AFFORDe: Automatic Allocation and Floorplanning FOR SPMD Architecture

Wissem Chouchene
Jean-Luc Dekeyser
University of Lille1, CRISTAL
Lille, France
wissem.chouchene@inria.fr
jean-luc.dekeyser@univ-lille1.fr

Rabie Ben Atitallah
University of Valenciennes, LAMIH
Valenciennes, France
rabie.benatitallah@univ-valenciennes.fr

***Abstract*—Nowadays, Field Programmable Gate Arrays (FPGAs) platforms offer a high density to allow designing Multi Processor-based System on Chip. SPMD (Single Program Multiple Data) is a massively parallel execution model based on the assembly of a given number of homogeneous Processing Elements (PEs). This model is often relaying on Master/Slaves architecture composed by a Master PE that manages the parallel execution of a set of identical slave PEs. Furthermore, Dynamic Partial Reconfiguration (DPR) feature allows such computing system to be reconfigured on the fly for a given application requirement. Given the growing number of PEs in Master/Slaves architecture, it is difficult to estimate the time of specification and design during the phase of allocation and floorplanning of Partial Reconfigurable Regions (PRRs) because it is still performed manually. In this work, we present AFFORDe, a tool enable to automate the Xilinx DPR flow for SPMD architecture that allows parsing the resource requirements of the static and the dynamically reconfigurable parts to perform an automatic floorplanning. The floorplanning is based on a Heuristic Algorithm for Automatic Floorplanning in SPMD Architectures (HAAFSA). This tool is used to generate a configuration file that allows floorplanning of reconfigurable regions in an automatic way of a given Master/Slaves configuration. Experimental results show the effectiveness of our tool to increase the design productivity for dynamically reconfigurable SPMD-based architecture.**

Keywords—FPGA; Floorplanning; DPR; SPMD; TCL;

I- Introduction

Thanks to the significant advantages provided by FPGAs, large scale reconfigurable parallel systems are implemented to reduce the time execution of a given application. Single Program Multiple Data (SPMD) is one of widely usable models for parallelism. This model is associated in general with a regular architecture based on a Master Processing Element (PE) and a set of homogeneous slave PEs that execute the same program at the same time. Slave PEs are connected by the mean of a regular network topology and synchronized with a specific mechanism. Due to the need of high performance computing in recent application, SPMD architecture require an important design time and significant effort at each time the hardware/software application requirements vary. In fact, it is common for a designer to spend several months to develop new application for such architecture, followed by testing, debugging and integration. All of these efforts represent burdens of time and effort that could be eliminated through automation. Furthermore, Dynamic Partial Reconfiguration feature allows reconfigurable region in each slave PE to be shared by partial reconfigurable modules at each time the application requirement changes. It allows reducing hardware resources. Otherwise, one of the difficulties to use DPR is: 1) the need of deep knowledge of the FPGA structure. 2) The time spent to locate by hand many partial reconfigurable regions. Thus, floorplanning automation in case of DPR based design is preferred to avoid hand manipulation of PRRs. Furthermore, the regularity of both DPR-based SPMD architecture and FPGA structures allows performing a regular floorplanning. The regular floorplanning is characterized by specifying homogeneous PRRs to be placed in an organized way. Such floorplanning can enable bitstream relocation technique that reduces the partial bitstream memory size.

In this work, we present AFFORDe, a tool that automates the Xilinx DPR flow for SPMD architectures that parses the resource requirements of the static and the dynamically reconfigurable parts to allow automating the design floorplanning by the mean of HAAFSA. AFFORDe generates an UCF description that can consider any bitstream relocation.

This paper is organized as follows: Section II describes the previous works related to bitstream relocation and floorplanning techniques. Section III details our proposed approach to automate the DPR flow for SPMD architectures. Our floorplanning algorithm is exposed in Section IV while experimental results are shown in Section V.

II- Related Works

Massively parallel computing system like SPMD architecture including a huge number of PEs can help designers to achieve promising results and allows exploring various problems related to High-Performance Computing (HPC) [1], real time embedded system [2] [3], prototyping and design space exploration [4], [5], etc. The number of PEs is a user setting limited by two aspects: resource limitation and design complexity. There are several works based on automated methods. We mention Archborn [6] which is an open source tool that automates the generation of base CHMP [6] system through Tool Command Language (TCL) scripting. In fact, Vivado and PlanAhead, the latest Xilinx tools suite, provide extensive functionalities for all programmable platform FPGAs. In order to integrate the DPR functionality in SPMD architecture, designers can employ automated mechanisms based on TCL commands that allow creating and configuring all components, Programmable Blocks (PBlocks) in our case study. Furthermore, Vivado and PlanAhead provide the interface to import a set of TCL commands as a script file to automate the design. Also, these tools allow the extraction of technological parameters of the used FPGA through specific queries which is an essential aspect in our automated floorplanning process.

Recent floorplanning contributions have focused on two different aspects [7] [8]: The first one focuses on reducing the bitstream size by using Columnar Kernel Tessellation method which minimizes the cost of wasted resources. The second optimization aims to reduce the overall wire length by using simulating annealing. Mixed-Integer Linear Programming (MILP) formulation presented in [9] has been used to enhance the previous methods in terms of optimization research space. It presents two optimization algorithms, the first one is the Heuristic Optimal (HO) and is used to reduce the time execution by making the first founded solution as a constraint to minimizing the solution search space. The second one is called Optimal (O) and used to explore the full solution space but requiring a larger duration. The two algorithms was been extended to support bitstream relocation in [10]. The extended version allows the design re-use but compromises the solution coast in term of wire length and partial bitstream size.

III- Proposed Approach

With recent Xilinx FPGA platforms that support DPR feature, we focus on automating the design flow of DPR-based SPMD architecture by mainly the mean of the automation of the floorplanning step. Making profit from the regular and the repetitive structure of FPGA, the particularity of AFFORDe is its capability to perform a regular floorplanning adapted to such massively parallel computing system. Figure 1 present the proposed design tool.

AFFORDe consists of three main phases that ensure the automation of the DPR flow based on floorplanning and generating a UCF description compatible with Xilinx tools. First, an initialization phase which involves the creation of various related projects of the static part and the reconfigurable modules. The second phase begins with the creation of a PlanAhead project and extracts the resource requirements (Static and dynamic) obtained from the synthesize step and technological parameters from the Xilinx FPGA database. The third phase consists of recovering these parameters in order to perform the floorplanning algorithm to floorplan the dynamically reconfigurable regions. We detail these phases in the following sections.

A- The initialization Phase

This step is manifested by collecting the required information to initialize the creation of the PlanAhead project and the floorplanning phase. In fact, this phase allows firstly fixing the number of reconfigurable PEs in the Master/Slaves architecture which is a parameter set by the user. This number (Net_D) is considered as an essential parameter in our design since it includes the dimension of the network. The network dimension is determined as:

$$\text{Net\_D} = \text{NX\_Slaves} \times \text{NY\_Slaves} \quad (1)$$

Secondly, we recover the synthesis information according to generated netlists files (.ngc) relative to the static part and Partial Reconfigurable Modules (PRMs). Finally, this step takes into account the initial user constraints description of the system (memory mapping, the assignment of IOs,…). This description is included in an initial UCF file. These different types of settings are the basic inputs for creating a PlanAhead project of a dynamically reconfigurable design.

B- Extraction of Technological Parameters and Ressources Requirements Phase

1) Creating the PlanAhead Project: In order to extract resource requirements and technological parameters, a PlanAhead project must be created to allow the architectural analyzing of the system and the used FPGA platform proprieties where the floorplanning algorithm will be performed. The architectural analyzing allows determining the system resource requirements in terms of CLB, RAMB and DSP corresponding to the static part and PRMs. The project is created in TCL mode through a set of routine commands that take all netlist files and the initial UCF file as inputs. Since the SPMD system based on DPR is characterized by the execution of the same reconfigurable task in each slave PE, a unique configuration is instantiated in PlanAhead flow to implement this task in each PRR at the same time. It means that the number of configurations is equal of the number of PRMs. An additional configuration is added to implement BLANK modules at each PRR. This additional configuration is considered as an initial configuration that aims to save power before starting DPR. Listing 1 shows an example of creating a configuration based on PRM (my_module.ngc).

```
// Create a  configuration based on the PRM my_module.ngc
for {set i 0} {$i<$Net_D} {incr i} {
set_property edif_top_file my_module.ngc [lindex [get_filesets ] $i ]
import_files -fileset [lindex [get_filesets ] $i] -force -norecurse my_module.ngc
 save_constraints
 }
```

Listing 1: A configuration based on the *my_module* module

According to the DPR technique, a given PRR is a rectangular and physical shape that can support an unlimited number of PRMs. In PlanAhead flow, a PBlock is an abstraction of a PRR and used to specify the location of a given physical area. In our case the number of PBlocks is equal to the Net_D parameter value since we take an assumption that a PE contains a single PRR. Initially in the PlanAhead project, a

PRR is represented as a BLACKBOX entity. Listing 2 show how to recover the BLACKBOX list from a given project.

```
//Detemination of the BLACKBOX list
set list_bbx {}
set BBX {}
for {set i 0} {$i<$Net_D} {incr i} {
set BBX [lindex [get_cells -hier -filter {IS_BLACKBOX}] $i]
lappend list_bbx $BBX
}
```

Listing 2: Recovering BLACKBOX list

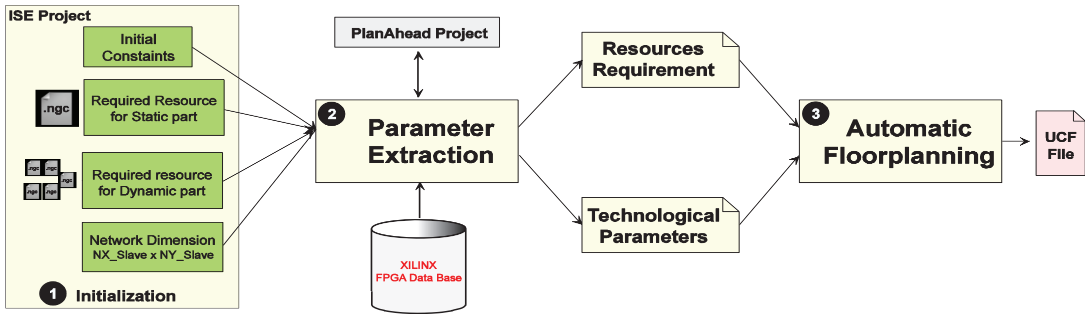


Fig. 1: AFFORDe Design Flow

2) Resources Requirement for Static and Dynamic Parts :Once the PlanAhead project is created, we proceed by analyzing the Master/Slaves architecture that we want to implement. This step is reflected in the determination of resources required by the static part and the different PRMs. The static part requirement can be determined by executing a specific TCL command to generate a report. This command is as follows: Report_state -all -file <Static_report_file _name>. To generate the different PRM reports, we perform different rules to obtain a detailed resources requirement for each PRM. To do so, we must create reconfigurables modules based on BLACKBOX list to let them associated with the corrsponding PBlocks. Listing 3 shows the TCL commands that allows creating these modules.

```
//Creating reconfigurable modules based on BLACKBOX list
for {set i 0} {$i<$Net_D} {incr i} {
create_reconfig_module -name RR$i -cell [lindex $list_bbx  $i] -
force
}
```

Listing 3: Creating reconfigurable modules

We generate the PBlocks_instantiation file that contains the PBlocks names reported with their associated PRRs (Black Box). This file will be used for the floorplanning phase and PBlocks instantiation will help us to generate the PRM reports.

```
set file [open " PBlocks_instantiation.dat " w]
set tiles_RR_name [get_reconfig_modules]
set pb_count [llength $tiles_RR_name]
for {set i 0} {$i<$pb_count} {incr i} {
set sites  [lindex $tiles_RR_name $i]
set s3 [string range $sites 0 [expr {[string last ":" $sites] - 1}]]
puts $file "$s3 [lindex [get_pblocks] $i]"
}
close $file
```

Listing 4: Generating the PBlocks_instantiation file

Listing 4 shows the TCL command to generate the PBlock_instantiation while the Listing 5 shows the TCL command used to generate the PRM reports.

```
for {set i 0} {$i<$Net_D} {incr i} {
report_stats -all  -tables   physicalResource -pblock [lindex
[get_pblocks] $i ] -file RR$i.txt
}
```

Listing 5: Generate resource requirement for reconfigurable modules

3) Technological Parameters: Technological parameters are obtained from a set of TCL queries to the Xilinx database. The extraction of these parameters is manifested by the exploration of the used FPGA platform structure. In fact, Xilinx FPGAs present a regular structure, they are divided by several columns (C) that can be symmetric or not. Columns are composed by a set of heterogeneous vertical lines of tiles. In Fig. 2, vertical lines of Tile_CLB, Tile_RAMB, and Tile_DSP are represented in blue, pink, and green respectively. Columns are also divided by equal Clock Regions (CR). Each CR is defined by its location in XY coordinates and is organized as a set of vertical sub-lines called frames. A frame is a set of tiles characterized by a fixed type and size. The frame size differs from an FPGA to another depending of the corresponding reference technology. With such structure, it is possible to determine the whole available resources in the FPGA in terms of tiles (locations and types) just by parsing a single horizontal line. To do so, it is important to calculate the FPGA total width which is equal to the sum of column widths.

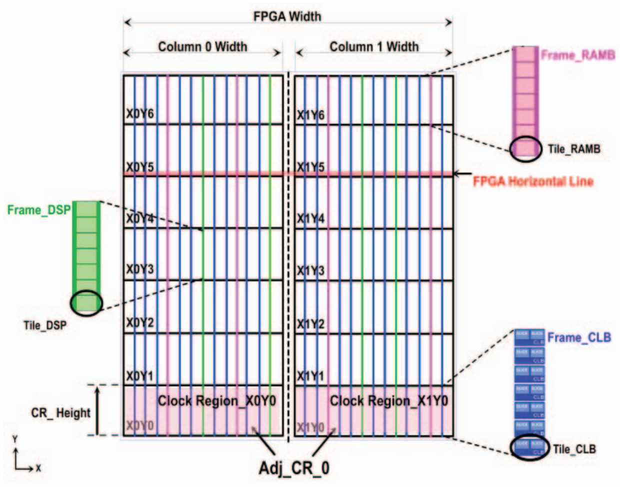


Fig. 2: Xilinx FPGA Structure

Then, based on the size and the type of the different frames and the number of CRs, the total of the available resources can be computed. A TCL command can generate a report that contains the available ressources of a given FPGA which is: Report_property [get_parts <FPGA reference>]. We give as follows TCL commands to determine the composition of a single horizontal line throught the following Listing 6.

```
set file [open "Ress_type_coords.dat" w]
set compo_list_name {}
set compo_list_type {}
set siteslen [llength [get_sites *X*Y0 -filter
    {SITE_TYPE=~RAMB* || SITE_TYPE=~SLICE* ||
    SITE_TYPE=~DSP*}]]
for {set i 0} {$i<$siteslen} {incr i} {
set sites  [lindex [get_sites *X*Y0 -filter
    {SITE_TYPE=~RAMB* || SITE_TYPE=~SLICE* ||
    SITE_TYPE=~DSP*}] $i ]
set prop [get_property NAME  $sites]
lappend compo_list_name $sites
set prop1 [get_property SITE_TYPE $sites]
lappend compo_list_type $sites
puts $file "$prop $prop1"
}
close $file
```

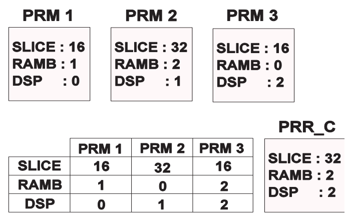
Listing 6: Composition of a single horizontal line

We mention some parameters to use: the number of available resources in terms of CLB, RAMB and DSP, the number of clock regions, the size of different type of frames per clock region, the CLB types, orientation and XY coordinates, and the useless areas ranges. We detail those parameters in the next section. We use the Xilinx TCL language to recover and generate setting files of all these parameters. The results of these queries are generated in the form of organized structures to be useful by HAAFSA.

C- Automatic Floorplanning

We have developed HAAFSA that takes advantage of the regularity of both FPGA platforms and SPMD architectures. This algorithm performs the floorplanning considering any bitstream relocation technique to provide factorized single partial bitstream functionality. Our algorithm presents different steps that aim to floorplan automatically Net_D identical PRRs. It consists in finding a set of PRR references in one Adj_CR, and then replicates them to the remaining Adj_CRs by using the translation vector CR_Height. According to Fig. 2, an Adj_CR is all adjacent clock regions along the X axis (in our case, Adj_CR= 7). Adj_CR_0 is considered as the reference Adj_CR of our circuit.

Before describing details of our algorithm it is important to transform the resource requirements by the system into useful architectural parameters. Since all PRRs are similar in our architecture and each one of them allows holding several PRMs with different resources requirement, it is necessary to define a common PRR (PRR_C). A PRR_C is the union of the maximum resources (SLICE, RAMB, DSP) required by each PRM as shown in the example of Fig. 3. The table presented in the example shows the resources requirement of the modules PRM1, PRM2 and PRM3. The PRR_C contains the maximum resources requirement of each type (SLICE = 32, BRAM=2 and DSP=2).



PRM 1	PRM 2	PRM 3
SLICE : 16 RAMB : 1 DSP : 0	SLICE : 32 RAMB : 2 DSP : 1	SLICE : 16 RAMB : 0 DSP : 2

	PRM 1	PRM 2	PRM 3
SLICE	16	32	16
RAMB	1	0	2
DSP	0	1	2

PRR_C
SLICE : 32 RAMB : 2 DSP : 2

Fig. 3: Determination of the PRR_C size from several PRM requirements

Now, it is possible to determine the feasibility (F) of our implementation. The equation below leads to compute F.

$$F = (AR - SR) - (Net\_D * PRR\_C) \quad (2)$$

Where AR is the available resources on FPGA, SR is the required resources by the static part, Net_D is the number of PRR_Cs. All resource requirements are presented in the form of vectors in function of (CLB, BRAM and DSP). If F is negative, an error notification is generated to the user. We mention that an additional resource in terms of LUT is added to a PRR_C due to the needed Proxy Logic [11].

Once the feasibility of the implementation is assured (positive), the algorithm can begin. We present as follows the different steps of HAAFSA.

1) The Floorplanning Mode: We defined four floorplanning modes according to the required heterogeneous resources with the additional amount of LUT for Proxy Logic. These modes are: Mode 1 (CLB), Mode 2 (CLB+BRAM), Mode 3 (CLB+DSP) and Mode 4 (CLB+BRAM+ DSP).

2) Determining the number of PRR_C references(Z) in a single Adj_CR: Z is calculated from the following equation:

$$Z = \left\lceil \frac{\text{Net\_D} * \text{C}}{\text{nbCR}} \right\rceil \quad (3)$$

Where C is number of columns and nbCR is the number of all clock regions in the FPGA device. C and nbCR are some technological parameters extracted from the privious step.

3) Detrmining the number of frames required by a PRR_C: Since CLB, BRAM and DSP frames have fixed sizes, the calculation of the required frames for a PRR_C is given by the following equation:

$$\text{Vect} = \begin{pmatrix} \text{PRR\_C}_{\text{CLB}} \\ \text{PRR\_C}_{\text{BRAM}} \\ \text{PRR\_C}_{\text{DSP}} \end{pmatrix} / \begin{pmatrix} \text{Frame\_C} \\ \text{Frame\_R} \\ \text{Frame\_D} \end{pmatrix} = \begin{pmatrix} N_{\text{CLB}} \\ N_{\text{BRAM}} \\ N_{\text{DSP}} \end{pmatrix} \quad (4)$$

Where $\text{PRR\_C}_{\text{CLB}}$, $\text{PRR\_C}_{\text{BRAM}}$ and $\text{PRR\_C}_{\text{DSP}}$ are the resources required by a single PRR_C. Frame_C, Frame_B and Frame_D are technological parameters and are respectively the frame sizes of CLB, BRAM and DSP. Vect is the resulting vector that indicates the required frame number by a PRR_C. We mention that if one of the Vect values is greater than 1 the PRR_C height will be equal to CR_Height since Frame_C, Frame_B and Frame_D are geometrically equal. Else the PRR_C height will be adjusted.

4) Determining the Z locations: The algorithm iterates through the FPGA width (X_max) in terms of tiles on a single horizontal line to find a minimal width of each PRR_C according to Vect vector. Each PRR_C width must be comprised between a CLB_L and a CLB_R respecting the Xilinx DPR flow. Hence, the PRR_C width is deduced from the CLB_L and the CLB_R locations.

In general, for a given FPGA, BRAMs are the most critical resources while CLBs are present with a huge quantity. Furthermore, the destruction of such resources along the X axis is not regular in general. This aspect leads us to optimize the PRRs width as a given partial bitstream size depends on it. The first step in this process is to find a first PRR_C then saving in an array (SEQ) its sequence of tiles forming its width. Finally, the SEQ sequence serves to looking for the same occurrence for the remaining (Z-1) PRR_Cs in a single Adj_CR. According to the floorplanning mode, three arrays are generated (CLB_loc, BRAM_loc and DSP_loc) to contain the Z*PRR_C locations.

5) Determining the PRR_C height: The PRR_C height (H) is determined from the equations below:

$$H = \begin{cases} \text{CR\_Height}, & N_{\text{CLB}} > 1 \text{ or } N_{\text{BRAM}} > 1 \text{ or } N_{\text{DSP}} > 1 \\ \lceil \text{MAX (H0)} \rceil, & \text{Otherwise} \end{cases} \quad (5)$$

H is geometrically equal to the maximum value of the PRR_C elements. In fact, since all frame types are geometrically equal, it is essential to make a common representation that express one frame type in function of the other frame types. It is performed through the normalization of the $\text{Frame}_{\text{size}}$ = (Frame_C, Frame_B, Frame_D) vector. We call H0 the resulting vector from the division operation of PRR_C by the normalized $\text{Frame}_{\text{size}}$ vector. By this way H is the maximum of the H0 elements.

$$\text{H0} = \begin{pmatrix} \text{PRR\_C}_{\text{CLB}} \\ \text{PRR\_C}_{\text{BRAM}} \\ \text{PRR\_C}_{\text{DSP}} \end{pmatrix} / \begin{pmatrix} \text{Frame\_C/Frame\_B} \\ \text{Frame\_B/Frame\_B} \\ \text{Frame\_D/Frame\_B} \end{pmatrix}_{\text{BRAM}} \quad (6)$$

Once H is determined all locations can be generated from the CLB_loc, BRAM_loc and DSP_loc arrays for Z*PRR_C in a single Adj_CR.

6) Replication Phase: A replication step is performed to duplicate the location listed in the CLB_loc, BRAM_loc and DSP_loc arrays through the $\textit{Frame}_{\textit{size}}$ vector to the remaining Adj_CRs. During the replication loop an UCF description is generated to perform the location constraints for Net_D comon partial reconfigurable regions (corresponding to its PBlocks).

7) HAAFSA Complexity: HAAFSA is a quadratic algorithm in function of $O(Z^2)$ and it is determened as follows: We consider the variable SEQ_L that represents the SEQ array length. We demenstrate the operation cost (mainly comparision) when floorplanning PRR_C references along the Adj_CR width (X_max). Let nbIT be the number of comparisons. We consider the most complex case study which is the Mode 4 with PRR height adjustment.

The first step consists in finding a BRAM as near as possible to a DSP. The number of maximum comparisons here is equal to X_max^2. The second step consists in satisfying the placement of the CLB_L and CLB_R constraints. It consists of iterating in two sides right and left around the I = [BRAM DSP] interval. The worst case is where I is positioned in the middle of the Adj_CR. Hence, both iterations right and left cost are equal to (X_max)/2. By this way, the number of total operations to find the first PRR_C is:

$$\text{nbI T}_1 = \text{X\_max}^2 + \text{X\_max} \quad (7)$$

We apply the same methodology to find the remaining (Z-1) PRR_Cs. It consists to find the next BRAM tile on the FPGA then compare the SEQ sequence with the corresponding BRAM and its neighbor's positions. The remaining operations number to locate the rest of PRR_Cs is:

$$\begin{aligned} \text{nbIT}_2 &= 2 * (\text{X\_max} - \text{SEQ\_L}) \\ \text{nbIT}_3 &= 2 * (\text{X\_max} - 2 * \text{SEQ\_L}) \\ \text{nbIT}_4 &= 2 * (\text{X\_max} - 3 * \text{SEQ\_L}) \\ &(\ldots) \\ \text{nbIT}_z &= 2 * (\text{X\_max} - (\text{Z-1}) * \text{SEQ\_L}) \quad (8) \end{aligned}$$

In general the complexity cost is:

$$\text{Complexity} = \sum_{i=1}^{Z} \text{nbIT}_i$$

$$\text{Complexity} = X\_max^2 + X\_max + 2[(Z-1).X_{max} - SEQ\_L.\sum_{i=1}^{Z-1} i]$$

$$\text{Complexity} = 2[(Z-1).X\_max - SEQ\_L.(\frac{1}{2}(Z^2 - Z))] + X\_max^2 + X\_max$$

$$\text{Complexity} = -Z^2.SEQ_L + Z.[SEQ_L + 2.X\_max] + X\_max^2 - X\_max \tag{9}$$

IV- Experimental Results

We have implemented a Master/Slaves architecture based on SPMD model. Slaves PEs are connected to a generic Mesh network with (NX_Slaves*NY_Slaves) dimension. The communication tasks are managed by the master PE. All PE slaves execute the same program and each of them contains a partial reconfigurable module that can be triggered by a dedicated instruction.

We consider a given PRR_C to be implemented in each slave PRR and we specify (7x3), (3x3), (5x5) and (8x8) network dimensions. We use the Xilinx Virtex7 XC7VX485T device with {Frame_C = 50, Frame_R = 10 and Frame_D = 20} to implement these configurations. We present in Table I the resources requirement for the static part for each system dimension.

TABLE I. Ressources Requirement For The Static Part

System Dimension	Resource Requirements	
	LUTs	BRAM
3x3	34.140 (4%)	99 (9%)
7x3	56.225 (7%)	187 (19%)
5x5	64.155 (8%)	189 (19%)
8x8	146.689(49%)	209 (21%)

TABLE II. Ressources Requirement For Reconfigurable Modules

PRMs	Resource Requirements		
	Slices	BRAM	DSP
PRM_1	48	3	5
PRM_2	32	7	3
PRM_3	96	5	0
PRR_C requirement	96	7	5

We use a set of reconfigurable modules to implement in every PRR in each slave PE. The resources requirements for these modules are presented in Table II. We consider PRMs with two input ports of 32 bits and an output port of 32 bits. The required Proxy Logic is equal to 96 LUTs (= 24 Slices). Note that a CLB contains 2 Slices the total required resources for a PRR_C are as follows:

$$PRR\_C = \begin{pmatrix} 48 \\ 7 \\ 5 \end{pmatrix}$$

According to the Vect vector, the PRR_C consumes one frame of each resource type and its height can be computed according to the H0 value.

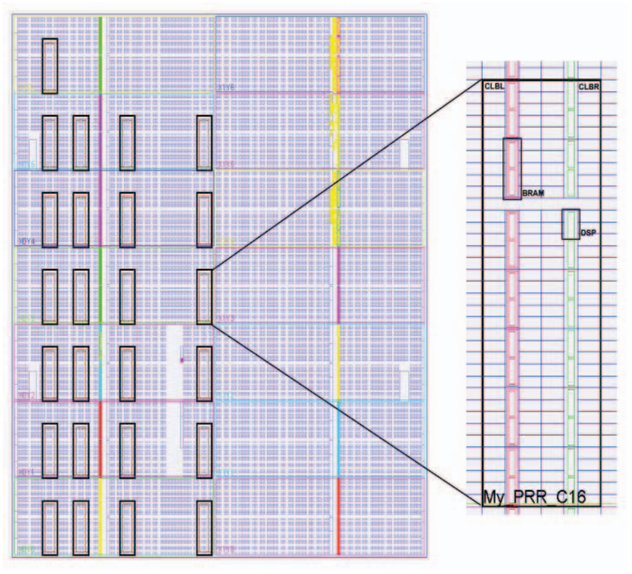


Fig. 4: Floorplanning view of 5x5 PRRs

Since the PRR_C require 2 CLB_R/L at its width limits, the required CLBs must be divided by 2.

$$H0 = \left\lceil \begin{pmatrix} 24 \\ 7 \\ 5 \end{pmatrix} / \begin{pmatrix} 5 \\ 1 \\ 2 \end{pmatrix}_{BRAM} \right\rceil = \left\lceil \begin{pmatrix} 4.8 \\ 7 \\ 2.5 \end{pmatrix} \right\rceil \Rightarrow H = 7$$

The PRR_C height is adjusted with 7 BRAMs in the same Frame_B. We show in Fig. 4 the resulting floorplanning for a system based on 5x5 dimension. We give additional technological parameters of the used FPGA: C =2, nbCR =14 and X_max = 256. According to the equation (3), Z = 4.

In order to evaluate the HAAFSA performance, we implement the system configurations presented in Table I. Experiments are conducted using an Intel Xeon CPU E31270-based host of 3.4GHZ and 16 GB of RAM. Figure 5 shows the theoretical evaluations of number of operations in function of the system size according to equation (9).

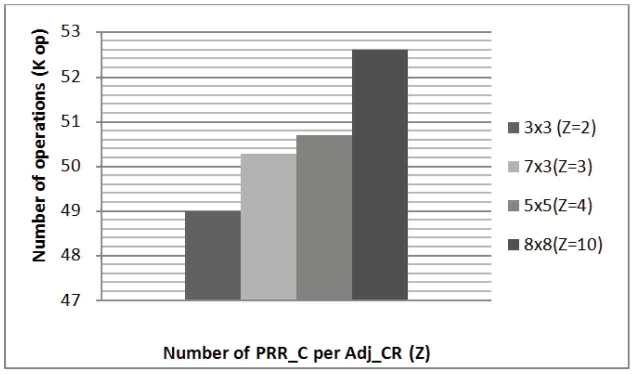


Fig. 5: Evaluation of operation number in function of the system dimension

Based on the obtained behavior of our algorithm complexity, we can confirm its efficiency to floorplan partial reconfigurable regions in large systems. According to real time execution of the corresponding configurations as shown in Fig.6, our automatic algorithm leads to a high design productivity compared to the manual floorplanning that can take several hours and require a deep knowledge of the FPGA architecture.

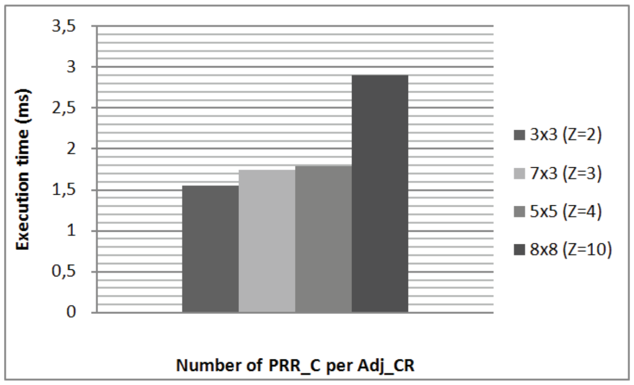


Fig. 6: Execution time evaluation

V- Conclusion And Future Works

We have presented AFFORDe, a tool that allows automating the Xilinx dynamic partial reconfiguration flow for SPMD architecture thanks to the TCL scripting. Our tool is based on an automatic floorplanning performed by our rapid algorithm that relies on the regularity of the FPGA structure and ensures high design productivity. Furthermore, our algorithm allows the bitstream relocation technique for such systems in order to allow partial bitstream reuse. Experimental results show the effectiveness of our algorithm to floorplan SPMD systems based on dynamic partial reconfiguration.

For future work, we focus on implementing a unification flow that allows obtaining an identical mapping and P&R for all instantiated PRRs. This flow permits to perform the relocation technique.

References

[1] D. Gohringer and J. Becker, “Fpga-based runtime adaptive multiprocessor approach for embedded high performance computing applications,” in VLSI (ISVLSI), 2010 IEEE Computer Society Annual Symposium on, July 2010, pp. 477–478.

[2] M. Samarawickrama, R. Rodrigo, and A. Pasqual, “Scalable fpga based multiprocessor architecture for real-time embedded vision,” in Information and Automation for Sustainability (ICIAFs), 2010 5th International Conference on, Dec 2010, pp. 172–176.

[3] T. Nguyen and A. Kumar, “Pr-hmpsoc: A versatile partially reconfigurable heterogeneous multiprocessor system-on-chip for dynamic fpga-based embedded systems,” in Field Programmable Logic and Applications (FPL), 2014 24th International Conference on, Sept 2014, pp. 1–6.

[4] S. Fernando, F. Siyoum, Y. He, A. Kumar, and H. Corporaal, “Mampsx: A design framework for rapid synthesis of predictable heterogeneous mpsocs,” in Rapid System Prototyping (RSP), 2013 International Symposium on, Oct 2013, pp. 136–142.

[5] R. Van Langendonck, A. Lusala, and J. Legat, “Mpsocdk: A framework for prototyping and validating mpsoc projects on fpgas,” in Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2012 7th International Workshop on, July 2012, pp. 1–8.

[6] Sen Ma; Hongyuan Ding; Miaoqing Huang; David Andrews,"Archborn: an open source tool for automated generation of chip heterogeneous multiprocessor architectures", in ReConFigurable Computing and FPGAs (ReConFig), 2015 International Conference, pp 1-6

[7] K. Vipin and S. A. Fahmy, “Architecture-aware reconfiguration-centric floorplanning for partial reconfiguration,” in Proc. Intl. Conf. on Reconfigurable Computing: architectures, tools and applications (ARC), 2012, pp. 13–25.

[8] C. Bolchini, A. Miele, and C. Sandionigi, “Automated Resource-Aware Floorplanning of Reconfigurable Areas in Partially-Reconfigurable FPGA Systems,” in Proc. Intl. Conf. on Field Programmable Logic and Applications (FPL), 2011, pp. 532–538.

[9] M. Rabozzi, J. Lillis, and M. D. Santambrogio, “Floorplanning for Partially-Reconfigurable FPGA Systems via Mixed-Integer Linear Programming,” in Proc. Intl. Symp. on Field-Programmable Custom Computing Machines (FCCM), 2014, pp. 186–193.

[10] M. Rabozzi; R. Cattaneo; T. Becker; W. Luk; M. D. Santambrogio, “Relocation-Aware Floorplanning for Partially-Reconfigurable FPGA-Based Systems” in Proc. Intl. Conf. on Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015, pp. 97–104.

[11] T. Drahonovský et al, “Relocation of reconfigurable modules on Xilinx FPGA”, in Proc. Intl. Conf. Design and Diagnostics of Electronic Circuits & Systems (DDECS), 2013 pp, 175-180.