

Floorplanning Automation for Partial-Reconfigurable FPGAs via Feasible Placements Generation

Marco Rabozzi, *Student Member, IEEE*, Gianluca C. Durelli, *Student Member, IEEE*, Antonio Miele, *Member, IEEE*, John Lillis, *Member, IEEE*, and Marco D. Santambrogio, *Senior Member, IEEE*

Abstract—When dealing with Partially-Reconfigurable designs on Field Programmable Gate Arrays (FPGAs), floorplanning represents a critical step that highly impacts system’s performance and reconfiguration overhead. However, current vendor design tools still require the floorplan to be manually defined by the designer. Within this work we provide a novel floorplanning automation framework, integrated in the Xilinx tool-chain, which is based on an explicit enumeration of the possible placements of each region. Moreover, we propose a genetic algorithm, enhanced with a local search strategy, to automate the floorplanning activity on the defined direct problem representation. The proposed approach has been experimentally evaluated with a synthetic benchmark suite and real case studies. We compared the designed solution against both state-of-the-art algorithms and alternative engines based on the same direct problem representation. Experimental results demonstrated the effectiveness of the proposed direct problem representation and the superiority of the defined genetic algorithm engine with respect to the other approaches in terms of exploration time and identified solution.

Index Terms—Field Programmable Gate Arrays, Floorplanning, Partial Reconfiguration, Mixed Integer Linear Programming, Genetic Algorithm, Local Search, Simulated Annealing.

I. INTRODUCTION

Field Programmable Gate Array (FPGA) devices are nowadays widely employed in commercial and industrial appliances in many scenarios (e.g. telecommunication, automotive, high performance computing, video and image processing), due to their reduced costs, good computational power, and high flexibility since they can be reconfigured in order to change their functionality. Moreover, Partial Reconfiguration (PR) [1] has received a considerable attention in the recent years since it even more enhances such flexibility, by enabling the possibility to dynamically change only part of the modules at runtime while the rest of the system keeps working. Indeed, PR offers new opportunities such as the possibility to execute at different times more functionalities than the ones physically placeable on the device or the possibility to update or vary their implementations. In order to enable PR two necessary conditions must hold: 1) the Field Programmable Gate Array

(FPGA) device has to physically support the change of only a part of the configuration at runtime, and 2) the companion design tools have to support the implementation of such reconfigurable systems. In this scenario, Xilinx [2] is the vendor presenting the most mature solution.

The role of floorplanning [3] in PR-based system design is even more prominent than to the standard FPGA design flow. In fact, while in the latter, this activity is mainly of interest for expert designers aiming at achieving advanced performance optimization, in the former the implementation of a partially reconfigurable system forces to define the specific regions on the device fabric that will host the interchangeable functionalities. Therefore, floorplanning directly affects the feasibility and the performance of the final solution. However, it is a quite complex activity since the area constraints for the reconfigurable regions have to meet specific placement requirements (reported in [1]), while covering a minimum amount of configurable resources that are needed by the modules reconfigured over time in each of the regions. Nevertheless the internal architecture of FPGAs is becoming more and more advanced, exacerbating the floorplanning complexity. In fact, the homogeneous grid of Configurable Logic Blocks (CLBs) is alternated, most of the time in an irregular way, with columns of dedicated elements such as Block RAMs (BRAMs) and Digital Signal Processors (DSPs).

Commercial tool-chains still support floorplanning through visual instruments, such as Xilinx Vivado [2] (which integrates the previous PlanAhead tool). However, the designer still has to manually define the shape and the position of the reconfigurable regions, since the tool provides a limited automation on the regions definition that generally leads to unfeasible solutions. Nevertheless, also other design flows for Xilinx FPGA devices proposed by the academia (e.g., [4], [5]) suffer from the same lack. On the other hand, several academic solutions have been presented in literature to automate floorplanning ([3], [6]–[12]). However, only few ones ([6]–[8]) take into account the requirements for PR, and, at the same time, accurately consider an arbitrary distribution of heterogeneous resources within the device. Indeed, most of the algorithms consider only one of the two aspects, i.e. the PR requirements (e.g. [9], [10]) or the resource distribution (e.g. [3], [11]–[13]). Finally, as it will be shown in this paper, such comprehensive approaches generate suboptimal solutions. Another relevant consideration that can be drawn on most of such automation solutions is the fact that they are actually unconnected from

M. Rabozzi, G.C. Durelli, A. Miele and M.D. Santambrogio are with Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milano, Italy e-mail: marco.rabozzi@polimi.it, gianluca.durelli@polimi.it, antonio.miele@polimi.it, marco.santambrogio@polimi.it. J. Lillis is with University of Illinois at Chicago, Chicago, USA e-mail: lillis@uic.edu.

Manuscript received April 19, 2005; revised September 17, 2014.

the real design flow; in fact, only few of these engines ([4], [5], [14], [15]) are tested on real circuits and synthesize their final outcome on a real board to check for feasibility.

In this paper we present a novel floorplanning automation framework fully integrated with the Xilinx design flow. The framework exploits a direct representation of the problem based on the enumeration of the feasible placements that is able to abstract the computational complexity of floorplanning exploration while taking into account all the relevant constraints for PR on recent devices and metrics such as area consumption and aspect ratio. We show that, differently from the classical problem for VLSI design, enumerating a suitable subset of the feasible placements for each reconfigurable region is a viable approach and can be also efficiently automated by means of classical optimization algorithms. In a previous work [16] we proposed a preliminary version of the framework where the design space exploration was automated by means of a Mixed-Integer Linear Programming (MILP) formulation. We here propose a more complete and mature framework featuring a new automation engine providing higher performance. In conclusion, we summarize our contributions as follow:

- We accurately model all the constraints in the current PR guidelines [1].
- We propose a direct formulation of the floorplanning problem based on a conflict graph of the feasible regions placements described in terms of the actual coordinates on the fabric grid.
- We propose a genetic algorithm extended with a local search strategy exploiting the defined problem representation. The algorithm is able to speed up the identification of near-optimal solutions in a limited elaboration time.
- Finally, we experimentally show the effectiveness of the proposed approach by comparing against various state-of-the-art solutions and alternative engines exploiting the same problem representation on both synthetic benchmarks and real case studies.

The remainder of the paper is organized as follows. Section II discusses the related work in the area, while Section III presents a formal description of the problem. Then, Section IV shows the proposed design flow, whose details are discussed in Sections V and VI in which the feasible placements generation process and the floorplanning automation algorithm are presented respectively. Finally, Section VII evaluates our approach on different problem instances, and Section VIII draws the conclusions. In addition, we report in Appendix A a revisited presentation of the MILP formulation [16] that has been used as a baseline for the experimental evaluations.

II. RELATED WORK

Several floorplanners for FPGAs have been proposed in literature; however, most of them produce solutions that are either not compliant with PR requirements and guidelines (e.g. [11]–[13]), or only focus on a simplified device model, not capable of representing modern FPGAs lacking a uniform distribution of heterogeneous resources (e.g. [9], [10]).

One of the first algorithms that considers the heterogeneity of FPGA resources has been presented in [3]. The algorithm

exploits simulated annealing over a slicing-tree representation, and, subsequently, performs a compaction step to recover from unfeasible solutions and to improve the shapes of the modules. However, the resulting floorplan unlikely produces shapes that meet the PR requirements. Furthermore, the approach assumes the FPGA to have a homogeneous resource distribution, i.e., BRAM and DSP columns are homogeneously spaced within the device fabric. Based on this assumption, the algorithm divides the fabric grid in a set of homogeneous blocks having the same size and containing the same amount of resources for each resource type (DSP, BRAM and CLB). However, this organization, which characterizes obsolete device families (such as Xilinx Virtex-II and Spartan 3), does not hold for the recent devices (e.g. Xilinx Virtex 6). Interesting aspects of such formulation are the Irreducible Realization List (IRL) and the dominance relation, that have been successfully borrowed in our problem representation as described in Section V.

A similar approach considering a heterogeneous FPGA device has been proposed in [13]; it consists in 1) a simulated annealing algorithm exploring a sequence-pair representation of the solution, and 2) a subsequent refinement of such solution by means of a Min-Cost Max-Flow formulation which alters the rectangular shapes of the reconfigurable regions. Due to this second phase, the approach in general does not satisfy PR requirements.

Another class of approaches ([9], [10]) introduces the time domain in the problem by handling the definition of reconfiguration operations together with the design of the floorplan. In [10] only logic blocks are taken into account while ignoring other types of resources available in the FPGA device. The work proposed in [9] considers both the partitioning of modules into reconfigurable regions and their floorplanning. During the partitioning phase, the algorithm assigns each of the modules to a reconfigurable region to minimize the wastage of resources over time. After partitioning, the resource requirements of the regions are known and the algorithm computes a floorplan by means of simulated annealing using moves that preserve the PR constraints. Even though the approach considers heterogeneous resources, similarly to [3] it assumes their regular and uniform distribution.

Differently from [9], other approaches ([11], [12]), called *multi-layer* floorplanners, analyze together the various circuit configurations the system assumes in different instants of time. Their aim is to identify a floorplan such that the common modules used in all the configurations are placed at the same position in all the circuit configurations. Such modules will represent the static area of the device, while the rest of the device is reconfigured as a whole. As a consequence, the reconfigurable part does not follow the Xilinx PR flow. Nevertheless, in [12] the device is assumed to have a homogeneous resource distribution as in [3].

A last class of floorplanners ([6], [7], [8]) considers both the PR constraints and an accurate description of the heterogeneous resource distribution. The work proposed in [6] stems from Parquet [17], the state-of-the-art fixed-outline floorplanner for VLSI design, and presents a non-trivial adaptation of the methodology to deal with partially reconfigurable FPGAs. The algorithm uses simulated annealing to perturb

a floorplan representation that consists of a sequence pair augmented with a vector characterizing the aspect-ratio of the modules. Moreover, to increase the probability to detect feasible floorplans, it implements smart moves to recover from solutions in which the resource requirements are not satisfied.

The approach devised in [7] characterizes the FPGA device in terms of minimal reconfigurable units [1] called tiles. Each tile spans multiple configurable frames on the horizontal direction and contains a specific type and number of resources. Thus, the resource requirements of the reconfigurable regions are translated in terms of tile requirements and a technique called Columnar Kernel Tessellation is applied to search for floorplans that minimize the overall estimated bitstream size. A post processing step moves the obtained areas on the vertical direction trying to locally improve the wire length without affecting the occupation of resources.

Even though [6] and [7] give better results than [9] in terms of wire length and area occupancy respectively, [8] shows that the quality of their solutions can still be improved by means of analytic methods at the cost of a longer execution time. Specifically, [8] proposes two algorithms both based on a compact MILP formulation. The first algorithm is meant to locally improve the quality of an initial feasible solution with a relatively small computational effort. Instead, the second algorithm is able, in principle, to explore the full solution space and to find provably optimal solutions. Unfortunately both the algorithms require, to some extent, an initial feasible floorplan to achieve good final solutions. In our previous publication [16] we have demonstrated that solutions achieved by the approaches proposed in [8] (and consequently in [7] and [6]) can be further optimized without additional time penalties.

A final aspect to be considered is the experimental validation of the proposed solutions. Actually, only in few approaches ([4], [5], [14], [15]) the produced results are synthesized on the target device to check their feasibility and the achieved performance in terms of maximum clock frequency. In particular, in [14] an in-depth analysis of the effects of modules aspect ratio on the maximum achievable clock frequency is performed, while no automation strategies are presented. In [15] a similar analysis is performed by concluding that squared aspect ratios are preferable, and a very simple semi-automated floorplanner for pipeline designs based on a single chain of components is proposed. In [4], the floorplanning problem is tackled from a different perspective: the system is first synthesized without any constraint, and, then, an automated engine tries to identify a suitable set of placement constraints around the area used for placing and routing each module; unfortunately the approach is tested with a single reconfigurable region and it is unlikely to work with a larger number of regions. Finally, in [5] a simulated annealing is directly integrated with the synthesis tool to implement each explored solution; even though such a strategy presents a huge cost in terms of elaboration time.

Table I recaps the characteristics of the existing approaches showing the supported features. It is worth noting that the most efficient approach has been proposed in [16]; in fact, it outperforms (possibly in an indirect way) most of the relevant previous solutions supporting PR (i.e. [6]–[9], [13]). However,

TABLE I
COMPARATIVE ANALYSIS OF PAST APPROACHES

Approach	FPGA Model*	Reconfig. aware	PR support	Experimental comparison**	Exp. verified
Wong [3]	HO			[17]	
Feng [13]	HE				
Yuh [10]	CLB	✓			
Montone [9]	HO	✓	✓		
Singhal [11]	CLB	✓			
Banerjee [12]	HO	✓			
Bolchini [6]	HE	✓	✓	[9], [13]	
Vipin [7]	HE	✓	✓	[9]	
Rabozzi [8]	HE	✓	✓	[6], [7]	
Lamprecht [14]	HE				✓
Neely [15]	CLB	✓	✓		✓
Beckhoff [4]	HE	✓	✓		✓
Yousuf [5]	HE	✓	✓		✓
Rabozzi [16]	HE	✓	✓	[6], [8]	
PA	HE	✓	✓	[16]	✓

(*) Device models with a homogeneous resource distribution (HO), heterogeneous one (HE) and considering only configurable logic blocks (CLB)

(**) The cell lists the approaches that have been tested and outperformed by the one of the current line

the weakness it presents is the lack of an experimental validation of the achieved solutions while not all the current PR constraints are taken into account. In this paper, we aim at proposing a novel floorplanning automation framework that, starting from the preliminary idea presented in [16], supports the peculiarities of modern FPGA devices and PR design flow, and features an even more efficient automation engine in terms of quality of the achieved solutions and elaboration time. Moreover, we also present an experimental validation of the approach by implementing real designs on a FPGA device.

III. FLOORPLANNING PROBLEM DESCRIPTION

This section provides some relevant background on the floorplanning problem, in particular focusing on the Xilinx FPGA devices and the design rules of the related PR flow.

As shown in Figure 1a, the reconfigurable fabric of an FPGA device is organized in a set of columns of resources of various types, that is $T = \{CLB, BRAM, DSP\}$. The grid is also divided in quadrants, called *clock regions* according to the structure of the clock tree and the organization of the configuration memory. Based on the memory organization, the basic reconfiguration portion of the device grid, that we call *tile*, spans one clock region height and one resource width. Each tile contains a single type of resource depending on the position of the tile, and the amount of units depends on the type of resource. Thus, as in [7], we consider a more abstract model of the FPGA organization in terms of a grid of tiles. Finally, we also define a coordinate system on the grid of tiles, starting from the bottom-left corner. We denote with W and H the maximum values on the X and Y axis respectively.

According to the PR design guidelines, as shown in Figure 1b, the reconfigurable system is specified in terms of a structural description of interconnected N top components called *reconfigurable regions*¹. Each region implements a partially-reconfigurable unit in which it will be possible to load in a mutually-exclusive fashion a set of *modules* implementing different functionalities. Thus, the reconfigurable

¹When clear from the context, we also refer to them simply as *regions*.

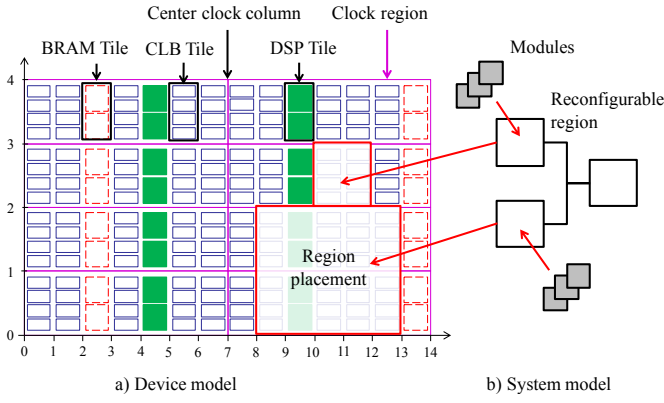


Fig. 1. Problem representation in terms of a) the reconfigurable FPGA device and b) the top level structural description of the system.

region n presents resource requirements that depends on the hosted modules; for each resource type t we denote the required amount as $r_{n,t}$. Moreover, the region is connected with the others and with the static part of the design (another component or set of components not featuring reconfiguration capabilities) by means of a set of interconnection buses, each one characterized by a width b in terms of number of wires. Do note that at floorplanning stage, positioning of the connections of the wires among the region boundaries is not handled; therefore, a center-to-center interconnection model is here adopted and the overall wire length is estimated using the classical half-perimeter wire length (HPWL) formula [12].

The goal of the floorplanning is to define a *placement* for each of the reconfigurable regions, in terms of rectangular shape and position on the FPGA resource grid. To this purpose, on the basis of the defined FPGA model, we denote with P the set of all possible placements that may be defined for the floorplanning of a single reconfigurable region:

$$P = \{(x, y, w, h) \mid x, y, w, h \in \mathbb{N}, x + w \leq W, y + h \leq H\} \quad (1)$$

where x and y represent the coordinates of the bottom-left corner of the placement, while w and h define its width and height respectively. Thus, the specific placement p can be characterized in terms of the available resource capacity, denoted as $c_{p,t}$ (for each resource type t), depending on the specific position and shape. It is worth noting that in some devices (e.g. the Zynq device) specific placements are forbidden since they overlap with hard processors, static logic or I/O blocks. We represent such placements with the subset $S \subset P$, that will be discarded during the floorplanning exploration. For a formal description of the floorplanning requirements it is convenient to define a relation \perp such that for $p_1, p_2 \in P$: $p_1 \perp p_2$ if and only if the two placements overlap on at least a tile. The non-overlapping relation $\not\perp$ is simply defined as the complement of \perp : $\not\perp = P \times P \setminus \perp$.

To be feasible, a floorplan must assign a placement p_n for each region n and satisfy a set of PR requirements:

REQ1: each assigned placement must contain at least the required resources for the corresponding region:

$$\forall n \in N, t \in T : c_{p_n,t} \geq r_{n,t} \quad (2)$$

REQ2: each assigned placement must not be forbidden:

$$\forall n \in N : p_n \notin S \quad (3)$$

REQ3: the left and right boundaries of a placement p_n must be aligned to specific coordinates that prevent splitting of interconnect resources [1] (VL and VR enumerate valid left and right coordinates, respectively):

$$\forall p_n = (x, y, w, h) \mid n \in N : x \in VL \wedge x + w \in VR \quad (4)$$

REQ4: CLB resources at both sides of the center clock column must lie in the static part of the design, an assigned placement can cross the center column but such resources are not available for the corresponding region:

$$\forall t \in T : c_{(x_{clk}-1,0,2,H),t} = 0 \quad (5)$$

REQ5: placements assigned to two different regions cannot overlap:

$$\forall p_{n1}, p_{n2} \mid n1, n2 \in N \wedge n1 \neq n2 : p_{n1} \not\perp p_{n2} \quad (6)$$

This list of constraints can be partitioned in two groups: REQ1-REQ4 are specifically related to the placement p_n for a single region n , while REQ5 rules the relative positions between different regions. Moreover, the first set can be summarized in a single definition by introducing a new set P_n , which represents all the feasible placements on the device for a reconfigurable region n .

In conclusion, the floorplanning problem can be stated as follows: *Given the sets P_n of feasible placements, a floorplan is a function f that assigns for each region $n \in N$ a placement $p \in P_n$ such that there is no overlapping among the placements.* More formally:

$$f : n \in N \rightarrow p \in P_n \quad (7)$$

$$f(n_1) \not\perp f(n_2) \quad \forall n_1, n_2 \in N : n_1 \neq n_2$$

IV. PROPOSED FLOORPLANNING FRAMEWORK

The structure of the proposed floorplanning automation framework and its integration in the Xilinx design flow is depicted in Figure 2. The design flow implemented in Xilinx Vivado consists in three main automated steps:

- 1) *Synthesis*, which takes the input Hardware Description Language (HDL) structural specification of the system and translates it in an intermediate netlist,
- 2) *Implementation*, which performs the place and route of the netlist on the selected FPGA device, and,
- 3) *Bitstream Generation*, which generates the final partial and complete configuration files.

Moreover, after each phase, a set of manual steps can be performed to define specific aspects and set parameters (such as the selection of the I/O pins or the floorplanning of the modules), while the obtained circuit can be analyzed by means of utility tools (for instance to estimate the power consumption or the timing of the netlist).

In this scenario, the PR design flow is an enhancement of the standard flow able to handle the fact that several modules can be implemented in the same reconfigurable region. During the three phases, partial specifications, related to the modules and

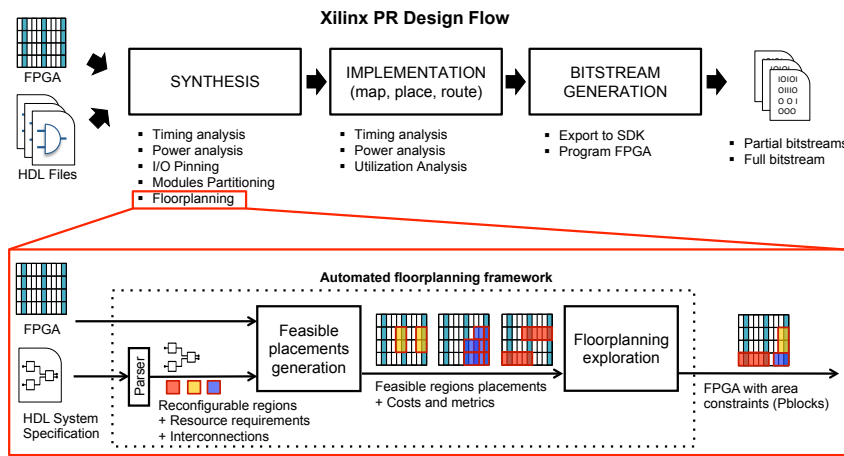


Fig. 2. The proposed floorplanning framework integrated within the Xilinx PR design flow.

the top level of the design, are used for the synthesis and the implementation of sub-circuits and the subsequent generation of partial bitstreams. Within this scenario, the floorplanning is a manual activity executed before the implementation phase, immediately after the definition of the reconfigurable regions within the top level specification.

The proposed floorplanning automation framework replaces the corresponding manual activity in the considered PR design flow. The framework takes in input the HDL structural specification of the system and translates it in an internal agile representation based on a graph. Moreover, it exploits synthesis reports to collect information on the resource requirements, that will be annotated on the system internal representation. The overall resource requirements of each reconfigurable region are computed as the maximum requirements among the various modules that will be hosted within the specific region. Finally, the framework takes in input the description of the considered FPGA device modeled as discussed in Section III. The output of the framework is the set of rules describing the floorplan solution, specified in the Xilinx constraint language to be imported in Vivado in order to continue with the subsequent implementation phase.

According to the formalization of the problem presented in the requirements REQ1-REQ5, the floorplanning automation framework is divided in two different phases:

- 1) **Feasible placements generation**, that consists in building a conflict graph where nodes represent the union of the P_n sets of possible feasible placements for each reconfigurable region n (REQ1-REQ4), and edges represent the overlapping among pairs of placements of different regions (REQ5).
- 2) **Floorplanning exploration**, that selects a possible placement in P_n for each reconfigurable region n such that there is no overlap among placements (REQ5) and an objective function specified by the designer is maximized.

We automated the two phases with different strategies according to their peculiarity and computational complexity. In particular, the first phase performs an exhaustive exploration for the definition of the conflict graph, since, as shown in the next section, the problem has a limited complexity. For the

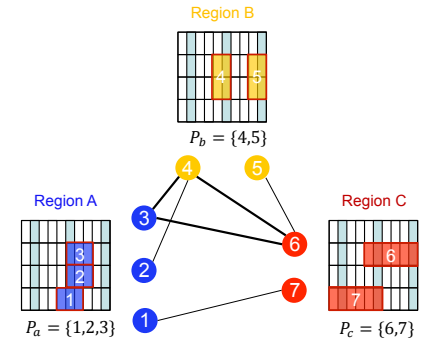


Fig. 3. Example of conflict graph.

second phase, characterized by a considerably larger design space, the framework features an efficient exploration engine powered by a genetic algorithm extended with a local search strategy. As shown in the experimental session, this strategy provides near-optimal solutions with a very limited execution time. Nevertheless, as shown in Section VII, the framework supports the integration of further automation strategies. The two phases are discussed in more details in the following sections.

V. FEASIBLE PLACEMENTS GENERATION

The first phase of the proposed framework is devoted to the definition of an abstract model called *conflict graph* that describes all the feasible placements for the various reconfigurable regions and the possible conflicts among pairs of placements. As shown in Figure 3, the conflict graph contains a group of nodes for each reconfigurable region n representing the overall enumeration of the feasible placements P_n , computed by fulfilling requirements REQ1-REQ4. Moreover, edges are used to represent conflicts between pairs of placements of two different regions, according to requirement REQ5.

It is worth noting that in the classical floorplanning for VLSI design it is commonly agreed that such a problem representation, based on the direct specification of all the possible region coordinates on the device grid, is extremely inefficient for automated optimization due to the huge solution space it defines. For this reason, past approaches have exploited various indirect representations, such as slicing trees, sequence pairs or other hierarchical tree-based representations [18]. At the opposite, the PR guidelines cause a considerable decrease in the number of feasible placements for a single reconfigurable region, thus allowing to effectively exploit such a direct representation of the placements. As an example, Figure 4 reports the number of feasible placements generated for a single reconfigurable region when varying its resource requirement on a specific Xilinx XC7V585TFF6 device; the number of all feasible placements under PR constraints (P_n) is two order of magnitude smaller with respect to the number of all the possible placements that can be generated on the device without

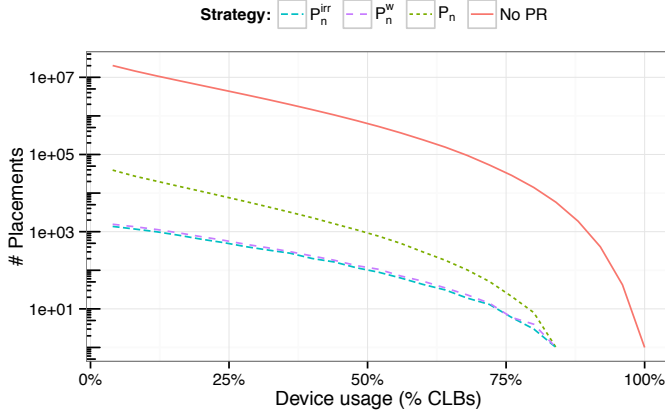


Fig. 4. Number of feasible placements for a single reconfigurable region.

constraints (namely *No PR* in the figure). It is worth noting that only CLBs are considered as resource requirements, while taking into account also other resource types would have even more decreased the number of placements.

The number of placements to be explored for the floorplanning can be even more reduced with respect to P_n , if we consider that in most of the cases, from an optimization point of view it is not efficient to define placements larger than the minimal bounding boxes containing the required resources. In fact, as discussed in [3] and [6], by using the minimal bounding boxes it is possible to reduce resource utilization, thus leaving space for additional functionalities and reducing the reconfiguration time. For this reason, we define a new set P_n^{irr} containing the *irreducible placements* of a region n as:

$$P_n^{irr} = \{p \in P_n \mid \nexists p_2 \in P_n : p_2 \neq p \wedge p_2 \prec p\} \quad (8)$$

where \prec represents a containment relation between two different placements of the same region. More formally, given two placements $p_1 = (x_1, y_1, w_1, h_1), p_2 = (x_2, y_2, w_2, h_2) \in P_n$, we have $p_1 \prec p_2$ if and only if $x_1 \geq x_2, y_1 \geq y_2, x_1 + w_1 \leq x_2 + w_2$ and $y_1 + h_1 \leq y_2 + h_2$. As shown in Figure 4, P_n^{irr} allows to reduce the size of the conflict graph of about another order of magnitude compared to P_n .

Even if P_n^{irr} sets are well-suited for optimizing resource occupation, we have experimentally noted that they may produce suboptimal results in terms of global wire length among various regions. For this reason, we have slightly relaxed the definition of P_n^{irr} to consider also placements that are required to be minimal only with respect to the horizontal direction. For a formal definition of this set of placements we consider a weaker containment relation \prec^w : given the two above placements p_1 and p_2 , we have $p_1 \prec^w p_2$ if and only if $x_1 = x_2, y_1 = y_2, h_1 = h_2$ and $w_1 \leq w_2$. Thus the corresponding width-reduced placements set for region n is defined as:

$$P_n^w = \{p \in P_n \mid \nexists p_2 \in P_n : p_2 \neq p \wedge p_2 \prec^w p\} \quad (9)$$

For the three defined sets the following relation holds:

$$P_n^{irr} \subseteq P_n^w \subseteq P_n \quad (10)$$

It is worth noting that the choice of reducing the placements only on the horizontal direction is suggested from the structure

Algorithm 1 Width-reduced placements generation

```

1: for each  $n \in N$  do
2:    $P_n^w \leftarrow \emptyset$ 
3:   for each  $x \in VL$  do
4:     for  $y \leftarrow 0$  to  $H - 1$  do
5:       for  $h \leftarrow 1$  to  $H - y$  do
6:          $w \leftarrow \text{searchMinimalWidth}(x, y, h, n)$ 
7:          $v \leftarrow \text{validAspectRatio}(x, y, h, w)$ 
8:         if  $w > 0 \wedge v = \text{true}$  then
9:            $P_n^w \leftarrow P_n^w \cup (x, y, w, h)$ 

```

of current devices. Usually H is much more coarse-grained than W ; as an example a Xilinx Virtex-5 XC5VLX110T is described using 8 rows and 62 columns of tiles ($W = 62$ and $H = 8$) [8]. On average, with respect to different CLB resource requirements, this relaxed strategy leads to 14% more placements with respect to P_n^{irr} , as shown in Figure 4.

A last relevant issue related to the generation of the feasible placement is the *aspect ratio*, that is the ratio between the width and the height of a placement. Indeed, as discussed in [15] and [14], extreme aspect ratios (e.g. lower than 1:5 or higher than 5:1) often lead to implementations with high routing congestion and low performance. This issue is mainly suffered on the vertical direction since its axis is more coarse-grained. Thus, placements with such elongated shapes can be filtered during the placement generation process; then, among the available ones, higher cost can be attributed to placements with extreme aspect ratios during the exploration phase.

Algorithm 1 automates the computation of the sets P_n^w . The procedure executes an extensive search of the possible placements by scanning all the valid coordinates of the device starting from the bottom left corner of the FPGA. For each point on the coordinate system, the algorithm considers all the possible placement heights that do not exceed the boundaries of the device and search for the minimal width needed to cover the required resources. Notice that depending on the resource requirements and on the presence of hard processors and static logic, the search for the minimal width can fail; in these situations the corresponding placement is not generated. In case of success, the *searchMinimalWidth* function returns the minimal width for the current starting point and height. If it is not possible to find a feasible placement for the given height and position, the method returns 0. At the same time the *validAspectRatio* function is called to check if the current placement has to be discarded due to extreme aspect ratios.

From an asymptotic complexity point of view, *searchMinimalWidth* function is the most time consuming operation in the innermost loop. By using a binary search and pre-computing the resources occupied by each placement on the device, this function can be implemented with a $\mathcal{O}(\log W)$ time complexity. Since the set VL , representing valid left coordinates for the regions, has a size proportional to W , *searchMinimalWidth* function is invoked $\mathcal{O}(|N| \cdot H^2 \cdot W)$ times. In conclusion, Algorithm 1 has an overall time complexity of $\mathcal{O}(|N| \cdot H^2 \cdot W \cdot \log W)$. Notice that H and W are usually small numbers and the placements generation takes only a small amount of time compared to the overall optimization of the floorplan. Another observation, deriving directly from

the algorithm, is that the number of width-reduced feasible placements for a region cannot exceed $H^2 \cdot W$. As discussed in Section VII, in real situations the number of feasible placements is limited up to few thousands items per set, and therefore leads to manageable conflict graphs and to a placements generation time of few seconds.

VI. FLOORPLANNING EXPLORATION

Once the feasible placements are generated, the second phase of the proposed floorplanning framework consists in the choice of the most suitable placement p for each region n among the available ones, such that 1) all selected placements do not overlap, meaning that no conflict edge exists among pairs of such placements, and 2) a specified objective function is optimized. The proposed floorplanning automation framework supports the integration of any automation engine capable of solving such an exploration problem by working on the defined conflict graph. It is worth noting that even if the conflict graph has a small size, the solution space, represented by the Cartesian product of the sets of feasible placements of each reconfigurable region, has a size that grows exponentially with respect to the number of regions, thus motivating the necessity of an efficient exploration engine.

In the preliminary formulation of the framework [16] we adopted an exploration engine based on an exact MILP model, whereas, within this work we designed and tested various heuristic methods in order to improve the floorplan exploration time. During our experimental sessions, we identified the Genetic Algorithm (GA) engine extended with steepest descent local search to be the most effective approach in finding near-optimal solutions in a reduced amount of time. Therefore, we here present the GA engine in details, while we refer the reader to Appendix A for a description of the MILP model that has been used as a baseline. From experiments conducted with different placement sets, we also noticed that the width-reduced placements P_n^w offer the best trade off in terms of size of the resulting solution space and quality of the achievable results. Hence, in the following discussion we refer to sets P_n^w , even though the approach is still valid when other sets of placements such as P_n or P_n^{irr} are considered.

The proposed GA engine for automating the floorplanning exploration is based on the classical simple Genetic Algorithm formulation [19]. We defined a solution encoding exploiting the enumeration of the feasible placements identified during the first phase of the floorplanning framework. More precisely, the *chromosome* is a linear vector where each position represents a reconfigurable region n , and the contained value refers to the feasible placement p in the corresponding set P_n^w . Then, the standard *crossover* and *mutation* operators have been employed. The *crossover* operator cuts in a random point the chromosomes of two parent solutions and exchanges the second parts to generate two children, while the *mutation* operator replaces with a given probability the placement of a region with another randomly selected placement in P_n^w .

In order to evaluate the solution we consider two different cost metrics:

- A_{cost} , the cost directly related to placement selection.

Algorithm 2 GA local search

```

1: function IMPROVESOLUTION(solution)
2:   obj ← solution.evaluate()
3:   repeat
4:     oldObj ← obj
5:     for each  $n \in N$  do
6:       for each  $p \in P_n$  do
7:         obj' ← solution.evaluatePlacement( $n, p$ )
8:         if obj' < obj then
9:           solution.setPlacement( $n, p$ )
10:        obj ← obj'
11:   until obj < oldObj
12:   return solution

```

- W_{cost} , the cost deriving from inter region wire length.

The first contribution can be easily computed summing the cost $a_{p,n}$ associated to each placement $p \in P_n^w$ that is selected for the current floorplan. As an example, the cost $a_{p,n}$ can refer to the aspect ratio of the placement, amount of wasted resources, or wire length of a connection to a fixed I/O pin. On the other hand, the second metric estimates the inter region wire length using the HPWL formula. HPWL considers the wire connections concentrated in the center of the regions and measures the wire length using the Manhattan distance. In conclusion, the considered *fitness* function is a linear combination of the two defined metrics and an additional parameter λ able to handle unfeasible situations:

$$obj = q_a \cdot \frac{A_{cost}}{A_{max}} + q_{wl} \cdot \frac{WL_{cost}}{WL_{max}} + \lambda \quad (11)$$

In the formula, A_{max} and WL_{max} represent the maximum values that A_{cost} and WL_{cost} can assume respectively; they are used to normalize the two contributions. Then, q_a and q_{wl} are user-defined weights. In particular, the fitness function first analyzes the feasibility of the solution only in terms of fulfillment of the non-overlapping condition (REQ5), and then computes the cost according to the selected metrics. If the solution is unfeasible, a penalty value λ , defined as the number of pairs of regions that overlap, is summed to the objective value. In Equation (11) we force $q_a + q_{wl} = 1$ so that, valid floorplans are represented by $0 < obj \leq 1$, while $obj > 1$ identifies unfeasible solutions. The goal of the λ factor is to enable the *selection* operator of the GA (we use the classical tournament selection) to rank unfeasible solutions in terms of the criticality of the constraint violation.

The choice of a simple solution encoding and operators has been driven by the possibility to directly manage the solution space, as motivated in Section V. However, as a drawback, we have noted during a preliminary experimental evaluation that such GA engine is not able to obtain better performance than the preliminary MILP approach since it generates too many unfeasible solutions. In fact, the direct problem formulation leads feasible solutions to evolve in unfeasible ones with a high probability. The main cause is the crossover operator that, due to its nature, applies “global changes” to each explored solution; at the opposite the mutation operator which performs local moves, has higher possibilities to make a feasible solution to evolve to another feasible one, that is a “neighbor” in the solution space. For this reason, such engine

has been enhanced with a *local search* function, based on a steepest descent heuristic, that improves the current solution with iterated local modifications until no further improvement is possible. The strategy, shown in Algorithm 2, is invoked in the GA fitness function and guarantees to reach a local optimum from the input solution. We empirically demonstrated that the adoption of local search within GA leads to a hybrid approach able to converge faster towards global optima [20].

VII. EXPERIMENTAL EVALUATION

The proposed floorplanning automation framework has been implemented in C++; GALib [21] has been used for the GA engine. Within the following experimental sessions, we also integrated in the framework a Simulated Annealing (SA) engine, implemented in C++ by using the GNU Scientific library [22], and the preliminary MILP formulation [16]. For the other considered state-of-the-art approaches, the original algorithms provided by the related authors have been adopted, while all the MILP models have been solved using Gurobi 6.5. Finally, we integrated a Graphical User Interface (GUI) for analyzing and possibly modifying the floorplan solutions by means of a Web application in Javascript and HTML5².

In the first experimental session, we performed an extensive testing campaign considering a large set of synthetic circuits aimed at demonstrating that the proposed GA engine outperforms the state-of-the-art approaches, whereas in the second section we performed a more in-depth comparison of various engines (such as GA, SA and MILP) exploiting the same direct problem representation proposed in this paper. Finally, we carried out two real case studies to show that the framework generates feasible floorplanning solutions possibly without any manual action of the designer, and, moreover, it is able to improve the system performance. The three sessions are presented in the following sections. As a final note, all the experiments have been performed on a 2.2GHz Intel Core Duo T6600 processor running a Linux operating system.

A. Comparison with respect to past approaches

The first experimental session considered the test suite of synthetic circuits from [8] targeted for the Virtex-5 XC5VLX110T device. This suite consists of 20 circuits with different area occupancy and number of reconfigurable regions; specifically there are 4 circuits having a number of reconfigurable regions in the range $\{5, 10, 15, 20, 25\}$, while with respect to area utilization there are 5 circuits for each fixed device occupancy in the range $\{70\%, 75\%, 80\%, 85\%\}$. It is worth noting that the maximum number of regions for the considered circuits has been set by taking into account that reasonably a reconfigurable system does not feature a larger number of reconfigurable regions. Moreover, we also made comparisons on some circuits from MCNC and GSRC suites, adapted as done in [6]; more precisely, we considered *apte*, *xerox*, *hp*, *ami33* and *ami49* targeted for a more recent Virtex-7 XC7K160T device.

In this first session, we compared our GA engine (dubbed PA-GA³) against the most efficient state-of-the-art approaches discussed in Section II, i.e., [6], the HO and O MILP-based algorithms presented in [8] and our preliminary MILP formulation [16] (dubbed as PA-MILP). Nevertheless, in order to perform the comparison it was necessary to remove requirements REQ3 and REQ4 from the placement generation process since previous approaches do not support them and their integration for [6] and [8] is not straightforward. Regarding the objective function, in this section we only considered the overall wire length since it has been noted to be the most challenging optimization goal. For each experiment we executed 10 runs of [6] and considered the best result as its final solution. According to the approach defined in [8], we run the HO approach by starting from some of the best solutions found by [6] (the ones within 10% from the best one), and subsequently O by using the final solution achieved by HO. For all the MILP formulations, the Gurobi solver execution time was limited to 1800 seconds, whereas, for PA-GA engine we used a stopping criterion based on elapsed time and the same time limit was applied. The elaboration was parallelized on all the available cores by using the *Threads* Gurobi setting and by running different processes for PA-GA with different random seeds. Notice however that the time limit does not take into account the time needed by PA-MILP and PA-GA for the generation of the feasible placements and the additional time to generate the initial solution for O [8]. Finally, PA-GA and PA-MILP used P_n^w as input.

Tables II and III show the results of this first experimental session both in terms of execution time and quality of the achieved solutions. PA-GA was always able to find equivalent or better solutions than PA-MILP that in turns provided better results than [8] and [6]. Furthermore, the highest improvements were achieved for the most challenging circuits consisting of high number of reconfigurable regions. Specifically, for the test cases having 20 and 25 regions PA-GA reduced the wire length of PA-MILP solutions by 8.2% on average while using the same amount of time. On the other hand, when considering the variation of resource usage (reported in Table III), as expected, the best results for both PA-GA and PA-MILP are obtained for circuits with lower resource requirements even if there is no pronounced trend.

Table IV reports the results for the re-adapted MCNC and GSRC benchmarks. Results show that PA-MILP gives an improvement with respect to O that varies from 1.2% on *ami33* to 49.5% on *hp* circuits, while with respect to the comparison between PA-MILP and PA-GA, the results are aligned to the synthetic benchmark trend. Indeed, for *apte*, *xerox* and *hp* circuits both PA-MILP and PA-GA provides a similar overall wire length, whereas, when dealing with the bigger *ami33* and *ami49* circuits, PA-GA is able to improve PA-MILP solutions by 18.8% and 18.0% respectively.

It is worth noting that for the synthetic circuits having 5 reconfigurable regions (Table II), PA-MILP is able to certify the optimality of the solutions and thus complete its execution

²A preview of the GUI can be accessed at: <http://floorplacer.necst.it>

³The label PA, standing for Proposed Approach, here identifies all the engines based on the direct problem formulation proposed in this work.

TABLE II
RESULTS WITH DIFFERENT NUMBERS OF RECONFIGURABLE REGIONS

# RRs	Average wire length improvement w.r.t. [6]				Average execution time (sec)				
	HO[8]	O [8]	PA-MILP [16]	PA-GA	[6]	HO [8]	O [8]	PA-MILP [16]	PA-GA
5	6.99%	7.48%	7.46%	7.46%	11.0	13.3	84.1	32.5	1801.2
10	7.59%	11.65%	17.98%	19.07%	24.1	48.3	1848.3	1421.8	1801.4
15	8.88%	16.25%	34.03%	36.14%	41.1	74.6	1874.6	1802.1	1801.7
20	5.47%	14.80%	33.29%	39.17%	65.5	82.8	1882.9	1802.6	1801.9
25	5.67%	24.01%	40.72%	45.32%	94.0	119.1	1919.2	1803.0	1802.3

TABLE III
RESULTS WITH DIFFERENT OVERALL DEVICE OCCUPANCY

Occupancy	Average wire length improvement w.r.t. [6]				Average execution time (sec)				
	HO [8]	O [8]	PA-MILP [16]	PA-GA	[6]	HO [8]	O [8]	PA-MILP [16]	PA-GA
70%	8.51%	17.40%	30.19%	32.15%	47.4	87.6	1559.0	1454.7	1801.7
75%	5.49%	18.89%	26.21%	28.74%	47.3	67.1	1514.2	1138.9	1801.7
80%	6.20%	12.86%	27.57%	30.73%	47.1	60.0	1508.1	1448.9	1801.7
85%	7.48%	10.20%	22.82%	26.10%	46.7	55.8	1505.9	1447.1	1801.7

TABLE IV
APPROACHES COMPARISON ON DIFFERENT TEST CASES

Circuit	# RRs	Wire length					Execution time (sec)				
		[6]	HO [8]	O [8]	PA-MILP [16]	PA-GA	[6]	HO [8]	O [8]	PA-MILP [16]	PA-GA
apte	9	12789	12029	9682	5313	5206	21.75	147.93	1947.93	1801.87	1801.88
xerox	10	26589	25878	22974	12643	12813	25.53	307.97	2107.98	1802.36	1802.37
hp	11	12403	11796	11036	5568	5298	29.42	201.04	2001.05	1802.41	1802.45
ami33	33	172332	157583	130121	128538	104414	164.61	324.54	2124.57	1803.93	1803.98
ami49	49	55819	53178	36930	22669	18583	180.54	192.25	1992.26	1809.35	1809.47

before the given time limit. PA-GA, being a meta-heuristic approach, cannot state if the identified solution is optimal, and, hence, the execution time of the algorithm depends on the time budget assigned. However, in these cases, we noted that PA-GA converge faster to the optimal solution than PA-MILP, while as the problem grows above the 10 regions, no MILP formulation is able to reach the optimal solution in a reasonable time when the inter-region wire length is considered. In fact, in our tests we run the PA-MILP, i.e. the most efficient MILP formulation for several hours; however, after an initial very fast convergence to a near-optimal solution, the engine was not able to improve the solution or certify its optimality. This is related to the weak linear relaxation bounds provided by the MILP formulation with respect to inter-region wire length; the issue was only partially mitigated by using additional cuts to the model (see Appendix A). As an example, we report in Figure 5 the graph representing the improvement of the best solution for the considered algorithms. We may note from the figure that PA-GA is the faster to evolve towards near-optimal solutions. This trend is representative for all the performed tests. Moreover, we noted that on average PA-GA and PA-MILP tends to stabilize their solutions in less than 600 seconds, while after that, no relevant improvement is reported.

As a final note, the generation of the definition of the conflict graph model had a negligible impact on the overall execution time of the proposed algorithms. Indeed, in real

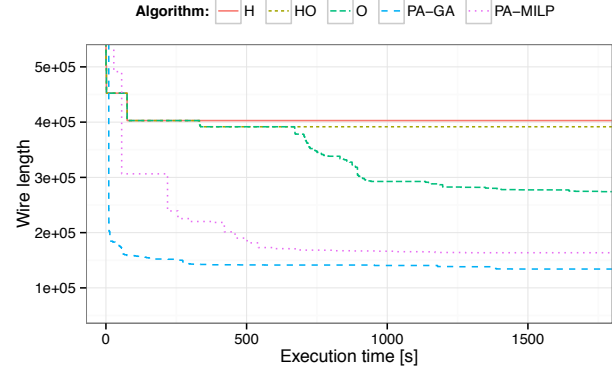


Fig. 5. Solution improvement over time for different approaches on the ami49 test case.

situations, the size of the conflict graph is manageable; as an example, for the circuit having the highest number of regions (ami49), the feasible placements generation process produced 146446 nodes in less than 10 seconds.

B. Analysis of engines based on the proposed representation

In a second session, we performed a more challenging comparison of PA-GA against other engines exploiting the same direct problem representations. In particular, we considered 1) PA-MILP, 2) an SA engine (called PA-SA), since it represents

TABLE V
PROPOSED APPROACHES COMPARISON WITH DIFFERENT NUMBER OF REGIONS

# RRs	Solutions improvements w.r.t. PA-MILP [16]								
	$q_a = 1.0, q_{wl} = 0.0$			$q_a = 0.5, q_{wl} = 0.5$			$q_a = 0.0, q_{wl} = 1.0$		
	PA-SA	PA-GAn	PA-GA	PA-SA	PA-GAn	PA-GA	PA-SA	PA-GAn	PA-GA
5	0.00%	-3.78%	0.00%	-0.77%	-11.95%	0.00%	-3.11%	-2.77%	0.00%
10	-4.31%	-23.02%	0.00%	-2.32%	-27.09%	0.18%	-8.59%	-29.16%	0.55%
15	-7.25%	-31.62%	-0.27%	-6.19%	-21.16%	5.28%	-12.69%	-78.73%	5.35%
20	-9.97%	-31.53%	-0.94%	-4.44%	-17.45%	7.09%	-15.29%	-30.36%	10.41%
25	-17.02%	-24.34%	-0.35%	-7.38%	-20.66%	9.42%	-3.75%	-8.78%	19.39%
30	-10.19%	-18.97%	-0.60%	-10.74%	-16.80%	8.95%	0.21%	-21.48%	21.86%

TABLE VI
PROPOSED APPROACHES COMPARISON WITH VARYING RESOURCE USAGE

Usage	Solutions improvements w.r.t. PA-MILP [16]								
	$q_a = 1.0, q_{wl} = 0.0$			$q_a = 0.5, q_{wl} = 0.5$			$q_a = 0.0, q_{wl} = 1.0$		
	PA-SA	PA-GAn	PA-GA	PA-SA	PA-GAn	PA-GA	PA-SA	PA-GAn	PA-GA
70%	-9.63%	-17.24%	0.00%	-1.02%	-13.23%	7.50%	1.44%	-15.95%	10.67%
75%	-7.72%	-24.94%	-0.32%	-3.77%	-11.13%	5.98%	-3.16%	-21.95%	9.29%
80%	-6.37%	-19.66%	-0.28%	-11.66%	-28.70%	3.21%	-17.54%	-53.37%	7.95%
85%	-8.75%	-26.81%	-0.82%	-4.76%	-25.84%	3.94%	-9.52%	-25.12%	10.49%

the classical approach for floorplanning strategies, and 3) a GA engine exploiting the same solution encoding but without any local search strategy (dubbed as PA-GAn). The SA engine was defined on the basis of the standard SA algorithm and re-using the evaluation and mutation functions of the GA engine. In this experimental session we considered PA-MILP as a baseline, since it has been the preliminary automating solution designed for the proposed framework. For this analysis, we included requirements REQ3 and REQ4, and we re-adapted the suite of the first session to target a Virtex-7 XC7V585T device; we actually modified the resource requirements of the circuits according to the size of the new device and we considered a new set of circuits with 30 regions. It is worth noting that the considered Virtex-7 device presents a higher heterogeneous distribution of resources and requires the introduction of more forbidden placements than the Virtex-5 one used in the first experimental session. Indeed, in some preliminary tests, we found that the considered state-of-the-art engines in [6], [8] failed in finding any feasible solution on this device.

Moreover, the tests have been performed considering different settings of the objective function ranging from an optimization based only on placements cost ($q_a = 1.0, q_{wl} = 0.0$), one considering only wire length ($q_a = 0.0, q_{wl} = 1.0$) and finally a mixed objective function taking into account both metrics to the same extent ($q_a = 0.5, q_{wl} = 0.5$). In order to perform a fair comparison of the approaches in terms of exploration efficiency, according to the discussion in Section VII-A we fixed a limited time budget of 600 seconds that includes the time for the generation of the feasible placements. Except for the new time limit, the run settings for PA-MILP and PA-GA were as in the previous section, whereas, we restarted PA-SA engine several times on the available cores using different random seeds until the available time budget elapsed.

Table V and Table VI compare the obtained results according to the number of regions and device usage respectively.

For the problem instances consisting of 5 regions the MILP approach and the GA were both able to find the optimal solution in all cases, whereas the SA engine found optimal solutions only when the objective function was set to consider uniquely region placements cost. In general, the SA engine was almost never able to achieve better solutions than the MILP based algorithm except for some problem instances consisting of large number of regions. This result is quite interesting since SA is the commonly used approach for automating the floorplanning exploration. On the contrary, we may conclude that it is not well-suited for the defined problem representation due to the fact that many unfeasible solutions can be generated.

PA-GA proved to be an effective approach, leading to almost the same results for an optimization based on placement cost, while greatly outperforming the MILP engine when considering the most difficult problem, that is the wire length optimization. This is especially highlighted when the circuits feature a large number of regions or a low resource usage. This improvement was mainly obtained exploiting local search within GA that allows to quickly explore a solution space consisting of local optimal solutions. Indeed, as it is possible to notice from the two tables, PA-GAn provides results considerably far from its enhanced counterpart since it spends a large amount of time exploring unfeasible regions of the solution space, without the capability to recover to a feasible solution. It is worth noting that in some experiments PA-GAn was not able even to find an initial feasible solution; such situations were discarded from the results reported in the two tables. Thus, we may conclude that the GA engine is the most promising solution for the proposed floorplanning framework.

C. Case studies

Finally, we validated the proposed approach on two real case studies to be implemented on a Virtex-7 XC7V585T device.

The first case study is a re-adaptation of a Xilinx sample design (*project_cpu_virtex7*), consisting of five modules connected using a star topology, whereas the second case study is an in-house design of an image processing pipeline with reconfigurable components. As already noted in the previous section, the approaches proposed in [6] and [8] do not consider requirements REQ3 and REQ4 thus leading to potentially invalid floorplan solutions with respect to the subsequent place and route phase. Hence, we compared the results achieved by the proposed floorplanner with respect to solutions designed manually by starting from the initial placements provided by the *place pblock* feature available in Vivado. This Vivado feature provides the user with suggestions on where to place the reconfigurable regions, however the identified placements do not meet PR guidelines and require manual modifications.

Similarly to [15], the Xilinx case study has been re-adapted considering each of the five available modules as reconfigurable, thus leading to five distinct reconfigurable regions each containing a single module. Notice however that this does not represent a limitation for the evaluation of the proposed approach since the employed implementation flow is the same. The resource requirements of the reconfigurable regions are derived from the requirements of the corresponding modules (shown in Table VII) in which the number of LUTs was augmented by approximately 25% to ensure enough space for the insertion of proxy logic [1]. Furthermore, the number of interconnections among reconfigurable regions together with interconnections to I/O are summarized in Table VIII.

For the exploration, the objective function was set to consider wire length and resource consumption to the same extent in order to reduce both reconfiguration overhead and improve the possibility to meet timing constraints. The floorplan solution identified by PA-GA is shown in Figure 6a together with the placed and routed circuit. Overall the implementation phase was successful and the timing constraint requiring a 100MHz frequency was met. On the other hand, Figure 6b presents the initial solution provided by Vivado *place pblock*, and Figure 6c the subsequent manually re-adapted floorplan. It is clearly visible that the designer has to perform a considerable and intrusive change of the solution proposed by Vivado; from our experience we may report that such activity requires around 2 hours of time, while our automated engine requires a few minutes. Moreover when considering the quality of the achieved solutions, this second one was not able to meet timing during implementation due to not optimized inter-region interconnections. By lowering the timing constraint it was possible to meet timing at 80MHz, however no place and route solution was found satisfying timing constraints with frequency equal or higher than 85MHz. Furthermore, the floorplan produced by the GA engine was able to reduce the overall size of the partial bitstreams by 25.7% with respect to the manual solution.

As a second case study, we realized a design in the context of image analysis consisting of seven different modules whose interconnections are shown in Figure 7. In particular, the design is composed of two main computational pipelines that operate on a gray scaled image. The first pipeline includes the *histogram*, *Otsu filter* and *threshold 1* modules, it binarizes

TABLE VII
RESOURCE REQUIREMENTS OF MODULES FROM THE XILINX CASE STUDY

Module	LUTs	Registers	F7 Muxes	F8 Muxes	BRAMs	DSPs
cpuEngine	7440	3892	297	0	21	4
fftEngine	2837	1679	0	0	16	96
usbEngine0	6000	4699	259	81	36	0
usbEngine1	6080	4699	259	81	36	0
wbArbEngine	6800	1044	1959	172	0	0

TABLE VIII
INTERCONNECTION MATRIX OF REGIONS FOR THE XILINX CASE STUDY

Region	fftEngine	usbEngine0	usbEngine1	wbArbEngine	I/O
cpuEngine	1	0	0	311	0
fftEngine	-	0	0	106	69
usbEngine0	-	-	0	118	69
usbEngine1	-	-	-	118	0
wbArbEngine	-	-	-	-	0

the given image by applying the Otsu separation algorithm [23], while, the second pipeline is configured to perform edge detection by exploiting the *Gauss filter*, *Laplace filter* and *Threshold 2* modules. The modules were generated by using Vivado HLS and AXI stream interfaces were used for communication. Furthermore, we implemented different alternative versions of each module within the design (employing different algorithms or having a different trade-off between results accuracy and execution time) to exploit partial reconfiguration to switch from one to the other one; thus, seven reconfigurable regions were considered (one for each component within the original design). The resource requirements for each version of the various modules are listed in Table IX, whereas the requirements for the corresponding reconfigurable regions were obtained as in the previous case study.

Even in this scenario, the objective function was tuned to take into account inter-region wire length, regions aspect ratio and wasted resources to the same extent. Within this case study, an analysis of the post-implementation results showed that the critical paths were represented by the internal interconnections between the computational logic and the local BRAMs of the modules, whereas, inter-region interconnections were easily routed. Due to the peculiar characteristics of this design, both the PA-GA and manually re-adapted floorplans were able to meet timing at 120MHz and failing at a frequency equal or higher than 125MHz. Moreover, the floorplan produced by the GA engine was able to reduce the overall size of partial bitstreams of the manual solution from 9695 KB to 8815 KB, hence leading to a smaller reconfiguration time. Finally, similarly to the previous case, the manual definition of the floorplan required about 2 hours of activity.

VIII. CONCLUSIONS

This work has proposed a novel floorplanning automation framework, compatible with the Xilinx tool-chain and its PR

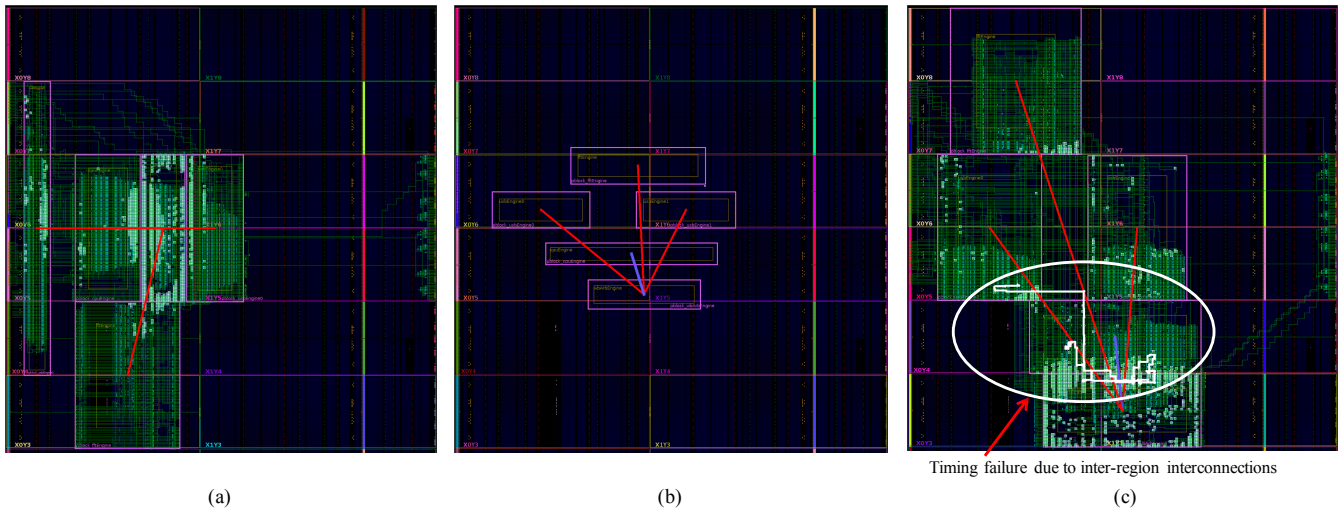


Fig. 6. Floorplans for the Xilinx case study: (a) place and route of PA-GA floorplan at 100MHz constraint, (b) initial floorplan obtained with Vivado *place pblock*, and, (c) place and route of manually adapted Vivado floorplan at 90MHz constraint.

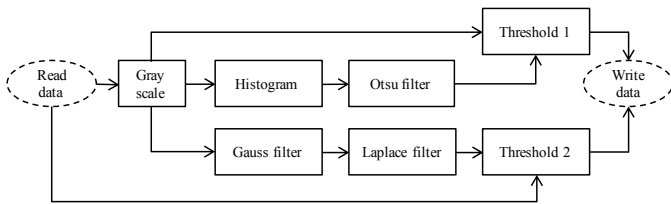


Fig. 7. Modules interconnections for the image processing case study.

TABLE IX
RESOURCE REQUIREMENTS OF MODULES FOR THE IMAGE PROCESSING CASE STUDY

Region	Module	LUTs	FFs	BRAMs	DSPs
Laplace filter	LF_v1	628	332	64	2
Gauss filter	GF_3x3	807	465	64	0
	GF_3x3 float	881	809	32	5
	GF_5x5 float	815	760	32	5
Gray scale	GS_v1	334	238	64	4
Histogram	Hi_v1	256	180	1	0
	Hi_v2	104	87	1	0
Otsu filter	OF_v1	1205	1164	0	13
	OF_v2	726	517	0	2
Threshold 1	Th_v1	115	71	0	0
Threshold 2	Th_v1	115	71	0	0

flow. The framework considers a direct problem representation, which consists in an explicit enumeration of the possible placements of each region. The defined model allows to simplify the development of efficient floorplanning algorithms devoted to the optimization of different metrics such as aspect ratio, inter-region wire length and resource consumption. Various engines, based on an exact MILP formulation, GA and SA heuristic approach, possibly enhanced with local search strategies, have been designed for automating the floorplanning

activity. Such algorithms are experimentally evaluated with a challenging synthetic benchmark suite and real case studies. Experimental results demonstrated the effectiveness of the proposed direct problem representation and superiority of the defined GA engine with respect to the other defined strategies and the state-of-the-art approaches in terms of exploration time and identified solution.

APPENDIX A MILP FORMULATION

Within this appendix we report the MILP model proposed in the preliminary version of the framework [16] and used within Section VII as a baseline for algorithms comparison. The variables, sets and parameters of the formulation are listed in Table X, whereas the model constraints and objective function are summarized in Table XI.

In the proposed model, the binary variables $x_{n,p}$ represent the current solution by stating which placement p is chosen for a given region n ; when considering the conflict graph, these variables state which specific node is selected for each region. Thus, a first class of constraints, dubbed as *placements constraints*, guarantees that a given solution is feasible: in particular, exactly one placement for each region must be selected (C1), and no pairs of placements connected by an edge can be selected since they are overlapping (C2). It is worth noting that constraint C2 is defined on maximal cliques (i.e. fully-connected subgraphs) in the conflict graph instead of single edges. This allows to reduce the size of the model while, at the same time, improves the linear relaxation bounds during the MILP solving process [24], so that the overall effect is a speed-up in the MILP solver performance. Such a reduction in the problem size can be clearly seen in Figure 8, by analyzing the number of edges between the placements generated for two regions when varying equally their requirements in terms of CLBs; more precisely, the figure compares the number of non-zero terms within the constraint matrix of the complete MILP model using the single edges approach and the maximal cliques approach.

TABLE X
MILP VARIABLES, SETS AND PARAMETERS

Sets	
N	set of reconfigurable regions to floorplan
P_n^w	set of width-reduced feasible placements for region n
I	set of interconnections between regions. Each element is a tuple of the form: $(n1, n2, b)$ where $n1$ and $n2$ are the regions involved in the interconnections and b is its width
Parameters	
W	maximum value on the horizontal direction
H	maximum value on the vertical direction
$tileW$	the width of a tile within the FPGA
$tileH$	the height of a tile within the FPGA
$a_{n,p}$	cost associated to placement $p \in P_n^w$ for region $n \in N$
q_a	weight associated to the area cost
q_{wl}	weight associated to the wire length cost
A_{max}	maximum cost due to placements selection
WL_{max}	maximum cost related to global inter-region wire length
Variables	
$x_{n,p}$	binary variable set to 1 if and only if the placement $p \in P_n^w$ is selected for region $n \in N$
cx_n	x coordinate of region $n \in N$ centroid
cy_n	y coordinate of region $n \in N$ centroid
$dx_{n1,n2}$	horizontal distance between centroids of regions $n1, n2 \in N$
$dy_{n1,n2}$	vertical distance between centroids of regions $n1, n2 \in N$
A_{cost}	floorplan cost due to placements selection
WL_{cost}	floorplan cost related to global inter-region wire length

TABLE XI
MILP MODEL CONSTRAINTS AND OBJECTIVE FUNCTION.

Placements constraints	
C1	$\sum_{p \in P_n^w} x_{n,p} = 1, \forall n \in N$
C2	$\sum_{n \in N, p \in P_n^w : p \perp (xp, yp, 1, 1)} x_{n,p} \leq 1,$ $\forall xp \in [0, W - 1], yp \in [0, H - 1]$
Wire length semantics	
C3	$cx_n = \sum_{p=(xp,yp,wp,hp) \in P_n^w} x_{n,p} \cdot (xp + wp/2), \forall n \in N$
C4	$cy_n = \sum_{p=(xp,yp,wp,hp) \in P_n^w} x_{n,p} \cdot (yp + hp/2), \forall n \in N$
C5	$dx_{n1,n2} \geq cx_{n1} - cx_{n2}, \forall n1, n2 \in N \mid n1 \neq n2$
C6	$dx_{n1,n2} \geq cx_{n2} - cx_{n1}, \forall n1, n2 \in N \mid n1 \neq n2$
C7	$dy_{n1,n2} \geq cy_{n1} - cy_{n2}, \forall n1, n2 \in N \mid n1 \neq n2$
C8	$dy_{n1,n2} \geq cy_{n2} - cy_{n1}, \forall n1, n2 \in N \mid n1 \neq n2$
Cost functions definition	
C9	$A_{cost} = \sum_{n \in N, p \in P_n^w} a_{n,p} \cdot x_{n,p}$
C10	$WL_{cost} = \sum_{(n1,n2,b) \in I} (dx_{n1,n2} \cdot tileW + dy_{n1,n2} \cdot tileH) \cdot b$
Additional cuts	
C11	$dx_{n1,n2} + dy_{n1,n2} \geq$ $\sum_{p=(xp,yp,wp,hp) \in P_{n1}^w} x_{n1,p} \cdot \min\{wp/2, hp/2\} +$ $\sum_{p=(xp,yp,wp,hp) \in P_{n2}^w} x_{n2,p} \cdot \min\{wp/2, hp/2\},$ $\forall n1, n2 \in N \mid n1 \neq n2$
Objective function	
OBJ	$\min \left\{ q_a \cdot \frac{A_{cost}}{A_{max}} + q_{wl} \cdot \frac{WL_{cost}}{WL_{max}} \right\}$

variables constraints C3-C8 are specified; constraints C3-C4 compute the coordinates of the centroids, while constraints C5-C8 ensure that the distances among regions cannot be less than expected.

An in-depth analysis of the formulation has shown that constraints C3-C8 give weak bounds when the linear relaxation of the MILP model is solved. For this reason we introduced the additional cut C11, stating that that the centroid distance of two regions has to be at least the sum of the distances to reach the centroids of the selected placements from their nearest borders. Indeed a wire connecting the centroids of two placements has to cross at least one border for each of them.

ACKNOWLEDGMENTS

This work was supported by the European Commission in the context of the H2020 FETHPC EXTRA project (#671653). Authors would also like to thank Xilinx, especially Patrick Lysaght, Jason Wong and Raymond Kong, for their support during this work.

REFERENCES

- [1] Xilinx Inc, "Vivado Design Suite User Guide: Partial Reconfiguration," 2015. [Online]. Available: <http://www.xilinx.com>
- [2] —, "PlanAhead." [Online]. Available: <http://www.xilinx.com/tools/planahead.htm>
- [3] L. Cheng and M. Wong, "Floorplan Design for Multimillion Gate FPGAs," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 12, pp. 2795–2805, 2006.
- [4] C. Beckhoff, D. Koch, and J. Torresson, "Automatic floorplanning and interface synthesis of island style reconfigurable systems with goahead," in *Architecture of Computing Systems (ARCS)*, 2013, pp. 303–316.
- [5] S. Yousuf and A. Gordon-Ross, "DAPR: Design Automation for Partially Reconfigurable FPGAs," in *Proc. Int. Conf. on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, 2010, pp. 1–7.
- [6] C. Bolchini, A. Miele, and C. Sandionigi, "Automated Resource-Aware Floorplanning of Reconfigurable Areas in Partially-Reconfigurable FPGA Systems," in *Proc. Intl. Conf. on Field Programmable Logic and Applications (FPL)*, 2011, pp. 532–538.

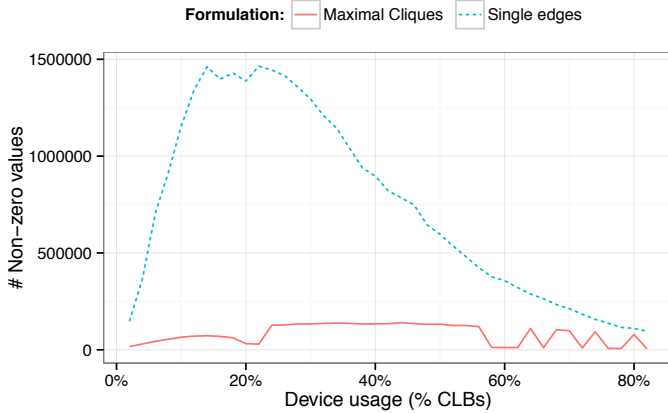


Fig. 8. Comparison of model size for the *maximal cliques* and *single edges* MILP formulations on a problem consisting of two regions having equal resource requirements.

A second class of constraints, parameters and variables are used to compute the cost OBJ of a given solution, that is equivalent to the one used for the GA engine without the penalty contribution. Constraint C9 computes the A_{cost} metric by summing the cost $a_{p,n}$ of each selected placement, while constraint C10 computes the global HPWL. The specification of C10 requires the introduction of variables cx_n and cy_n to compute the coordinates of the centroid of each region n , and variables $dx_{n1,n2}$ and $dy_{n1,n2}$ to compute the Manhattan distance between the centroids of each couple of regions $n1$ and $n2$. Moreover, in order to guarantee the semantics of these

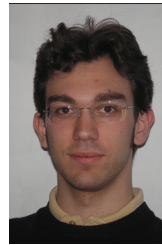
- [7] K. Vipin and S. A. Fahmy, "Architecture-aware reconfiguration-centric floorplanning for partial reconfiguration," in *Proc. Intl. Conf. on Reconfigurable Computing: architectures, tools and applications (ARC)*, 2012, pp. 13–25.
- [8] M. Rabozzi, J. Lillis, and M. D. Santambrogio, "Floorplanning for Partially-Reconfigurable FPGA Systems via Mixed-Integer Linear Programming," in *Proc. Intl. Symp. on Field-Programmable Custom Computing Machines (FCCM)*, 2014, pp. 186–193.
- [9] A. Montone, M. D. Santambrogio, D. Sciuto, and S. O. Memik, "Placement and Floorplanning in Dynamically Reconfigurable FPGAs," *ACM Trans. on Reconfigurable Technology and Systems*, vol. 3, no. 4, 2010.
- [10] P.-H. Yuh, C.-L. Yang, and Y.-W. Chang, "Temporal floorplanning using the three-dimensional transitive closure subGraph," *ACM Trans. on Design Automation of Electronic Systems*, vol. 12, no. 4, 2007.
- [11] L. Singhal and E. Bozorgzadeh, "Multi-layer Floorplanning on a Sequence of Reconfigurable Designs," in *Proc. Intl. Conf. on Field Programmable Logic and Applications (FPL)*, 2006, pp. 1–8.
- [12] P. Banerjee, M. Sangtani, and S. Sur-Kolay, "Floorplanning for Partially Reconfigurable FPGAs," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 30, no. 1, pp. 8–17, 2011.
- [13] Y. Feng and D. P. Mehta, "Heterogeneous Floorplanning for FPGAs," in *Proc. Intl. Conf. on VLSI Design*, 2006, pp. 257–262.
- [14] J. Lamprecht and B. Hutchings, "Profiling FPGA floor-planning effects on timing closure," in *Proc. Intl. Conf. on Field Programmable Logic and Applications (FPL)*, 2012, pp. 151–156.
- [15] C. E. Neely, G. Brebner, and W. Shang, "ReShape: Towards a High-Level Approach to Design and Operation of Modular Reconfigurable Systems," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 6, no. 1, pp. 5:1–5:23, May 2013.
- [16] M. Rabozzi, A. Miele, and M. Santambrogio, "Floorplanning for Partially-Reconfigurable FPGAs via Feasible Placements Detection," in *Proc. Intl. Symp. on Field-Programmable Custom Computing Machines (FCCM)*, 2015, pp. 252–255.
- [17] S. N. Adya and I. L. Markov, "Fixed-outline floorplanning: enabling hierarchical design," *IEEE Trans. on VLSI Systems*, vol. 11, no. 6, pp. 1120–1135, 2003.
- [18] S. H. Gerez, *Algorithms for VLSI design automation*. Wiley New York, 1999, vol. 8.
- [19] D. E. Goldberg, *Genetic Algorithms in Search Optimization and Machine Learning*. Addison-Wesley, 1989.
- [20] W. Wan and J. B. Birch, "An improved hybrid genetic algorithm with a new local search procedure," *Journal of Applied Mathematics*, 2013.
- [21] M. Wall, "GALib," 2007. [Online]. Available: <http://lancet.mit.edu/gal/>
- [22] "GNU Scientific Library," 2014. [Online]. Available: <http://www.gnu.org/software/gsl>
- [23] N. Otsu, "A threshold selection method from gray-level histograms," *Automatica*, vol. 11, no. 285–296, pp. 23–27, 1975.
- [24] S. Rebennack, "Stable set problem: Branch & cut algorithms," in *Encyclopedia of Optimization*, C. A. Floudas and P. M. Pardalos, Eds. Springer, 2009, pp. 3676–3688.



Marco Rabozzi is a PhD student at Politecnico di Milano. In 2014 He received his M.Sc. degree in Computer Science from the University of Illinois at Chicago and the master degree in Computer Science and Engineering from Politecnico di Milano. His main research interests are related to combinatorial optimization and computer-aided design, with a focus on mixed-integer linear programming, algorithms for resource constrained scheduling problems and automated floorplanning tools for FPGA devices.



Gianluca Carlo Durelli is a Research Affiliate at Politecnico di Milano. He received its B.Sc. and M.Sc. in Computer Engineering in 2009 and 2012 respectively, and a Ph.D. degree in Computer Science in 2016 from Politecnico di Milano. His research interests revolve around embedded systems, computer architectures and operating systems, with a particular focus on runtime resource management.



/many-core systems and FPGA-based systems design.

Antonio Miele is an Assistant Professor at Politecnico di Milano since 2014. He holds a master degree in Computer Engineering from Politecnico di Milano and a M.Sc. in Computer Science from the University of Illinois at Chicago. In 2010 he received a Ph.D. degree in Information Technology from Politecnico di Milano. His main research interests are related to the definition of design and analysis methodologies for embedded systems, in particular focusing fault tolerance and reliability issues, runtime resource management in heterogeneous multi-



John Lillis is currently an Associate Professor in Computer Science at the University of Illinois at Chicago where he has been a faculty member since 1997. He received his Ph.D. in Computer Science from the University of California at San Diego in 1996. His research interests lie primarily in combinatorial optimization for EDA including physical design and logic synthesis.



Marco Domenico Santambrogio is an Assistant professor at Politecnico di Milano. He received his laurea (M.Sc. equivalent) degree in Computer Engineering from the Politecnico di Milano (2004), his second M. Sc. degree in Computer Science from the University of Illinois at Chicago (UIC) in 2005 and his PhD degree in Computer Engineering from the Politecnico di Milano (2008). Dr. Santambrogio was a postdoc fellow at CSAIL, MIT, and he has also held visiting positions at the Department of Electrical Engineering and Computer Science of the Northwestern University (2006 and 2007) and Heinz Nixdorf Institut (2006). Marco D. Santambrogio is a senior member of the IEEE and ACM. He has been with the NECST Lab at the Politecnico di Milano, where he founded the Dynamic Reconfigurability in Embedded System Design (DRESD) project in 2004 and the CHANGE (self-adaptive computing system) project in 2010. He conducts research and teaches in the areas of reconfigurable computing, self-aware and autonomic systems, hardware/software co-design, embedded systems, and high performance processors and systems.