Paper Title*

*Note: Sub-titles are not captured in Xplore and should not be used

1st Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address

2nd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address

***Abstract*—Floorplanning is a mandatory step in the design of hardware accelerators on FPGA platforms consisting in placing the hardware modules to specific FPGA areas. In particular, if the design makes use of partial reconfiguration (PR), the overall system performance is greatly influenced by the allocation strategy. Existing tools for PR, which are provided by FPGA vendors, allow floorplanning to be performed manually. This method besides being highly time-consuming, demands a good knowledge of the architecture and organization of the hardware for an optimal floorplan. To solve this problem, this paper presents an automated floorplanner based on Mixed Integer Programming (MILP). The most common approach by the state of the art automated MILP (or other optimization method) based floorplanners is to apply MILP (or other optimization methods) for finding the optimal placement from a pre-enumerated list of possible placements. Meanwhile, in this work, the floorplanning problem was directly solved as an optimization problem by modeling the FPGA resources as a set of MILP constraints along with the standard PR related constraints. The performance of the proposed approach was tested by using a synthethic benchmark suite. Its performance was also compared with the state of the art MILP based floorplanners for the average execution time. Finally, the floorplanner was used in a real case study. This approach was observed to improve the execution time reported in the state of the art by orders of magnitude.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

par 1

what is floorplanning and how is it applied (what's its role) in the context of PR, what is the benefit of having a good floorplan or a floorplan at all

par 2

How is fp done currently and what are the shortcomings of those methods

par 3

In short what did I do and what are my contributions (clearly stated)

par 4

organization of the paper

Identify applicable funding agency here. If none, delete this.

Floorplanning is one of the major challenges in the field of dynamic parital reconfiguration. The placement of the static and reconfigurable slots on the FPGA fabric must satisfy the application requirements set by the application designer while also respecting the technological constraints set by the manufacturer. The conventional approach for an automated generation of FPGA floorplans usually involves two steps. First for each slot, all the possible rectangular slots that satisfy the resources requirement of the slot are enumerated. This is done by starting a scan on the fpga fabric from the bottom left corner and lisitng all the rectangles that contain all the necessary resources for the respective slots. Then some sort of heuristics/optimization is applied to choose the optimal ones from the set of possible slots. This approach has many problems {*to be listed later*}

Our approach instead focuses on applying the optimization process on a lower level of abstraction of the fpga fabric i.e., rather than applying optimization to select the most optimal one from a set of pre-scanned slots, we modeled the different types of resources on the fpga and their distribution along with forbidden regions as a set of constraints and added these constraints to the predefined constraints related to dpr.

Let us consider a floorplanning example where we have to make a floorplan for two slots S_1 and S_2 on the FPGA fabric. Each slot has resource requirements denoted as $\{D_1, B_1, C_1\}$ and $\{D_2, B_2, C_2\}$ where D, B and C represent DSP, BRAM and CLB respectively.
Our proposed system takes as an input in the resource requirement of each slot and a description of the resource distribution of the FPGA fabric and it returns the placement coordinates of the slots on the fpga fabric.
A slot is represented using 4 parameters i.e. the two bottom left coordinates and the width and the height of the slot. In our considered example the slots S_1 and S_2 are represented as (x_1, y_1, w_1, h_1) and (x_2, y_2, w_2, h_2). A forbidden region is also represented as a slot hence a forbidden region F_i can be represented as fx_i, fy_k, fw_i, fy_i

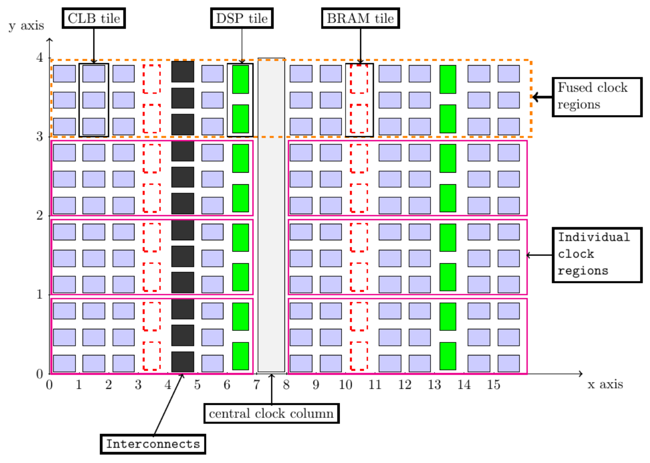

Fig. 1. FPGA architecture

II. Related Works

III. Background and Modeling

In this section we breifly describe the general architecture of FPGAs, the design flow in partial reconfiguration (PR) and the assumptions that led to the fromulation of PR floorplanning problem as a MILP problem. This work is based on the 7 series FPGA family from Xilinx.

A. FPGA Architecture and Partial-Reconfiguration

The configurable fabric of Xilinx FPGAs is divided into quadrants named clock regions. Within each clock region there are columns of different configurable resources such as CLBs, BRAMs or DSPs. Resources within a clock region share the same clock. A single column in a clock region is referred to as a tile. The number of resources in a tile varies depending on the device family. For example in Virtex 7z a CLB tile contains 50 clbs a BRAM tile contains 10 brams and a DSP tile contains 20 dsps. The functional logic compoenets (clbs, brams and dsps) and the routing logic components (switches, interconnects etc...) on the FPGA are configured based on a bit file stored in the configuration memory of an FPGA. This memory is organized into minimal configurable units called frames. A single frame in the configuration memory corresponds to a single tile on the fabric. In addition to the above mentioned resources, FPGAs also contain other components such as clock and clock modifying logic, I/O logic, configuration logic etc... Depending on the type of device family some of these components may or may not be included in a reconfigurable region. FPGAs, in particular 7 series devices, which are subject of this work, also contian routing resources called interconnect tiles. These tiles are placed back-to-back as shown in Fig 1. When floorplanning for partial reconfiguration, the position of these back-to-back boundaries must be known inorder not to split them and violate PR restriction.

Partially reconfigurable applications are often composed of $\boldsymbol{M_s}$ number of static and $\boldsymbol{M_r}$ number of reconfigurable modules that are to be placed in $\mathbf{N}_s$ static and $\mathbf{N}_r$ reconfigurable regions on the FPGA respectively. In PR applications the number of static modules equals to the number of static regions i.e., $M_s = N_s$ while the number of reconfigurable modules is always greater than reconfigurable regions i.e., $M_r > N_r$. Floorplanning in PR can then be defined as the process of allocating placement for N_r reconfigurable regions on the FPGA fabric.

state how PR-fp is currently done in vivado

B. Combining clock regions

On Xilinx FPGAs, the central clock column divides the FPGA into left and right regions as shown on fig 1. But in our model all the horizontally adjacent clock regions were fused into a single clock region. This simplifies our modeling with no penalty. As also shown in the figure on fig 1, a cartesian coordinate system can be overlayed on FPGAs to uniquely identify each resource on the logic fabric. The x axis represents each column of resources while the rows on the y axis represent the fused horizontally adjacent clock regions as indicated on fig 1. Combining the horizontally adjacent clock regions[1], in addition to resulting in a lower range of variables on the y axis, organizes resources on the y axis on a per tile (per clock region) basis instead of as individual clbs, brams or dsps. This reduces the search space for the solution in the MILP formulation at the expense of wasting resources. Based on this abstraction the resources on the FPGA fabric on a tile basis, the FPGA would become **W** columns wide and **H** clock regions high.

The ith reconfigurable region R$_i$ is a rectangular region on the FPGA fabric that hosts, at different durations, the set of reconfigurable modules assigned to it. R$_i$ can be represented as

$$R_i = (x_i, y_i, w_i, h_i) \mid x_i + w_i \leq W, y_i + h_i \leq H \qquad (1)$$

where x$_i$ and y$_i$ represent the bottom left coordinate and w$_i$ and h$_i$ represent the width and height of R$_i$ resectively. A resource type t that is required by reconfigurable module M_r is denoted as c_{rt} while the same type of resource that is incorporated inside a reconfigurable region R$_i$ is denoted as η_{it}. The upper and lower boundaries of R$_i$ are aligned to clock region boundaries since the resources on the fabric are organized on a tile basis.

talk about the modeling of forbidden regions

C. Discretization of the axis

Floorplanning is a two dimensional (quadratic) problem in that determining the number of resource contained in R$_i$

[1]henceforth clock regions implies horizontally fused clock regions

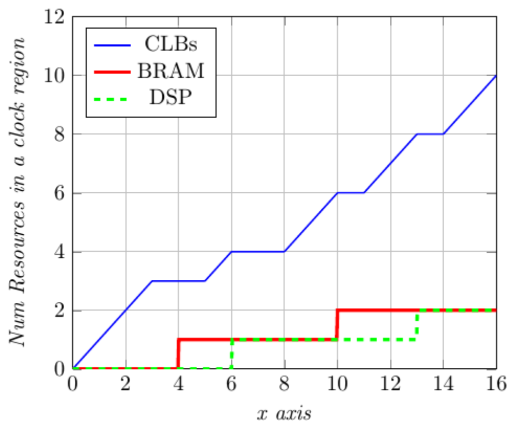

Fig. 2. Resource distribution finger print of fig 1

invloves determining the resources on both axis i.e., the area of the rectangle. To linearize this problem either of the axis must be discretized. A good strategy for discretization would be choosing the axis with the lower range of variables as this leads to an easily scalable model. In all the FPGA families that were chosen to be studied for this project, the range of the y axis (the number of clock regions on the y axis) was less than the range of the x axis i.e., H $<$ W. For example in kintex xc7z045fbv676 there are 100 columns on the x axis as opposed to 7 clock regions on the y axis. Hence the binary variable β_{ij} denotes clock region j in region in R$_i$.

D. *FPGA resource finger-printing*

Resources in most Xilinx FPGAs are distributed in a redundant manner this is to say vertically adjacent clock regions have a fairly similar distribution of resources with the possibility of different forbidden regions being included in them. Hence the reconfigurable fabric of most Xilinx FPGAs can accurately be represented by describing the resources in a single clock region and the location of forbidden regions in all clock regions. This process is equivalent to developing the resource distribution finger-print of a specifc type of FPGA. For each resource type *t* a piecewise function f$_t$(x) can be used to describe the distribution of that particular resource inside the first clock region of the FPGA along the x axis. As an example f$_t$(x) for the first clock region on fig 1 would be

If μ_t is a constant that denotes the number of resource *t* per tile, the amount of each type of resource included inside a region R$_i$ i.e., $\eta_i t$ can be defined as

$$\eta_{it} = \sum_{j=1}^{H} \beta_{ij} \cdot (f_t(x_i + w_i) - f_t(x_i)) * \mu_t \tag{2}$$

IV. Floorplanning problem formulation

put a picture of PR design flow in Xilinx

Partial-reconfiguration involves dynamically switching modules in a reconfigurable region whilst other reconfigurable and static regions continue to be operational. **Fig** depicts the PR design flow in Xilinx FPGAs as implemented in Xilinx Vivado. The major steps in the automated PR flow are

- *Synthesis*: At this level the behavioral description of modules written in hardware description langauges (Verilog or VHDL) is converted into a gate-level netlist. In PR design flow, floorplanning is mandatory (it is optional in the standard flow) and it must be done before the implementation step.
- *Implementation*: In this step the gate-level netlist output of the previous stage is functionally mapped to specific device resources on the FPGA and then placed and routed.
- *Bitstream generation*: In PR desgin flow, the final output of an FPGA design tool is a set of bit files which contain the configuration information for both the logic and routing resources of the static and reconfigurable regions.

In floorplanning for PR, each R$_i$, which are also named *Pblocks* in the Xilinx design flow, are subject to the following restrictions and requirements.

1) Reconfigurable regions must contain only valid reconfigurable resources (for example in 7 series FPGAs only CLBs, BRAMs and DSPs must be in R$_i$)
2) The minimum number of resources incorporated by a reconfigurable region $\mathbf{R}_i$ (Pblock) must at least be equal to the resource requirement of the largest module hosted in the region
3) Two reconfigurable regions must not overlap. Overlapping is equivalent to sharing at least a single tile
4) The left and right edges of R$_i$ must not split interconnect columns
5) Reconfigurable regions(Pblocks) can span non reconfigurable components such as configuration blocks, central clock column etc... without considering them as part of the region(Pblocks)
6) The height of R$_i$ must be aligned to clock regions. This is an optional requirement but enforcing it results in starting the reconfigurable module from a known initial state re-configuration

As described in the previous section the proposed floorplanner takes as an input the resource description of the FPGA fabric and the resource requirement of each reconfigurable region and produces a feasible placement for each region.

V. MILP FORMULATION

In this section the MILP formulation of the PR floorplanning is presented. The section first summarizes the list of optimization variables then presents the MILP contraints and the objective function to be optimized. The constraints are divided into three sections namely *structural*, *resource* and *non-interference* constraints.

A. Optimization Variables

To encode the PR-floorplanning of N$_r$ reconfigurable regions as a MILP formulation the following binary and real variables are defined.

For each reconfigurable region R$_i$ where i = 1....,N$_r$ and for each FPGA resource type *t* we define

- W, H $\in \mathbb{Z}$ integer variable that represent the width and the height (number of clock regions) of the fpga fabric
- x$_i$, y$_i$ w$_i$, h$_i$ $\in \mathbb{Z}$, integer variables that represent the bottom left coordinates, the width and the height of R$_i$ respectively
- β_{ij} $\in$ [0,1] a binary variable such that β_{ij} = 1 iff clock region j is included in R$_i$
- c$_{kt}$ $\mathbb{Z}$ an integer variable that specifies the *t* type resource requirement of the k$^{\text{th}}$ reconfigurable module
- η_{it} $\mathbb{Z}$ an integer variable that specifies the *t* type of resource contained in R$_i$
- γ_{ik} $\in$ [0,1] is a binary variable used to identify whether R$_i$ is found on the left or on the right of R$_k$
  γ_{ik} = 1 if x$_i$ $\leq$ x$_k$ [i.e. R$_i$ is on the left of R$_k$]
- F: set of forbidden regions
- F$_k$: forbidden region k $\in$ F

B. Structural constraints

The following constraints ensures the structural integrity of the reconfigurable regions

***Constraint** 1:* $\forall$ i$_1$ i = 1...,N$_r$, $\forall$ x$_i$ = 0...,W, $\forall$ y$_i$ = 0...,H

$$\begin{aligned} x_i + w_i \leq W \\ y_i + h_i \leq H \end{aligned} \tag{3}$$

meaning: the right most x coordinate and the top y coordinates of R$_i$ must not exceed the boundaries of the fabric

***Constraint** 2:* $\forall$ i = 1..., N$_r$, $\forall$ j = 1...,H

$$h_i = \sum_{j=1}^{H} \beta_{ij} \tag{4}$$

meaning: The height h$_i$ of a reconfigurable region R$_i$ is the sum of all the clock regions included in the region

***Constraint** 3:* $\forall$ i = 1...,N$_r$, $\forall$ j = 1...,H

$$\beta_{i(j+1)} \geq \beta_{ij} + \beta_{i(j+2)} - 1 \tag{5}$$

meaning: This contraint imposes contiguity of clock regions in a reconfigruable region R$_i$ i.e., if β_{i0} = 1 & β_{i2} = 1 thenβ_{i1} must also be equal to 1.

C. Resource constraints

A reconfigurable region must satisfy the resource requirements of the largest module it hosts. The expression for η_{it} in 2 is not linear and needs to be linearized to be used in a MILP constraint. The linearization that is done as [**?**]equires the definition of auxilary integer variable τ_{ijt} such that

- τ_{ijt} $\in \mathbb{Z}$ an integer variable such that

$$\tau_{ijt} = \beta_{ij} \cdot (f_t(x_i + w_i) - f_t(x_i)) \tag{6}$$

By enforcing the following constraints on η_{it} can now be restated as

$$\eta_{it} = \sum_{j=1}^{H} \tau_{ijt} \tag{7}$$

The linearization subject to the following constraints

$$\begin{aligned} &\tau_{ijt} \geq 0 \\ &\tau_{ijt} \leq BIG\_M \cdot \beta_{ij} \\ &\tau_{ijt} \leq (f_t(x_i + w_i) - f_t(x_i)) \\ &\tau_{ijt} \geq (f_t(x_i + w_i) - f_t(x_i)) - (1 - \beta_{ij}) \end{aligned} \tag{8}$$

The resource requirment constraint is stated as

***Constraint** 4:* $\forall$ i = 1...,N$_r$, $\forall$ j = 1...,H, $\forall$ r = 1...,number of vector elements

$$\eta_{it} \geq c_{rt} \tag{9}$$

D. *Non-interference constraints*

Two reconfigurable regions R$_i$ and R$_k$ are considered to be interfering with eachother (ovelapping) if there is a shared clock region between them while at the same time the right most side x coordiante of one of the regions is greater than the left most x coordinate of the other region. Hence the non interference constraint can be expressed as

***Constraint** 5:* $\forall$ i = 1...,N$_r$, k = 1...,N$_r$, $\forall$ j = 1...,H

$$x_i \geq x_k + w_k - (3 - \gamma_{ik} - \beta_{ij} - \beta_{kj}) \cdot BIG\_M \tag{10}$$

The non interference constraint between a region R$_i$ and a forbidden region can also be stated in the same way

VI. EXPERIEMTNAL RESULTS

A. Experimental Setup

How was the experiment implmented i.e, with what kind of prog. langaguge, on what optimization tool, on what platforms... What is the system being tested for ? What is the compositon of the synthethic task suite used for testing. What type of FPGAs are modeled ? What are the challenges when switiching between models.

The system is tested for Exec time Vs Num of Reconfigurable regions and Exec time Vs %of resources used by the system on both zynq and virtex

The system is also tested for % of wasted resources (coeficients used to set allowed percentage of resources to be wasted %)VS Exec time on both virtex and zynq. The average wasted resources are also reported when not imposing these constraints and what effect it has on exec time and feasibility in general. my guess is that upto a certain point (upto a certain % of utilization of resources) not imposing these constraints improves the exec time but after the utilization increases (i.e., more applications require more resources) not imposing these coeffecients leads to infeasible constrants. This has to be tested with a carefully designed test suit which will test the limist of the platform.

Finally a case study on a real application on Zynq.