2015 IEEE International Parallel and Distributed Processing Symposium Workshop

Relocation-aware Floorplanning for Partially-Reconfigurable FPGA-based Systems

Marco Rabozzi[1], Riccardo Cattaneo[1], Tobias Becker[2], Wayne Luk[2], Marco D. Santambrogio[1]
[1]Politecnico di Milano, Milan, Italy
[2]Imperial College, London, UK
marco.rabozzi@mail.polimi.it, riccardo.cattaneo@polimi.it, marco.santambrogio@polimi.it,
tbecker@doc.ic.ac.uk, w.luk@imperial.ac.uk

***Abstract*—Within this paper we present a floorplanner for partially-reconfigurable FPGAs that allow the designer to consider bitstream relocation constraints during the design of the system. The presented approach is an extension of our previous work on floorplanning based on a Mixed-Integer Linear Programming (MILP) formulation, thus allowing the designer to optimize a set of different metrics within a user defined objective function while considering preferences related directly to relocation capabilities. Experimental results show that the presented approach is able to reserve multiple free areas for a reconfigurable region with a small impact on the solution cost in terms of wire length and size of the configuration data.**

I. Introduction

Within the context of floorplanning for partially-reconfigurable FPGAs [1], bitstream relocation is the capability of moving a task from an area of the FPGA to another one simply by moving the configuration data from the initial location to the corresponding target location. In practice, to perform the relocation of a task it is necessary to change the addresses contained in the partial bitstream and recompute the Cyclic Redundancy Check (CRC) before sending the bitstream to the configuration memory interface of the device [2].

The main benefit of bitstream relocation is to enable design re-use: instead of developing different instances of the same design to meet the constraints at different locations, a single relocation-aware design is able to meet such constraints. Moreover, as FPGA gets larger, it takes longer to reconfigure the entire chip; partial reconfiguration allows fast reconfiguration, since only part of the device would be reconfigured. This capability is exploited by bitstream relocation to deliver rapid changes to a design at run time, while reducing design effort by supporting design re-use at compile time.

There are various techniques available for bitstream relocation in the literature. Within [2] and [3] the authors present techniques for 1D relocation on Xilinx Virtex-2 and Virtex-2 Pro FPGAs. These works provide a communication infrastructure that allow multiple locations for a task on the horizontal direction without compromising the functionality of the system. In order to effectively perform the relocation, the approaches consider identical areas having the same footprint in terms of heterogeneous resources. In this fashion efficient hardware filters can be used to update the configuration data without having to generate different bitstreams for different locations of the same module.

Other important works in this direction are [4] and its enhancement [5] that introduce the BiRF filter. The authors provide both a hardware and a software implementation for the filter and within [5] BiRF is extended to handle also 2D-partial reconfiguration allowing relocation on the vertical direction. The latter approach has been validated on the more recent Xilinx Virtex-4 and Virtex-5 device families.

The work proposed in [6] relaxes the identical areas requirement and gives the possibility to perform relocation among areas that do not necessarily have the same distribution of resources. Given a set of areas involved in the relocation of a module, the main idea is to instruct the placement program to prevent configuration of mismatching resources among the areas. Afterwords the relocation process is performed by a software filter updating the frame addresses and changing the configuration frames of the mismatching logic. This manipulation of the bitstream does not affect the functionality of the module as soon as the configuration data of the routing resources are kept consistent among the areas involved in the relocation.

The floorplaning extension presented here is complementary with respect to the filters aforementioned. Exploiting our methodology the designer can identify areas suitable for task relocation while a bitstream filter such as [5] and [6] or a different technique can be used to effectively perform the migration of the bitstream among the identified areas. In this work we consider a practical scenario in which the areas used for relocation must have the same footprint of heterogeneous resources as in [3] and [5]. This is done to avoid wastage of non usable mismatching resources on one side and to reduce the size of the solution space on the other.

Recently several floorplanner able to consider both a non uniform distribution of heterogeneous resources and the guidelines and constraints for Partial Reconfiguration (PR) [7] have been devised in the literature [8]–[10].

The approaches presented in [8] and [9] focus on two different types of optimization during the exploration of the solution space. The former, by means of a method called Columnar Kernel Tessellation, meanly focuses on reducing the overall amount of wasted resources to minimize the bitstream size, while the latter, exploits simulated annealing to reduce the overall wire length. On the other hand, the two algorithms

978-1-4673-7684-6/15 $31.00 © 2015 IEEE
DOI 10.1109/IPDPSW.2015.52

presented in [10], based on a MILP formulation, allow to improve the quality of the solutions achieved by [9] and [8] at the cost of a generally higher execution time. The first approach called HO (Heuristic Optimal), extracts the sequence pair representation of a first feasible solution and uses it as an additional constraint to reduce the size of the search space so that the initial solution can be locally improved in a small amount of time. The second approach, named O (Optimal), is able to explore the full solution space but, in general, it requires a larger amount of time.

Within this work, we propose an extension of our previous approach [10] adding support for bitstream relocation for both HO and O algorithm. We first propose a revised FPGA partitioning procedure that eases the extension of the model, then, the MILP model at the core of [10] is enhanced to address both constraints and metrics related to bitstream relocation.

The remainder of the paper is organized as follows: Section II gives a description of the problem and the proposed approach, Section III describes the device model and the revised partitioning procedure, Section IV and Section V discusses how we extended the MILP formulation to take into account relocation as a design constraint and metrics respectively, Section VI evaluates the impact of bitstream relocation constraints on a case study and, finally, Section VII presents final observations and remarks.

II. Problem description and proposed approach

In order to extend the MILP formulation to take into account bitstream relocation, we need a description of the FPGA that models all the relevant aspects. The basic block considered in the floorplanner of [10] is a tile, that is the minimal area considered for reconfiguration. A tile is described in terms of the resources that it contains, but we do not have any information about how the resources are located within the tile and which is the mapping between these resources and the configuration memory where the bitstream is loaded. For this reason, we need to strengthen the definition of tile type to address bitstream relocation:

Definition .1. Two tiles are of the same type if they have the same number and types of resources and if the configuration data needed to configure the resources is the same across the two tiles.

Notice that since we have redefined the notion of tile type, also the FPGA partitioning into portion can produce a different result and more portions could be needed to describe the FPGA structure. We recall, that a portion is a fixed rectangular area on the FPGA containing tiles of the same type.

With the new definition of tile type we are now able to define when bitstream relocation is possible. A bitstream can be relocated from one area to another one if the two areas are compatible. Two areas are compatible if they have the same shape, size and relative positioning of tiles of the same type. In this scenario, ideally, a functionality could be relocated from an area to a compatible one simply by changing the frame addresses. Notice however that the communication infrastructure is not considered here and should be carefully planned by the designer so that bitstream relocation could be effectively performed. To clarify the concept of compatible areas, we show in Figure 1 an example of compatible and non compatible areas.

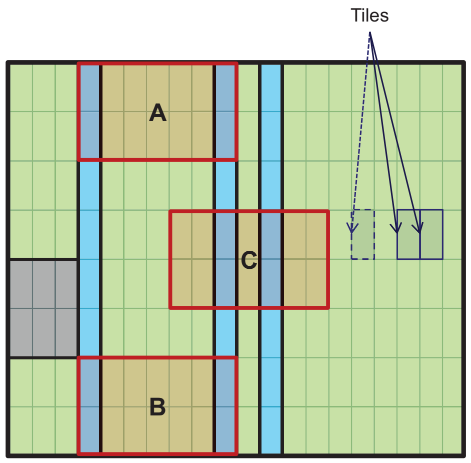


Fig. 1: Example of compatible and non-compatible areas

In Figure 1 the color of a tile identifies its type, tiles of the same color are of the same type. Areas A and B are compatible because they have the same shape, size and their tiles are in the same relative positions (blue and green tiles in the same positions). On the other hand areas A and C are not compatible, indeed, even if they have the same shape, size and cover the same amount of resources, the relative positioning of tiles is not the same (the first column of tiles occupied by region A is blue, while the first column of tiles covered by area C is green). Another important aspect to take into account when bitstream relocation is performed, is that the target area for relocation must not overlap with areas occupied by other tasks to avoid malfunctions. Within this context we introduce the following useful definition:

Definition .2. An area A is said to be free-compatible with respect to another area B, if A and B are compatible and A does not overlap with another free-compatible area or with an area assigned to a reconfigurable region.

Considering Definition .2, that takes also into account areas occupied by other tasks, a bitstream can be relocated from an area to another one if the target area is free-compatible with respect to the source area.

A. On how to consider bitstream relocation

The designer willing to relocate the bitstreams of the tasks assigned to a reconfigurable region needs to identify a free-compatible area into which the configuration data can be moved. There are two ways in which the process of identifying free-compatible areas can be automated by the floorplanner:

Relocation as a constraint: The designer specifies for which reconfigurable regions he/she needs one or more free-compatible areas where the bitstreams of the regions can be relocated. In this context a solution is feasible only if the algorithm can find a placement for all the regions and the corresponding free-compatible areas;

Relocation as a metrics: The designer specifies for each region the maximum number of free-compatible areas he/she wishes to identify. The number of successfully identified free-compatible areas is considered as a metrics within the objective function and it affects the desirability of a solution. This approach is more flexible than the previous one, but does not guarantee the identification of free-compatible areas within a feasible solution.

The two approaches presented above can also be considered together. As an example, the designer may decide to obtain a certain number of free-compatible areas as a constraint, while if extra free-compatible areas are identified, the desirability of the solution increases. In the following sections we are going to provide a description on how to integrate both relocation as a constraint and as a metrics in the floorplanner proposed in [10]. The extension presented can be adopted for both O and HO. The only remark is that when relocation as a constraint is considered in HO, the input heuristic solution should contain, other than the regions placement, also the free-compatible areas positions. In this fashion the sequence-pair is naturally extended to consider also the free-compatible areas, so that the non-overlapping constraints are guaranteed for all the areas.

III. Device model description

Before going on in the details of the MILP model, it is convenient to recall from [10] the sets, parameters and variables that are also referred in this context. Sets and parameters:

$P :=$	set of portions in which the FPGA has been partitioned;
$R :=$	set of rows of the FPGA numbered from 1 to $\|R\|$;
$N :=$	set of reconfigurable regions to place;
$T :=$	set of resource types considered (CLB, BRAM, etc.);
$c_{n,t} :=$	resources of type t required by reconfigurable region n;
$maxW :=$	maximum value on the x axis.

Variables:

$x_n :=$	integer positive variable (≥ 1) representing the leftmost position of region n;
$w_n :=$	integer positive variable (≥ 1) representing the width of 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;
$k_{n,p} :=$	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).

A. Model simplification

In order to introduce bitstream relocation within our methodology, we need to add additional variables and new constraints to the MILP model. Even though this is possible for an arbitrary resource distribution of the FPGA, the problem that is obtained in the general case is quite hard to be solved in the context of [10] MILP formulation. The need to consider both the x and y axes to identify free-compatible areas greatly increases the number of constraints within the formulation, as a result the execution time of the solver increases when the linear programming relaxations are solved during the branch and cut procedure.

To simplify the problem we get rid of one of the two dimensions by addressing FPGAs that can be described in terms of portions extending for the entire FPGA height. This simplification is not practical in cases in which hard processors placed in the middle of the device break the contiguity of a column (e.g. the PowerPC in Virtex-5 FX70T). For this reason, we also allow to define forbidden areas that cannot be crossed by reconfigurable regions and free-compatible areas. The set of the portions, also called columnar portions, is identified by P while the set of forbidden areas is denoted by A. The set F defined in [10] is discarded together with all the constraints related to it. This is done to avoid confusion between the two formulations, indeed A and F are defined in a quite different way. While in [10] the set F is a subset of the set of portion P, here the sets A and P are disjoint. This is done to preserve the FPGA partitioning of set P in which no two portions overlap and all the portions in the set cover the FPGA area entirely. Here the forbidden areas in A overlap with the portions in P. This is an important difference with respect to the FPGA partitioning presented in [10] and we need to define the parameters, variables and constraints of the forbidden areas differently from the ones of the portions. The parameters related to the new forbidden areas are as follows:

$A :=$	set of forbidden areas;
$ra_{a,r} :=$	1 if forbidden area a lies on row r, 0 otherwise;
$xa1_a :=$	leftmost position of a tile in forbidden area a;
$xa2_a :=$	rightmost position of a tile in forbidden area a.

A new set of variables, similar to the one defined for the reconfigurable regions in O, is introduced for both O and HO formulations to ensure non overlapping with forbidden areas:

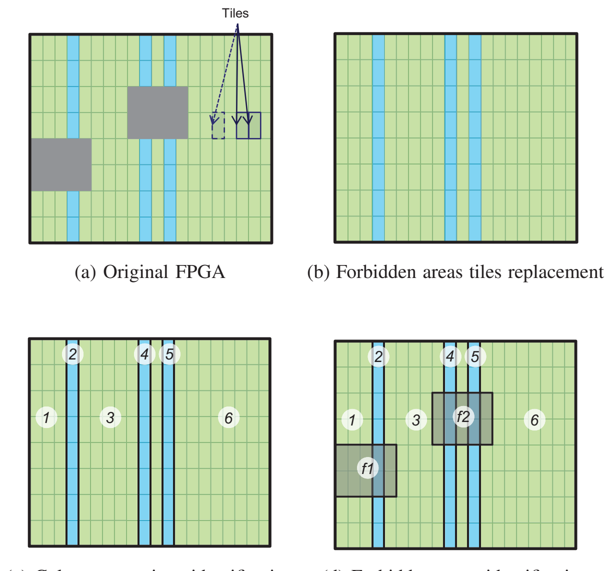
$q_{n,a} :=$	binary variable forced to 1 if region n is not to the left of forbidden area a;

The semantics of variables $q_{n,a}$ is guaranteed by means of the following constraint:

$$\begin{gathered}\forall n \in N, a \in A : \\ x_n + w_n \leq xa1_a + q_{n,a} \cdot maxW\end{gathered} \tag{1}$$

While these are the constraints that ensure non overlapping between reconfigurable regions and forbidden areas:

$$\begin{gathered}\forall n \in N, a \in A, r \in R \mid ra_{a,r} = 1 : \\ x_n \geq xa2_a + 1 - (2 - q_{n,a} - a_{n,r}) \cdot maxW\end{gathered} \tag{2}$$



(a) Original FPGA (b) Forbidden areas tiles replacement

(c) Columnar portions identification (d) Forbidden areas identification

Fig. 2: Columnar partitioning example

B. Revised FPGA partitioning procedure

Thanks to the previous model simplification we are now able to define the steps of the revised partitioning procedure called columnar partitioning:

1) Each tile belonging to a forbidden area is replaced by a tile that lies on the same column and does not belong to any forbidden area;
2) 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, thus a new portion is created containing that tile;
3) The portion is extended to the right side until free tiles of the same type are encountered;
4) The portion is extended to the bottom side until all the tiles on the row below the portion are free and of the same type. If the portion cannot be extended completely to the bottom of the FPGA, then the FPGA cannot be columnar partitioned;
5) If there are still free tiles, the process is repeated from step number 2 until all the tiles are part of one and only one portion;
6) At the end, each forbidden area is identified by its position and size.

To clarify how the columnar partitioning is performed, we show an example in Figure 2, representing the initial FPGA with hard processors shown in gray (Figure 2a) and the actions taken during step 1 (Figure 2b), steps 2-5 (Figure 2c) and step 6 (Figure 2d).

As we can see from Figure 2d we obtain the following sets of portions and forbidden areas:

$$P = \{1, 2, 3, 4, 5, 6\}, \qquad A = \{f1, f2\} \qquad (3)$$

Even though columnar partitioning cannot be applied in the general case, most of the commercially available FPGAs, including Xilinx devices of Virtex-7 family, are compliant with this simplified columnar description. A columnar partitioning enjoys two important properties that directly derive from the partitioning construction:

Property .3. Two adjacent columnar portions always have tiles of different types.

Property .4. The columnar portions can be orderly numbered from left to right.

Properties .3 and .4 are exploited in the following sections to introduce the needed constraints for the identification of free-compatible areas.

IV. Relocation as a constraint

In this subsection we show how to introduce bitstream relocation as a constraint considering an FPGA that has been successfully partitioned using the columnar partitioning procedure presented in Section III. The designer has to specify, as additional input information, how many free-compatible areas should be identified and for each area which are the regions for which compatibility has to be ensured. Within the following subsections we define the new parameters, variables and constraints that have to be added to the MILP model.

A. Constants definition

Exploiting Property .4 of the columnar partitioning, we enumerate the columnar portions from 1 to $|P|$ starting from the left side of the FPGA. The additional parameters and set needed for the new specifications are the following:

$FC :=$	set of free-compatible areas that have to be placed;
$s_{c,n} :=$	binary parameter set to 1 if area c has to be free-compatible with respect to reconfigurable region n;
$nTypes :=$	the number of different tile types present within the FPGA;
$tid_p :=$	integer number in the range $[1, nTypes]$ identifying the type of tiles present in portion p.

A free-compatible area is conceptually similar to a reconfigurable region: it is rectangular because it must have the same shape of a region to which it is compatible and it cannot overlap with other regions, free-compatible areas and forbidden areas. For this reason, the easiest way to introduce a free-compatible area is to consider it as special reconfigurable region for which additional constraints are added to ensure compatibility, more formally we have $FC \subset N$. By considering free-compatible areas as reconfigurable regions, we get for free all the necessary non overlapping constraints together with the constraints defining the amount of intersection between areas and portions defined in [10]. However, unlike a reconfigurable region, a free-compatible area does not require a certain amount of resources by itself, but the number and types of resources covered must be equal to the ones occupied by the region for which the compatibility is required. The latter constraint is addressed in the next subsections, while for a given free-compatible area n the parameters $c_{n,t}$ and the corresponding constraints in which the parameters are used are discarded from the MILP formulation.

B. Variables identification and semantic constraints

Since the portions that can be covered by the reconfigurable regions are columnar, the variable $k_{n,p}$ can be used to check whether the reconfigurable region n intersect the columnar portion p or not. In order to properly state the compatibility constraints we need a set of support variables that defines for a reconfigurable region, or a free-compatible area, the offset of the first columnar portion covered:

$o_{n,p}$:= real non negative variable (≥ 0) set to 1 when p is the first columnar portion (from left to right) covered by reconfigurable region or free-compatible area n, 0 otherwise.

As done also for other variables in the MILP model, such as h_n and $l_{n,p,r}$, the variable $o_{n,p}$ is declared as real even though the values that it can assume are integer. The reason for this is to reduce the problem complexity since a MILP solver can deal much easier with real variables rather than integer ones. What follows are the constraints needed to ensure the semantics and integrity of the variable $o_{n,p}$ representing the offset of a region or a free-compatible area.

Offset uniqueness:

$$\begin{gathered}\forall n \in N : \\ \sum_{p \in P} o_{n,p} = 1\end{gathered} \tag{4}$$

Offset assignment deriving from covered portions:

$$\begin{gathered}\forall n \in N : \\ o_{n,1} = k_{n,1} \\ \forall n \in N, p \in P \mid p > 1 : \\ o_{n,p} \geq -k_{n,p-1} + k_{n,p}\end{gathered} \tag{5}$$

Since the meaning of these new offset variables may be unclear to the reader, we show in Figure 3 an example of a reconfigurable region placed within a columnar partitioned FPGA together with the values assumed by the variables $o_{n,p}$ and $k_{n,p}$ for the specific placement represented.

C. Bitstream relocation constraints

At this point we are ready to introduce the constraints that ensure the compatibility between a free-compatible area and the corresponding regions. If we consider a columnar partitioned FPGA such as the one in Figure 3, we can see that a region, such as the one represented, intersects a set of adjacent portions. From Property .3 we know that at the edge between two different portions covered by a region the tile type changes. In the case of the represented region, 3 portions are covered and from left to right the tile types follow the sequence blue-green-blue. If we need to find a compatible area with respect to a region, we need an area that cover exactly the same number of portions in the same sequence in terms of tile types. The other constraint is that the height of the

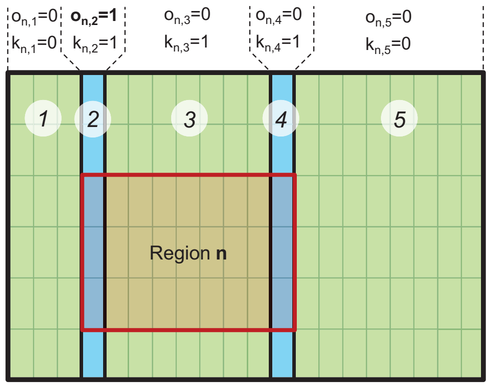


Fig. 3: Columnar portions offset example

region and the free-compatible area must be the same and the amount of tiles covered in each corresponding portion is also the same. Notice that having fixed the height and the number of tiles covered within a portion, the width of the intersection to that portion is automatically fixed. Overall we need four different set of constraints to ensure the compatibility between a reconfigurable region and a free-compatible area:

1) The height of the reconfigurable region and the free-compatible area must be the same;
2) The number of portions covered by the reconfigurable region and the free-compatible area must be the same;
3) Portions intersected in the same relative positions must have tiles of the same type;
4) The number of tiles intersected with portions covered in the same relative positions must be the same.

The first two constraints can be easily stated as follows:

$$\begin{gathered}\forall n \in N, c \in FC \mid s_{c,n} = 1 : \\ h_c = h_n\end{gathered} \tag{6}$$

$$\begin{gathered}\forall n \in N, c \in FC \mid s_{c,n} = 1 : \\ \sum_{p \in P} k_{c,p} = \sum_{p \in P} k_{n,p}\end{gathered} \tag{7}$$

For the remaining constraints we exploit the offset variables defined in the previous subsection together with Property .4 that allows an ordering of the columnar portions from left to right numbered from 1 to $|P|$. In the following inequalities pc and pn are meant to identify the first portion intersected by the free-compatible area c and region n respectively, while i is an index used to iterate over the set of portions. The latter two set of inequalities are as follows:

$$\begin{gathered}\forall n \in N, c \in FC, pc, pn \in P, i \in \{-|P|+1, \dots |P|-1\} \mid \\ s_{c,n} = 1 \land 1 \leq pc + i, pn + i \leq |P| : \\ tid_{pc+i} \leq tid_{pn+i} + nTypes \cdot (3 - o_{c,pc} - o_{n,pn} - k_{n,pn+i}) \\ tid_{pc+i} \geq tid_{pn+i} - nTypes \cdot (3 - o_{c,pc} - o_{n,pn} - k_{n,pn+i})\end{gathered} \tag{8}$$

$$\forall n \in N, c \in FC, pc, pn \in P, i \in \{-|P|+1, \dots |P|-1\} \mid s_{c,n} = 1 \wedge 1 \leq pc+i, pn+i \leq |P| : \\ \sum_{r \in R} l_{c,pc+i,r} \leq \sum_{r \in R} l_{n,pn+i,r} + \\ + maxW \cdot |R| \cdot (3 - o_{c,pc} - o_{n,pn} - k_{n,pn+i}) \\ \sum_{r \in R} l_{c,pc+i,r} \geq \sum_{r \in R} l_{n,pn+i,r} + \\ - maxW \cdot |R| \cdot (3 - o_{c,pc} - o_{n,pn} - k_{n,pn+i}) \tag{9}$$

Notice that the constraints are active only when pc and pn effectively represent the first occupied portions and when i is iterating over a portion that is covered. When the "big M" constants at the right hand sides are cancelled, each couple of inequalities ensure that the remaining terms coincide.

By looking at Equation 8 we can notice that tid_{pc+i} and tid_{pn+i} are known parameters denoting the tile type of two columnar portions, while the only variables involved in the formulas are $o_{c,pc}$, $o_{n,pn}$ and $k_{n,pn+i}$. If the tile types of the two portions are the same, then nothing is implied over the variables. On the other hand, if the tile types differ, the variables cannot be all equal to 1 at same time. More formally Equation 8 can be tightened and rewritten as:

$$\forall n \in N, c \in FC, pc, pn \in P, i \in \{-|P|+1, \dots |P|-1\} \mid s_{c,n} = 1 \wedge 1 \leq pc+i, pn+i \leq |P| \wedge tid_{pc+i} \neq tid_{pn+i} : \\ o_{c,pc} + o_{n,pn} + k_{n,pn+i} \leq 2 \tag{10}$$

V. Relocation as a metrics

The idea behind considering bitstream relocation as a metrics is quite simple: all the constraints defined in the previous section, together with the non overlapping constraints for the free-compatible areas are translated in soft constraints. By soft constraints we mean a relaxed constraint that can be always satisfied, but depending on how the constraint is satisfied the value of the objective function varies. To measure the level of satisfaction of the constraints related to free-compatible areas we introduce the following set of variables:

$v_c :=$	binary variable set to 1 if almost one of the constraints regarding the free-compatible area c is violated.

The variable v_c must be introduced in all the constraints related to the free-compatible area c that can compromise the feasibility of the final solution if violated. It is enough to introduce v_c in Equation 9 and Equation 10 of the previous section and within the non overlapping constraints of [10] for the purpose of maintaining the solution feasibility even when the free-compatible area c cannot be identified. This is the modified Equation 9:

$$\forall n \in N, c \in FC, pc, pn \in P, i \in \{-|P|+1, \dots, |P|-1\} \mid s_{c,n} = 1 \wedge 1 \leq pc+i, pn+i \leq |P| : \\ \sum_{r \in R} l_{c,pc+i,r} \leq \sum_{r \in R} l_{n,pn+i,r} + \\ + maxW \cdot |R| \cdot (3 - o_{c,pc} - o_{n,pn} - k_{n,pn+i} + v_c) \\ \sum_{r \in R} l_{c,pc+i,r} \geq \sum_{r \in R} l_{n,pn+i,r} + \\ - maxW \cdot |R| \cdot (3 - o_{c,pc} - o_{n,pn} - k_{n,pn+i} + v_c) \tag{11}$$

While this is the modified Equation 10:

$$\forall n \in N, c \in FC, pc, pn \in P, i \in \{-|P|+1, \dots |P|-1\} \mid s_{c,n} = 1 \wedge 1 \leq pc+i, pn+i \leq |P| \wedge tid_{pc+i} \neq tid_{pn+i} : \\ o_{c,pc} + o_{n,pn} + k_{n,pn+i} \leq 2 + v_c \tag{12}$$

The non overlapping constraints for O and HO described in [10] are modified in a similar fashion by adding or subtracting an appropriate "big M" term multiplied by the variable v_c.

Exploiting the variables v_c we can introduce a cost term in the objective function that measures how many of the requested free-compatible areas have not been identified. To increase the flexibility of this approach, we let the designer decide the weight or importance for each free-compatible area:

$cw_c :=$	weight associated with free-compatible area c.

Considering also the weights, the cost function for bitstream relocation becomes:

$$RL_{cost} = \sum_{c \in FC} cw_c \cdot v_c \tag{13}$$

The resulting objective function integrated with the one proposed in [10] becomes:

$$\min \left\{ q_1 \cdot \frac{WL_{cost}}{WL_{max}} + q_2 \cdot \frac{P_{cost}}{P_{max}} + q_3 \cdot \frac{R_{cost}}{R_{max}} + q_4 \cdot \frac{RL_{cost}}{RL_{max}} \right\} \tag{14}$$

where RL_{max} is used to normalize the relocation cost term and can be computed as:

$$RL_{max} = \sum_{c \in FC} cw_c \tag{15}$$

VI. Experimental evaluation

In this section we analyze the impact of bitstream relocation on the software defined radio (SDR) design proposed in [8]. The SDR design consists of the following five modules: matched filter, carrier recovery, demodulator, signal decoder and video decoder. For each module different *modes* requiring different resources are configured one at a time. The modes are mutually exclusive implementations of the module with the same set of inputs and outputs. All the modules are connected in sequential order with a 64 bit wide bus, moreover, the modes of a module are assumed to be all assigned to a specific region. Hence, there are 5 reconfigurable regions (one for each module).

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. Table I reports the number and type of resources required by each reconfigurable region expressed in terms of tiles.

TABLE I: Resource requirements for the SDR design

Region	CLB tiles	BRAM tiles	DSP tiles	# Frames
Matched Filter	25	0	5	1040
Carrier Recovery	7	0	1	280
Demodulator	5	2	0	240
Signal Decoder	12	1	0	462
Video Decoder	55	2	5	2180
Total	104	5	11	4202

As we can see from table I the resource requirements are heterogeneous and vary across the regions. The last column of the table shows also the least amount of configurable frames that each region needs to cover.

As a first analysis, we performed a feasibility test in which we checked the possibility to find at least a free-compatible area for each reconfigurable region at a time. The solver determined that no solution exists for the SDR design in which we require a free-compatible area for the matched filter or the video decoder region. Indeed, even if the amount of DSP, BRAM and CLB within the FPGA would suffice to accommodate one of the two areas, the rectangular geometry of the regions does not allow to exploit the resources completely. On the other hand, the solver was able to find a placement for each of the free-compatible areas related to the carrier recovery, demodulator and signal decoder region. From now one we refer to these regions as relocatable regions.

In light of the feasibility analysis, we defined two new problem instances derived from the SDR design in which we considered the same objective function as [8] and [10] in which the objective was to first optimize the wasted area and, without increasing the area cost, minimizing the overall wire length. Within the first instance, named SDR2, we required to find 2 free-compatible areas for each relocatable region, while, in the second instance, called SDR3, we requested 3 free-compatible areas for each relocatable region. The resources wasted by the extra free-compatible areas are not considered here as an additional cost, indeed these areas are needed only to reserve free spaces for the relocation of the bitstreams of the reconfigurable regions. Since the number of regions in the designs is manageable, we used O to solve the problems.

Table II compares the results achieved by our floorplanning extension (named PA) on SDR2 and SDR3 designs against the solutions obtained by [8] and [10] on the original SDR design. Notice that the proposed floorplanner extension is equivalent to our previous work [10] if relocation requirements are not considered. Thus, in Table II we only report the solution achieved by [10] with respect to the original SDR design.

The optimal solution for SDR2 was found by our approach in approximately 1160 seconds, however about 5 hours were needed to prove its optimality. For this problem, the quality of the result, in terms of the objective function, was equal to the one achieved by [10] on the SDR design without relocation requirements. The SDR3 instance is more complex than SDR2 due to the additional free-compatible areas requested, indeed, even if we let the solver run for 6 hours the best solution found was not proven to be optimal. For this problem, the free-compatible areas constraints affected the quality of the result and the final solution achieved 346 wasted frames: 40 more frames than SDR2 but still 120 frames less than the ones required for the solution presented in [8] without relocation constraints.

The floorplans resulting from solving SDR2 and SDR3 are shown in figures 4 and 5 respectively. The names of the free-compatible areas are composed using the name of the region to which they are compatible followed by a number (e.g. Signal Decoder 2).

TABLE II: Comparison of different floorplan solutions

Algorithm	Design	Free-compatible areas	Wasted frames
[8]	SDR	0	466
[10]	SDR	0	306
PA	SDR2	6	306
PA	SDR3	9	346

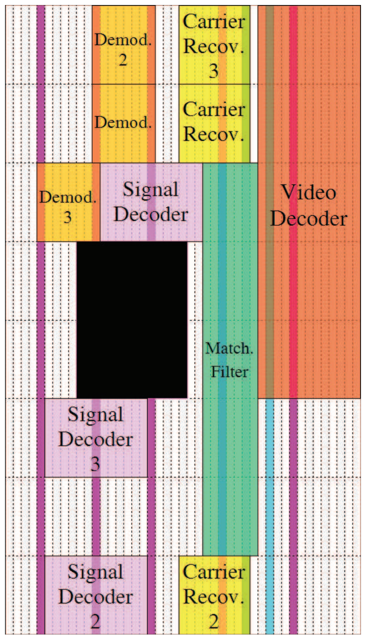


Fig. 4: SDR2 floorplan (6 free-compatible areas)

VII. Conclusions

Within this work we presented an extension of a floorplanning algorithm able to deal with bitstream relocation. The approach allows the designer to specify constraints on the number of compatible areas required by each reconfigurable region, or to introduce the relocability of a region as a metrics within the objective function to be optimized. Moreover, a revised FPGA partitioning technique compatible with most of

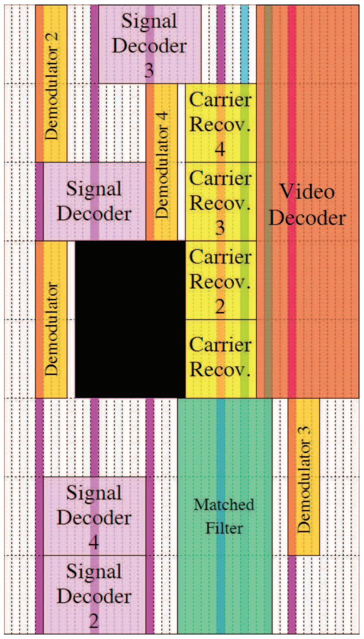


Fig. 5: SDR3 floorplan (9 free-compatible areas)

the current devices has been devised to simplify the description of the problem.

Acknowledgments

This work was partially funded by the European Commission in the context of the FP7 FASTER project [11](#287804).

References

[1] 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.

[2] H. Kalte, G. Lee, M. Porrmann, and U. Rückert, "REPLICA: A bitstream manipulation filter for module relocation in partial reconfigurable systems," in *Proc. IPDPS Workshops*, 2005.

[3] H. Kalte and M. Porrmann, "REPLICA2Pro: task relocation by bitstream manipulation in virtex-II/Pro FPGAs," in *Proc. Conf. on Computing Frontiers*, 2006, pp. 403–412.

[4] F. Ferrandi, M. Morandi, M. Novati, M. D. Santambrogio, and D. Sciuto, "Dynamic reconfiguration: Core relocation via partial bitstreams filtering with minimal overhead," in *Proc. Intl. Symp. on System-on-Chip (SOC)*, 2006, pp. 1–4.

[5] S. Corbetta, M. Morandi, M. Novati, M. D. Santambrogio, D. Sciuto, and P. Spoletini, "Internal and external bitstream relocation for partial dynamic reconfiguration," *IEEE Trans. on VLSI Systems*, vol. 17, no. 11, pp. 1650–1654, 2009.

[6] T. Becker, W. Luk, and P. Y. Cheung, "Enhancing relocatability of partial bitstreams for run-time reconfiguration," in *Field-Programmable Custom Computing Machines, 2007. FCCM 2007. 15th Annual IEEE Symposium on*. IEEE, 2007, pp. 35–44.

[7] Xilinx Inc, "Vivado Design Suite User Guide: Partial Reconfiguration," 2014.

[8] 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.

[9] 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.

[10] 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.

[11] Faster website. [Online]. Available: http://www.fp7-faster.eu/