Floorplanning for Partially-Reconfigurable FPGA Systems via Mixed-Integer Linear Programming

Marco Rabozzi[1], John Lillis[2], Marco D. Santambrogio[1]
[1]Politecnico di Milano, Milan, Italy
[2]University of Illinois at Chicago, Chicago, USA
marco.rabozzi@mail.polimi.it, lillis@uic.edu, marco.santambrogio@polimi.it

***Abstract*—The aim of this paper is to show a novel floorplanner based on Mixed-Integer Linear Programming (MILP), providing a suitable formulation that makes the problem tractable using state-of-the-art solvers. The proposed method takes into account an accurate description of heterogeneous resources and partially reconfigurable constraints of recent FPGAs. A global optimum can be found for small instances in a small amount of time. For large instances, with a time limited search, a 20% average improvement can be achieved over floorplanners based on simulated annealing. Our approach allows the designer to customize the objective function to be minimized, so that different weights can be assigned to a linear combination of metrics such as total wire length, aspect ratio and area occupancy.**

I. Introduction

Floorplanning [1] using Partial Reconfiguration (PR) on modern FPGAs requires the designer to take into account the resources needed by each reconfigurable region with respect to different resource types [2], satisfying at the same time placement constrains specific for PR [3]. In the last few years, several solutions have been presented to automate floorplanning ([4]–[12]). However, the only ones that accurately consider the collocation of heterogeneous resources within the device and take into account the constraints for PR are [4] and [5].

The approach proposed in [4] stems from Parquet [13], which is the state-of-the-art fixed-outline floorplanner for VLSI design, and copes with partial-reconfigurable heterogeneous FPGAs. This algorithm has been proven to give better results in terms of total wire length with respect to [6] and [9], both based on simulated annealing. On the other hand, the work described in [5] is a two steps optimization algorithm. The first step tries to reduce the amount of wasted resources using a method called Columnar Kernel Tessellation, while the second step locally improves the total wire length without changing the overall occupied area. As stated in [5] this approach provides a better floorplan in term of resource usage with respect to [6].

We propose two novel MILP based algorithms, called [HO] (Heuristic-Optimal) and [O] (Optimal). They improve the value of the objective function with respect to [4] and [5] while still satisfying the constraints for PR. Our algorithms let the designer decide to what extent optimize each term in a set of different metrics. We provide a comparison with previous works using total wire length and wasted resources as the metrics of choice.

Within this context we may summarize our contributions as follows:

- we provide a MILP formulation in [O], that can be used to solve the problem to optimality or within a certified gap from the optimum;
- we guarantee the PR constraints satisfaction and give a simple technique to fully characterize an FPGA device;
- our algorithms let the designer decide which metrics to consider for the optimization process and to what extent. Moreover, the flexibility of the formulation gives the user the possibility to define other metrics different from the ones presented in this paper;
- finally, a simplified MILP model defined for [HO] can be used to locally improve the goodness of a solution obtained from a heuristic such as [4] or [5].

The remainder of this paper is organised as follows: Section II discusses, at the best of our knowledge, the related work in the area, Section III presents the description of the problem and the proposed approach, Section IV provides a suitable device characterization used within the MILP models, Section V describes the proposed MILP formulation used for [HO] and [O], Section VI discusses the obtained results and Section VII concludes the the paper.

II. Related Work

In literature, various floorplanners for FPGAs have been proposed. A first class of algorithms focuses on static placement such as [8] and [9]. The solution proposed in [8] considers device models in which the resources are homogeneously distributed, restricting the validity of the approach to less recent heterogeneous FPGAs. The method proposed in [9] is a two steps algorithm based on the formulation introduced in [8]. The first step exploits a fixed outline simulated annealing, augmented with a penalty term in the objective function to address the violation of resource requirements. The second step, by means of a Min-Cost Max-Flow formulation, modifies the rectangular shape of the previously placed modules to guarantee the feasibility of the solution. Unfortunately, the final result of the algorithm unlikely meets the PR constraints as required by [3].

Another class of approaches introduces reconfiguration and takes into account the time domain. One of the most important

TABLE I
COMPARISON OF FLOORPLANNERS FEATURES

	[8]	[9]	[10]	[11]	[12]	[6], [7]	[4], [5]	[HO]	[O]
Resource distribution-aware		✓					✓	✓	✓
Reconfiguration-aware			✓	✓	✓	✓	✓	✓	✓
Compliant with PR						✓	✓	✓	✓
Optimize interconnections	✓	✓	✓	✓	✓		✓	✓	✓
Considers IO pins						✓	✓	✓	✓
Customizable objective function								✓	✓
Reaches the optimum									✓

contributions in this area is [10]. However, the types of resources considered by their methodology are restricted to logic blocks while other resources are ignored. Other works such as [11] and [12] aim at reconfiguring the smallest amount of the system between two subsequent configurations. Their approaches, as stated in [4], are not compliant with PR.

Among the works in this category, [7] and its enhancement [6] are worth mentioning. The work proposed in [6] consists of two main steps: firstly, each task that has to be reconfigured on the device is assigned to a reconfigurable region while trying to minimize the variance of different used resources over time (temporal floorplacement). Subsequently, each reconfigurable region, defined by the maximum number of requested heterogeneous resources over time, is placed on the device. This step uses a simulated annealing based algorithm whose moves satisfy the PR constraints. However, this last step considers the different resources as homogeneously distributed within the device and does not take into account the complex structure of modern FPGAs.

Both PR constraints and an accurate description of the heterogeneous resource distribution are considered in [4]. The algorithm is based on simulated annealing and explores a reduced search space through an indirect representation of the solution. It exploits the well known *sequence pair* [14], augmented with a vector representing the height of each reconfigurable region, whose possible values are selected according to PR constraints on the device of choice. Firstly, [4] obtains the coordinates of the reconfigurable regions, ensuring the geometric relation between modules as stated in the sequence pair; secondly, it fixes the height according to the height vector and finally, greedy widens and moves the regions on the x axis to meet the resource requirements. Since not all the possible combinations of sequence pairs and height vectors lead to a feasible solution, [4] also implements a constraint violation term in the objective function, to guide the solver in critical situations. Moreover, the algorithm is able to detect free spaces and perform smart moves to recover from solutions in which not all the resource requirements are satisfied.

The indirect representation of a solution in [4], described by means of its sequence pair and height vector, can be, in general, mapped to a vast number of different placements on the device. The fast greedy approach used to widen and move the regions finds, in general, a sub-optimal placement that satisfies the resource requirements and meets the restriction imposed by the indirect representation. Indeed the mapping from an indirect to a direct representation has to be performed at each iteration of the simulated annealing, thus, it is not convenient to solve the mapping to optimality.

The work presented in [5], similarly to [4], produces floorplans that satisfy PR constraints and takes into account non homogeneous resource distributions. Their approach characterizes the FPGA device in terms of minimal reconfigurable units called tiles. Each tile contains a specific type and number of resources and consists of several configurable frames. Hence, the reconfigurable regions requirements are translated in terms of tiles requirements with some unavoidable resource overhead due to inexact divisions. The reconfigurable regions are assigned a priority based on the types and number of required tiles and are placed sequentially starting from those using rarer tiles such as DSP and BRAM tiles. The placing is performed by merging adjacent tiles on the same row to form kernels that contain at least some instances of the type of needed tiles. The smallest kernel in terms of configurable frames is selected and vertically extended to meet the region requirements. If other tiles, different from the rarest are required, the region is then extended horizontally trying to satisfy the region needs. The process is repeated several times using different kernels for packing. At the end of each iteration some post processing is performed on the columnar direction to locally improve the total wire length without changing the shapes of the regions. The best outcome with respect to an objective function is then considered as the solution.

The fixed schedule of the regions and the greedy tile packing procedure may result in completely unexplored and potentially promising solutions of the search space. This issue is partially mitigated restarting the algorithm several times with different kernels, but still, there is no guarantee on the goodness of the solution found with respect to the optimal one.

Table I summarizes the capabilities of existing floorplanners together with our approaches [HO] and [O].

III. PROBLEM DESCRIPTION AND PROPOSED APPROACH

The floorplanning problem is defined by means of a set of reconfigurable regions with their resource requirements (i.e., number of DSPs, BRAMs, CLBs, ...), the desired FPGA device and the objective function that needs to be optimized. The FPGA can be seen as a grid where each point, described by its Cartesian integer coordinates, contains one, none or more than one resource. Moreover, since we are interested in reconfiguring regions on the FPGA, we also have to consider the technological constraints of the specific circuit when partial reconfiguration takes place. The reconfiguration process involves a minimal reconfiguration unit, from now on referred as a *tile*. A solution to the floorplanning problem gives, for each region, the coordinates of the bottom left corner, the width and the height with respect to the device grid. A valid solution of the problem must ensure that (i) for each region, all resource requirements are satisfied (ii) there is no overlapping between two different regions (iii) the regions are placed into rectangular areas of the FPGA that include complete tiles (PR requirement). The latter constraint avoid incomplete tile boundaries, that would require extra circuitry and increased latency to modify, read and write configuration information [5]. Depending on the device, a tile has a different size in terms of CLB resources (on a Xilinx [15] Virtex-5 XC5VLX110T it spans 20 CLBs height and 1 CLB width). The width and height of a tile are respectively denoted by $frameW$ and $frameH$.

Although [4] and [5] have been proven to give better performance with respect to previous approaches, they still look for a solution in a sub-optimal or incomplete search space respectively. We propose two approaches based on a suitable MILP formulation that overcome these issues and give better results in terms of the objective function, at the cost of a generally higher execution time.

The first algorithm that we propose, [HO], improves the quality of a solution produced by a heuristic such as [4] or [5]. [HO] selects one or more good solutions from a heuristic, then, for each solution found, considers the sequence pair representation of the floorplan. Each sequence pair is used within a different MILP formulation to fix the geometric relations between reconfigurable regions. Finally, each instance is solved quickly using a state-of-the-art solver such as [16], and the best outcome is considered. [HO] can give good improvements of the solution with a small overhead in terms of execution time. This approach is well suited for heuristics that already consider the sequence pair representation such as [4].

The second approach that we propose, [O], describes the entire problem using a MILP formulation that ensures non overlapping of reconfigurable regions without the need of a fixed sequence pair. [O] is able to explore the full solution space of the problem and can in general give the optimal solution for small instances. When the solver is faced with big instances, it can be warm started using the solution achieved by [HO] or from a different algorithm. If the instance is fairly hard, after a fixed amount of time the search can be stopped and the best solution found is retrieved. [O] gives better results than [HO] but is in general quite time consuming; hence the designer can select which of the two algorithms to adopt depending on his needs.

IV. Device Characterization

In this section we provide a characterization of the FPGA device that eases the MILP formulation, reducing as much as possible the need of integer variables that would make the problem hard to solve.

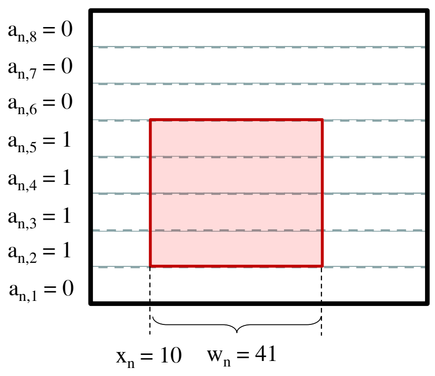

Fig. 1. Variables values of a reconfigurable region placed within the device

A. Grid reduction

The first step toward the simplification of the problem is to reduce the device grid granularity as done in [5]. Instead of considering the single slice resource on the chip, we can just take into account, without loss of generality, the tiles as the minimal area units. Using a grid whose integer coordinates address tiles reduces both the solution space and guarantees PR constraints, since regions are placed using integer coordinates.

B. Problem linearization

Floorplanning is intrinsically a non linear problem, since we have to ensure that each reconfigurable region occupies a two dimensional area large enough to include all the required resources. As a second step, to linearize the problem, we need to discretize one of the two axes. In general, the grid obtained after the previous step is much larger on the x axis and has only a few possible values on the y axis. As an example, the Xilinx Virtex-5 XC5VLX110T can be described using a *grid of tiles* with 8 rows and 62 columns. This suggests to discretize the device on the y axis. Figure 1 shows how the *linearization* process is performed: for each reconfigurable region like the one shown in the figure in red, a set of binary variables indexing the rows are used to state if the region occupies a specific row. On the x axis instead, a couple of integer variables representing the leftmost position and the width of the region suffice.

C. FPGA partitioning

The last step needed to fully characterize the device is to define for each tile the number and type of resources available. Fortunately, there is no need to consider all the tiles separately: even though different resources are available in different locations of the chip, the FPGA structure is quite regular and there are big areas characterized by the same *type* of tile. Two tiles are of the same *type* if they have the same amount and type of resources (e.g., two tiles having both 20 CLBs, 2 BRAMs and no other resources are of the same type). The FPGA can be partitioned into several rectangular areas named *portions*. All the tiles within a portion are required to be of the same type. A simple technique that can be used to create the FPGA partitioning is the following:

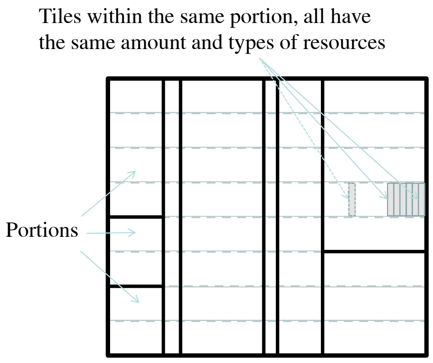
Tiles within the same portion, all have the same amount and types of resources
Portions

Fig. 2. Partitioning of the FPGA into portions

1) the FPGA is scanned top to bottom, left to right and the first tile that is still not part of any portion (free tile) is selected, then a new portion is created containing that tile;
2) the portion is extended on the right side until other free tiles of the same type are encountered;
3) the portion is extended on the bottom side until all the tiles on the row below the portion are free and of the same type;
4) if there are still free tiles, the process is repeated from step 1 until all the tiles are part of one and only one portion.

An example of the result of *FPGA partitioning* is shown in Figure 2. To give an idea, 20 portions are enough to correctly characterize a Virtex-5 XC5VLX110T in terms of DSPs, BRAMs and CLBs resources. Notice that non purely columnar partitions are also possible, indeed hard processors and transceivers may break the contiguity of a column. If the reconfigurable regions are not allow to intersect a specific portion, the portion is said to be in the set of *forbidden areas*.

Even though partitioning resembles the kernel construction shown in [5], the two processes should not be confused. Here partitioning is used as a mean to describe the FPGA device, it is not related to the placement of the reconfigurable regions. Moreover, portions must contain only tiles of the same type as opposed to kernels that can include different types of tiles.

V. MILP Formulation

In this section we present the MILP models used by [HO] and [O] algorithms. The two models are quite similar and differ only in the definition of the non overlapping constraints. If not explicitly specified, all the parameters, variables and constraints defined in this section apply to both the formulations. Since the models are fairly big to be described, we first introduce the constants, variables and constraints related to the description and feasibility of a solution; then, in the subsection dedicated to the objective function, we add the real variables, parameters and constraints that are solely needed to define the solution cost.

A. Constants definition

An FPGA can be fully described by means of the portions in which it has been divided. Each portion is described in turn by its position and its type of tiles. Furthermore, we also know the number of reconfigurable regions to place along with their resource requirements. What follows are the sets and parameters of the model:

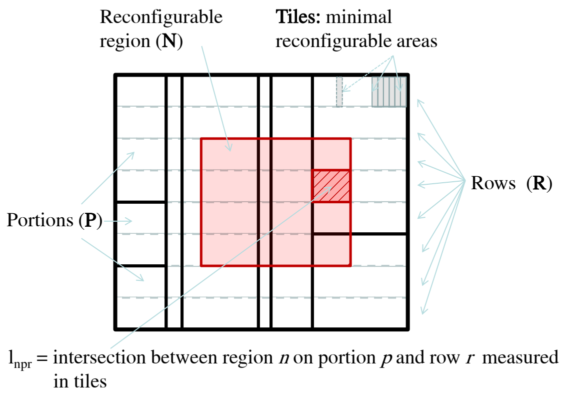
Reconfigurable region (**N**)
Tiles: minimal reconfigurable areas
Rows (**R**)
Portions (**P**)
l_{npr} = intersection between region n on portion p and row r measured in tiles

Fig. 3. Computation of covered resources

$P :=$	set of portions;
$F :=$	set of forbidden areas (F $\subset$ P);
$R :=$	set of rows of the FPGA numbered from 1 to $\lvert R \rvert$;
$N :=$	set of reconfigurable regions to place;
$T :=$	set of resource types considered (CLB, DSP, etc.);
$c_{n,t} :=$	resources of type t required by reconfigurable region n;
$rp_{p,r} :=$	1 if portion p lies on row r, 0 otherwise;
$x1_p :=$	leftmost position of a tile in portion p;
$x2_p :=$	rightmost position of a tile in portion p;
$d_{p,t} :=$	number of resources of type t available in a tile within portion p;
$maxW :=$	maximum value on the x axis.

If the algorithm [HO] is used, we also need to specify the sequence pair describing the geometrical relation between the regions. To do so, we define the following parameters:

$pair1_n :=$	defines for a region n its index within the first sequence of the pair;
$pair2_n :=$	defines for a region n its index within the second sequence of the pair.

B. Variables identification

What we need to compute are the position and dimensions of each reconfigurable region. In order to be able to properly state the resource occupancy constraints, we also need some support variables that define the amount of intersection, in terms of tiles, between a region n on a portion p and row r (Figure 3).

What follows are the variables shared by both formulations for [HO] and [O]:

$a_{n,r} :=$	binary variable set to 1 if and only if region n occupies row r;
$x_n :=$	integer positive variable (≥ 1) representing the left-most position of region n;

$w_n :=$ integer positive variable (≥ 1) representing the width of region n. A value of 1 means that only the tiles at x_n can be covered by the region;

$yl_n :=$ real non negative variable (≥ 0) denoting the lowest row occupied by region n;

$yh_n :=$ real non negative variable (≥ 0) denoting the highest row occupied by region n;

$h_n :=$ real non negative variable (≥ 0) denoting the height of region n;

$l_{n,p,r} :=$ real non negative variable (≥ 0) defining the amount of intersection, in terms of tiles, between region n on portion p and row r such that $rp_{p,r} = 1$;

$k_{n,p} :=$ binary variable set to 0 if the projections on the x axis of a region n and a portion p do not intersect (i.e., the region is to the right or to the left with respect to the portion).

Even though the variables $yl_n, yh_n, h_n, l_{n,p,r}$ have been declared as real, the constraints of the model ensure the integrity of their values. This gives an advantage to the solver that will not branch on real variables.

The MILP model of [O] requires another set of binary variables in order to avoid overlapping between regions. This is the reason why the MILP model of [O] is much harder to solve than the one for [HO], but it can explore a much bigger search space and find better solutions. The number of these extra variables is quadratic with respect to the reconfigurable regions to place and can be computed as $|N| \cdot (|N| - 1)/2$. Considering a total ordering of the regions in N, for every $n1, n2 \in N$ such that $n1 < n2$, we introduce:

$g_{n1,n2} :=$ binary variable forced to 1 if region $n1$ is not to the left of region $n2$;

C. Semantic constraints

Before stating the requirements strictly related to our problem, we first introduce the constraints that ensure the *soundness* of the semantics of variables.

Rows contiguity. Ensures that if a region n occupies rows $r1$ and $r3 > r1$, then it also occupies the rows between $r1$ and $r3$:

$$\begin{aligned} &\forall n \in N, r1, r2, r3 \in R \mid r3 > r2 > r1 : \\ &a_{n,r2} \geq a_{n,r1} + a_{n,r3} - 1 \end{aligned} \tag{1}$$

Consistency of the definition of variables $k_{n,p}$:

$$\begin{aligned} &\forall n \in N, p \in P : \\ &x_n + w_n - 1 \geq x1_p \cdot k_{n,p} \\ &x_n \leq x2_p + (1 - k_{n,p}) \cdot (maxW - x2_p) \end{aligned} \tag{2}$$

Limit of the right side of the regions on the x axis:

$$\forall n \in N : x_n + w_n \leq maxW \tag{3}$$

Height definition with respect to the single rows occupied:

$$\forall n \in N : h_n = \sum_{r \in R} a_{n,r} \tag{4}$$

Vertical positions bounds. Constrains variable yl_n to be not greater than the lowest row and yh_n to be not smaller than the highest row occupied by region n:

$$\begin{aligned} &\forall n \in N, r \in R : yl_n \leq |R| - a_{n,r} \cdot (|R| - r) \\ &\forall n \in N, r \in R : yh_n \geq a_{n,r} \cdot r \end{aligned} \tag{5}$$

Vertical gap. Fixes the gap between yl_n and yh_n with respect to the height of the region. This constraint together with (5), sets the variables to the correct integer values:

$$\forall n \in N : yh_n - yl_n + 1 = h_n \tag{6}$$

Intersection upper bounds. Constrains the intersection between a region n on a portion p and row r to be not greater than expected in all the possible cases:

$$\begin{aligned} &\forall n \in N, p \in P, r \in R \mid rp_{p,r} = 1 : \\ &l_{n,p,r} \leq a_{n,r} \cdot (x2_p - x1_p + 1) \\ &l_{n,p,r} \leq k1_{n,p} \cdot (x2_p - x1_p + 1) \\ &l_{n,p,r} \leq w_n \\ &l_{n,p,r} \leq x_n + w_n - k_{n,p} \cdot x1_p \\ &l_{n,p,r} \leq x2_p - x_n + 1 + (1 - k_{n,p}) \cdot (maxW - x2_p) \end{aligned} \tag{7}$$

the inequalities in (7) have the meaning (same order):

1) No tiles are covered if the row is not occupied by the region. The constraint implies also that the covered area cannot exceed the availability of the portion row;
2) No tiles are covered if the region is on the left or on the right of the portion;
3) The covered tiles cannot exceed the width of the region;
4) If the region intersects the portion on the left side, no more than the tiles on a row between the right side of the region and the left side of the portion can be covered;
5) If the region intersects the portion on the right side, no more than the tiles on a row between the right side of the portion and the left side of the region can be covered;

Intersection lower bound. Ensures that the overall intersection between a region n over a row r occupied by the region covers at least w_n tiles (this constraint, together with (7), fixes the amount of intersection to the correct integer value):

$$\begin{aligned} &\forall n \in N, r \in R : \\ &\sum_{p \in P \mid rp_{p,r} = 1} l_{n,p,r} \geq w_n - (1 - a_{n,r}) \cdot maxW \end{aligned} \tag{8}$$

Forbidden areas constraint. Ensures no intersection with a portion in the set of forbidden areas:

$$\begin{aligned} &\forall n \in N, p \in F, r \in R \mid rp_{p,r} = 1 : \\ &l_{n,p,r} = 0 \end{aligned} \tag{9}$$

For the MILP formulation of [O] we also need to guarantee the semantics of variables $g_{n1,n2}$:

$$\begin{aligned} &\forall n1, n2 \in N \mid n1 < n2 : \\ &x_{n1} + w_{n1} \leq x_{n2} + g_{n1,n2} \cdot maxW \end{aligned} \tag{10}$$

D. Problem constraints

After having guaranteed the correct meaning of each variable, we can move on and define the constraints that are tightly coupled with the problem.

Ensures that each reconfigurable region covers all the needed resources for each resource type:

$$\forall n \in N, t \in T : \quad \sum_{p \in P, r \in R | rp_{p,r}=1} l_{n,p,r} \cdot d_{p,t} \geq c_{n,t} \tag{11}$$

The non overlapping constraints for the MILP formulation in [HO] are defined by means of the sequence pair obtained after the execution of a heuristic:

$$\forall n1, n2 \in N \mid pair1_{n1} < pair1_{n2} \wedge pair2_{n1} < pair2_{n2} : \quad x_{n1} + w_{n1} \leq x_{n2} \tag{12}$$

$$\forall n1, n2 \in N \mid pair1_{n1} < pair1_{n2} \wedge pair2_{n1} > pair2_{n2} : \quad yl_{n1} \geq yl_{n2} + h_{n2} \tag{13}$$

Instead, regarding [O], there are no fixed geometrical relations between the reconfigurable regions. The non overlapping constraints exploit the variables $g_{n1,n2}$ and, in this case, are expressed saying that two regions cannot overlap on the same row:

$$\forall r \in R, n1 \in N, n2 \in N \mid n1 < n2 : \quad x_{n1} \geq x_{n2} + w_{n2} - (3 - g_{n1,n2} - a_{n1,r} - a_{n2,r}) \cdot maxW \tag{14}$$

E. Objective function

The objective function to be minimized in both models is a linear combination of different metrics. A weight can be assigned to each cost component, depending on the designer needs. We consider the following cost functions:

Global wire length (WL_{cost}): an estimation of the overall wire length measured using the commonly adopted half-perimeter wire length (HPWL) as in [4]. Both connections to IOs and between reconfigurable regions are taken into account. HPWL assumes the pins concentrated in the center of the regions/IO ports, so the wire length of a connection is estimated with the Manhattan distance between the centroids of the components weighted by the interconnection width;

Regions perimeter (P_{cost}): is the sum of all the regions perimeters. This cost penalizes those reconfigurable regions that differ much from a squared shape;

Wasted resources (R_{cost}): is the difference between the resource occupied by the regions and their real requirements. A weight can be also associated to each resource type, considering for instance the rareness or importance of the resource type.

The objective of our models can be written as:

$$\min \left\{ q_1 \cdot \frac{WL_{cost}}{WL_{max}} + q_2 \cdot \frac{P_{cost}}{P_{max}} + q_3 \cdot \frac{R_{cost}}{R_{max}} \right\} \tag{15}$$

where q_1, q_2 and q_3 are the user defined weights. WL_{max}, P_{max} and R_{max} represent the maximum values that the related cost functions can assume and are used to normalize each component.

To compute the value for WL_{cost} we need to define some additional parameters:

$IO :=$ set of the interconnections to the IO ports. Each element of the set is described by a 4-dimensional tuple of the form: (n, iox, ioy, b) where n is the reconfigurable region connected to the IO, iox and ioy represent the coordinates of the IO centroid with respect to the original device grid before the reduction, while b is the interconnection width;

$C :=$ set of the interconnections between reconfigurable regions. Each element is a 3-dimensional tuple of the form: $(n1, n2, b)$ where $n1$ and $n2$ are the regions involved in the interconnections and b is its width;

The following variables are also defined to compute the required Manhattan distances:

$(cx_n,\ cy_n) :=$ real variables representing the (x, y) coordinates of region n centroid;

$(dcx_{n1,n2},\ dcy_{n1,n2}) :=$ real variables representing the distances between centroids of regions $n1$ and $n2$ on x and y axes;

$(dpx_{io},\ dpy_{io}) :=$ real variables representing the distances on x, y axes between centroids of region n and the IO port defined in $io = (n, iox, ioy, b) \in IO$;

To guarantee the semantics of the previously defined variables, we have to state the following constraints.
First of all, we have to compute the regions centroids with respect to the original device grid:

$$\begin{aligned} &\forall n \in N : \\ &cx_n = frameW \cdot (x_n + w_n/2) \\ &cy_n = frameH \cdot (yl_n - 1 + h_n/2) \end{aligned} \tag{16}$$

Secondly, we ensure that the Manhattan distance between two regions centroids is not less than the correct value (notice that there is no need to give an exact assignment because the distances are going to increase the solution cost to be minimized):

$$\begin{aligned} &\forall n1 \in N, n2 \in N \mid n1 \neq n2 : \\ &dcx_{n1,n2} \geq cx_{n1} - cx_{n2} \\ &dcx_{n1,n2} \geq cx_{n2} - cx_{n1} \\ &dcy_{n1,n2} \geq cy_{n1} - cy_{n2} \\ &dcy_{n1,n2} \geq cy_{n2} - cy_{n1} \end{aligned} \tag{17}$$

Finally, the Manhattan distance between the centroids of a region and its IO connection has to be not less than the correct value (as before there is no need for an exact assignment):

$$\begin{aligned} &\forall io = (n, iox, ioy, b) \in IO : \\ &dpx_{io} \geq cx_n - iox \\ &dpx_{io} \geq iox - cx_n \\ &dpy_{io} \geq cy_n - ioy \\ &dpy_{io} \geq ioy - cy_n \end{aligned} \tag{18}$$

Now we are ready to compute the value of WL_{cost} as the sum of the wire length of the IO connections and the internal connections between reconfigurable regions:

$$\begin{aligned} WL_{cost} = &\sum_{(n1,n2,b)\in C} ((dcx_{n1,n2} + dcy_{n1,n2}) \cdot b) + \\ &\sum_{io=(n,iox,ioy,b)\in IO} ((dpx_{io} + dpy_{io}) \cdot b) \end{aligned} \tag{19}$$

Instead, the value of P_{cost} does not require extra variables and can be simply computed as:

$$P_{cost} = 2 \cdot \sum_{n\in N} (w_n \cdot frameW + h_n \cdot frameH) \tag{20}$$

To compute the last metric R_{cost}, let's denote with rc_t the user defined penalty that is given if a resource of type t is wasted (i.e. is covered by a reconfigurable region but not required), then, R_{cost} can be written as:

$$R_{cost} = \sum_{n\in N, t\in T} waste_{n,t} \cdot rc_t \tag{21}$$

where:

$$waste_{n,t} := \sum_{p\in P, r\in R | rp_{p,r}=1} (l_{n,p,r} \cdot d_{p,t}) - c_{n,t} \tag{22}$$

VI. Results Evaluation

The proposed models [HO] and [O] have been translated using MathProg, a subset of the AMPL language, and solved with Gurobi Optimizer [16]. We compared the results achieved by [HO] and [O] with respect to [4], that already showed better results than [6] and [9]. An extended testing has been performed on a Virtex-5 XC5VLX110T using the global wire length as the metric of choice (i.e. weights of the objective function have been set to: $q_1 = 1.0$, $q_2 = 0.0$, $q_1 = 0.0$). We generated a set of pseudo-random circuits with a number of reconfigurable regions in the range [5, 10, 15, 20, 25] and, for each value in the range, 4 circuits have been generated having an occupancy rate of device slices of 70%, 75%, 80% and 85%. In each circuit, we ensured that from 3 to 7 reconfigurable regions required BRAMs, while from 1 to 2 required DSPs.

We performed 10 executions of [4] on every circuit and the best result has been considered for comparison. [HO] instead, selected the executions of [4] that did not differ by more than 10% from the considered [4] solution. For each sequence pair in the selected solutions, [HO] re-optimized the problem and the best outcome represented the final result of [HO]. Concerning [O], we decided to warm start the solver using the best solution found by [HO] and limit the searching time to 1800 seconds. This because, in case of big instances, the solver had difficulties to find an initial solution and the overhead to compute a solution using [HO] still was less than starting the solver from scratch. Since [HO] relies on [4], the overall execution time of [HO] takes also into account the time spent by [4] to solve the problem. The same is true for [O] since we provided an initial solution computed by [HO]. Notice that [O] does not require [4] or any other algorithm to solve the problem, however a good initial solution can give a good speed up in terms of execution time. The tests have been executed on a 2.2GHz Intel Core Duo processor, performing in parallel the 10 executions of [4] and enabling multi-threading in Gurobi Optimizer to solve the MILP models.

Table II reports the average wire length reduction achieved by the two approaches with respect to [4].

TABLE II
Algorithms execution with different numbers of Reconfigurable Regions (RRs)

# RRs	Average wire length improvement w.r.t. [4]		Average execution time (sec)		
	[HO]	[O]	[4]	[HO]	[O]
5	6.99%	7.48%	10.9	17.2	139.7
10	7.56%	12.05%	23.9	113.8	1913.9
15	8.83%	19.46%	40.6	143.2	1943.2
20	5.47%	19.98%	65.0	124.4	1924.4
25	5.47%	21.42%	93.5	178.2	1978.2

Execution times of all the algorithms are also shown. For small numbers of reconfigurable regions [HO] gives an improvement comparable to [O] but using much less time. It was interesting to notice that with 5 reconfigurable regions, in 3 out of 4 instances [O] proved the optimality of the solution found by [HO], while in the remaining circuit another 2% was gained. When the problems become more complex and a higher number of reconfigurable regions need to be placed on the device, the gap between [4] and [O] is remarkable and on average a 20% reduction is provided within an acceptable time. Better results can be achieved by giving more time to the solver, so the designer can trade off the computation time with respect to the desired wire length improvement.

Table III perform a similar analysis but based on the device occupancy rate.

TABLE III
Algorithms execution with different overall device occupancy

Occupancy	Average wire length improvement w.r.t. [4]		Average execution time (sec)		
	[HO]	[O]	[4]	[HO]	[O]
70%	8.44%	19.87%	47.0	165.5	1638.6
75%	5.49%	20.29%	46.9	122.5	1571.6
80%	6.20%	13.33%	46.8	95.6	1560.9
85%	7.32%	10.82%	46.4	77.8	1548.5

The best improvements are achieved for an occupancy rate smaller than 80%, whereas the wire length reduction diminishes as soon as the device gets more occupied and the slacks between regions are also reduced.

Using the same processor we also tested our approach on a real design provided in [5]. The design considered is

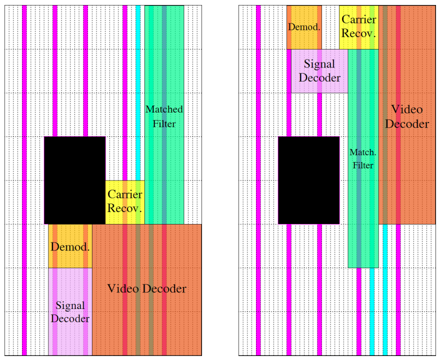

Fig. 4. Floorplans using (a) the algorithm proposed in [5], (b) using [O]

a software defined radio (SDR) that includes the following modules: matched filter, carrier recovery circuit, demodulator, signal decoder and video decoder. In each module different modes requiring different resources are configured one at a time. All the modules are connected in sequential order with a 64 bit wide bus. The target device is a Virtex-5 FX70T that contains three different type of tiles: CLB tile, BRAM tile and DSP tile consisting of 36, 30 and 28 configurable frames respectively.

The resource requirements of the reconfigurable regions are taken from [5] and considered in terms of tiles. We compared our approach with respect to the floorplan performed in [5] when the goal was to optimize the number of wasted frames and secondly to reduce the wire length without modifying the main objective. It was easy to set up the parameters of the objective function of our MILP model to achieve the same type of optimization: $q_1 = 1.0$, $q_2 = 0.0$, $q_3 = R_{max}$. With this settings the wire length cost component produce real positive values no greater than 1, while the wasted resource component takes only integer values. Optimizing this cost function means finding the floorplan having the lowest possible resource wastage and, among all these optimal solutions, finding the one that gives the lowest wire length. We also set the parameter rc_t for t in $\{CLB, DSP, BRAM\}$ to properly consider the number of frames per tile. The floorplan shown in [5] achieved 466 wasted frame, while the one obtained after the execution of [O] provided 306 wasted frame, an area reduction of roughly 34% maintaining a similar overall wire length. The optimal solution was found by [O] in approximately 29 seconds, however more than a hour was needed to prove its optimality. Both floorplans are shown in Figure 4.

In general our approaches are suitable in two scenarios: (i) when the number of reconfigurable regions to place is high, so that an advantage over simulated annealing can be gained by exploring more deeply the solution space; (ii) when the occupancy rate of the device is not so high and a good improvement can be gained over the greedy placement performed in [4] and [5].

VII. Conclusions

This paper presented two new approaches to automate floorplanning for partial reconfigurable FPGAs. They are based on a Mixed-Integer Linear Programming model that allows a deep exploration of the solution space using state-of-the-art solvers. The results are compliant with PR requirements and a detailed characterization of current and future device can be easily handled using the FPGA partitioning technique.

The proposed floorplanners allows the designer to perform a quick local improvement from a first heuristic solution, or to search the entire solution space for better results. The optimization process is guided by a customizable objective function able to consider different metrics, so that more control is given over the kind of desired solutions.

Future work will extend the proposed model to take into consideration also other metrics, furthermore the proposed algorithms will be integrated in a comprehensive flow were it will be combined with a scheduler for partial dynamic reconfigurable FPGA-based system. Finally, ongoing works are also investigating the possibility of extending the proposed solution to support bitstreams relocation at runtime.

Acknowledgments

This work was partially funded by the European Commission in the context of the FP7 FASTER project [17](#287804).

References

[1] L. Cheng and M. D. F. Wong, “Floorplan Design for Multimillion Gate FPGAs,” *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 25, no. 12, pp. 2795–2805, 2006.
[2] Xilinx Inc, “Floorplanning Methodology Guide ,” 2010.
[3] ——, “Partial Reconfiguration User Guide,” 2010.
[4] C. Bolchini, A. Miele, and C. Sandionigi, “Automated Resource-Aware Floorplanning of Reconfigurable Areas in Partially-Reconfigurable FPGA Systems,” in *FPL*, 2011, pp. 532–538.
[5] K. Vipin and S. A. Fahmy, “Architecture-aware reconfiguration-centric floorplanning for partial reconfiguration,” in *ARC*, 2012, pp. 13–25.
[6] A. Montone, M. D. Santambrogio, and D. Sciuto, “Wirelength driven floorplacement for FPGA-based partial reconfigurable systems,” in *IPDPS Workshops*, 2010, pp. 1–8.
[7] A. Montone, M. D. Santambrogio, D. Sciuto, and S. O. Memik, “Placement and Floorplanning in Dynamically Reconfigurable FPGAs,” *TRETS*, vol. 3, no. 4, p. 24, 2010.
[8] L. Cheng and M. D. F. Wong, “Floorplan design for multi-million gate FPGAs,” in *ICCAD*, 2004, pp. 292–299.
[9] Y. Feng and D. P. Mehta, “Heterogeneous Floorplanning for FPGAs,” in *VLSI Design*, 2006, pp. 257–262.
[10] P.-H. Yuh, C.-L. Yang, and Y.-W. Chang, “Temporal floorplanning using the three-dimensional transitive closure subGraph,” *ACM Trans. Design Autom. Electr. Syst.*, vol. 12, no. 4, 2007.
[11] L. Singhal and E. Bozorgzadeh, “Multi-layer Floorplanning on a Sequence of Reconfigurable Designs,” in *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] S. N. Adya and I. L. Markov, “Fixed-outline floorplanning: enabling hierarchical design,” *IEEE Trans. VLSI Syst.*, vol. 11, no. 6, pp. 1120–1135, 2003.
[14] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, “VLSI module placement based on rectangle-packing by the sequence-pair,” *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 15, no. 12, pp. 1518–1524, 1996.
[15] Xilinx Inc, http://www.xilinx.com/tools/planahead.htm.
[16] Gurobi Optimization Inc, “Gurobi optimizer.” [Online]. Available: http://www.gurobi.com/download/gurobi-optimizer
[17] Faster website. [Online]. Available: http://www.fp7-faster.eu/