Multiobjective Floorplanning for Partially-Reconfigurable FPGA Systems

BY

MARCO RABOZZI
B.S., Politecnico di Milano, Milan, Italy, July 2012

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2014

Chicago, Illinois

Defense Committee:

John Lillis, Chair and Advisor
Wenjing Rao
Marco D. Santambrogio, Politecnico di Milano

ACKNOWLEDGEMENTS

Several people made this work possible, they gave me inspiration, suggestions, support, precious time, guidance, help and patience.

First of all I want to thank my advisor, Marco Domenico Santambrogio, who encouraged me and stimulated my interest and passion on our work. Without his guidance this thesis would not have been possible. Advice and comments given by Prof. John Lillis, my UIC advisor, has been a great help in enhancing the work and showing me the right direction to follow.

A thank you also to Fabrizio Spada, Riccardo Cattaneo and Gianluca Durelli for the support and precious help they gave me.

Thank you also to all the people in the NECSTLab at Politecnico di Milano, with whom I spend the working time in a stimulating atmosphere sharing useful suggestions and knowledge.

A thank you to the PoliMi-UIC students of Fall 2013, who shared with me such a great new experience.

Thank you also to Giovanni Riso and Alessandro Riva for all the study hours spent together at Politecnico di Milano.

Last but not least, I would like to express my gratitude to my family and Emilia Maulini for the encouragement they gave me and their patience.

MR

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

CHAPTER **PAGE**

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

3D-subTCG 3-Dimensional Transitive Closure sub-Graph. vii, 35

ASIC Application-Specific Integrated Circuit. vii, 2, 4

BRAM Block RAM. vii, 5, 13, 34, 38, 44, 77, 78

CLB Configurable Logic Block. vii, 4–6, 9, 13, 14, 34, 44, 46, 59, 61, 77, 78

CMP Chip-MultiProcessor. vii, ix

DSP Digital Signal Processor. vii, 5, 13, 34, 38, 44, 46, 59, 77, 78

FPGA Field Programmable Gate Array. vii, ix, x, 1–18, 20, 24–26, 29, 30, 33, 34, 37–39, 42–46, 59–64, 76, 79, 80, 82

GPP General-Purpose Processor. vii, 2

HDL Hardware Description Language. vii, 3, 4, 8, 10

HOF Heuristic-Optimal Flooplanner. vii, 41, 42, 45, 47, 49, 53, 68, 70–76, 80–83

HPWL Half-Perimeter Wirelength. vii, 15, 17, 22, 54

IC Integrated Circuit. vii, 4

ICAP Internal Configuration Access Port. vii, 3, 12

IO Input/Output. vii, 2, 40, 54–57, 71, 72, 74, 81

IOB Input/Output Block. vii, 4–6

LCS Longest Common Subsequence. vii, 33

LP Linear Programming. vii, 59, 69

LUT Look-Up Table. vii, 6

LIST OF ABBREVIATIONS (Continued)

MILP Mixed-Integer Linear Programming. vii, x, 25, 39, 41, 42, 45, 49, 53, 58, 59, 61, 68, 70–72, 76, 78, 80–82

OF Optimal Flooplanner. vii, 41, 42, 45, 47, 49, 53, 54, 67, 68, 70–76, 79–82

PR Partial Reconfiguration. vii, 8, 12, 14, 34–40, 42, 64, 80, 81

SDR Software Defined Radio. vii, 25, 77, 79

TCG Transitive Closure subGraph. vii, 35

TSP Travelling Salesman Problem. vii, 18–20, 22–24

VLSI Very Large Scale Integration. vii, 15, 16, 26, 29

SUMMARY

The exponential performance improvement achieved by single processors, starting from the early 80s, has slowed down during the last decade. On one hand the power wall was faced, it was no more possible to obtain faster processors simply by augmenting the clock frequency, indeed the power consumption would be too high and cooling with air not feasible. On the other hand the gap between the time needed to access the main memory and the time required by the processor to complete an instruction has steadily increased in the last years.

These issues have led to the advent of Chip-MultiProcessor (CMP) architectures on one side and on a renewed interest in reconfigurable computing on the other. The trends is moving towards heterogeneous architectures in which CMPs and reconfigurable devices such as Field Programmable Gate Array (FPGA) cooperate to achieve effective solutions in terms of both performance and power consumption.

The development of applications able to exploit the capabilities of such heterogeneous systems requires a completely different design flow. The programmer has to deal with a bigger solution space in which hybrid software/hardware solutions can be devised. New challenges arise when some of the tasks of the application are executed in hardware. each task should be mapped to a semantically equivalent circuit, scheduled and floorplanned on the device ensuring the reconfiguration constraints.

This work aims at optimizing the floorplanning on partially-reconfigurable FPGAs taking into account a set of different metrics whose weights can be specified by the designer. We

propose a Mixed-Integer Linear Programming (MILP) formulation to solve the problem. Our approach is able to find an optimal floorplan, with respect to the specified set of metrics, while considering the complex structure and heterogeneous resources of modern FPGAs.

The thesis is organized as follows:

- in Chapter 1 we provide a view on reconfigurable computing and describe the design flow for partially-reconfigurable FPGAs
- in Chapter 2 we present a more detailed description of the floorplanning on FPGAs problem together with a NP-completeness proof
- Chapter 3 discuss the most important floorplan representations and the state-of-the-art floorplanners
- Chapter 4 shows the proposed floorplanner and describe its implementation
- Chapter 5 reports the results obtained with our methodology on a set of problem instances and a real case study
- In Chapter 6 we discuss the contributions of our approach together with its limits and possible future developments

CHAPTER 1

INTRODUCTION AND MOTIVATIONS

In this chapter we present the context in which our work is developed. Section 1.1 introduces reconfigurable computing and architectures motivating the use of Field Programmable Gate Arrays (FPGAs). Section 1.2 describes the FPGA technology and reconfigurations features of modern devices. The chapter is concluded with Section 1.3 that presents the steps needed to perform partial dynamic reconfiguration on FPGAs, underlining the lack of an automated floorplanner to help the designer in defining the area constraints.

1.1 Reconfigurable computing

The concept of reconfigurable computing is not new, a first proposal of a reconfigurable system was given by Estrin [1] in 1960. The basic scheme of the proposed architecture was a fixed standard processor coupled with an array of reconfigurable hardware, corresponding to the variable part of the system. The idea was to configure the variable part of the architecture to address more effectively the specific type of computation required and, once the computation was performed, to reconfigure the system to solve new tasks. The concept of reconfiguration can be intuitively associated to a change of functionality of a system, however, to be more precise, we report here an interesting formal definition taken from [2]:

Definition 1.1. (Reconfiguration) Given a System S able to interact with the environment E by means of an input set I and an output set O, reconfiguration means changing the current

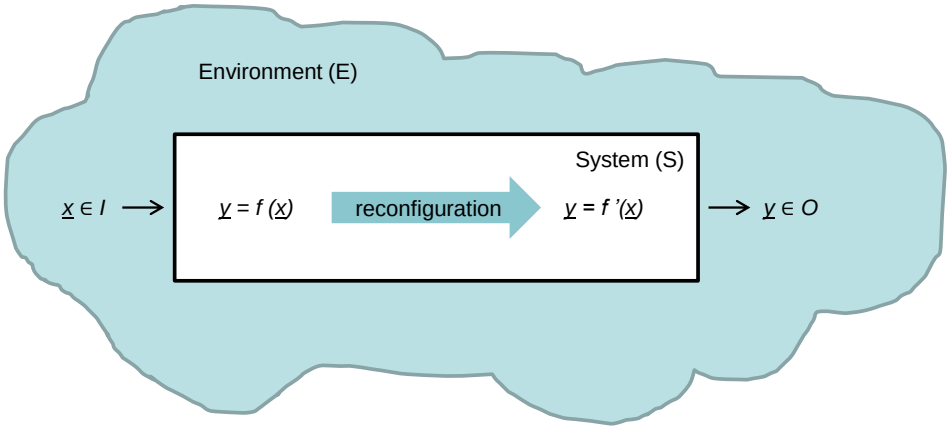


Figure 1: System reconfiguration

behaviour or functionality $f : I \to O$ of S to a new functionality $f' : I \to O$ on the same input and output sets.

A simple representation of the given definition is shown in Figure 1.

FPGAs are well suited for reconfigurable systems, since they can be configured on the field after the manufacturing process to achieve the desired hardware functionality. Several reconfigurable architectures are possible exploiting these devices, among which we can have systems consisting of General-Purpose Processors (GPPs) coupled with FPGAs and systems-on-chip entirely developed on FPGAs connected through the Input/Output (IO) with only few physical components [2]. Reconfigurable computing combines the features of both software and hardware solutions, it gives the possibility to achieve higher performance than systems entirely based on GPPs, while allowing higher flexibility than Application-Specific Integrated

Circuits (ASICs). To better classify different models of reconfiguration, we consider, from [3], the following characterizing features:

- *who* controls the reconfiguration;
- *when* the configuration is generated;
- *what* is the level of reconfiguration granularity.

The first subdivision (*who*) distinguishes from systems that completely control and perform the reconfiguration internally to systems in which the reconfiguration is initiated and managed by an external source. When the reconfiguration is controlled and executed within the FPGA boundaries, there must be a specific part of the device that is configured to communicate with an internal reconfiguration interface, such as the Internal Configuration Access Port (ICAP) for Xilinx [4] devices.

The generation of the configurations (*when*) ranges from completely static techniques to fully dynamic ones. Currently, support is given to the creation of configurations at design time, in which all the possible implementations and relative positions of the modules are considered. Once generated, the configurations can be subsequently loaded to reconfigure, even partially [5], the device. Other possibilities rely on adapting or completely generating the configurations dynamically. However, the last approach is currently not feasible due to the high amount of time required to synthesize modules from Hardware Description Languages (HDLs).

The level at which reconfiguration takes place (*what*) can vary a lot. We can basically distinguish between two different approaches referred as *smallbits* and *module based* [6]. *small-*

bits consists in manipulating the single configuration bits of a Configurable Logic Block (CLB) or in modifying the parameters of an Input/Output Block (IOB) within the device. On the other hand, the *module based* technique involves the modification of larger FPGA areas into which different modules or functionalities can be loaded. In the context of this work we are going to consider reconfiguration at module level, following the latest guidelines for partial reconfiguration provided by Xilinx [5].

1.2 FPGA Technology

Within this section, we describe the FPGA technology referring specifically to Xilinx [4] devices, even though the underlining concepts also hold for other vendors such as Altera [7]. In subsection 1.2.1 we give an overall description of the device structure, while in subsection 1.2.2 we characterize the device according to the types of allowed reconfiguration.

1.2.1 FPGA overview

An FPGA device is a particular type of Integrated Circuit (IC) whose hardware can be configured to execute a desired functionality. The main property of these devices, is their possibility of being reconfigured an infinite number of times, so that they can adapt to the specific task to solve [8]. The reconfiguration process of the FPGA is conceptually equivalent to realize a new piece of hardware whose specification can be given using a HDL, as done for ASIC.

The FPGA architecture can be seen as matrix in which each cell contains a resource. We refer to resources as the basic blocks that can be used to realize our circuit on the device. Depending on the FPGA model there can be different resources and their collocation within the

device may differ from one family to another. At least three types of resources are present within the device, namely: CLBs, IOBs and interconnections. An FPGA containing only these types of resources is called homogeneous, while a device containing other resources such as Digital Signal Processors (DSPs), Block RAMs (BRAMs) or multipliers, is defined as heterogeneous. In figure 2 the structure of a homogeneous FPGA is represented.

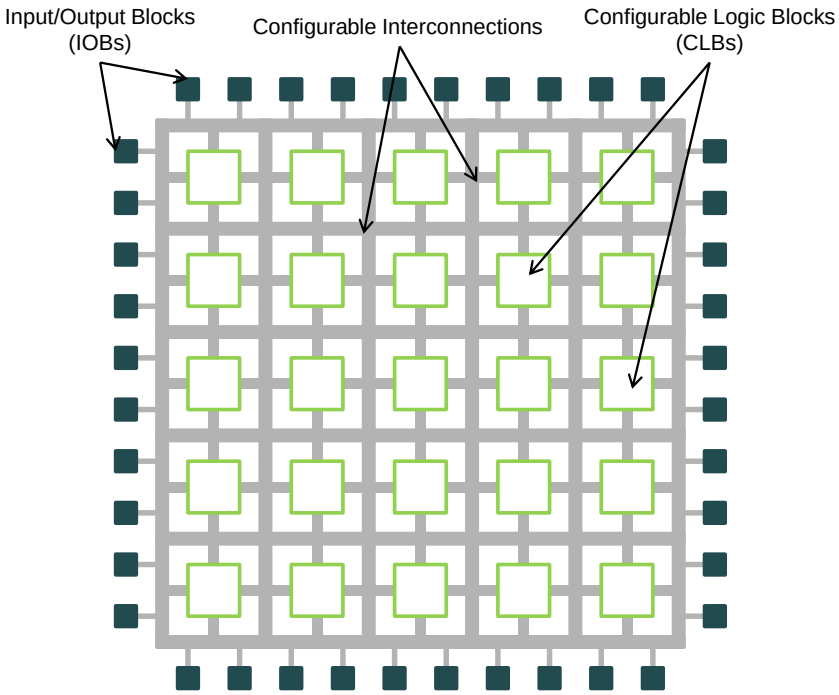


Figure 2: Homogeneous FPGA structure

CLBs are the main components of the FPGA and are subdivided into slices [9]. Each slice in turn, contains different components, among which we usually have Look-Up Tables (LUTs), latches and multiplexers. LUTs are used to implement custom combinatorial functions and, depending on the device, a LUT can have 4 inputs and 1 output or also have 6 inputs and 2 outputs. The key property of a LUT is that its output values can be configured for each combination of the inputs.

IOBs are usually located at the boundary of the chip and they are needed to perform input/output communications from and to the other peripherals connected to the FPGA. IOBs can be configured to select the desired operating voltage and the direction of the communication, while the interconnections can be configured to route the signals among CLBs and to connect them to the IOBs.

1.2.2 Reconfiguration characterization

In order to configure the desired interconnections and functionalities, a configuration memory integrated into the FPGA is used. A bit contained within a specified address in the memory is in a one-to-one correspondence to the underlining resource being configured, while the file that contains the configuration data that is loaded into the configuration memory is called *bitstream*. The configuration memory is organized into frames, that are the minimal configuration units that can be addressed [5]. With respect to the matrix organization of CLBs resources, each frame can span multiple CLBs height, while multiple frames are needed on the horizontal direction to fully configure a CLB. Depending on the FPGA technology different types of re-

configuration are possible [8], a simple taxonomy is obtained using the following two different features:

- execution requirements;
- reconfiguration size.

With execution requirements we mean the prerequisites to perform a reconfiguration with respect to the current execution status of the device. If we must interrupt the current FPGA execution and reboot the device to load a new bitstream in the configuration memory, the reconfiguration is called *static*. On the other hand, if there are no temporal requirements and we are allowed the load a bitstream even when the FPGA is executing a task, we have *dynamic* reconfiguration. The reconfiguration size refers to the least amount of area that can be reconfigured at once. If the reconfiguration process requires to load a bitstream characterizing the overall FPGA device, the reconfiguration is called *complete*. Whereas, if we are allowed to reconfigure a subset of the configuration memory, we have *partial* reconfiguration and a portion of the device can be reconfigured using a partial bitstream.

In case of partial reconfiguration, we can have a further classification based on the dimension at which reconfiguration is performed [8]. If complete columns of the device must be reconfigured the reconfiguration is 1D-partial, because we only have a degree of freedom on the horizontal direction. Otherwise, if we are also allowed to reconfigure parts of the columns and describe rectangular regions, the reconfiguration is 2D-partial thanks to the added degree of freedom on the vertical direction. The work developed in this thesis addresses the most general

reconfiguration: 2D-partial dynamic reconfiguration for FPGAs with heterogeneous resources. Partial dynamic reconfiguration has the great benefit of giving the designer the possibility to change part of the functionalities of the FPGA, without interrupting and compromising the execution of other tasks.

1.3 Partial dynamic design flow

In this section we describe, within the context of Xilinx devices, a simplified version of the design flow for FPGAs supporting partial dynamic reconfiguration [5].

The description of a design targeted for FPGAs is suitably given with an HDL such as VHDL or Verilog together with a set of constraints defining the prerequisites for the project. When Partial Reconfiguration (PR) is considered, the designer has to specify the description of each of the modules that has to be reconfigured onto the device. In a PR design it is possible to identify two types of logic, namely: *static logic* and *reconfigurable logic*. Static logic refers to the logic on the device that, once configured, is never changed, whereas the reconfigurable logic can be modified across different reconfigurations of the system. As a requirement for PR design, the static logic and the reconfigurable logic must lie in different separated areas of the device [5]. The reconfigurable logic in turns can be divided into several reconfigurable regions, while each reconfigurable region can host a specific set of tasks that are reconfigured one at a time by loading the corresponding partial bitstream on the FPGA. Xilinx also strongly recommends to define rectangular reconfigurable regions that include complete tiles to avoid performance degradation when the overall system is deployed. A tile consists of several adjacent configurable frames containing complete resources and is referred as a reconfigurable frame or

minimal reconfigurable unit within [5]. To clarify the notion of tiles, we underline them in Figure 3 where an heterogeneous FPGA having frames spanning 4 CLBs height is considered.

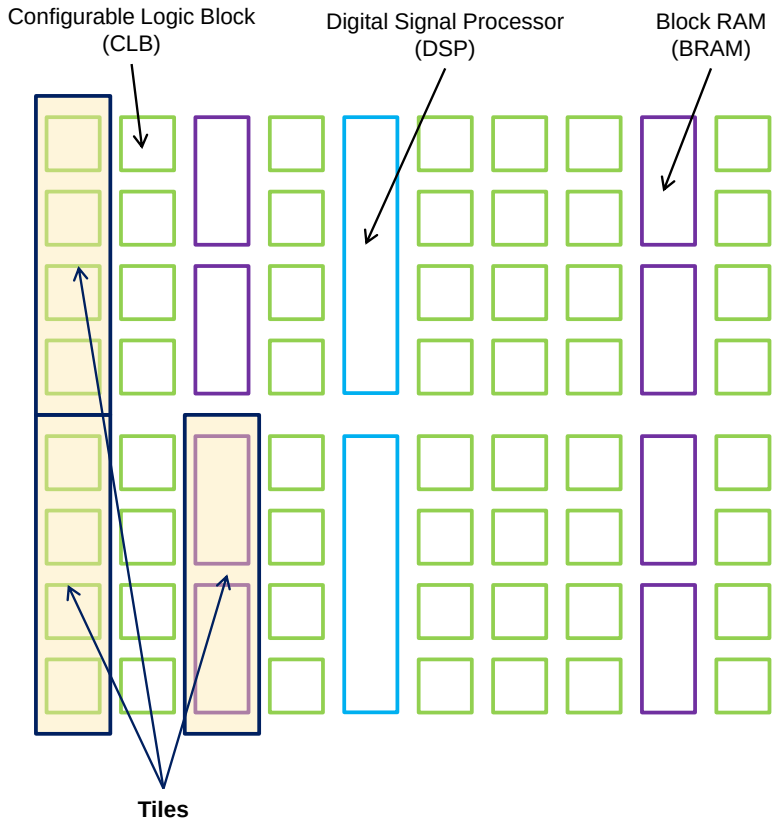


Figure 3: Heterogeneous FPGA structure and tiles locations

The overall design flow intended for partially-reconfigurable FPGA devices is represented in Figure 4. As a first step, the designer is required to provide a functional description of all

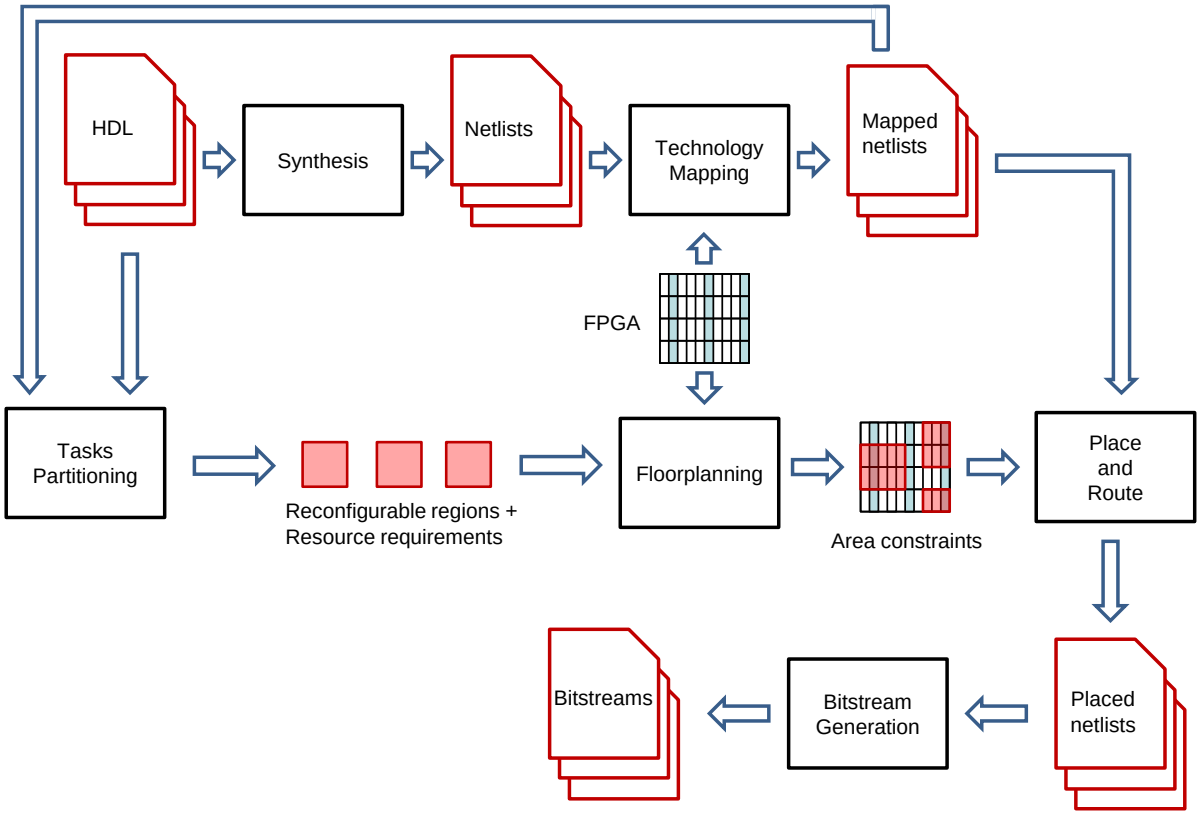


Figure 4: Partial reconfiguration design flow

the tasks or modules by means of HDL, that is subsequently synthesized to produce a set of netlists. The synthesis translates the functional description of the system into a set of basic components, such as logic ports and flip-flops, among which interconnections are established.

A further refinement of the netlists is performed during technology mapping, here the basic components involved in the netlists are logically mapped to the resources available on the target FPGA and we are provided with the numbers of device resources required by each

netlist. Exploiting this last information, the designer can decide how to partition the tasks into reconfigurable regions, so that eventually each reconfigurable region is characterized by the resource requirements of the tasks assigned to it.

The area constraints for the design are derived from the floorplanning step that is manually performed by the user. The reconfigurable regions must cover enough resources to meet the requirements of the tasks being reconfigured over time. Since the shape of a reconfigurable region cannot be modified during the execution of the system, the area of the region must be chosen large enough to accommodate the maximum number of resources used by the assigned tasks.

At this point, both the area constraints and the mapped netlists can be given as input to the place and route phase. Here each mapped netlist is placed onto the FPGA and signals among different components are routed. This step is aware of the definition of the reconfigurable regions, thus the placement is performed so that the area constraints are not violated. The final step of the flow produces the partial bitstreams associated to the reconfigurable regions and a complete bitstream for the configuration of the static logic and tasks initially present within the reconfigurable regions. The communication between reconfigurable regions and static logic is guaranteed by the insertion of proxy logic that is automatically generated by the tools within the design flow. Proxy logic remains fixed during the reconfiguration of the regions and all the tasks implemented in the same region must have the same connections to the proxy logic to avoid dangling wires.

Once the bitstreams are obtained, they can be loaded onto the FPGA by means of the JTAG port and possibly using ICAP for subsequent partial reconfigurations. The ICAP has to be inserted in the static logic of the system and allows the chip to reconfigure itself. On the other hand, the JTAG port can be accessed externally to reconfigure parts of the system and it is more suitable for debugging.

Notice that the overall design flow presented here is not meant to be followed sequentially, indeed during each step of the flow, the designer is warned about possible not satisfied project constraints such as timing closures. Thus, the design can be interrupted at any point and previous phases may be re-executed taking into account the feedbacks received. Further, during synthesis, technology mapping, place and route phases several optimization parameters can be set to drive the deployment of the system.

Floorplanning is a critical step of the design flow, since the designer as to take into account several constraints while trying to achieve a good device area subdivision manually. In our work, we propose a fully automated floorplanner, able to take into account the PR constraints, while searching for optimal solutions with respect to a customizable objective function. A first version of our work has been accepted as a full paper for the 22nd IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM) conference [10].

CHAPTER 2

FLOORPLANNING PROBLEM

In this chapter we present a detailed description of the floorplanning problem and prove its complexity class. First, in Section 2.1, we specifically consider and describe floorplanning for partially-reconfigurable FPGAs having heterogeneous resources. Then, within Section 2.2, we compare it to other related problems whose complexity classes are already known and we conclude the chapter proving that the decisional version of the floorplanning problem is NP-complete.

2.1 Problem description

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 matrix where each cell, described by its integer coordinates, contains one, none or more than one resource. To fix the notion of a cell, we can consider it as spanning 1 CLB resource width and 1 CLB resource height on the FPGA device. Moreover, since we are interested in reconfiguring regions of 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, called tile, that is a rectangle including several cells. Figure 5 shows an example of a FPGA matrix structure in terms of cells and tiles.

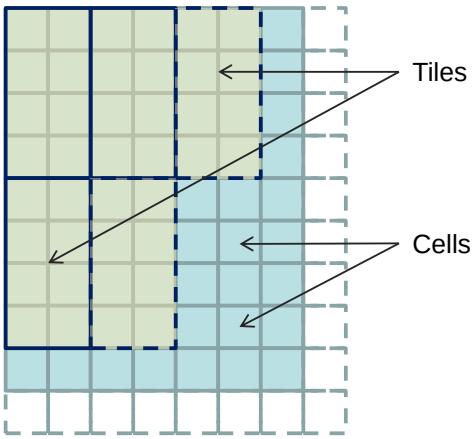


Figure 5: FPGA matrix

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 matrix. 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 to ease the place and route process [5] (PR requirement). The latter constraint avoid incomplete tile boundaries, that would require extra circuitry and increased latency to modify, read and write configuration information [11]. Depending on the device, a tile has a different size in terms of CLB resources (on a Xilinx [4] Virtex-5 XC5VLX110T it spans 20 CLBs height and 1 CLB width). The width and height of a tile are respectively denoted by $tileW$ and $tileH$.

2.2 Complexity

Even though floorplanning for partially-reconfigurable heterogeneous FPGAs may resemble floorplanning for Very Large Scale Integration (VLSI) circuits [12], the two problems are different. We discuss how the two problems are related and prove that the decisional version of the FPGA floorplanning problem is NP-complete in the general case.

2.2.1 VLSI floorplanning problem

In VLSI floorplanning we are given a set of modules described in terms of rectangles with fixed size, or having some constraints on the aspect ratio, that have to be packed optimizing the overall area occupancy and the overall wirelength. A common technique used to estimate the wirelength between two modules is to use the Half-Perimeter Wirelength (HPWL). HPWL is computed as the product of the Manhattan distance between the modules centroids multiplied by the interconnection width. If needed, it is also possible to add the constraint that the resulting modules packing cannot exceed a fixed-outline due to a fixed chip size. A simplified decision problem of the fixed-outline floorplanning can be stated as follows:

Problem 2.1. (VLSI fixed-outline floorplanning) Given M modules with fixed rectangular shapes, decide if it is possible to pack all the rectangles in a bounding box of height H and width W, such that no two rectangles overlap.

This problem has been shown to be NP-complete [13]. Moreover, if we remove the fixed-outline requirement we can derive a less constrained decision problem:

Problem 2.2. (VLSI floorplanning) Given M modules with fixed rectangular shapes, decide if it is possible to pack all the rectangles in a bounding box of area A, such that no two rectangles overlap.

Murata et al. [14], show a polynomial reduction from the fixed-outline version of the problem that prove the NP-completeness of VLSI floorplanning. Allowing the modules to be rotated by ninety degrees do not change the complexity class, hence the unoriented and the oriented version of the problem are both NP-complete [15].

2.2.2 FPGA floorplanning problem

We would like to derive a complexity result also for the floorplanning problem on partially-reconfigurable FPGAs [16]. A direct polynomial reduction from the VLSI floorplanning problem, or from the make-span minimization problem [13], is not straight forward, mainly because our problem differs in three aspects:

- a reconfigurable region even though rectangular, does not have a fixed width and height but requires a certain amount of area;
- modern FPGAs have heterogeneous resources in specified positions of the chip, thus a reconfigurable region may need to cover different resources, disallowing some region placements and shapes;
- reconfigurable regions must cover complete tiles with respect to the FPGA matrix.

The objective function of the floorplanning can take into account different metrics and a tile can consist of several cells. However, for the purpose of proving the NP-completeness, we

just consider a simplified decisional version of the problem where cells and tiles coincide and only overall area and wirelength are taken into account:

Problem 2.3. (floorplanning on heterogeneous partially-reconfigurable FPGAs) We are given a device matrix M of width W and height H and n reconfigurable regions. Each tile contained in M is characterized by an integer number of resources along with the resource type identifier. Each reconfigurable region can require different types and numbers of resources and the wirelength between regions is computed using the HPWL measure. The goal is to decide whether it is possible to assign all the reconfigurable regions to rectangles on matrix M such that: (i) no two regions overlap, (ii) all the regions cover the required number and type of resources, (iii) the covered tiles are not greater than α (area bound) and (iv) the sum of the wirelength between regions is not greater than γ (wirelength bound).

To clarify the description of the problem, we show in Figure 6 an example containing a valid and an invalid floorplan where we assume the values for α and γ big enough to satisfy area and wirelength bounds. The device consists of two different types of resources, namely resource 1 and resource 2. In this scenario we consider one resource for each tile, hence there are 28 resources of type 1 and 14 resources of type 2. The resource requirements of the two reconfigurable regions to place on the device are shown on top of the figures. The placement 6a is valid, indeed the two regions do not overlap, are assigned device matrix rectangles containing complete tiles and cover all the required resources. On the other hand, 6b is not a valid floorplan, because even though the regions do not overlap and are assigned to rectangles covering complete tiles, we can easily see that region 1 does not meet the resource requirement for type 1 resources.

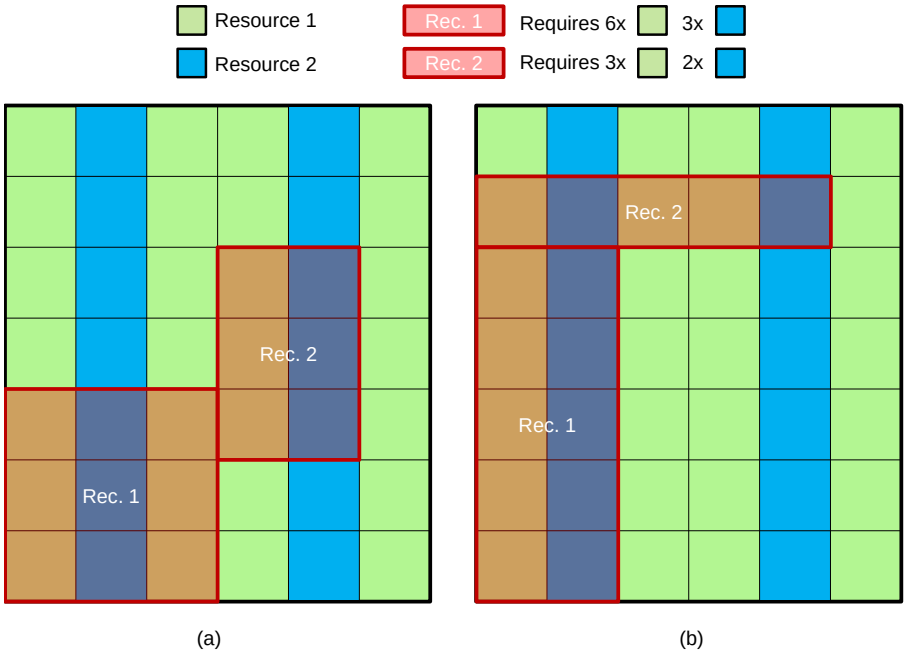


Figure 6: Valid (a) and invalid (b) floorplans

We are now ready to prove the following theorem:

Theorem 2.4. *The floorplanning on heterogeneous partially-reconfigurable FPGAs problem is NP-complete*

Proof. To prove the claim we consider a polynomial reduction from the metric Travelling Salesman Problem (TSP)[1] with Manhattan distance [17, p. 212], to the floorplanning on heterogeneous partially-reconfigurable FPGAs problem, from now on simply called floorplanning prob-

[1]In the metric TSP a metric is used to compute the distances between nodes, this ensures that the edges connecting them satisfy the triangular inequality.

lem. In the metric TSP with Manhattan distance we are given a complete undirected graph $G(N, E)$ where N is the set of nodes and E the set of edges. The nodes are placed within the Cartesian plane at integer coordinates, for every node $i \in N$ its coordinates are (i_x, i_y). The cost of edges connecting each couple of nodes is computed using the Manhattan distance, so, for every edge $e = \{i, j\} \in E$ its cost is $c_e = |j_x - i_x| + |j_y - i_y|$. The goal is to state whether there exists an Hamiltonian cycle[1] on graph G of cost not greater than ϵ (cost bound). Papadimitriou proved that both Euclidean and Manhattan metric TSPs are NP-complete [18].

We now show that given any instance of the Manhattan metric TSP we can derive in polynomial time an instance of the floorplanning problem, such that the answer to the floorplanning porblem is "Yes" if and only if also the answer to the Manhattan metric TSP is so. This would imply that solving the floorplanning problem would be at least as hard as solving the Manhattan metric TSP.

Before going on in the proof, we assume, without loss of generality, that no two nodes of a Manhattan metric TSP lie on the same Cartesian coordinates. Indeed, thanks to triangular inequality, it is possible to devoid the problem instance from such multiple nodes, obtaining a new problem instance that is equivalent to the original one in terms of the least cost Hamiltonian cycle.

Consider now an instance of the Manhattan metric TSP, the construction of the corresponding floorplanning problem instance can be computed in polynomial time as follows:

---

[1]An Hamiltonian cycle of an undirected graph G with $|N|$ nodes, is a cycle of G visiting all the $|N|$ nodes exactly once, whose cost is the sum of the costs of all the edges in the cycle.

- generate a bounding box containing all the nodes of the graph, this is accomplished searching for the minimum and maximum values of the nodes coordinates: the left bottom corner and the top right corner of the bounding box are set to $(\min_{i \in N} i_x - 0.5, \min_{i \in N} i_y - 0.5)$ and $(\max_{i \in N} i_x + 0.5, \max_{i \in N} i_y + 0.5)$ respectively;
- consider an FPGA matrix having the same width and height of the bounding box, in which all the tiles are squares with unitary side (i.e. $tileW = 1$ and $tileH = 1$);
- move the FPGA matrix onto the bounding box, so that the matrix and the bounding box overlap exactly. This ensures that nodes within the bounding box are located in the center of tiles of the FPGA matrix;
- for each tile containing a node, assign a resource of type S (we refer to these as S tiles);
- assign to the remaining tiles an arbitrary resource different from S;
- consider $n = |N|$ reconfigurable regions numbered from 1 to n;
- set the resource requirements for each reconfigurable region to exactly one resource of type S;
- consider the regions connected in circular order: 1 connected to 2, 2 connected to 3, ..., n connected to 1, where the bandwidth of each connection is unitary;
- set the area bound α to n;
- set the wirelength bound γ to the TSP cost bound ϵ.

The construction should appear clearer looking at Figure 7.

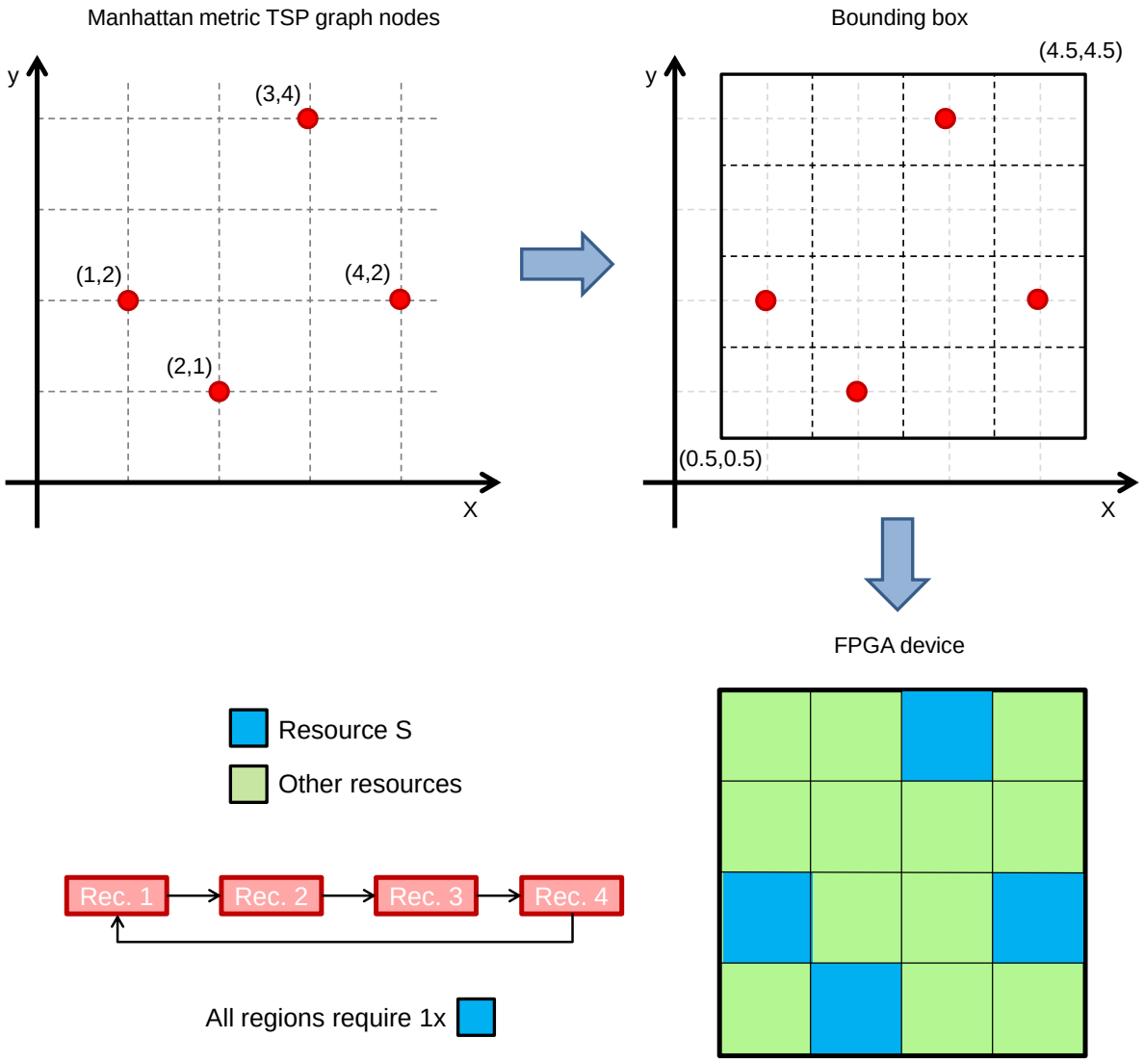


Figure 7: From TSP to floorplanning problem instance

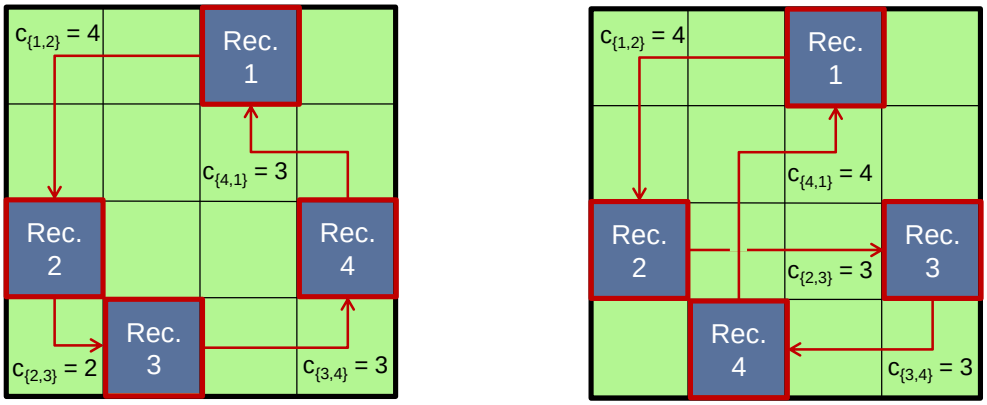
An important consequence derived from our construction, is that each tile containing a resource of type S has its center in a direct correspondence to a node of the original TSP problem. Moreover, the Manhattan distance between the center of two S tiles is equal to the Manhattan distance between the corresponding nodes in the TSP problem.

Another property derived from construction regards the placement of the n reconfigurable regions. Since we have $\gamma = n$ and each region requires exactly one resource, we cannot have regions covering more than one tile. Furthermore, the resource required by each region is of type S and since there are n S tiles, a valid floorplan must assign each region to a rectangle covering exactly an S tile not covered by other regions.

So far we have not yet considered the validity of the floorplan with respect to the wirelength. Figure 8 shows two floorplans satisfying the resource constraints and area bound but with different wirelength.

The wirelength of a link is computed with the HPWL measure, since the bandwidth have been set to 1 for all the connections, the HPWL is simply the Manhattan distance between the center of the two regions involved in the communication. Moreover, from what argued before, each region covers exactly an S tile, hence the region center corresponds to the covered S tile center. In conclusion, we have the following result:

Property 2.5. the wirelength between two connected regions is equal to the Manhattan distance of the two nodes in the TSP graph that correspond to the S tiles covered by the regions.



(a) Wirelength = 12.

(b) Wirelength = 14.

Figure 8: Wirelength of different floorplans

Thanks to Property 2.5 we are now able to show the equivalence of the two problem instances.

Given a florplan that satisfy the resource requirements and area bound, we can derive an Hamiltonian cycle in the TSP graph having the same cost as the wirelength of the floorplan. To achieve this, we simply label each node in the TSP graph with the number of the region that covers the S tile corresponding to the node. Then, the edges of the Hamiltonian cycle are those connecting nodes 1 and 2, nodes 2 and 3, ..., nodes n and 1. Since the connection between regions are circular we have a one to one correspondence between the connection of the regions and the edges within the Hamiltonian cycle. Thus applying Property 2.5 on each couple of connected regions, and summing up all the contributes, we have that the overall wirelength of the floorplan is equal to the Hamiltonian cycle cost.

On the other hand, given an Hamiltonian cycle in the TSP graph of cost c, we can derive a floorplan that satisfy the resource requirements and area bound with a wirelength equal to c. The first step is to convert the Hamiltonian cycle into a circuit and label the nodes from 1 to n following the arc directions. Then, we place each reconfigurable region i onto the S tile that correspond to node i. Thus, in the Hamiltonian cycle we have connections between nodes 1 and 2, nodes 2 and 3, ..., nodes n and 1 that corresponds to the same connections between regions 1 and 2, regions 2 and 3, ... regions n and 1. Again using Property 2.5 on each couple of regions, we get that the floorplan wirelength and the Hamiltonian cycle cost are the same.

The previous statements implies that an Hamiltonian cycle of cost no greater than ϵ exists if and only if there exists a valid floorplan, that is a floorplan satisfying the resource requirements, the area bound and with a wirelength not greater than $\gamma = \epsilon$.

We have proven that if we can solve the floorplanning problem in polynomial time, we are also able to solve the metric TSP with Manhattan distance in polynomial time that is NP-complete. Hence the floorplanning problem is NP-hard. It is easy to see that the floorplanning problem is also in NP. Indeed, given a floorplan solution consisting of the coordinates of the reconfigurable regions within the FPGA device, we can check in polynomial time if the solution is correct. This concludes the argument and shows that the floorplanning problem is NP-complete.

□

Theorem 2.4 states that if P is different from NP it is not possible to solve the the floorplanning problem in polynomial time. However, this should not prevent us from trying to develop

an exact algorithm, the previous proof relies on an unbounded number of reconfigurable regions and to an arbitrary complex structure of the FPGA device. In real applications, such as the Software Defined Radio (SDR) design presented in [11] and discussed in Section 5.3, the number of reconfigurable regions is limited while the FPGA structure is quite regular. In this work we shows an exact approach based on a Mixed-Integer Linear Programming (MILP) model, that can find an optimal solution to the floorplanning problem.

CHAPTER 3

STATE OF THE ART

In this chapter we present the contribution of previous works regarding the floorplanning problem. Section 3.1 presents different floorplan representations that have been devised in literature for VLSI floorplanning. Then, within subsection 3.1.1, we describe more specifically the *sequence pair* representation, that has been originally invented for VLSI design [14] and recently exploited for floorplanning on partially-reconfigurable FPGAs with heterogeneous resources [16]. In Section 3.2 we consider previous works on FPGA floorplanning, while Section 3.3 concludes the chapter underlining the limits of current methodologies and introducing our approach.

3.1 Floorplan representation

One of the main problems arising in VLSI floorplanning is how to characterize the solution space. Indeed modules, in principle, could be placed in the plane at any position, thus producing an infinite number of alternatives among which find a good and feasible solution. To overcome this issue, several representations on a finite space domain have been devised. We denote with Π the set of feasible floorplans, that is, the set of modules placements in the plane such that no two modules overlap, while we define Γ as the set of all the possible codes for a given representation. A representation is defined by means of two functions: the translating function $\tau : \Pi \rightarrow \Gamma$ that maps a feasible placement to its code and a realization function $\rho : \Gamma \rightarrow \Pi$ that,

given a code, produce a feasible placement. Since the number of feasible placements is infinite while there should be a finite number of possible codes, τ cannot be injective and ρ cannot be surjective.

Notice that neither τ nor ρ are required to be complete functions, they can also be partially defined[1]. In literature several representations have been proposed [19–29] with ρ functions defined on different $\Delta \subseteq \Pi$. Depending on the set of representable floorplans Δ, we can have different type of floorplans [30]. From general to specific we have:

general floorplans: all the feasible floorplans in the space [20];

LB-compacted floorplans: the feasible floorplans in the space such that no module can be moved bottom or left with respect to a containing rectangle [21];

mosaic floorplans: obtained from the dissection of a rectangle in which no line crossings are allowed and such that all the dissections include exactly one module [22];

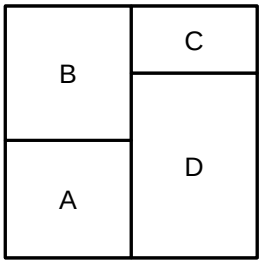
slicing floorplans: obtained recursively dividing a rectangle using horizontal and vertical lines and such that all the divisions contain exactly one module [23].

Among the set of possible floorplans, the following strict inclusions hold:

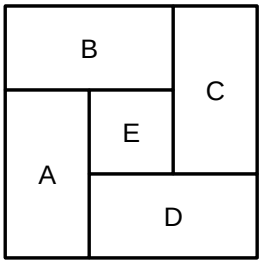
$$slicing \subsetneq mosaic \subsetneq LB - compacted \subsetneq general \tag{3.1}$$

---

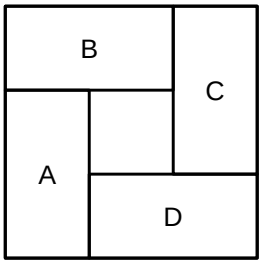
[1]A function is partial, or partially defined, if not all the elements in its domain have an image.



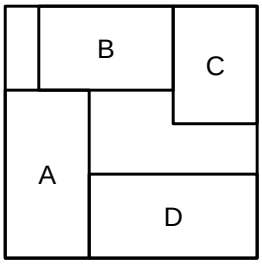
(a) Slicing floorplan.



(b) Non slicing, mosaic floorplan.



(c) Non mosaic, LB-compacted floorplan.



(d) Non LB-compacted, general floorplan.

Figure 9: Examples of floorplan types

Figure 9 shows an example for each type of floorplan previously defined, the capital letters within the rectangles denote the presence of a module. Floorplan 9a is slicing since it can be obtained recursively dividing a rectangle with horizontal and vertical lines, while Figure 9b shows a mosaic floorplan having a wheel structure that cannot be represented as a slicing floorplan. Figure 9c represents a LB-compacted floorplan in which no module can be moved left or bottom, furthermore the empty area in the middle cannot be represented by a mosaic floorplan. A general floorplan is shown in figure 9d, notice that modules B and C can be moved left and bottom respectively, thus the floorplan is not LB-compacted.

Another characterization of floorplan representations is given in [14]. If we denote with Ω the set of optimal floorplans with respect to some objective function, a representation is P-admissible if and only if it satisfies the following 4 conditions:

1. Γ is finite;
2. ρ is defined for all the codes in Γ;
3. ρ can be computed in polynomial time;
4. $\exists \gamma \in \Gamma \mid \rho(\gamma) \in \Omega$.

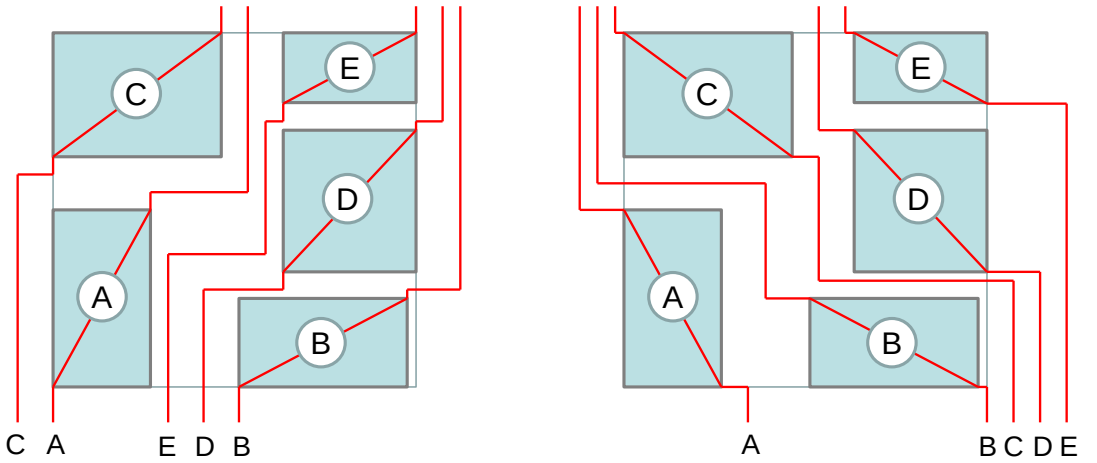
The underlying reason of using a P-admissible representation, is that it is well suited for meta-heuristics such as simulated annealing. Indeed, thanks to property 1) 2) and 3), the algorithm converges efficiently to a feasible solution and by 4) we know that an optimal solution could be found. Much of the effort in VLSI design has been done on finding P-admissible representations trading off the type of representable floorplans Δ to reduce the code space Γ. However, the flooplanning problem on FPGA devices having heterogeneous resource is quite different. First of all, the set of feasible placements Π is not infinite. In principle, one could enumerate all the possible placements, since the device matrix is finite and the reconfigurable regions must be placed at discrete coordinates. Secondly, and more important, the reconfigurable regions, as opposed to modules, cannot be placed at arbitrary positions in the space but have to cover specific resources at fixed positions within the FPGA. For these reasons and since, regarding FPGA floorplanning, feasibility is often an issue, it is advisable a representation that consider the overall generality of possible placements. One of the most simple and

elegant P-admissible representations for general floorplans is the *sequence-pair* representation, introduced in [19] and later in [14]. In the following subsection we describe the sequence-pair representation and how it can be exploited for floorplanning on partially-reconfigurable FPGA devices with heterogeneous resources.

3.1.1 Sequence pair representation

Given a set of M modules, a sequence pair is a floorplan representation defined by means of two sequences each containing a permutation of the modules. More formally, the set of codes of the representation is $\Gamma = (pair_1, pair_2)$ where $pair_1, pair_2$ are permutations of M. For instance, considering $M = \{A, B, C, D\}$ a sequence pair can be $(\langle A, B, D, C\rangle, \langle D, A, C, B\rangle)$. In order to fully characterize the representation we also need to show how to compute τ and ρ.

First of all, we consider the translation τ from a general floorplan to the corresponding sequence pair, this computation is referred as *gridding* within [14]. For each module of the floorplan, we draw the so called *positive step-line* and *negative step-line*. The positive step-line of a module $x \in M$ consists of three lines, namely: the *down-left step-line*, the principal diagonal of x and the *up-right step-line*. The up-right step-line starts at the up right corner of the module and goes up and right alternatively, while the down-left step-line is similarly defined starting from the bottom left corner of x. The positive step-line of x cannot cross other modules and other positive step-lines. Each positive step-line is identified by the name of the module that it traverses. Considering the identifiers of the positive step-lines from left to right, we obtain the first pair of the sequence. Independently from the positive step-lines, we can also draw the negative step-lines. The negative step-line of a module x consists again of three lines:



(a) positive step-lines: $pair_1 = \langle C, A, E, D, B \rangle$ (b) negative step-lines: $pair_2 = \langle A, B, C, D, E \rangle$

Figure 10: From a floorplan to its sequence pair representation

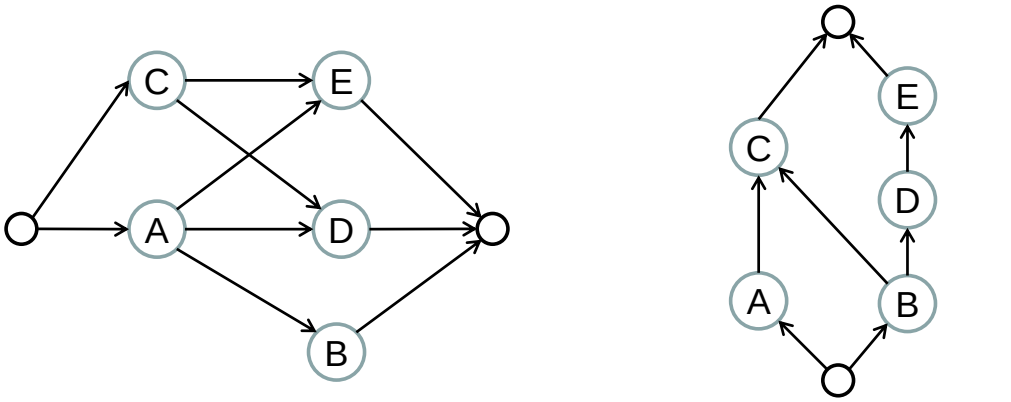
the *left-up step-line*, the secondary diagonal of x and the *right-down step-line*. The drawing of these lines is performed similarly but alternating left-up steps and right-down steps. Since also the negative step-lines do not cross each other, we can order them from left to right and obtain the second pair of the sequence from the lines identifiers. An example of gridding is shown in Figure 10 together with the corresponding sequence pair. Since gridding can be performed for each floorplan, the function τ is defined for all the possible sequence pairs.

The second function that we are going to define is ρ, it maps a sequence pair to a floorplan by inducing the topological relations between the modules. The geometrical relation between modules $x, y \in M$ depends on their relative positions, or indexes, within the two pairs of the sequence:

$$
\begin{aligned}
(\langle ..., x, ..., y, ... \rangle, \langle ..., x, ..., y, ... \rangle) &\implies x \text{ at the left of } y \\
(\langle ..., x, ..., y, ... \rangle, \langle ..., y, ..., x, ... \rangle) &\implies x \text{ above } y
\end{aligned}
\tag{3.2}
$$

For each couple of modules there can be only 4 possible relative orderings within the sequence pair. Two of these orders are given in Equation 3.2, while other two can be obtained by exchanging the names of the modules. Every order states a different relation between the two modules, namely: x at the left of y, x above y, y at the left of x, y above x. Regardless of the order in which the two modules appear, the sequence pair guarantees that they do not overlap. Starting from the *at the left of* relation and from the *above* relation, it is possible to construct the horizontal and vertical graphs respectively. In these graphs, the nodes are the modules, while the directed edges represent a horizontal or vertical order relation. An example of constraints graphs for the sequence pair $(\langle C, A, E, D, B \rangle, \langle A, B, C, D, E \rangle)$ is shown in Figure 11. Transitive edges have not been drawn for simplicity, while sink and source nodes have been inserted.

Both the graphs constructed starting from the sequence pair are guaranteed to be acyclic [14], thus every sequence pair produce a feasible floorplan in terms of the geometrical relations among modules. From the constraints graphs, it is possible to compute the x and y coordinates of all the modules, using a longest path algorithm over the horizontal and vertical graphs respectively. Other more efficient approaches have been devised to generate the floorplan from



(a) Horizontal-constraint graph (b) Vertical-constraint graph

Figure 11: Constraints graphs derived from a sequence pair

the sequence pair, such as [31] and its enhancement [32] that do not generate the constraints graphs but exploit Longest Common Subsequences (LCSs).

Regarding floorplanning on partially-reconfigurable FPGA with heterogeneous resources, it is not so straight forward to produce a feasible floorplan starting only from the sequence pair. The problem is to decide the width, height and position of the reconfigurable regions within the device in order to cover the required resources. However, with some modifications, sequence pair can still be successfully used [16]. The idea is to exploit the sequence pair to keep track of the geometrical relations between modules so that no overlaps occur. Then, new techniques must be devised to produce feasible and good floorplans starting from the sequence pair constraints. Notice that even if a fixed constraint graph is given, there still can be a lot of different floorplans on the FPGA with different costs in term of the objective function to be optimized. One of the two approaches that we are going to propose, starts from a heuristic

solution given in terms of a sequence pair and find the optimal floorplan under the specified geometrical constraints.

3.2 Related work

In literature, various floorplanners for FPGAs able to handle different features of commercial devices have been proposed. Subsection 3.2.1 presents floorplanners that deal with static placement without considering reconfiguration capabilities. Subsection 3.2.2 describes floorplanners that take into account also the time domain, while subsection 3.2.3 presents floorplanners aware of the FPGA architecture considering both non homogeneous distributions of heterogeneous resources and PR constraints.

3.2.1 Static floorplanners

A first class of algorithms focuses on static placement such as [33] and [34]. The solution proposed in [33] is based on the slicing tree representation [23] that is perturbed using simulated annealing to find a good floorplan. At each iteration of the annealer, a post processing step is performed trying to meet the resource requirements by modifying the rectangular shapes of the regions. The approach deals with heterogeneous resources and considers resource vectors containing CLBs, BRAMs and DSPs. Unfortunately, the proposed solution relies on a regular device structure in which even if the resources are of different types, they are organized in a repetitive pattern and are homogeneously distributed within the FPGA. Modern FPGAs are far from having such regular structures and this restrict the applicability of the approach to less recent devices.

The method proposed in [34] is a two steps algorithm based on the formulation introduced in [33]. 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 shapes of the previously placed modules to guarantee the feasibility of the solution. However, if PR is taken into account, the shapes of the reconfigurable regions cannot deviate from rectangular shapes. This prevent [33] from using the post processing step, while regarding [34], their final solution unlikely meets the PR constraints as required by [5].

3.2.2 Reconfiguration-aware floorplanners

Another class of approaches introduces reconfiguration and takes into account the time domain. One of the most important contributions in this area is [35]. The algorithm relies on an extension of the Transitive Closure subGraph (TCG) representation [26] to deal with the temporal dimension. A 3-Dimensional Transitive Closure sub-Graph (3D-subTCG) representation is devised to take into account the reconfiguration process and the precedence relations between the modules on the device. However, the types of resources considered by this approach are restricted to logic blocks while other resources are ignored.

Other works such as [36] and [37] aim at reconfiguring the smallest amount of the system between two subsequent configurations. The work presented in [36] exploits a multi-layer sequence pair representation to solve the floorplanning problem, in which the types of resources considered are still restricted to basic blocks. On the other hand, the algorithm presented in [37] takes into account heterogeneous resources but only having a homogeneous distribution

within the device. Both approaches, as stated in [16], are not compliant with the PR design flow, since they do not guarantee identical organizations in terms of number, shape and position of reconfigurable regions at different configurations.

Among the works in this category, [38] and its enhancement [39] are worth mentioning. The work proposed in [39] 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.

3.2.3 Architecture-aware floorplanners

Both PR constraints and an accurate description of the heterogeneous resource distribution are considered in [16] and [11]. The algorithm [16] uses a floorplan representation whose set of codes Γ is defined by means of a sequence pair and a height vector. The sequence pair is needed to enforce the geometrical relations between reconfigurable regions, while the height vector sets for each reconfigurable region its height. The values of the height vector are positive integers and are selected according to the PR constraints of the device of choice. The search space Γ is explored with simulated annealing perturbing the floorplan representation. For each code $\gamma \in \Gamma$ the realization function ρ producing the floorplan $\rho(\gamma)$, is computed in three steps: firstly, ρ

obtains the vertical and horizontal constraints graphs from the sequence pair; secondly, using the vertical graph and the height vector, it computes the y coordinates of the regions and finally, greedy widens and moves the regions on the x axis to meet the resource requirements without violating the horizontal constraints graph precedences. Since not all the possible combinations of sequence pairs and height vectors lead to a feasible solution, [16] also implements a constraint violation term in the objective function, to guide the annealer 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 representation of a solution in [16], described by means of its sequence pair and height vector, can be, in general, mapped to a vast number of different floorplans on the device. The function ρ is computed using a fast greedy approach that does not give any guarantee on the quality of the floorplan. For this reason, the representation is not P-admissible, since it does not guarantee the existence of a code γ such that the floorplan $\rho(\gamma)$ is optimal with respect to a given objective function. Thus, the annealer explores in general a sub-optimal solution space. On the other hand, computing ρ to optimality would not be convenient, since the computation has to be done at each iteration of the simulated annealing algorithm.

The work presented in [11], similarly to [16], produces floorplans that satisfy PR constraints and takes into account non homogeneous resource distributions. Their approach characterizes the FPGA device in terms of tiles (minimal reconfigurable units). 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 wirelength 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.

3.3 Limits of current approaches

Floorplanning on modern partially-reconfigurable FPGAs requires the floorplanner to meet both PR constraints and to cope with non homogeneous resource distributions. Among the works in this area, the only ones that take into account both these aspects are [16] and [11]. The approach proposed in [16] proved to give better results in terms of total wirelength with

respect to [34] and [39], while [11] produced a better floorplan in terms of resource usage with respect to [39].

Although [16] and [11] give better results with respect to previous approaches, they still look for a solution in a sub-optimal or incomplete search space respectively. Furthermore, none of the algorithms give information about the quality of the solution found with respect to the optimum.

We propose two novel 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. Our algorithms let the designer decide to what extent optimize each term in a set of different metrics, giving also the possibility to modify and extend our models to take into account other objectives different from the ones presented in this work. The floorplans generated by our methodologies are compliant with PR design flow and can be used for FPGAs having non homogeneous resource distributions.

Table I, an update of the one presented in [16], recaps the features of state-of-the-art floorplanners.

TABLE I: COMPARISON OF STATE-OF-THE-ART FLOORPLANNERS

	[33]	[34]	[35]	[36]	[37]	[38, 39]	[11, 16]
Resource distribution-aware		✓					✓
Reconfiguration-aware			✓	✓	✓	✓	✓
Compliant with PR						✓	✓
Optimize interconnections	✓	✓	✓	✓	✓		✓
Considers IO pins						✓	✓
Customizable objective function							
Reaches the optimum							

CHAPTER 4

PROPOSED FLOORPLANNER

The first algorithm that we propose, Heuristic-Optimal Flooplanner (HOF), improves the quality of a solution produced by a heuristic such as [16] or [11]. HOF 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 Gurobi [40], and the best outcome is considered. HOF 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 [16].

The second approach that we propose, Optimal Flooplanner (OF), describes the entire problem using a MILP formulation that ensures non overlapping of reconfigurable regions without the need of a fixed sequence pair. OF 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 HOF 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. OF gives better results than HOF but is in general quite time consuming; hence the designer can select which of the two algorithms to adopt depending on his needs.

The rest of the chapter is organized as follows: Section 4.1 shows a suitable representation of the FPGA device that will be used within the algorithms, subsequently, Section 4.2 presents the MILP model used in OF and HOF, then, Section 4.3 introduces some additional constraints whose goal is to provide the MILP solver more information about the structure of the problem, finally, Section 4.4 concludes the chapter with some final considerations on the proposed model.

4.1 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.

4.1.1 Matrix reduction

The first step toward the simplification of the problem is to reduce the device matrix granularity as done in [11]. Instead of considering the single resource on the chip, we can just take into account, without loss of generality, the tiles as the minimal area units. Using a matrix whose integer coordinates address tiles reduces both the solution space and guarantees PR constraints, since regions are placed using integer coordinates.

4.1.2 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 matrix 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

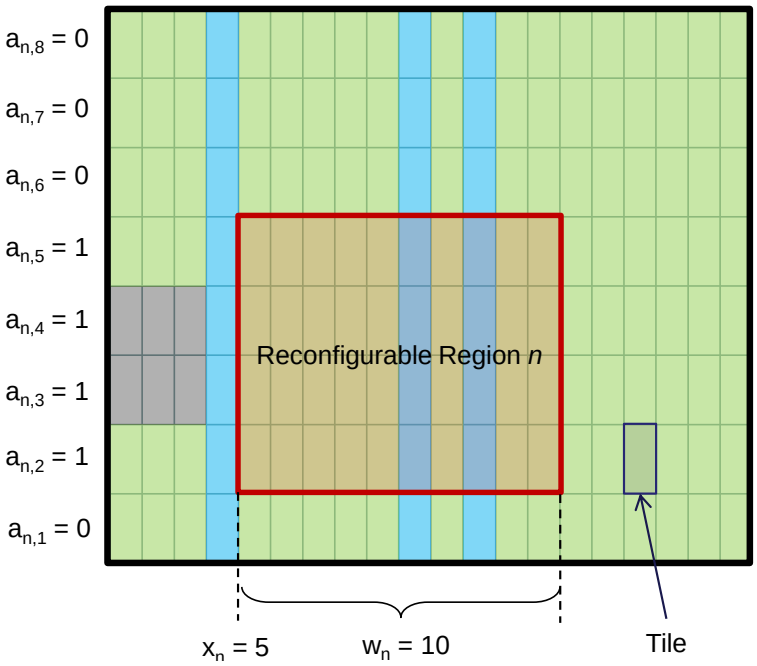


Figure 12: Variables values of a region placed within the device

be described using a *matrix of tiles* with 8 rows and 62 columns. This suggests to discretize the device on the y axis. Figure 12 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.

4.1.3 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:

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 13. 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*.

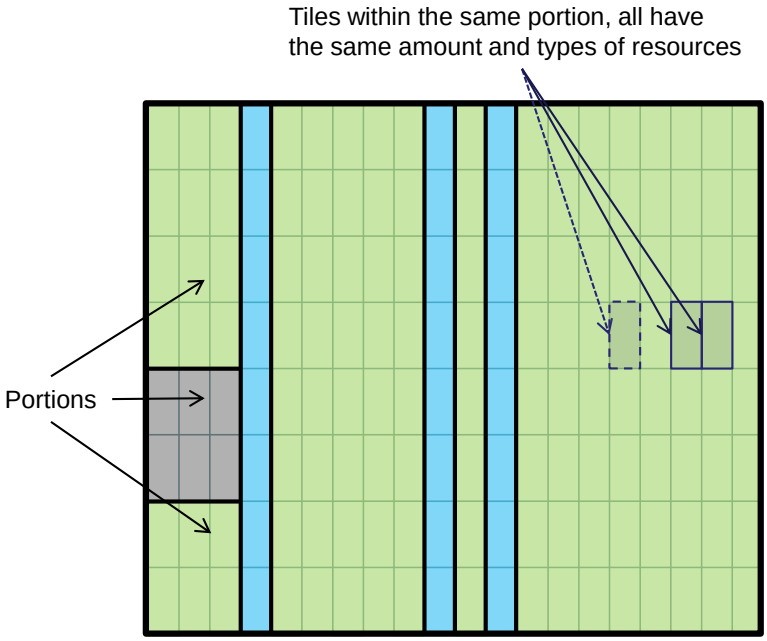


Figure 13: Partitioning of the FPGA into portions

Even though partitioning resembles the kernel construction shown in [11], 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.

4.2 MILP model

In this section we present the MILP models used by HOF and OF 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.

4.2.1 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:

$P :=$ set of portions;

$F :=$ set of forbidden areas (F $\subset$ P);

$R :=$ 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, 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 HOF 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.

4.2.2 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 14).

What follows are the variables shared by both formulations for HOF and OF:

$a_{n,r} :=$ binary variable set to 1 if and only if region n occupies row r;

$x_n :=$ integer positive variable (≥ 1) representing the leftmost 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;

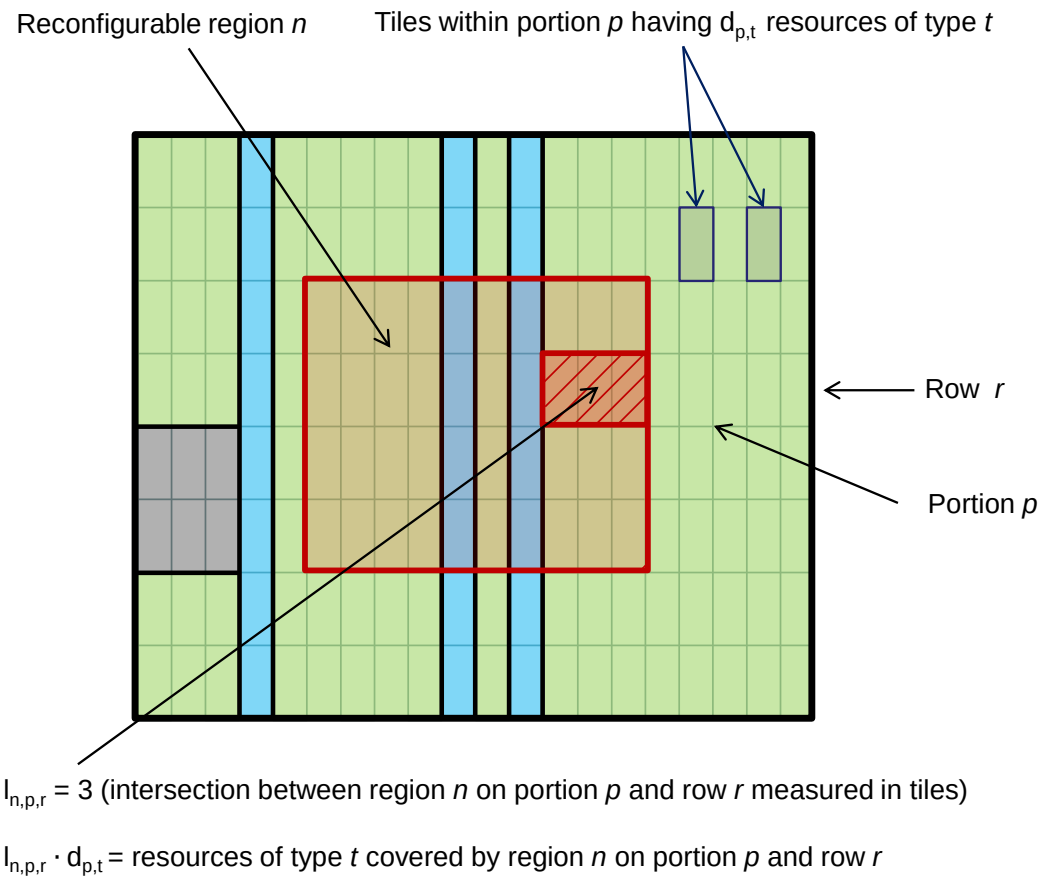


Figure 14: Computation of covered resources

$l_{n,p,r}$:= 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 OF requires another set of binary variables in order to avoid overlapping between regions. This is the reason why the MILP model of OF is much harder to solve than the one for HOF, 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 $n2$;

4.2.3 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 $r3$:

$$\begin{gathered} \forall n \in N, r1, r2, r3 \in R \mid r3 > r2 > r1 : \\ a_{n,r2} \geq a_{n,r1} + a_{n,r3} - 1 \end{gathered} \tag{4.1}$$

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

$$\begin{gathered} \forall n \in N, p \in P : \\ x_n + w_n - 1 \geq x1_p \cdot k_{n,p} \\ x_n \leq x2_p + (1 - k_{n,p}) \cdot (maxW - x2_p) \end{gathered} \tag{4.2}$$

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

$$\forall n \in N : x_n + w_n \leq maxW \tag{4.3}$$

Height definition with respect to the single rows occupied:

$$\forall n \in N : h_n = \sum_{r \in R} a_{n,r} \tag{4.4}$$

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:

$$\begin{aligned} &\forall n \in N, r \in R : yl_n \leq |R| - a_{n,r} \cdot (|R| - r) \\ &\forall n \in N, r \in R : yh_n \geq a_{n,r} \cdot r \end{aligned} \tag{4.5}$$

Vertical gap. Fixes the gap between yl_n and yh_n with respect to the height of the region. This constraint together with Equation 4.5, sets the variables to the correct integer values:

$$\forall n \in N : yh_n - yl_n + 1 = h_n \tag{4.6}$$

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:

$$\begin{aligned} &\forall n \in N, p \in P, r \in R \mid rp_{p,r} = 1 : \\ &l_{n,p,r} \leq a_{n,r} \cdot (x2_p - x1_p + 1) \\ &l_{n,p,r} \leq k_{n,p} \cdot (x2_p - x1_p + 1) \\ &l_{n,p,r} \leq w_n \\ &l_{n,p,r} \leq x_n + w_n - k_{n,p} \cdot x1_p \\ &l_{n,p,r} \leq x2_p - x_n + 1 + (1 - k_{n,p}) \cdot (maxW - x2_p) \end{aligned} \tag{4.7}$$

the inequalities in Equation 4.7 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 Equation 4.7, fixes the amount of intersection to the correct integer value):

$$\begin{gathered} \forall n \in N, r \in R : \\ \sum_{p \in P | rp_{p,r}=1} l_{n,p,r} \geq w_n - (1 - a_{n,r}) \cdot maxW \end{gathered} \tag{4.8}$$

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

$$\begin{gathered} \forall n \in N, p \in F, r \in R \mid rp_{p,r} = 1 : \\ l_{n,p,r} = 0 \end{gathered} \tag{4.9}$$

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

$$\begin{gathered} \forall n1, n2 \in N \mid n1 < n2 : \\ x_{n1} + w_{n1} \leq x_{n2} + g_{n1,n2} \cdot maxW \end{gathered} \tag{4.10}$$

4.2.4 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:

$$\begin{gathered} \forall n \in N, t \in T : \\ \sum_{p \in P, r \in R \mid rp_{p,r}=1} l_{n,p,r} \cdot d_{p,t} \geq c_{n,t} \end{gathered} \tag{4.11}$$

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

$$\begin{gathered} \forall n1, n2 \in N \mid pair1_{n1} < pair1_{n2} \wedge pair2_{n1} < pair2_{n2} : \\ x_{n1} + w_{n1} \leq x_{n2} \end{gathered} \tag{4.12}$$

$$\forall n1, n2 \in N \mid pair1_{n1} < pair1_{n2} \wedge pair2_{n1} > pair2_{n2} : \\ yl_{n1} \geq yl_{n2} + h_{n2} \tag{4.13}$$

Instead, regarding OF, 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:

$$\forall r \in R, n1 \in N, n2 \in N \mid n1 < n2 : \\ x_{n1} \geq x_{n2} + w_{n2} - (3 - g_{n1,n2} - a_{n1,r} - a_{n2,r}) \cdot maxW \tag{4.14}$$

4.2.5 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 wirelength (WL_{cost}): an estimation of the overall wirelength measured using the commonly adopted HPWL as in [16]. Both connections to the IO and between reconfigurable regions are taken into account. HPWL assumes the pins concentrated in the center of the regions/IO ports, so the wirelength 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:

$$\min \left\{ q_1 \cdot \frac{WL_{cost}}{WL_{max}} + q_2 \cdot \frac{P_{cost}}{P_{max}} + q_3 \cdot \frac{R_{cost}}{R_{max}} \right\} \tag{4.15}$$

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 matrix 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 some new constraints.

First of all, we have to compute the regions centroids with respect to the original device matrix:

$$\begin{aligned} &\forall n \in N : \\ &cx_n = tileW \cdot (x_n + w_n/2) \\ &cy_n = tileH \cdot (yl_n - 1 + h_n/2) \end{aligned} \tag{4.16}$$

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):

$$
\begin{aligned}
&\forall n1 \in N, n2 \in N \mid n1 \neq n2 : \\
&dcx_{n1,n2} \geq cx_{n1} - cx_{n2} \\
&dcx_{n1,n2} \geq cx_{n2} - cx_{n1} \\
&dcy_{n1,n2} \geq cy_{n1} - cy_{n2} \\
&dcy_{n1,n2} \geq cy_{n2} - cy_{n1}
\end{aligned}
\tag{4.17}
$$

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):

$$
\begin{aligned}
&\forall io = (n, iox, ioy, b) \in IO : \\
&dpx_{io} \geq cx_n - iox \\
&dpx_{io} \geq iox - cx_n \\
&dpy_{io} \geq cy_n - ioy \\
&dpy_{io} \geq ioy - cy_n
\end{aligned}
\tag{4.18}
$$

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

$$
\begin{aligned}
WL_{cost} = &\sum_{(n1,n2,b) \in C} ((dcx_{n1,n2} + dcy_{n1,n2}) \cdot b) + \\
&\sum_{io=(n,iox,ioy,b) \in IO} ((dpx_{io} + dpy_{io}) \cdot b)
\end{aligned}
\tag{4.19}
$$

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

$$P_{cost} = 2 \cdot \sum_{n \in N} (w_n \cdot tileW + h_n \cdot tileH) \tag{4.20}$$

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:

$$R_{cost} = \sum_{n \in N, t \in T} waste_{n,t} \cdot rc_t \tag{4.21}$$

where:

$$waste_{n,t} := \sum_{p \in P, r \in R | rp_{p,r}=1} (l_{n,p,r} \cdot d_{p,t}) - c_{n,t} \tag{4.22}$$

4.3 Formulation refinement

The model described so far would be sufficient to be used within a MILP solver such as [40] or [41]. In general, regarding MILP, there can be a lot of different models describing the same problem correctly, but not all the models have the same quality and are solved efficiently using state of the art solvers. There are several techniques that can be used to enhance the goodness of a MILP model [42]. Here we try to better characterize the solution space of the problem adding

some cuts. A cut is a constraint that remove solutions with respect to the Linear Programming (LP) relaxation[1] while preserving feasible alternatives regarding the MILP formulation. Hence, if the number of cuts is not too high and they make the model description tighter, the problem can be solved more efficiently. We consider two types of cuts: Section 4.3.1 takes into account cuts derived from resource requirements, while Section 4.3.2 presents several cuts obtained considering the geometry of the regions.

4.3.1 Resource cuts

Even though an FPGA matrix can contain tiles having different numbers and types of resources, it is often the case that in real devices, only a small amount of different tiles do exist. Moreover, tiles usually contain one single type of resource with a fixed availability. In this scenario, for instance, it makes sense to talk of DSP tiles and saying that a DSP tile always contain z DSPs. This information about the device can be exploited to tighten the MILP formulation. If we know that a resource of type $t \in T$ is present only in tiles in which z resources of type t are present, then for sure a reconfigurable region can only cover this resource in multiples of z. For example consider an FPGA in which CLB resources are contained only in tiles where 20 CLBs are present. If we have a region requiring 35 CLBs then it has to cover at least 2 CLB tiles and needs at least 40 CLBs. More formally, we define TF, the set of resource types always having a fixed availability within all the tiles, as follows:

---

[1] A LP relaxation of a MILP model is a new model in which the integrity constraints of the variables are removed. To give an example, a binary variable in the MILP model would be considered as a real variable in the closed interval $[0, 1]$ within the relaxation.

$$TF := \{t \in T \mid \forall p \in P : d_{p,t} = 0 \vee d_{p,t} = z_t\} \tag{4.23}$$

Where z_t represents the fixed availability of resource $t \in TF$ within all the tiles. Now, for all the reconfigurable regions and resource types in TF we can add a resource constraint tighter than Equation 4.11:

$$\begin{gathered} \forall t \in TF, n \in N : \\ \sum_{p \in P, r \in R \mid rp_{p,r}=1} l_{n,p,r} \cdot d_{p,t} \geq \lceil c_{n,t}/z_t \rceil \cdot z_t \end{gathered} \tag{4.24}$$

Notice that often we have $T = TF$ and each tile containing only one type of resource. In this situation, another approach would be to simply consider the resource requirements of the reconfigurable regions in terms of tiles instead of single resources [11].

Modern FPGAs contain, together with ordinary reconfigurable resources, also hard processors that break the regular columnar organization of the device. In this situation there can be rows containing different numbers of overall resources and it makes sense to define the parameter:

$resRow_{r,t} :=$ number of resources of type t available in row r;

Now, since not all the rows have the same amount of resources, there can be combinations of selected rows that cannot satisfy the resource requirements of a region regardless of its width. Thus we can remove these invalid selections with the following cuts:

$$\forall n \in N, t \in T : \sum_{r \in R} resRow_{r,t} \cdot a_{n,r} \geq c_{n,t} \tag{4.25}$$

4.3.2 Geometrical cuts

Another type of cuts that can be added to our MILP formulation refer to minimal geometrical constraints of the reconfigurable regions to be placed. Often the most common resources on an FPGA device are CLBs, using this information we can derive constraints on the shapes that the reconfigurable regions can assume.

To introduce our geometrical cuts, we first need the following parameters:

$maxD_t :=$ maximum number of resources of type t available in a single tile of the FPGA matrix;

$maxResRow_t :=$ maximum number of resources of type t available in a single row of the FPGA matrix ($= \max_{r \in R} resRow_{r,t}$);

$maxResCol_t :=$ maximum number of resources of type t available in a single column of the FPGA matrix;

The previous parameters can be easily computed starting from the FPGA portions before solving the problem. If a reconfigurable region, for a given resource $t \in T$, needs more than the resources available in a column, then it must span multiple rows. Similarly, the reasoning also holds if we exchange rows and columns in the previous statement. With this information we can define two cuts in terms of minimal width and height required by a region.

Width lower bound:

$$\begin{aligned} &\forall n \in N : \\ &w_n \geq max_{t \in T} \lceil c_{n,t} / maxResCol_t \rceil \end{aligned} \tag{4.26}$$

Height lower bound:

$$\begin{aligned} &\forall n \in N : \\ &h_n \geq max_{t \in T} \lceil c_{n,t} / maxResRow_t \rceil \end{aligned} \tag{4.27}$$

Notice that in both Equation 4.26 and Equation 4.27 the right terms can be computed and replaced with known numbers.

By knowing the resource requirements we are also able to estimate the amount of area, in terms of tiles, needed by each reconfigurable region. In general, a simple lower bound for the area required by a region can be computed as follows:

$$leastArea_n := max_{t \in T} \lceil c_{n,t} / maxD_t \rceil \tag{4.28}$$

Equation 4.28 is valid for any FPGA matrix. However, as usually happens, if the FPGA matrix consists of tiles having at most one type of resource, then, the least area bound can be improved considering the contributes of all the types of resources independently:

$$\begin{gathered} \forall n \in N : \\ leastArea_n := \sum_{t \in T} \lceil c_{n,t} / maxD_t \rceil \end{gathered} \tag{4.29}$$

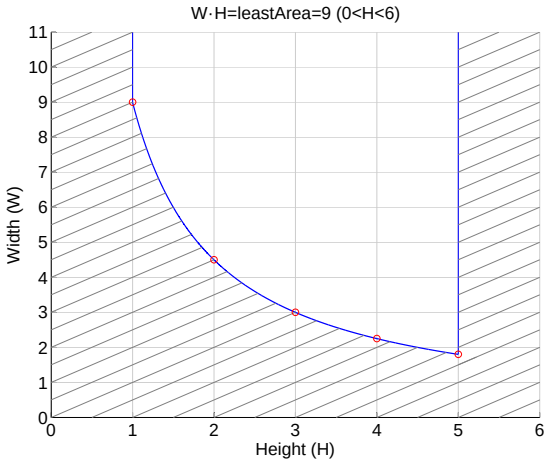
The region least area can be exploited to give the solver more information about the width and height that the region can assume. Since a reconfigurable region have a rectangular shape, the area can be simply computed multiplying the width and the height. Hence, the following non linear cuts can be derived:

$$\begin{gathered} \forall n \in N : \\ w_n \cdot h_n \geq leastArea_n \end{gathered} \tag{4.30}$$

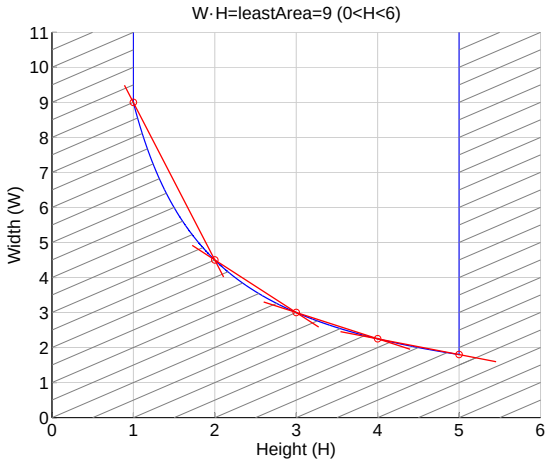
Even though Equation 4.30 is not linear, it describes a convex region in a 2-dimensional space and can be approximated with a set of linear inequalities as done in [43]. Furthermore, we are only interested in integer values of h_n, thus we just need to approximate exactly a set of discrete points, from now on referred as extreme points. To clarify the explanation, consider an FPGA of 5 rows numbered from 1 to 5 and a reconfigurable region having a least area of 9 tiles. Figure 15a shows the space of possible width and height combinations, the hyperbola represents the least area bound, while the two vertical lines derive directly from the geometrical constraints of the device. The points shown in red are the extreme points, they represent the least width of the region when the height is fixed. Figure 15b shows, with red lines, a simple set of constraints achieving the goal of describing the space of width and height combinations. Each of the cuts is an half-space whose frontier contains two adjacent extreme points. The linear inequalities defining the width-height cuts are as follows:

$$\begin{aligned} &\forall n \in N, r \in R \mid r < |R| : \\ &w_n \geq leastArea_n/r + (leastArea_n/(r+1) - leastArea_n/r) \cdot (a_n - r) \end{aligned} \tag{4.31}$$

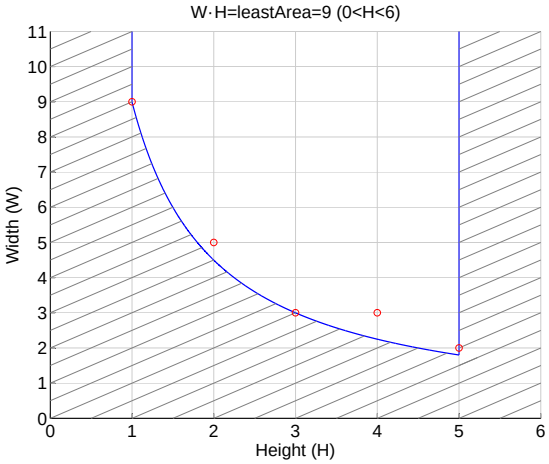
Equation 4.31 can be improved, indeed also the values of w_n must be integer because the regions can only contain complete tiles (PR requirement). We can exploit this information to tighten the previous cuts. The first step is to round up the extreme points with respect to the width axes to get integer extreme points (Figure 15c). The second step is to find a set of linear



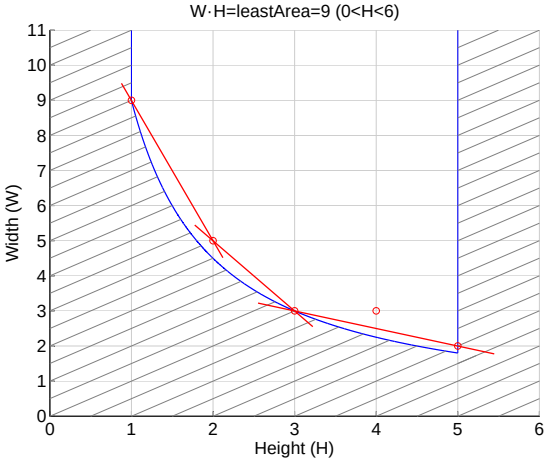
(a) Width-height extreme points.



(b) Width-height region approximation.



(c) Width-height extreme integer points.



(d) Width-height region integer approximation.

Figure 15: Width-height cuts

cuts describing the smaller region[1] containing all the integer extreme points without removing any feasible solution. In our example these cuts are shown in Figure 15d. Notice that the integer extreme point at coordinates $(4, 3)$ cannot be in the frontier of any of the constraints, otherwise one of the other integer extreme points would be excluded from the convex region. Notice also that the number of integer extreme points is exactly $|R|$.

A general method to generate these tighter cuts, is to consider all the half-spaces having at least two integer extreme points on the frontier and including all the other integer extreme points. In other words, for each couple of extreme points, we generate the associated cut only if it does not exclude other feasible solutions. Formally, the tighter width-height cuts can be defined for each reconfigurable region as follows:

$$\begin{aligned} &\forall n \in N, r1 \in R, r2 \in R \mid (r2 > r1) \wedge (\forall r \in R : extrW_{n,r} \geq cut_{n,r1,r2}(r)) : \\ &w_n \geq cut_{n,r1,r2}(h_n) \end{aligned} \tag{4.32}$$

Where $cut_{n,r1,r1}(h)$ is a linear function passing through the integer extreme points at rows $r1$ and $r2$, while $extrW_{n,r}$ represents the width of the extreme point at row r. Formally we can define them by the equations:

---

[1]The region described is convex because it is obtained from the conjunction of linear inequalities or, equivalently, from the intersection of half-spaces.

$$cut_{n,r1,r2}(h) = extrW_{n,r1} + \frac{extrW_{n,r2} - extrW_{n,r1}}{r2 - r1} \cdot (h - r1) \tag{4.33}$$

$$extrW_{n,r} = \lceil leastArea_n / r \rceil \tag{4.34}$$

Using the tight width-height cuts with Gurobi solver [40], we experienced a great improvement in terms of execution time. On average the solver, provided with these cuts, was able to find and certify the optimal solution three time faster.

4.4 Model remarks

While developing our model, we stressed the importance of reducing the amount of required integer variables because these are the ones that make the problem hard to be solved. Ideally, if we were able to completely avoid the use of integer variables while keeping a limited amount of constraints, we would be able to solve the problem in polynomial time [44, 45], however, this is probably not feasible since in Section 2.2 we proved that the decisional version of our problem is NP-complete. A first simple way to understand the sources of complexity of our model is to check where the integer variables are needed. Regarding OF we need:

- $2 \cdot |N|$ integer variables to describe the widths and leftmost positions of the regions;
- $|R| \cdot |N|$ integer (binary) variables to consider the rows occupied by the regions;
- $|P| \cdot |N|$ integer (binary) variables to check the intersection between portions and regions;

- $|N| \cdot (|N| - 1)/2$ integer (binary) variables to ensure the non overlapping constraints between regions.

Notice that the number of integer variables grows quadratically in terms of the regions to place and linearly with respect to the complexity of the device, that is related to the number of portions and rows. These remarks agree with the hint on the sources of complexity discussed in Section 2.2, where we exploited an arbitrary number of reconfigurable regions and an arbitrary complex device to prove the NP-completeness of the problem. On the other hand, HOF is much easier to solve than OF, mainly because its number of integer variables grows linearly also with respect to the reconfigurable regions to place, since the non overlapping constraints are already guaranteed by the fixed geometrical relations derived from the sequence pair.

Nevertheless, when solving MILP models, the number of integer variables required is not the the sole aspect to consider. Indeed, modern solvers take advantage of the LP relaxation of the problem and a tight MILP formulation can improve the execution time as stated in Section 4.3. The proposed MILP model is not minimum in terms of integer variables required. We were able to devise a different OF formulation whose number of integer variables was linear with respect to the regions to place and logarithmic to the dimension of the device. The idea was to describe the positions of the regions using a binary representation and exploit a large set of real variables to state if a region occupies a specific tile. A set of constraints binding the real variables to the integer ones were used to force the real variables to assume binary values, then, with some additional constraints on the real variables, it was possible to guarantee the non overlapping of regions and the resource requirements.

However, this alternative model required a much higher number of constraints while its LP relaxation was less tight than the one of the proposed model, so, in practice, even if the number of integer variables was reduced, the time required to find even a feasible solution was higher.

CHAPTER 5

EXPERIMENTAL RESULTS

In this chapter we present the results achieved by experimenting with OF and HOF algorithms. Section 5.1 describes the setting in which the experiments have been performed and how the MILP models are translated and solved. Section 5.2 performs a systematic campaign to test the performance of our algorithms with respect to [16]. Different problem instances are considered varying in terms of reconfigurable regions and resource usage, the achieved results are shown and a cost benefit analysis among OF and HOF is performed. Section 5.3 concludes the chapter considering a case study taken from [11] and comparing the resulting floorplans.

5.1 Experimental environment

All the experiments have been executed on a 2.2GHz Intel Core Duo processor under Linux. To solve the MILP models for OF and HOF we used the state-of-the-art solver Gurobi Optimizer 5.6.0 [40]. To exploit the full capability of the solver we enabled the multi-threading option setting the number of parallel threads to 2, while we leaved all the other parameters to their default values.

The MILP models of our experiments are the ones described in section 4.2 and include the additional cuts defined in Section 4.3. To ease the generation of the models, we defined all the constraints, objective and parameters using MathProg, a subset of the AMPL language. The models were then translated to the LP file format and solved with Gurobi.

5.2 Pseudo-random benchmark

In this section we experiment our approaches on a set of pseudo-randomly generated circuits, the goal of this test is to characterize the performance of the algorithms with respect to different parameters. Specifically, we take into account the number of reconfigurable regions and the device resource usage as parameters characterizing different problem instances. We consider the approach proposed in [16] both for comparison and to warm start the MILP solver.

Subsection 5.2.1 describes how the problem instances have been generated and shows the settings used within our models. In subsection 5.2.2 we present the results achieved, while subsection 5.2.3 perform a cost benefit analysis weighting time and floorplan quality among OF and HOF.

5.2.1 Problems generation and setup

An extended testing campaign has been performed on a Virtex-5 XC5VLX110T using the global wirelength as the metric of choice (i.e. weights of the objective function of the MILP models have been set to: $q_1 = 1.0$, $q_2 = 0.0$, $q_3 = 0.0$). 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%. To have a reasonable usage of heterogeneous resources, we ensured that from 3 to 7 reconfigurable regions required BRAMs, while from 1 to 2 required DSPs. The interconnections between regions and IOs have been randomly generated. We considered an interconnection probability between each couple of regions of $1/n$, where n is the number of

regions, and ensured a least number of interconnections to the IO ports. The interconnection bandwidths have been sampled from a uniform distribution with values between 5 and 40.

We compared our results with respect to the ones achieved by [16] on the same set of circuits and using the same objective function. We performed 10 executions of [16] on every circuit and the best result has been considered for comparison. HOF instead, selected the executions of [16] that did not differ by more than 10% from the considered [16] solution. For each sequence pair in the selected solutions, HOF re-optimized the problem and the best outcome represented the final result of HOF.

Concerning OF, we decided to warm start the solver using the best solution found by HOF 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 HOF still was less than starting the solver from scratch. Since HOF relies on [16], the overall execution time of HOF takes also into account the time spent by [16] to solve the problem. The same is true for OF since we provided an initial solution computed by HOF. Notice that OF does not require [16] 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 performing in parallel the 10 executions of [16] for a fair comparison to the multi-threaded MILP solver.

5.2.2 Results analysis

Table II reports the average wirelength reduction achieved by the two approaches with respect to [16]. The results are grouped with respect to different numbers of reconfigurable regions.

TABLE II: RESULTS WITH DIFFERENT NUMBERS OF REGIONS

# Regions	Average wirelength improvement w.r.t. [16]		Average execution time (sec)		
	HOF	OF	[16]	HOF	OF
5	6.99%	7.48%	10.9	12.9	56.0
10	7.59%	11.65%	23.8	45.3	1845.3
15	8.88%	20.06%	40.6	69.6	1869.7
20	5.47%	19.13%	64.9	83.3	1883.4
25	5.67%	21.97%	93.2	121.0	1921.0

Execution times of all the algorithms are also shown. For small numbers of reconfigurable regions HOF gives an improvement comparable to OF but using much less time. It was interesting to notice that with 5 reconfigurable regions, in 3 out of 4 instances OF proved the optimality of the solution found by HOF, 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 [16] and OF 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 wirelength improvement.

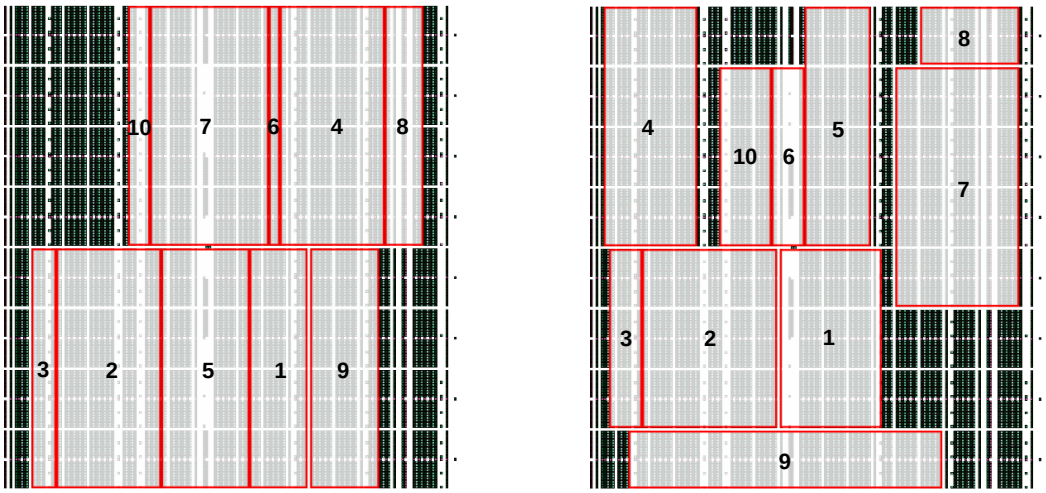
Table III perform a similar analysis but based on the device occupancy rate. The best improvements are achieved for an occupancy rate smaller than 80%, whereas the wirelength reduction diminishes as soon as the device gets more occupied and the slacks between regions are also reduced.

TABLE III: RESULTS WITH DIFFERENT DEVICE OCCUPANCY

Occupancy	Average wirelength improvement w.r.t. [16]		Average execution time (sec)		
	HOF	OF	[16]	HOF	OF
70%	8.51%	19.19%	47.0	89.2	1544.1
75%	5.49%	21.50%	46.8	62.7	1509.3
80%	6.20%	13.80%	46.7	59.2	1506.9
85%	7.48%	9.75%	46.3	54.6	1500.0

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 both [16] and [11].

In figures 16a and 16b we show the solutions found by [16] and OF on the same instance involving 10 reconfigurable regions with a 75% device occupancy. Even though the solution found by [16] is more regular, the one achieved by OF gives a 16% wirelength improvement for this specific problem. The improvement is obtained on one hand, by moving closer regions having an higher interconnection width such as 1 and 2, on the other hand, modifying the shape of regions to ease the interconnections (e.g. region 8 centroid is moved nearer to the required IO port at the top right side of the device).



(a) Floorplan produced by [16].

(b) Floorplan produced by OF.

Figure 16: Floorplans comparison on 10 reconfigurable regions

5.2.3 Cost benefit analysis

To guide the designer in the choice of the algorithm, a cost benefit analysis has been performed for HOF and OF. For each test instance, we computed the ratio between the wirelength improvement of OF with respect to HOF and the time overhead required by OF to achieve the result. This ratio has been normalized and reported in figure 17 for different rates of device occupancy and numbers of reconfigurable regions.

A green/yellow color indicates that the use of OF can be convenient and good improvements can be achieved with small efforts with respect to combinations of parameters that lie in the blue area of the graph.

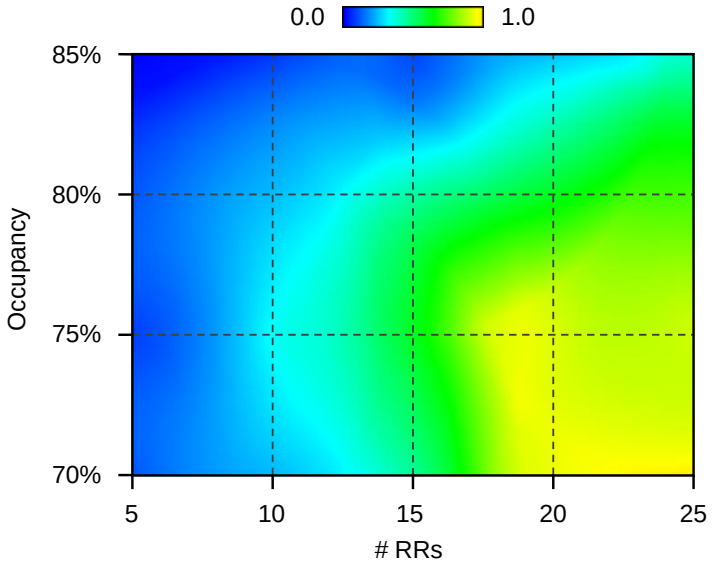


Figure 17: Normalized improvement over time overhead.

Figure 17 suggests that when the device occupancy is high or when the number of regions to place is small, HOF provides good improvements more efficiently than OF. On the other hand, when we are faced with a higher number of regions and the device occupancy rate is not higher than 80%, using OF could be convenient to achieve good quality floorplans.

5.3 Software defined radio case study

In this section we prove the effectiveness of our methodology experimenting it on a real case study taken from [11]. In Section 5.3.1 we present the design used for testing and the target FPGA device. Within Section 5.3.2 we show how the parameters of the MILP models have been set together with the objective function being optimized. Section 5.3.3 concludes the chapter comparing our floorplan to the one achieved by [11].

5.3.1 System design

The design considered is a SDR taken from [11]. The SDR chain consists of the following modules: matched filter, carrier recovery circuit, 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) and the number of type t resources required by a region is set to the maximum of the type t resources required by a modes assigned to it.

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. Each tile contains a fixed number of resources whose type is the type of the tile. Table IV reports the number and type of resources required by each reconfigurable region expressed in terms of tiles.

TABLE IV: 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
Decoder	12	1	0	462
Video Decoder	55	2	5	2180
Total	104	5	11	4202

As we can see from table IV 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 the each region needs to cover.

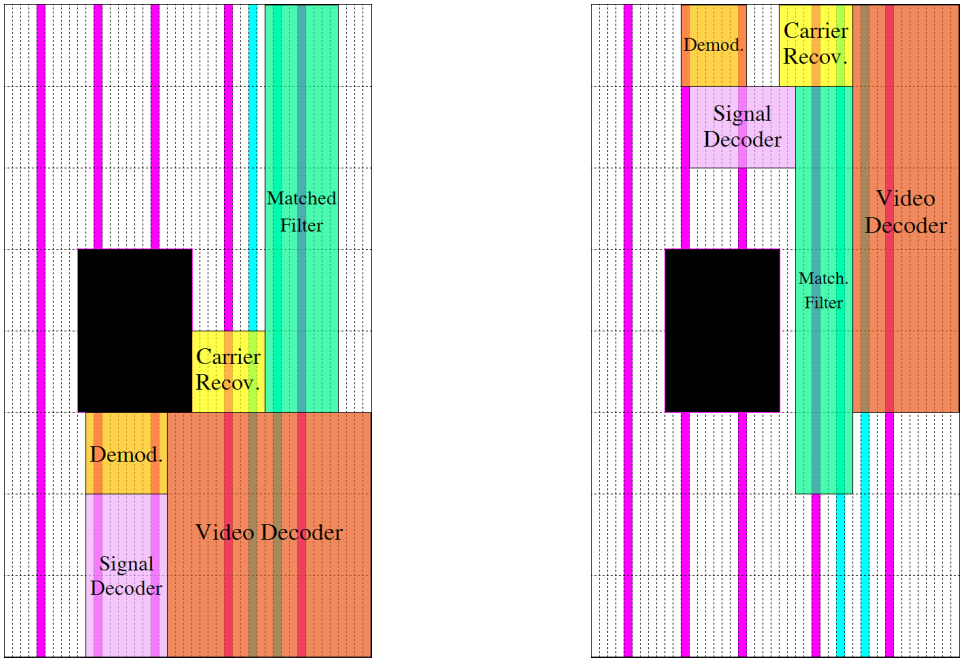
5.3.2 Floorplanner settings

To have a fair comparison to the floorplan performed in [11], we used the same objective function. Specifically, we considered the floorplan achieved by [11] when the objective was to reduce as much as possible the number of wasted configurable frames, that is, the number of frames covered by the regions that are not required. Furthermore, we considered the fact that [11], as a post processing step, tries to optimize the overall wirelength without changing the value of the main objective. To summarize, the optimization perfermod by [11], gives priority to the best floorplan in terms of wasted frames and among these prefers the one having the least wirelength.

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 wirelength 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 floorplans having the lowest possible resource wastage and, among all these solutions, finding the one that gives the lowest wirelength. We also set the parameter rc_t for t in $\{CLB, DSP, BRAM\}$ to properly consider the number of frames per tile.

5.3.3 Results comparison

Since the number of reconfigurable regions for the SDR design is not high and the resource requirements are far below the 75% of the overall FPGA availability, from the analysis shown in Section 5.2 we decided to use directly OF without warm start. The floorplan shown in [11] achieved 466 wasted frame, while the one obtained after the execution of OF provided 306 wasted frame, a wasted area reduction of roughly 34% maintaining a similar overall wirelength. The optimal solution was found by OF in approximately 29 seconds, however about 1028 seconds were needed to prove its optimality. Both floorplans are shown in Figure 18.



(a) Floorplan produced by [11].

(b) Floorplan produced by OF.

Figure 18: Floorplans comparison on the SDR design

CHAPTER 6

CONCLUSIONS

Here we present the final considerations about our MILP-based algorithms for floorplannig on partially-reconfigurable FPGAs. Within Section 6.1 we describe the contribution of our methodology together with its limit, whereas in Section 6.2 we discuss some possible future works and improvements that can derive from HOF and OF.

6.1 Contributions and limits

This work presented two new approaches to automate floorplanning for partially-reconfigurable FPGAs. They are based on a MILP 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 devices can be easily handled using the FPGA partitioning technique described in Section 4.1.3.

The proposed floorplanners allow 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. Within this context we may summarize our contributions in details as follows:

- we provided a MILP formulation in OF, that can be used to solve the problem to optimality or within a certified gap from the optimum;

- we guaranteed the PR constraints satisfaction and gave 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 work;
- finally, a simplified MILP model defined for HOF can be used to locally improve the goodness of a solution obtained from a heuristic such as [16] or [11].

For a better understanding of the capabilities of our algorithms, we report in Table V the features of existing floorplanners together with HOF and OF.

TABLE V: FLOORPLANNERS FEATURES

	[33]	[34]	[35]	[36]	[37]	[38, 39]	[11, 16]	HOF	OF
Resource distribution-aware		✓					✓	✓	✓
Reconfiguration-aware			✓	✓	✓	✓	✓	✓	✓
Compliant with PR						✓	✓	✓	✓
Optimize interconnections	✓	✓	✓	✓	✓		✓	✓	✓
Considers IO pins						✓	✓	✓	✓
Customizable objective function								✓	✓
Reaches the optimum									✓

On the other hand, one of the main drawbacks of OF is its high execution time. This issue limits the applicability of the approach in scenarios in which the designer do not need a quick answer but is interested in good quality floorplans. Morover, HOF needs a heuristic and feasible solution to perform re-optimization, thus it cannot be executed as a standalone algorithm.

6.2 Future work

A non trivial extension of the current work would be to address power consumption as a new optimization metrics. This would give the designer the possibility to achieve a floorplan with a more uniform thermal distribution reducing also the average temperature.

Another possible work deriving from the current MILP model would be to take into account *bitstream relocation* while searching for a floorplan. *Bitstream relocation* refers to the possibility of loading a partial bitstream into a different address of the configuration memory, still obtaining the same underlining functionality on the FPGA. This enhancement would give the designer the possibility to prefer floorplans in which one or more reconfigurable regions could be relocated at other available areas of the FPGA, without compromising the overall system functionality. The latter is an important aspect to take into account when designing fault tolerant systems.

Focusing on the developed MILP model, further studies could be brought forward to better characterize the polyhedron of the feasible solutions. Additional and smart cuts could improve the performance of our approaches, making them suitable for bigger problem instances and able to find better solutions in a smaller amount of time. Moreover, the effectiveness of HOF could be enhanced trying to integrate it into a simulated annealing framework or trading off the size of the solution space to achieve better results in a limited amount of time.

An approach similar to HOF could be devised to find good initial solutions quickly without the need of other heuristics. An idea would be to test the effectiveness of incremental approaches that consider the reconfigurable regions iteratively, fixing at each step some constraints on their geometrical relations. This would avoid to explore a huge search space while still looking for good solutions.

CITED LITERATURE

1. Estrin, G.: Organization of computer systems: the fixed plus variable structure computer. In Papers presented at the May 3-5, 1960, western joint IRE-AIEE-ACM computer conference, pages 33–40. ACM, 1960.
2. Santambrogio, M. D.: Hardware-Software codesign methodologies for dynamically reconfigurable systems. Doctoral dissertation, Ph. D. thesis, Politecnico Di Milano, Italy, 2008.
3. Williams, J. A. and Bergmann, N. W.: Embedded linux as a platform for dynamically self-reconfiguring systems-on-chip. In Ersa'04: the 2004 International Conference On Engineering of Reconfigurable Systems and Algorithms, pages 163–169. CSREA Press, 2004.
4. Xilinx Inc. `http://www.xilinx.com/tools/planahead.htm`.
5. Xilinx Inc: Partial Reconfiguration User Guide.,
6. Lim, D. and Peattie, M.: Two flows for partial reconfiguration: Module based or small bit manipulations. Application Note XAPP, 290, 2002.
7. Altera Corporation. `http://www.altera.com/`.
8. Hsiung, P.-A., Santambrogio, M. D., and Huang, C.-H.: Reconfigurable System Design and Verification. CRC Press, 2009.
9. Xilinx Inc: 7 Series FPGAs Configurable Logic Block.,
10. Rabozzi, M., Lillis, J., and Santambrogio, M. D.: Floorplanning for partially-reconfigurable fpga systems via mixed-integer linear programming. In FCCM, 2014. To appear.
11. Vipin, K. and Fahmy, S. A.: Architecture-aware reconfiguration-centric floorplanning for partial reconfiguration. In ARC, pages 13–25, 2012.
12. Adya, S. N. and Markov, I. L.: Fixed-outline floorplanning: enabling hierarchical design. IEEE Trans. VLSI Syst., 11(6):1120–1135, 2003.
13. Baker, B. S., Jr., E. G. C., and Rivest, R. L.: Orthogonal packings in two dimensions. SIAM J. Comput., 9(4):846–855, 1980.
14. Murata, H., Fujiyoshi, K., Nakatake, S., and Kajitani, Y.: VLSI module placement based on rectangle-packing by the sequence-pair. IEEE Trans. on CAD of Integrated Circuits and Systems, 15(12):1518–1524, 1996.
15. Korf, R. E., Moffitt, M. D., and Pollack, M. E.: Optimal rectangle packing. Annals OR, 179(1):261–295, 2010.

16. Bolchini, C., Miele, A., and Sandionigi, C.: Automated Resource-Aware Floorplanning of Reconfigurable Areas in Partially-Reconfigurable FPGA Systems. In FPL, pages 532–538, 2011.

17. Garey, M. R. and Johnson, D. S.: Computers and intractability, volume 174. freeman San Francisco, 1979.

18. Papadimitriou, C. H.: The euclidean traveling salesman problem is np-complete. Theor. Comput. Sci., 4(3):237–244, 1977.

19. Murata, H., Fujiyoshi, K., Nakatake, S., and Kajitani, Y.: Rectangle-packing-based module placement. In ICCAD, pages 472–479, 1995.

20. Ohtsuki, T., Sugiyama, N., and Kawanishi, H.: An optimization technique for integrated circuit layout design. Proc. ICCST, pages 67–68, 1970.

21. Guo, P.-N., Cheng, C.-K., and Yoshimura, T.: An o-tree representation of non-slicing floorplan and its applications. In Proceedings of the 36th annual ACM/IEEE Design Automation Conference, pages 268–273. ACM, 1999.

22. Hong, X., Huang, G., Cai, Y., Gu, J., Dong, S., Cheng, C.-K., and Gu, J.: Corner block list: an effective and efficient topological representation of non-slicing floorplan. In Computer Aided Design, 2000. ICCAD-2000. IEEE/ACM International Conference on, pages 8–12. IEEE, 2000.

23. Wong, D. and Liu, C. L.: A new algorithm for floorplan design. In Proceedings of the 23rd ACM/IEEE Design Automation Conference, pages 101–107. IEEE Press, 1986.

24. Lin, J.-M., Chang, Y.-W., and Lin, S.-P.: Corner sequence-a p-admissible floorplan representation with a worst case linear-time packing scheme. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, 11(4):679–686, 2003.

25. Nakatake, S., Fujiyoshi, K., Murata, H., and Kajitani, Y.: Module placement on bsg-structure and ic layout applications. In Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design, pages 484–491. IEEE Computer Society, 1997.

26. Lin, J.-M. and Chang, Y.-W.: Tcg: a transitive closure graph-based representation for non-slicing floorplans. In Proceedings of the 38th annual Design Automation Conference, pages 764–769. ACM, 2001.

27. Chang, Y.-C., Chang, Y.-W., Wu, G.-M., and Wu, S.-W.: B*-trees: a new representation for non-slicing floorplans. In Proceedings of the 37th Annual Design Automation Conference, pages 458–463. ACM, 2000.

28. Young, E. F., Chu, C. C., and Shen, Z. C.: Twin binary sequences: a nonredundant representation for general nonslicing floorplan. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 22(4):457–469, 2003.

29. Zhou, H. and Wang, J.: Acg-adjacent constraint graph for general floorplans. In Computer Design: VLSI in Computers and Processors, 2004. ICCD 2004. Proceedings. IEEE International Conference on, pages 572–575. IEEE, 2004.

30. Shen, Z. C. and Chu, C. C.: Bounds on the number of slicing, mosaic, and general floorplans. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 22(10):1354–1361, 2003.

31. Tang, X., Tian, R., and Wong, D.: Fast evaluation of sequence pair in block placement by longest common subsequence computation. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 20(12):1406–1413, 2001.

32. Tang, X. and Wong, D.: Fast-sp: a fast algorithm for block placement based on sequence pair. In Proceedings of the 2001 Asia and South Pacific design automation conference, pages 521–526. ACM, 2001.

33. Cheng, L. and Wong, M. D. F.: Floorplan design for multi-million gate FPGAs. In ICCAD, pages 292–299, 2004.

34. Feng, Y. and Mehta, D. P.: Heterogeneous Floorplanning for FPGAs. In VLSI Design, pages 257–262, 2006.

35. Yuh, P.-H., Yang, C.-L., and Chang, Y.-W.: Temporal floorplanning using the three-dimensional transitive closure subGraph. ACM Trans. Design Autom. Electr. Syst., 12(4), 2007.

36. Singhal, L. and Bozorgzadeh, E.: Multi-layer Floorplanning on a Sequence of Reconfigurable Designs. In FPL, pages 1–8, 2006.

37. Banerjee, P., Sangtani, M., and Sur-Kolay, S.: Floorplanning for Partially Reconfigurable FPGAs. IEEE Trans. on CAD of Integrated Circuits and Systems, 30(1):8–17, 2011.

38. Montone, A., Santambrogio, M. D., Sciuto, D., and Memik, S. O.: Placement and Floorplanning in Dynamically Reconfigurable FPGAs. TRETS, 3(4):24, 2010.

39. Montone, A., Santambrogio, M. D., and Sciuto, D.: Wirelength driven floorplacement for FPGA-based partial reconfigurable systems. In IPDPS Workshops, pages 1–8, 2010.

40. Gurobi optimization inc. `http://www.gurobi.com/download/gurobi-optimizer`.

41. Ibm ilog cplex optimizer. `http://www.ibm.com/software/commerce/optimization/cplex-optimizer`.

42. Vielma, J. P.: Mixed integer linear programming formulation techniques. 2013.

43. Chen, P. and Kuh, E. S.: Floorplan sizing by linear programming approximation. In Proceedings of the 37th Annual Design Automation Conference, pages 468–471. ACM, 2000.

44. Khachian, L, G.: A polynomial algorithm in linear programming. Doklady Akademii Nauk SSSR, 244:1093–1096, 1979. English translation: Soviet Mathematics Doklady 20:191–194.

45. Karmarkar, N.: A new polynomial-time algorithm for linear programming. In Proceedings of the sixteenth annual ACM symposium on Theory of computing, pages 302–311. ACM, 1984.

VITA

Marco Rabozzi

Education

B.S., Engineering of Computing Systems
Politecnico di Milano
2012

M.S., Computer Science (current)
University of Illinois at Chicago, Chicago, IL
May 2014

Publications

Marco Rabozzi, John Lillis, Marco D. Santambrogio. **Floorplanning for Partially-Reconfigurable FPGA Systems via Mixed-Integer Linear Programming**. In *Accepted full papers of the 2014 FCCM Symposium on Field-Programmable Custom Computing Machines* , Boston, Massachusetts, USA, May 11-13, 2014