRs has to be fixed at the stage of optimization). Conversely, the [O] algorithm is the most mature solution working on a direct representation of the FPGA area (see Section 2). In the second experimental session, FLORA has been compared against [10] and [9] for benchmark circuits. Finally, in the third experimental session, FLORA has been tested upon a case study and compared against the case of a manual floorplanning: the solution computed by FLORA has also been synthesized on a Xilinx Zynq-7020 and executed.

6.1 State-of-the-art Test Suite

A first comparison was performed by using the same synthetic test suite used by the authors of [10] and [9] Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.[2] This test suite, which is targeted at Virtex-5XC5VLX110T devices, consists of 20 circuits with $\{5, 10, 15, 20, 25\}$ RRs and a percentage of occupancy of resources in the set $\{70\%, 75\%, 80\%, 85\%\}$ that in the following will be referred to as *utilizations*. Since both FLORA and [O] can guarantee the optimality of solutions, the comparison with [O] has two objectives. The first one is to demonstrate that the optimal solutions computed by FLORA (when optimizing singularly for each metric or for the combination of the metrics) are better than the optimal solutions from [O]. This allows demonstrating the benefits of adopting the fine-grained modeling strategy of the FPGA area employed by FLORA. The second objective of the comparison with [O] is to demonstrate that FLORA reaches optimal solutions faster than [O].

Since the GA-based floorplanner in [9] cannot guarantee the optimality of its solutions (in essence, it is a heuristic approach), our objective in comparing FLORA with [9] is to demonstrate that in the same running time FLORA provides better solutions than [9] (even for cases where FLORA did not find the optimal solution in the limited running time). To compare FLORA with the GA-based algorithm in [9] we generated an indirect representation of the FPGA area, called conflict graph, as mandated by the authors in their paper. The conflict graph contains all the feasible placements for the RRs and the corresponding placement conflicts.

The first experimental session was conducted in two different settings. The first setting involves evaluating the quality of solutions obtained by all the three algorithms **(i)** for each individual metric supported by the objective function of Section 5.5, i.e., for the cases $(a = 1, b = 0)$ and $(a = 0, b = 1)$; and **(ii)** for the combination of both metrics $(a = 1, b = 1)$. In this setting, the running time of all algorithms was limited to 60 minutes. This entails limiting the execution time of the MILP solver in both FLORA and [O] to 3600s and applying an input stopping parameter based on elapsed time for the approach of [9]. Please note that the time limit was required to compare against [9], as genetic algorithms cannot guarantee optimal solutions and hence a user-defined stopping condition is required to terminate the algorithm.

In the second setting, FLORA and [O] were compared with respect to the elapsed time required to obtain optimal solutions when the objective function accounts for both MIW and WR metrics ($a = 1$ and $b = 1$ in Equation (16)). This configuration of the objective function was chosen as it was the most challenging optimization problem out of the three mentioned for the first setting. Both algorithms were run without a time limit.

The experiments of the first setting were conducted on a 2.4 GHz Intel Core Duo machine running Linux equipped with 2GB of RAM, while in the second setting a 40-core Intel Xeon E5 machine running at 2.40 GHz with 130 GB of RAM was used. Each test was run five times and the average measurements are reported here. It should be noted that, in order to find good solutions in faster times, the authors of [10] provided to [O] initial sub-optimal solutions obtained from their other algorithm named HO still proposed in [10] (see Section 2). The same can be done with FLORA (by enforcing a lower-bound to the objective function, as it was done for the HO algorithm). However, to better illustrate the net difference between the two approaches, this comparison targeted the case in which no initial solutions are provided.

First setting. Figures 3, 4 and 5 report the results of the experiments conducted in the first mode by grouping the benchmarks in the test-suite by the number of RRs and reporting the results for each utilization value. In these figures, the Δ symbol on the top of each bar implies that both FLORA and [O] reached optimality (within the given time limit), whereas the * symbol is meant to indicate that

[2]At "http://home.deib.polimi.it/santambr/prj/floorplacer/floorplanner.tar.gz", the authors had kindly shared both their work and the test suite they used. The test suite has been adapted to convert slices into CLBs.

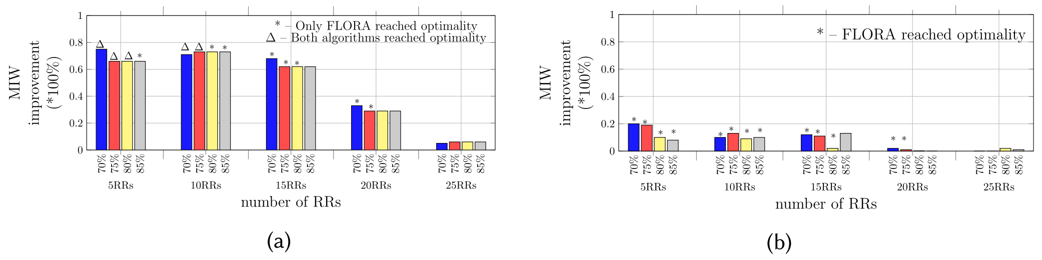


Fig. 3. MIW improvement of (a) FLORA w.r.t [O], and (b) FLORA w.r.t. [9].

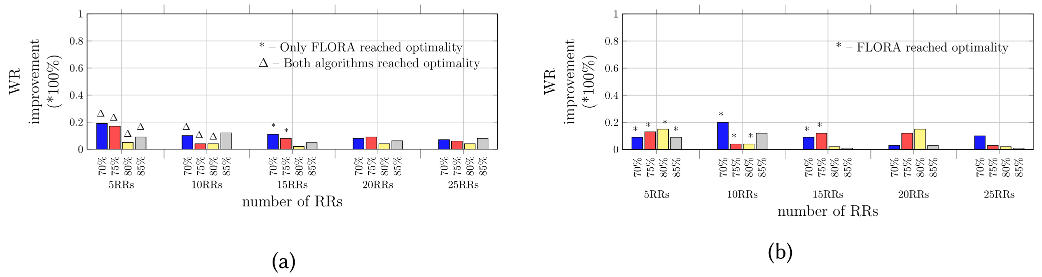


Fig. 4. WR improvement of (a) FLORA w.r.t [O], and (b) FLORA w.r.t. [9].

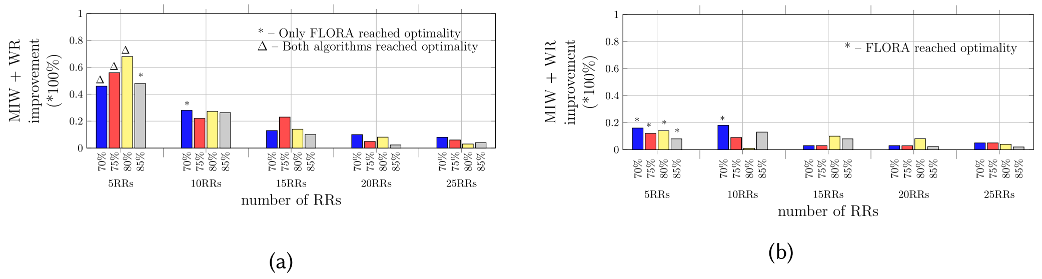


Fig. 5. WR and MIW improvement of (a) FLORA w.r.t [O], and (b) FLORA w.r.t. [9].

only FLORA was able to achieve optimality. The absence of these symbols on the top of the bars means that the comparison was based on the best sub-optimal solutions obtained within the time limit. Hence, for each test case, the comparison between FLORA and [O] was either (i) between two optimal solutions from both algorithms, or (ii) two sub-optimal solutions from both algorithms, or (iii) an optimal solution from FLORA and a sub-optimal solution from [O]. It is important to point out that *there was no case in which only [O] was able to achieve optimality.*

Figures 3(a) and 3(b) report the average inter-region wire-length improvement of FLORA against [O] and [9], respectively. As it can be seen from Figure 3(a), FLORA significantly improved the MIW metric with respect to [O], while producing optimal solutions for most of the test cases (70% of the solutions by FLORA were optimal, while only the 25% of [O]'s solutions were optimal). From Figure 3(a), it can also be noted that, in the cases with less than 20 RRs, the improvement of the MIW metric provided by FLORA is considerably high (up to 75.5%). By exploiting the more fine-grained modeling of the FPGA, FLORA was able to explore solutions that cannot be represented in the model adopted by [O], and was consequently able to produce placements with significantly lower MIW values compared to [O]. A similar trend can be observed in Figure 3(b) against the

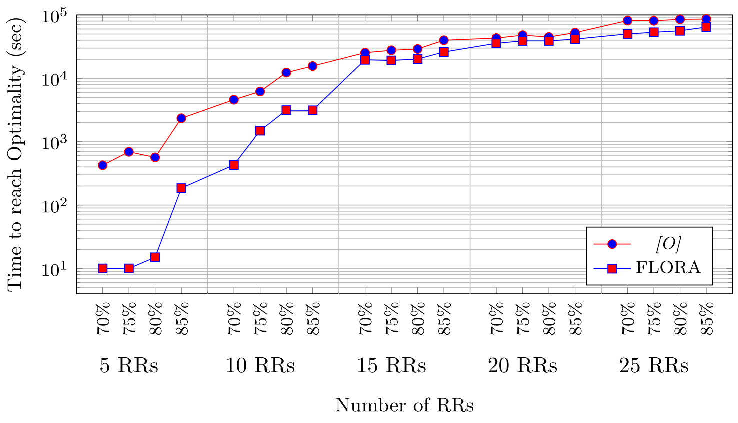


Fig. 6. Run-time until optimality for FLORA and [O].

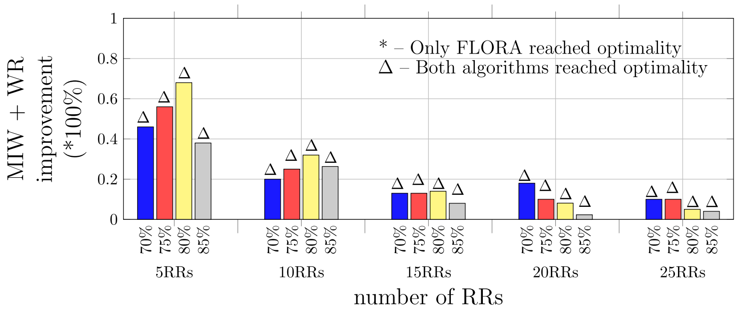


Fig. 7. WR and MIW improvement of FLORA w.r.t. [O] by comparing optimal solutions.

approach from [9], although the improvement is not as high as the one against [O] (up to 20%), but still definitively not negligible. As one can expect, even when utilizing a fine-grained model, the improvement of the MIW metric diminishes for larger numbers of RRs against both algorithms: this is because larger numbers of RRs lead to more "compact" placements with no much room to move the RRs around

Figure 4 reports the improvements in terms of the WR metric. Also in this case, FLORA improves upon both [O] and [9] for all the cases, and it was possible to obtain performance improvements up to 20% and 22% compared to [O] and [9], respectively.

Finally both the evaluation metrics were combined in the objective function: the results of this experiment are reported in Figure 5. It should be noted that this is a difficult optimization problem. Indeed, as it can be seen in Figure 5(a), neither [O] nor FLORA managed to provide optimal solutions for test cases beyond 10 RRs and 70% of utilization within the time limit. However, the sub-optimal solutions produced by FLORA resulted in better placements in all the test cases against [O] and [9]. With FLORA, it was also possible to compute solutions with up to 7% and 5% improvements against [O] and [9], respectively, under the most difficult cases (25 RRs).

Second setting. The results related to the second experimental setting (i.e., where both FLORA and [O] freely run until they both reach optimality) are reported in Figures 6 and 7. Figure 6 reports the total run-time required by the algorithms to compute their corresponding optimal solutions. As it can be noted from the figure, FLORA shows improvements of more than one order of magnitude for the cases with 5 and 10 RRs. Furthermore, despite being based on a more fine-grained (i.e., more

Table 1. Experimental Results for the Benchmark Circuits

		Wire-length			Execution time (sec)		
Circuit	#RRs	[O]	[9]	FLORA	[O]	[9]	FLORA
apte	9	6171	4978	4296	1789	3600	1641
xerox	10	20012	12009	11546	3600	3600	3600
hp	11	11579	5912	5887	3600	3600	1428
ami33	33	128123	98123	94312	3600	3600	3600

complex) modeling of the FPGA area, the time required to produce optimal solutions are still lower than those of [O] even for the cases with 15 or more RRs. For instance, in the most challenging cases (20 and 25 RRs), FLORA reached optimality up to 5 hours before [O]. Figure 7 reports the improvements achieved by FLORA by comparing the value of the objective function computed for the optimal solutions produced by the two algorithms. As it can be seen, FLORA significantly outperforms [O] in all the cases. Note that this is mainly because the design space that can be expressed by FLORA is larger than the one of [O] as FLORA adopts a more fine-grained modeling of the FPGA area.

6.2 Benchmark Circuits

A second experimental session was performed to compare FLORA against both [O] and the work in [9] on four benchmark circuits derived from the MCNC benchmark [2]. This standard ASIC benchmark was adapted for FPGA by following an approach similar to the one adopted in [7]. A time limit of 60 minutes was used also in this experimental session. The objective function was configured to deal with the MIW metric only ($a = 1$ and $b = 0$).

The experimental results are reported in Table 1. Out of the four benchmark circuits, FLORA was able to provide optimal solutions for two circuits (apte and hp), while [O] was able to produce the optimal solution only in one case (apte). As it can be seen from the table, FLORA had largely improved the MIW metric against [O] (by up to 95.8% for the hp circuit), while still achieving non-negligible improvements with respect to [9] (e.g., 15.8% for apte). Despite the improvements with respect to [9] are limited, it is worth observing that FLORA performs better or at least as [9] without working on an indirect representation of the FPGA area (see Section 2 for the limitations introduced by indirect representations).

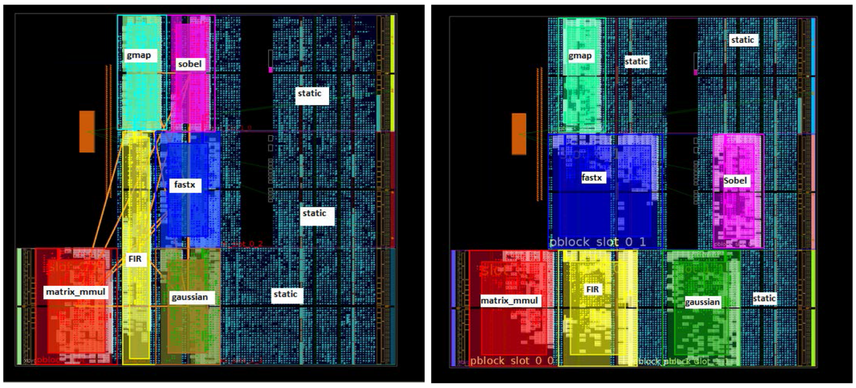
6.3 Case Study

The third experimental session targeted a case study focused on a partially-reconfigurable video processing design to be implemented on a Xilinx Zynq-7020 SoC (Pynq board). The Vivado tool by Xilinx was used. The design consists of 6 RRs, each hosting two alternative versions of a hardware module; specifically, five RRs host image filters (fastx, fir, gaussian, gmap, and sobel) and one hosts a matrix multiplier. All the hardware modules have been taken from the Xilinx HLS library. Each image filter is available in two versions: one designed for 3x3 kernels and the other for 5x5 kernels. The matrix multiplier is also available in two versions: one designed for a 4x4 matrix multiplication and another for an 8x8 multiplication. Each of such versions can be reconfigured in the corresponding RR. The resource requirement of the reconfigurable modules is summarized in Table 2. The ”hosting region” column of Table 2 denotes the id of the RR which hosts the corrosponding accelerator in the same row. The design is completed by static modules (i.e., not subject to PR) that are automatically placed by the Vivado tool (no floorplanning is required).

Here, we compare the floorplan generated by FLORA against the best floorplan we were able to *manually* find in terms of (i) time needed to obtain the floorplan, and (ii) sizes of the bitstreams for

Table 2. Resource Requirement of the Reconfigurable Modules in the Case Study

Filter type	Kernel size	LUT	FF	BRAM	DSP	Hosting Region
fastx	3×3	5004	4331	14	3	1
	5×5	6128	6371	19	8	1
fir	3×3	1629	2400	2	39	2
	5×5	2025	3476	4	48	2
gaussian	3×3	1889	2104	7	12	3
	5×5	2414	3096	11	16	3
gmap	3×3	1710	2204	6	3	4
	5×5	2014	3176	10	8	4
matrix_mul	4×4	3546	6081	14	12	5
	8×8	4296	8945	24	20	5
sobel	3×3	1465	1923	4	3	6
	5×5	1845	2212	7	6	6



(a) (b)

Fig. 8. Floorplans generated (a) by FLORA and (b) manually.

each RR. FLORA was configured to optimize the WR metric only (i.e., $a = 0$ and $b = 1$) as the case study does not involve direct connections between the RRs (the hardware modules communicate with a shared-memory paradigm).

The two floorplans are illustrated in Figure 8, which has been obtained by taking screenshots from the Xilinx Vivado tool. The entire design utilizes 94% of the total FPGA resources of the Zynq-7020: hence, generating a valid floorplan that satisfies all the resource requirements of the RRs without violating the PR constraints is particularly challenging. To the best of our abilities, the manual floorplanning of this design took approximately one hour and a half to obtain a valid solution. Conversely, FLORA computed the optimal solution in only 10 seconds. Table 3 summarizes the sizes of the partial bitstream of the modules used in the case study. Note that the floorplan produced by FLORA led to partial bitstreams up to 26% smaller (see the 3×3 FIR filter module) than those corresponding to the manual floorplan, while both designs (the manual and the one by FLORA) are able to run at a clock speed of 125 MHz on the Pynq board (the input clock was used without modification in the RRs). For instance, considering a reconfiguration interface with clock speed of 100 MHz (a total reconfiguration bandwidth of 400 MB/s), in the manual floorplan

Table 3. Partial Bitstream Sizes for Manual and FLORA Generated Floorplans Along with Clock Speed of Modules in the Case Study

Filter type	Kernel size	Manual (KB)	FLORA (KB)
fastx	3×3	806	737
	5×5	1100	1000
fir	3×3	749	554
	5×5	956	745
gaussian	3×3	1000	898
	5×5	1200	1000
gmap	3×3	621	515
	5×5	788	699
matrix_mul	4×4	1100	925
	8×8	1300	1100
sobel	3×3	545	490
	5×5	810	728

it takes 1.87 ms to reconfigure the 3×3 FIR filter (the partial bitstream size is 749 kB), while it takes 1.38 ms to reconfigure the same filter when the floorplan is generated by FLORA (the partial bitstream size is 554 KB). Note that this reduction in reconfiguration time is a direct consequence of minimizing the number of wasted resources in the RRs, which in turn led to smaller bitstreams.

7 CONCLUSIONS

This paper presented FLORA, a solution to automatically compute the floor planning of a set of reconfigurable regions. Our approach leveraged mixed-integer linear programming and a fine-grained direct modeling of the layout of the FPGA area, which also allows taking into account several realistic technological constraints mandated by commercial FPGA synthesis tools. Experimental results demonstrated that FLORA allows improving the maximum inter-region wire-length (up to 103%), the amount of wasted resources, and the solving time (even by one order of magnitude) with respect to other state-of-the-art approaches. For typical PR designs, which consist of an average of 5 reconfigurable regions, the experimental evaluation revealed a 70% average improvement in terms of inter-region wire-length, while exhibiting solving times that are one order of magnitude lower than those of state-of-the-art methods.

Future work will attempt at integrating FLORA in the Vivado suite to implement a completely automatic design flow under partial reconfiguration.

ACKNOWLEDGMENT

The Authors would like to thank Marco Pagani for providing the hardware modules in the video processing engine case study and for his extended support during the implementation.

REFERENCES

[1] P. Banerjee, M. Sangtani, and S. Sur-Kolay. 2011. Floorplanning for partially reconfigurable FPGAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 30, 1 (Jan 2011), 8–17.

[2] MCNC benchmark Netlists for Floorplanning and Placement. 2012. https://s2.smu.edu/~manikas/Benchmarks/MCNC_Benchmark_Netlists.html. [Online; accessed 27-March-2019].

[3] A. Biondi, A. Balsini, M. Pagani, E. Rossi, M. Marinoni, and G. Buttazzo. 2016. A framework for supporting real-time applications on dynamic reconfigurable FPGAs. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*.

[4] A. Biondi and G. Buttazzo. 2017. Timing-aware FPGA partitioning for real-time applications under dynamic partial reconfiguration. In *Proceedings of the 11th NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*.

[5] C. Bolchini, A. Miele, and C. Sandionigi. 2011. Automated resource-aware floorplanning of reconfigurable areas in partially-reconfigurable FPGA systems. In *2011 21st Int. Conf. on Field Programmable Logic and Applications*. DOI: https://doi.org/10.1109/FPL.2011.104

[6] L. Cheng and M. D. F. Wong. 2006. Floorplan design for multimillion gate FPGAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25, 12 (Dec 2006), 2795–2805. DOI: https://doi.org/10.1109/TCAD.2006.882481

[7] Yan Feng and D. P. Mehta. 2006. Heterogeneous floorplanning for FPGAs. In *19th Int. Conf. on VLSI Design Held Jointly with 5th Int. Conf. on Embedded Systems Design (VLSID'06)*. DOI: https://doi.org/10.1109/VLSID.2006.96

[8] Tuan D.A. Nguyen and Akash Kumar. 2016. PRFloor: An automatic floorplanner for partially reconfigurable FPGA systems. In *Proceedings of the 2016 ACM/SIGDA Int. Symposium on Field-Programmable Gate Arrays (FPGA'16)*.

[9] M. Rabozzi, G. C. Durelli, A. Miele, J. Lillis, and M. D. Santambrogio. 2017. Floorplanning automation for partial-reconfigurable FPGAs via feasible placements generation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25, 1 (Jan 2017), 151–164. DOI: https://doi.org/10.1109/TVLSI.2016.2562361

[10] M. Rabozzi, J. Lillis, and M. D. Santambrogio. 2014. Floorplanning for partially-reconfigurable FPGA systems via mixed-integer linear programming. In *2014 IEEE 22nd Annual Int. Symposium on Field-Programmable Custom Computing Machines*. DOI: https://doi.org/10.1109/FCCM.2014.61

[11] M. Rabozzi, A. Miele, and M. D. Santambrogio. 2015. Floorplanning for partially-reconfigurable FPGAs via feasible placements detection. In *2015 IEEE 23rd Annual Int. Symposium on Field-Programmable Custom Computing Machines*. DOI: https://doi.org/10.1109/FCCM.2015.16

[12] Love Singhal and Eli Bozorgzadeh. 2007. SPECIAL SECTION ON FIELD PROGRAMMABLE LOGIC AND APPLICATIONS - Multi-layer floorplanning for reconfigurable designs. *Computers and Digital Techniques, IET* 1 (08 2007), 276–294. DOI: https://doi.org/10.1049/iet-cdt:20070012

[13] ug909-vivado-partial-reconfiguration user guide. 2018. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_1/ug909-vivado-partial-reconfiguration.pdf. [Online; accessed 27-March-2019].

[14] Kizheppatt Vipin and Suhaib A. Fahmy. 2012. Architecture-aware reconfiguration-centric floorplanning for partial reconfiguration. In *Proceedings of the 8th Int. Conf. on Reconfigurable Computing: Architectures, Tools and Applications (ARC'12)*.

[15] Jun Yuan, Sheqin Dong, Xianlong Hong, and Yuliang Wu. 2005. LFF algorithm for heterogeneous FPGA floorplanning. In *Proceedings of the ASP-DAC 2005. Asia and South Pacific Design Automation Conference, 2005*. DOI: https://doi.org/10.1109/ASPDAC.2005.1466538

[16] Ping-Hung Yuh, Chia-Lin Yang, and Yao-Wen Chang. 2004. Temporal floorplanning using the t-tree formulation. In *Proceedings of the 2004 IEEE/ACM Int. Conf. on Computer-aided Design (ICCAD'04)*.

Received April 2019; revised June 2019; accepted July 2019