

A Reconfiguration-aware Floorplacer for FPGAs

A.Montone, F. Redaelli, M.D. Santambrogio, S. Ogrenci Memik

Abstract—The goal of this paper is to introduce a partitioning and floorplanning algorithm tailored for reconfigurable architectures deployable on FPGAs. Our proposed algorithm specifically considers the feasibility of the associated communication infrastructure for a given floorplan. Different from existing approaches, our floorplanning algorithm takes specific physical constraints such as resource distribution and the granularity of reconfiguration possible for a given FPGA device into account. These physical constraints are typically considered at the later placement stage. In order to emphasize this fundamental difference with respect to traditional floorplanners, we refer to our approach as a floorplacer.

I. INTRODUCTION

Nowadays one of the most important design styles in VLSI is represented by Field Programmable Gate Arrays (FPGA). Since their introduction in the mid 80s, FPGAs provided new challenges to the existing VLSI design automation algorithms. FPGAs, due to their internal architecture, require new algorithms (or improvement to the existing ones) for each step of the design flow. The standard FPGA design flow starts from a high-level description of the circuit (e.g. provided by a HDL language) and ends with a configuration file (bitstream) configuring the desired circuit on the device. Much effort has been already placed into improving the early stages of VLSI design flow, e.g. logic synthesis, as well as to improve last steps, i.e. placement and routing. While these problems have been faced directly by the FPGA vendors [1] as much as by the academic world [2], the floorplanning automation for FPGAs is a current research topic, particularly in order to face the new challenges provided by FPGAs. FPGAs present two new aspects with respect to classical VLSI designs: *Resource Heterogeneity* and *Reconfigurability*. The first one refers to the fact that the silicon die of FPGAs generally provides several kinds of resources (e.g., programmable logic cells, memories, multipliers, DSPs and so on). Each architectural module can exploit the provided resources in order to increase its performance, hence each module has to be placed in an area region containing all the needed resources. One of the most studied use of FPGAs deals with their *reconfiguration capability*, i.e. one can change the architecture or the application implemented on the FPGA without needing any physical action on the device. One of the cutting-edge applications deals with partial dynamic reconfigurability, that provides the capability of changing at runtime (i.e., without having a disruption of provided functionality) a subset of the modules in order to improve or change its own behavior. Reconfigurability introduces time as a new variable into the floorplanning formulation.

A. Montone, F. Redaelli and M.D. Santambrogio are with Politecnico di Milano, Milano, Italy (e-mails: {fredaelli, santambr}@elet.polimi.it, alessio.montone@dresd.org).

S. Ogrenci Memik is with Northwestern University, Chicago, Illinois (e-mails: seda@eecs.northwestern.edu).

A floorplanner taking into account both heterogeneity and reconfigurability, needs to find for each module¹ and area assignment (according to the target device’s capabilities) considering that the modules are going to be replaced later on.

The aim of this work is to introduce a resource-aware *floorplacer* for reconfigurable architectures. The term floorplacer has been introduced in the context of standard cell-based physical design of ASICs [3], [4] in order to underline how recently developed algorithms for VLSI design automation face several problems concurrently that have been classically considered disjoint. Our problem can be reduced to the identification of groups of modules that are likely to be configured in the same rectangular area and, consequently, identify these chip areas according to the modules’ requirements, device capabilities and objective function.

The state of the art of floorplanning for FPGAs will be reviewed in Section II, while in Section III the problem will be formalized. The proposed approach will be discussed in Section IV providing validation data. Finally, conclusions will be summarized in Section V.

II. RELATED WORKS

The problem of resource aware floorplanning tailored to FPGAs has been addressed in [5]. They consider floorplanning on devices having a periodic chip die, like Xilinx Virtex II [6]. Each architectural module has a list of required resources, hence each module has to be placed in an area region containing all the needed resources. The approach proposed in [5] is based on a two-step algorithm: first the execution of Parquet [7] floorplanner with the following resource aware cost function (where $N_{\text{assigned},r}$ is the number of resources of type r assigned to a module, similarly $N_{\text{required},r}$ is the number of resources of type r required by a module)

$$C = \alpha \cdot C_{\text{parquet}} + \sum_{r \in \text{resources}} \beta_r \cdot \left(1 - \frac{N_{\text{assigned},r}}{N_{\text{required},r}} \right)$$

The result is then refined by solving a flow maximization with cost minimization. This approach is currently the state of the art in resource aware floorplanning tailored for FPGAs. It is able to reach high resource utilization configuration (an average of 80% CLBs usage has been observed during tests). One of the greatest limits is that such floorplans are, in the majority of cases, not compatible with the FPGA reconfigurability process (because reconfigurable modules have to be rectangles in order to exploit reconfigurability capabilities on the majority of FPGAs).

One of the earliest works dealing with reconfiguration aware floorplanning has been proposed by Bazargan et al. [8]. Aim of this work is to minimize execution time by

¹In this paper we use the term *module* to address an architectural component after technology mapping

implementing in hardware as many modules as possible, under the constraint of FPGA size. They define an offline floorplanner that, sequentially for each time instant, decides if a required functionality should be implemented in hardware on a reconfigurable device or has to be executed via software on a general purpose processor. They propose different heuristics in order to minimize execution time, and reduce implemented modules' fragmentation across the device area. It can be noticed how this formulation involves complex hardware-software co-design issues.

The first definition of *temporal floorplanning* has been introduced by Vasilko [9], where temporal floorplanning consists of two phases: i) partitioning and sequencing design modules into design configurations (also called temporal partitioning or scheduling), and ii) spatial positioning of design modules and wiring within the reconfigurable area of each configuration. One of the most important contributions in three dimensional floorplanning was made in [10], [11]. They introduced two full 3D floorplanners based on simulated annealing, one over a three dimensional transitive closure graph (TCG) and the other one on T-trees. They just evaluate the time required by each module to communicate with RAM chips outside the FPGA in order to store results and read input, but they do not evaluate if a found floorplan, particularly its communication infrastructure, is feasible on a given device or not. Furthermore, they do not consider other resources than logic blocks.

A most recent work [12] tackles the FPGA floorplanning problem considering partial dynamic reconfiguration, particularly module reusability. They introduce a variation to the well known sequence pair representation [13] in order to represent the floorplan in different time frames. Aim of the paper is to reduce the quantity of device area reconfigured between different time frames. Given two designs in two different time instants, the authors present a simulated-annealing based floorplanning able to reduce the reconfigured area between the two instants by exploiting re-use of already configured modules (i.e. given a module A configured on the FPGA at time $t = 0$ and assuming that such module is required again at time instant $t = 1$, instead of configuring a new module A before $t = 1$, one wants to re-use the already-configured module A).

In our work we will describe a novel approach for resource- and reconfiguration- aware floorplanning. Different aspects of the problem will be described, focusing particularly on the FPGA's resource heterogeneity and the temporal dimension typical of reconfigurable systems. Reconfiguration capability of FPGAs introduces new degree of freedoms in the floorplanning solution space. While several algorithms and approaches have been introduced in the past for static architectures floorplanning, such a problem has not been yet widely analyzed for reconfigurable architectures.

III. PROPOSED PROBLEM FORMULATION

In this section the proposed formulation of the floorplacement problem will be introduced. First some definitions will be given, then the formal definition will be introduced as two sequential steps.

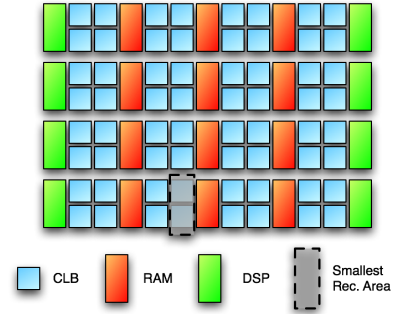


Figure 1. Common structure of FPGAs belonging to Xilinx Virtex 5 Family. This is just an abstract drawing for illustrative purposes and the amount of resources and relative frequency of occurrence of patterns are not realistic.

A. Target Devices and Architecture

Let a *Module* be a technology-mapped netlist implementing a required functionality, and a *Reconfigurable Area* (RA) be a rectangular FPGA area where two or more modules are going to be placed and routed (at design time) and configured (at runtime) according to the application implemented on the reconfigurable system. This work focuses on the last generation of Xilinx FPGA devices, hence on Xilinx Virtex 5 (even if it can be extended in order to support other vendors' FPGAs). According to their datasheets [14] all the devices, belonging to this family, share a common structure drawn in figure 1. The entire FPGA is made of programmable logic (CLBs) and periodically distributed BRAMs, while all the other resources (like DSPs) are placed on the vertical edges of the FPGAs. Such devices are divided into 4 rows and the smallest reconfigurable element is 1 row high and 1 CLB wide, which is referred to as *frame*. The height of the row expressed in CLBs is a device parameter. Given a generic module, the smallest area that can be involved in the reconfiguration process is a rectangle such that:

- The height in rows is the smallest integer greater than the module's height in rows;
- The width in CLBs is the smallest integer greater than the module's width in CLBs;
- The area is aligned with the grid made of rows and CLBs.

The Xilinx Early Access Partial Reconfiguration (EAPR) design flow [15] will be considered in the proposed approach. This flow is compatible with dynamic reconfiguration (in either internal and external version). EAPR allows the definition of a set of RAs, such that none of them overlaps (i.e. has a non empty intersection) with any other defined RA. All the *fixed logic* (i.e. all logic that has to remain configured on the device at all times as well as glue logic) is placed outside the reconfigurable areas, while it can be routed also using routing resources intersecting and even crossing RAs (usage of routing resources crossing RAs is the most relevant). Figure 2-a provides an example of a reconfigurable architecture developed with the EAPR design flow. Reconfigurable areas are aligned to the grid made by rows and CLBs according to the EAPR design rules. The communication between modules and fixed logic can be granted by the EAPR design flow,

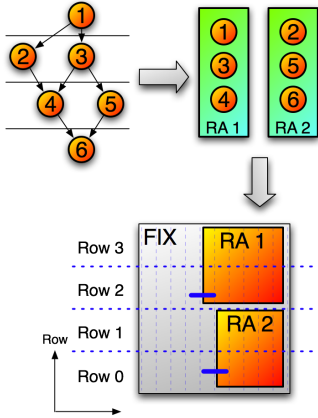


Figure 3. Overview of the proposed approach and formalization. The input task graph is partitioned in RAs and after these RAs are floorplaced in the target device.

hence it is up to the fixed logic to provide an inter-module communication (e.g., through a bus-based infrastructure or a NoC based infrastructure).

Furthermore, hardware macros can be placed on RAs' boundaries in order to define *pins* where modules can hook themselves. We will refer to the set of pins composing the access point to the communication infrastructure as *link*. Such macros are made with a set of pairs of CLBs, one side of CLB pair is connected to a module's signal while the other is connected to a fixed logic's signal².

Let us define *Simple Reconfigurable Area* (Simple RA) as a rectangular area of the FPGA containing a set of links, such that a reconfigurable module can be placed in this Reconfigurable Area. Generally Reconfigurable Areas are surrounded by glue logic, but adjacent Reconfigurable Areas may be grouped to form a so-called *grouped Reconfigurable Area* containing as many links as the contained Reconfigurable Areas. A module can be configured inside a single Reconfigurable Area as over several adjacent Reconfigurable Areas. It can be noticed how a simple RA can be seen as one grouped RA containing only one hardware macro.

In order to perform floorplanning (particularly to define a shape and a size) of the RA, one must know which resources must be provided by each RA (i.e., which on-chip resources must be included inside the Reconfigurable Area). The proposed problem formulation and consequently the approach will be divided into two phases, as shown in Figure 3:

- Partitioning into RAs: find a Reconfigurable Area (simple or grouped) where a given module can be placed. While a module can be placed in multiple partitions, the algorithm looks for the one minimizing the overall partition variance (Section III-B).
- RA Floorplacement: given the set of RAs, find the area constraints for each RA (Section III-C).

Next subsection will introduce the formulation of the partitioning into RAs followed by the Reconfigurable Area floorplacement.

²Actually such hardware macro may also connect two adjacent RAs, but such case will not be considered here.

B. Partitioning into RAs

Given a scheduled task graph (TG), for each time instant one can identify which modules (i.e., nodes of the TG) are required to be configured in the system and ready to be used. We define a *static photo* at a given time p as the set of modules required to be configured and running at time (static photo) p .

Let N be the number (constant along all the static photos of the TG) of grouped Reconfigurable Areas (let us recall that also simple Reconfigurable Areas can be seen as a particular case of grouped RAs) and \mathcal{M} the set of all the modules (i.e. the nodes of the TG).

For a module $m \in \mathcal{M}$ let $\text{TIME}(m)$ be the time instant when module m has to be configured and ready to be used. For the sake of simplicity this formulation will be carried on considering predicate $\text{TIME}(\cdot)$ returning a single value and not a set of adjacent values (e.g., when a module's execution spans across several static photos). It is straightforward to remove this assumption from the formulation (and the proposed approach will not be affected by this limitation) even if it requires a more verbose notation.

Let $r_{m,t}$ be an integer describing the number of resources of type t required by module m . While $c_{m,n}$ is the binary variable that is 0 if and only if m is not configured in the grouped Reconfigurable Area n , otherwise it is 1.

A feasible mapping solution consists in having each RFU bound with at least one RA³. The solution we are looking for in this paper is characterized by a stronger constraint where each node of the TG must be associated with one and only one RA..

Let $\rho_{n,p,t}$ be the number of resources of type t required by all the modules configured into Reconfigurable Area n at a given time p (static photo), hence:

$$\rho_{n,p,t} = \sum_{m \in \mathcal{M} \text{ s.t. } \text{TIME}(m)=p} r_{m,t} c_{m,n} \quad (1)$$

Let $\overline{\rho_{n,p,t}}$ be the temporal average of $\rho_{n,p,t}$, that is the average number of resources of type t required by the Reconfigurable Area n , hence (where P is the number of static photos):

$$\overline{\rho_{n,p,t}} = \frac{\sum_p \rho_{n,p,t}}{P} \quad (2)$$

The temporal variance of $\rho_{n,p,t}$ can be defined as:

$$\text{Var}(\rho_{n,p,t}) = \frac{1}{P} \sum_p (\rho_{n,p,t} - \overline{\rho_{n,p,t}})^2 \quad (3)$$

For each resource type t an associated cost_t is given. Such a cost is inversely proportional to the quantity of the on-chip available resource of type t , as the greater the availability of a resource t , the lesser its cost will be.

In such a context, we aim at maximizing the similarity of all the modules mapped in the same Reconfigurable Area.

³Consequently each RFU m has a set of RAs where it can be placed and such a situation can be exploited if bitstream relocation is available (i.e., given a placed and routed RFU, it can be placed at runtime in a different position by performing a rigid translation of all its own components and nets) [16], [17].

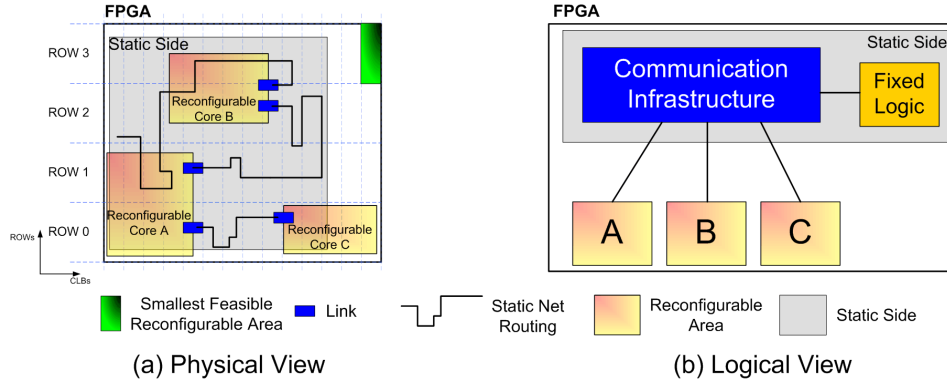


Figure 2. a) Physical view of the target architecture underlining how static nets can be routed crossing RAs b) Logical view of the static part of the architecture providing both logic and also a communication infrastructure used by modules.

This goal can be achieved via an objective function Φ which depends on variable $c_{m,n}$ and parameter N as shown below:

$$\Phi_N(c_{m,n}) = \sum_n \left(\sum_t \text{Var}(\rho_{n,p,t}) \text{cost}_t \right) \quad (4)$$

therefore, as we partition the task graph into N Reconfigurable Areas we aim to find an association between the nodes of the input TG and the available Reconfigurable Areas such that Φ_N is minimized, in formula:

$$c_{m,n} = \text{argmin} \Phi_N \quad (5)$$

In order to achieve this goal, the following constraints must be considered:

- Each module (node of the TG) must be associated with one and only one Reconfigurable Area, hence:

$$\sum_n c_{m,n} = 1 \quad \forall m \in \mathcal{M} \quad (6)$$

- In order to avoid trivial solutions each Reconfigurable Area must contains at least one module:

$$\sum_t \sum_p \rho_{n,p,t} > 1 \quad \forall n \in \{1 \dots N\} \quad (7)$$

The formulation can be extended to consider N as a variable of the problem, but to avoid trivial solutions, the values assumed by N must belong to a subset (whose boundaries are problem dependent and can be considered as a design trade-off) of the integer numbers. Let $\mathcal{N} \subset \mathbb{N}$, the optimal number of grouped Reconfigurable Areas be defined as follows (in order to preserve similarity of notation, N is still written as a subscript of the objective function even if it is a parameter):

$$N = \text{argmin}_{N \in \mathcal{N}} \min \Phi_N \quad (8)$$

C. Reconfigurable Areas Floorplacer

Once $c_{m,n}$ and N are computed, the floorplacement of the Reconfigurable Areas can be performed. In order to constraint the on-chip resources covered by each Reconfigurable Area the following quantity must be defined:

$$\xi_{n,t} = \max_p \rho_{n,p,t} \quad (9)$$

where $\xi_{n,t}$ is the minimum amount of on-chip resources of type t that has to be provided by Reconfigurable Area n .

For each Reconfigurable Area n the number of required links λ_n can be computed as follows:

$$\lambda_n = \max_p \# \{m \in \mathcal{M} : \text{TIME}(m) = p\} \quad (10)$$

Like in classical floorplanning, each Reconfigurable Area n is placed inside a rectangular area A_n identified by the coordinates (x_n, y_n) of the bottom-left corner and characterized by a height h_n and a width w_n , and different areas must not overlap each other. Like in classical placement each area must not exceed device area (due to their straightforward formulation such constraints are not written in formulae for brevity). Furthermore, considering devices' coarse grained horizontal reconfigurability, h_n can assume only 4 different values $h_n \in \{1, 2, 3, 4\}$.

Each area A_n must provide $\pi_{n,t}$ resources of type t according to the following constraint:

$$\pi_{n,t} \geq \xi_{n,t} \quad \forall n \quad (11)$$

The fragmentation index for a resource of type t and a module n is given by:

$$\varphi_{n,t} = \frac{\pi_{n,t} - \overline{\rho_{n,p,t}}}{\pi_{n,t}} \quad (12)$$

and the fragmentation index of a given Reconfigurable Area may be computed as the weighted sum of the fragmentation index of the different resources, c.e.:

$$\varphi_n = \sum_t w_t \varphi_{n,t} \quad (13)$$

Aim of the Reconfigurable Area floorplacement is to minimize a cost function that, in the simplest formulation, minimizes the overall fragmentation:

$$\Theta = \sum_n \varphi_n \quad (14)$$

IV. PROPOSED APPROACH

According to the problem formulation, also the proposed approach will be divided into two steps, the first one dealing with the Reconfigurable Area partitioning and the second one dealing with Reconfigurable Area floorplacement problem.

A. Reconfigurable Area Partitioning

The most straightforward solution to the problem consists in solving its quadratic and integer programming formulation. It is well-known that a generic quadratic programming problem is a NP-Hard problem, hence two approximate solutions are also considered, the first one based on a simulated annealing engine and the second one on hierarchical partitioning.

Let us consider a bucket data structure having a set of buckets B_n for each grouped Reconfigurable Area n . A given module m belongs to a bucket B_n (and only to that) if and only if m is going to be placed in the Reconfigurable Area n at TIME(m).

Let the simulated annealing moves be the following:

- Move one module between two buckets: randomly pick a module $m \in b_n$ and move to a bucket $b_{n'}$ where $n \neq n'$. This move can be performed if and only if B_n contains another module $m' \neq m$;
- Swap modules belonging to different buckets: randomly pick up two modules m and m' , respectively $m \in B_n$ and $m' \in B_{n'}$ such that $B_n \neq B_{n'}$. The move consists in swapping modules' buckets such that $m \in B_{m'}$ and $m' \in B_m$.

Once the data structure and a set of moves have been defined, a simulated annealing based algorithm can be formulated in order to minimize objective function (4). According to the classical simulated annealing process a move is always accepted if it implies an improvement in the objective function, otherwise it is accepted with a monotonically-decreasing probability (the decrease rate is stated by Boltzmann equation).

Another solution recalls the hierarchical pattern that is used in several algorithms for VLSI design automation. At each step the set of modules is partitioned in two subsets, then every subset is partitioned in other two subsets and so on so forth till as many subsets as required Reconfigurable Areas have been generated.

Let Σ be a set containing sets of modules, initially $\Sigma = \{\mathcal{M}\}$. The hierarchical partitioning can be described as follows:

- 1) Pick up the set $S \in \Sigma$ having the highest objective function Φ (eq. 4);
- 2) Perform partitioning over S obtaining two subset S' and S'' such that $S' \cup S'' = S$ and $S' \cap S'' = \emptyset$. A Fiduccia Mattheys partitioning algorithm may result suitable for such partitioning [7];
- 3) Let $\Sigma \leftarrow \Sigma - S + S' + S''$;
- 4) If $|\Sigma| = N$ end (i.e. the desired number of partitions is obtained, this number can be evaluated as stated in equation 8), otherwise go back to step 1.

B. Reconfigurable Area Floorplacer

Once the RAs and their corresponding resource requirements have been defined, the set of RAs goes through the core of the floorplacement algorithm. In order to represent the floorplan, a horizontal constraint list (HCL) is used for each row of the device. The HCL for row r is a list containing all the RAs placed in row r and ordered by increasing *column*.

Given this data structure the following moves are defined:

- Swap: randomly choose two modules and swap their position;
- Move: randomly change the position of a module;
- Span: randomly choose a module and increase its height expressed in device rows;
- Unspan: randomly choose a module and decrease its height.

Given a floorplan two quantities are defined: positive and negative area slack. Negative aerial slack is the area of the floorplan crossing target device boundaries, while positive aerial slack is the greatest contiguous empty area starting from the right-top most corner of the device and with non increasing width going bottom-ward. Given such slacks, the objective function is defined as (where P is the positive aerial slack, N is the negative one, $M \in \mathbb{N}$ and greater than the entire FPGA area):

$$\Theta = P - M \cdot N$$

. Aim of the simulated annealing process is to maximize Θ . According to such objective, the SA engine is biased toward floorplacements starting from the left-bottom most corner and leaving as much space as possible free on the top-right most corner (P area). Hence, the Simulated Annealing engine tries to minimize free space surrounded by RAs.

C. Validation

The proposed approach has been validated with randomly generated static photos and with a C++ tool implementing the algorithm. Such a tool takes in input a scheduled task graph and produces area constraint for each RFU. For space reasons hereby we provide only data regarding the simulated annealing phase of the floorplacement algorithm, representing the most relevant part of the proposed approach. The implementation of the floorplacer used an initial temperature of $3 \cdot 10^6 K$ and a cooling constant equal to 0.9 (i.e., $T_n = 0.9 \cdot T_{n-1}$), while the process is halted when $T < 10^{-6} K$. All the moves are coupled with a packing algorithm removing any occurrence of overlapping modules. In the average, the packing algorithm takes a time that is $O(n \cdot \log n)$, while in the worst case it is $O(n^2)$. The packing algorithm overcomes all the other operations of the SA's moves, that requires $O(n)$ time. We consider Xilinx Virtex 5 LX 85 FPGA and we randomly generated sets of reconfigurable areas. Each set requires from 50% to 92% of CLBs available on the FPGA. It can be noticed how increasing the number of CLBs required by the set involves a decrease in simulated annealer's success rate (i.e., the percentage of feasible solutions obtained from different executions of the algorithm). Figure 4 plots the success rate with respect to the total size of RAs' set.

It can be observed how almost 100% of times SA provides a feasible solution when RAs require less than 75% of the available CLBs, decreasing to a 30% success rate for set requiring more than 90% of available CLBs. When the set of reconfigurable areas requires less than 60% of FPGA's CLBs the final floorplan results biased toward module with high aspect ratio (i.e., the ratio between the longest side and the shortest expressed in CLBs), while with sets requiring much more resources the resulting floorplan is biased toward more

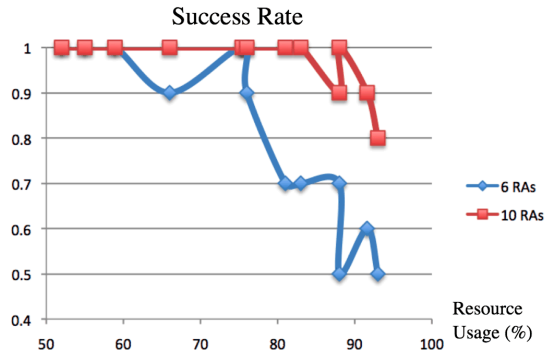


Figure 4. Success Rate with respect to resource requirements. X-axis: FPGA % area requested by all the RAs. Y-axis: average success rate of the annealer.

rectangular modules (i.e., with lowest aspect ratio). Figure 5 reports the average aspect ratio of the feasible solution with respect to resource requirements of randomly generated RAs sets.

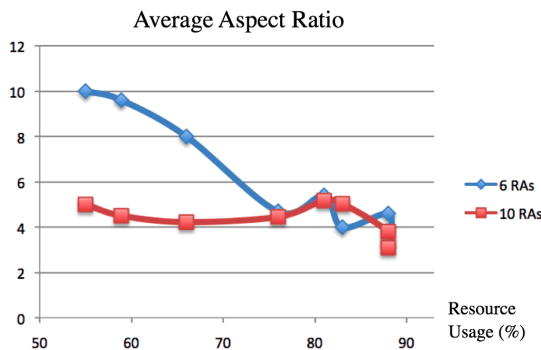


Figure 5. Average aspect ratio with respect to resource requirements. X-axis: FPGA % area requested by all the RAs. Y-axis: average aspect ratio of the feasible RAs.

On a Core 2 Duo 2.33 GHz our implementation takes an the average of 100 ms to perform an entire simulated annealing process on instances of both classes. The simulated annealer finds a feasible solution in more than 80% of the executions, while in the hard instances this rate falls below 10%.

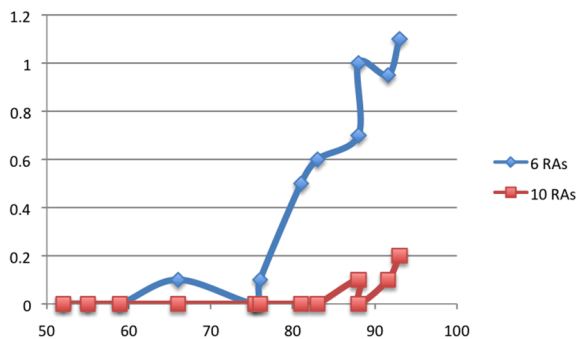


Figure 6. X-axis: FPGA % area requested by all the RAs. Y-axis: Average number of annealer's re-iteration, i.e. number of iterations excluding the first one, required before obtaining a feasible.

The low success rate can be easily overcome with multiple executions of the annealer. Due to the low execution time, such multiple re-iterations of the simulated annealing process does not limit our approach significantly. In Figure 6 we show a plot of the average number of iterations required to obtain the first feasible floorplacement.

V. CONCLUSIONS AND FUTURE WORKS

In this paper we have described a novel approach for resource- and reconfiguration- aware floorplanning. It has been shown how the reconfiguration capability of FPGAs introduces another degree of freedom in the floorplanning solution space. While several algorithms and approaches have been introduced in the past for static architectures floorplanning, such a problem has not been yet widely analyzed for reconfigurable architectures. The proposed Reconfigurable Area floorplacement can be improved by taking into account also inter reconfigurable modules communication. Furthermore, more complex objective functions can be defined in order to better consider reconfigurable modules aspect ratios and wirelengths.

REFERENCES

- [1] Xilinx Incorporation. *ISE 9.2i Manual*, 2007.
- [2] Vaughn Betz and Jonathan Rose. *VPR: A new packing, placement and routing tool for FPGA research*. Springer-Verlag, London, UK, 1997.
- [3] J.A. Roy, S.N. Adya, D.A. Papa, and I.L. Markov. Min-cut floorplacement. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 25(7):1313–1326, July 2006.
- [4] S.N. Adya, S. Chaturvedi, J.A. Roy, D.A. Papa, and I.L. Markov. Unification of partitioning, placement and floorplanning. *Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on*, pages 550–557, Nov. 2004.
- [5] Yan Feng and Dinesh P. Mehta. Heterogeneous floorplanning for fpgas. *vlsid*, 0:257–262, 2006.
- [6] Xilinx Inc. *Virtex II - Family Overview*, Dec 2007.
- [7] S.N. Adya and I.L. Markov. Fixed-outline floorplanning: enabling hierarchical design. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 11(6):1120–1135, Dec. 2003.
- [8] Kiarash Bazargan, Ryan Kastner, and Majid Sarrafzadeh. 3-d floorplanning: Simulated annealing and greedy placement methods for reconfigurable computing systems. *rsp*, 00:38, 1999.
- [9] Milan Vasilko. Dynasty: A temporal floorplanning based cad framework for dynamically reconfigurable logic systems. In *FPL '99: Proceedings of the 9th International Workshop on Field-Programmable Logic and Applications*, pages 124–133, London, UK, 1999. Springer-Verlag.
- [10] Ping-Hung Yuh, Chia-Lin Yang, and Yao-Wen Chang. Temporal floorplanning using the t-tree formulation. *Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on*, pages 300–305, Nov. 2004.
- [11] Ping-Hung Yuh, Chia-Lin Yang, and Yao-Wen Chang. Temporal floorplanning using the three-dimensional transitive closure subgraph. *ACM Trans. Des. Autom. Electron. Syst.*, 12(4):37, 2007.
- [12] L. Singhal and E. Bozorgzadeh. Multi-layer floorplanning on a sequence of reconfigurable designs. *Field Programmable Logic and Applications, 2006. FPL '06. International Conference on*, pages 1–8, Aug. 2006.
- [13] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani. Vlsi module placement based on rectangle-packing by the sequence-pair. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 15(12):1518–1524, Dec 1996.
- [14] Xilinx Incorporation. *Virtex 5 - Family Overview*, Dec 2007.
- [15] Xilinx Incorporation. *Early Access Partial Reconfiguration User Guide*, 2006.
- [16] S. Corbetta, F. Ferrandi, M. Morandi, M. Novati, M.D. Santambrogio, and D. Sciuto. Two novel approaches to online partial bitstream relocation in a dynamically reconfigurable system. *isvlsi*, 00:457–458, 2007.
- [17] H. Kalte, G. Lee, M. Pormann, and U. Ruckert. Replica: A bitstream manipulation filter for module relocation in partial reconfigurable systems. *ipdps*, 04:151b, 2005.