Efficient Region Allocation for Adaptive Partial Reconfiguration

Kizheppatt Vipin, Suhaib A. Fahmy

School of Computer Engineering
Nanyang Technological University
Nanyang Avenue, Singapore
vipin2@e.ntu.edu.sg, sfahmy@ntu.edu.sg

***Abstract*—While partial reconfiguration on FPGAs has attracted significant research interest in recent years, designing systems that leverage it remains a specialist skill. Systems with a large number of reconfigurable modules can be challenging to design. Deciding on how many reconfigurable regions to use is not always straightforward, yet this choice impacts area efficiency and configuration latency. Current FPGA partial reconfiguration tool flows require the designer to have detailed knowledge of the physical architecture of the FPGA. It is the responsibility of the designer to decide on the location and size of regions. In this paper we introduce a formulation for determining the tradeoff between the number of reconfigurable regions, the allocation of modules to regions, and the reconfiguration overhead, represented by area and reconfiguration time. Throughout the investigation, we consider the heterogeneous nature of modern FPGAs as well as limitations imposed by current tools. We then show that adopting an optimal allocation can result in both area savings and a reduction in reconfiguration time over a standard approach to allocation.**

I. Introduction

Partial reconfiguration has remained a constant research theme within the FPGA community since it was first mooted. The core premise — that since the full configuration of an FPGA is stored in volatile memory it can be changed at runtime — while simple and enticing, has seen many technical hurdles prevent it from being adopted in mainstream design. Coupled with these challenges is the search for applications that can truly take advantage of this feature.

The circuit implemented in an FPGA can be altered by modifying the contents of its configuration memory. In a full FPGA configuration, the contents of the whole configuration memory are modified. Selectively modifying the contents of only part of it is called Partial Reconfiguration (PR). Reconfiguring only a portion of the FPGA means the time required, and energy consumed, for reconfiguration are reduced [1]. Some modern applications can adapt their processing to environmental conditions. This adaptation typically relies on switching between multiple *modes* of operation; this adaptation can be efficiently implemented using partial reconfiguration.

Although runtime partial reconfiguration has advantages over full reconfiguration, it presents a greater challenge to the designer, and current tools impose a number of limitations. The designer must decide how to partition the FPGA between the *static*, always-on, parts and reconfigurable regions, requiring detailed knowledge of the FPGA architecture. The designer must also determine which modules should be assigned to which regions and hence the granularity of reconfiguration. Some work has been done to mitigate these limitations, but existing approaches are only suitable for use by FPGA experts. Hence the system designer and the module designer are typically one and the same, with extensive hardware experience. For systems incorporating partial reconfiguration, design decisions can impact the cost of using PR. The number of regions used, and the allocation of reconfigurable modules to regions can impact resource requirements considerably. Hence for greater savings, this allocation should be done intelligently.

The primary cost associated with partial reconfiguration is reconfiguration time. In a circuit in which all functionality is present on chip, selecting between modes typically takes just a few clock cycles. When using PR, the system must pause, and the PR region must be reconfigured, which takes time. Smaller regions result in shorter reconfiguration times, but can impact area efficiency. Hence, in this work, we try to optimise these two factors together to arrive at an optimal allocation.

In this paper, we propose techniques for automatically determining the optimal reconfiguration scheme for a given adaptive application. Based on a representation of the application, the tools propose the best arrangement of reconfigurable regions and how modules should be assigned to those regions. In this paper, we are most interested in adaptive applications where reconfiguration occurs at the module level, such as a change in coding scheme in a radio system. This sort of reconfiguration occurs on an unpredictable, as-needed basis, and is at the functional, rather than task, level. The aim is that an adaptive systems designer can design a system that takes full advantage of partial reconfiguration, using a library of components developed by RTL designers, without needing experience in low-level FPGA details.

The rest of this paper is organised as follows: Section II discusses related work, Section III investigates the existing partial reconfiguration tool flow for Xilinx FPGAs, Section IV discusses the reasoning behind the method and the mathematical formulation, Section V presents some results of optimal region allocation for an example application and Section VI concludes the paper.

II. Related Work

Much has been published in recent years concerning partial reconfiguration (PR). This has included methods for efficient partial reconfiguration and practical applications using PR. When Xilinx introduced partial reconfiguration for the first time in Virtex devices, there were severe restrictions on how reconfigurable regions could be arranged. They had to extend the whole height of the device and the designer had to manually place additional hardware blocks called "bus-macros" between the static and dynamic regions to preserve routing between them. In [2], the authors describe a rapid prototyping methodology for Xilinx FPGAs taking into account these limitations. Modules are implemented in slots extending the whole height of the device and are connected to a standard shared bus. The designer merges the module bitstream with the configuration bitstream using a tool called JBits. In [3], two methods are proposed for partial reconfiguration in Virtex devices. The first method uses partially reconfigurable regions extending the whole height of the device, while the second method allows modules to be assigned arbitrary rectangular shapes, by way of a new bitstream merging process and reserved routing.

In [4] a framework for dynamic reconfiguration is introduced and the authors describe challenges in PR design including partitioning and floor planning, but no solution for partitioning is suggested. [5] and [6] describe floorplanning methods for partial reconfiguration once manual partitioning has been performed by the designer. A method to assign modules to regions, given a fixed number of regions, based on communication graphs, is presented in [7].

Work on high-level abstraction of PR systems at runtime has been done. In [8] a software framework for implementing adaptive systems on FPGAs is described. This work uses a layered structure separating control and processing planes and decoupling system adaptation from FPGA hardware reconfiguration.

In terms of cost analysis of partial reconfiguration, performance bounds for high-performance computing are evaluated in [9]. This study develops an execution model for runtime partial reconfiguration and analyses the results on the Cray XD1 reconfigurable computer. In [10] the authors investigate the energy efficiency of partial reconfiguration and show that the energy savings depend significantly on the speed of the reconfiguration process.

While bottom-up, architecture-based research such as online placement has proven interesting, recent devices with more heterogeneous blocks and more complex non-uniform routing make these approaches harder to implement and keep up to date [11]. Hence we feel it is more effective to develop tools that integrate with vendor-supported tool flows to address the challenges of partial reconfiguration, focussing more on top-down simplification of this unique feature of FPGAs for use by those who are not architecture experts. Furthermore, as vendor tools increase in sophistication, the effectiveness of these tools can also increase.

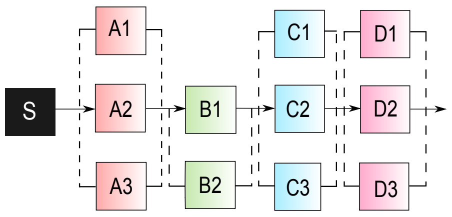

Fig. 1. Example PR design.

III. Partial Reconfiguration Tool Flow

It is first important to define the terms we use in this paper. A PR *region* is an area on the device allocated to logic, that can be reconfigured at runtime. It includes different types of basic primitives such as Slices, BlockRAMs, DSP Slices etc. A *module* is a processing unit in the design and may have multiple *modes*. In this discussion, *modes* are mutually exclusive implementations of the module with the same set of inputs and outputs. At runtime, we may wish to switch a *module* from one *mode* to another. A *configuration* is a set of possible co-existent modes for all the *modules* in the system. Since not all possible combinations of module modes will be valid, this allows us to focus on solutions that are optimal for valid sets of modes.

For efficient implementation and management, a hierarchical module based design approach should be followed for PR designs [1]. Fig. 1 shows an example PR design. The first task is to divide the design into static logic and reconfigurable modules. The functionality of the static logic does not change during FPGA operation. There can be one or more reconfigurable modules. In Fig. 1, module S represents the static logic and modules A, B, C, and D represent reconfigurable modules. A_1, A_2, and A_3 are different modes of reconfigurable module A. The designer should ensure that each region contains sufficient resources to implement all modes of the modules assigned to that region. This requires FPGA architecture knowledge and manual floorplanning. For the example design, $S \rightarrow A_1 \rightarrow B_1 \rightarrow C_1 \rightarrow D_1$ is a possible configuration. Each configuration contains the static logic and one of the possible modes for each reconfigurable module. In the current supported tool flow, configurations do not play any role in synthesis, since the reconfigurable modules and the assignment of regions, are performed manually.

IV. Problem Formulation

A. Fundamentals

From Section III, it is clear that the current supported PR tool flow requires extensive input from the designer, who is expected to know the details of the target FPGA architecture, as well as preparing all region netlists. In this paper, we formulate an analytical model, which can integrate with the existing PR tool flow in order to generate efficient PR designs without detailed input from the designer. Given an application description this tool will define the optimal number, and size, of partial regions and the assignment of modules to those regions. It will then pass this information and the resulting netlists to the PR implementation tools.

The simplest way to partition the FPGA for partial reconfiguration is to divide the whole FPGA into two: one static region and one PR region. All the static logic in the design is implemented in the static region, while modules that require reconfiguration are implemented in the PR region. This approach has some benefits such as the designer only needs to allocate a single region, large enough to hold the most resource hungry configuration and the tools can optimise area usage and timing across all modules, resulting in the best possible timing performance and area. However, some major drawbacks mean this method is not ideal. Firstly, each time any module in the region needs to be reconfigured, the whole region must be reconfigured, and so the frequency of reconfiguration will be the sum of the frequencies of reconfiguration for all modules. Secondly, since the whole region must be reconfigured even if a small module is being changed, the reconfiguration time is increased, in some cases significantly. Finally, designs with many possible combinations of modes will require a large bitstream for each possible configuration, resulting in significant storage being required to store bitsreams. Hence, simply allocating all reconfigurable modules to a single region is not ideal for systems that rely on minimising reconfiguration time and bitstream storage.

Generally, a one-region-per-module approach will offer the lowest worst-case reconfiguration times, since a region will only reconfigure when its sole module needs to, and the size of the region is only as large as the largest mode of that single module. However, a one-region-per-module approach is the least area-efficient way to allocate regions.

An analytical approach would allow us to determine how many regions to use, and which region each module should be allocated to, with a view to minimising reconfiguration time, while also maintaining area efficiency. System configurations play an important part in the following discussion. For most systems, not all combinations of module modes will be encountered at run time. In fact, there are normally a small number of possible combinations, that are referred to as configurations. Configurations greatly reduce the search space, since we only need to consider possible mode combinations to arrive at an optimal allocation. By using *configurations*, we limit the cost of combining modules into regions, and allow further analysis of the reconfiguration time. Consider two modules, each with a large and small mode, as shown in Fig. 2. If they are allocated to separate regions, the regions must each be large enough for the largest mode of the corresponding modules. However if we know that the largest modes for both modules never exist together, then if they are combined in a single region, that region only needs to be large enough for the largest overall configuration.

B. Proposed Tool Flow

Our proposed tool flow for solving this PR region allocation problem is shown in Fig. 3. The designer provides design files for all modules (in all modes), a list of allowable

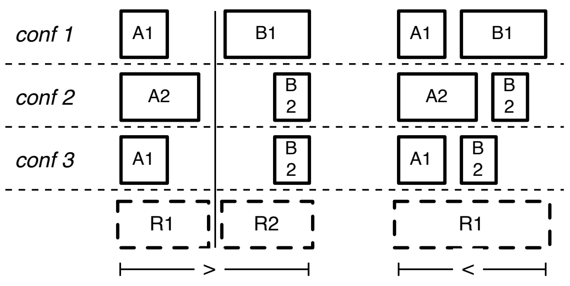

Fig. 2. When assigning modules to separate regions, if some configurations do not exist, combining modules into a single region can save area.

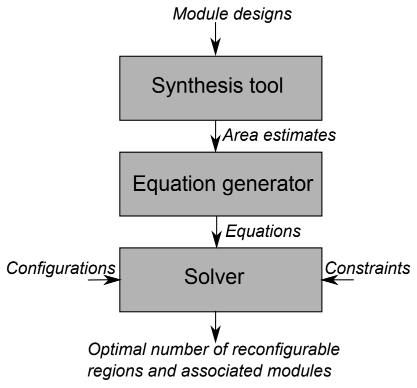

Fig. 3. Proposed tool flow.

configurations and design implementation constraints (such as timing requirements) to the automation script in XML format. The script performs the following steps:

1) Vendor supplied *XST* tool is used to synthesise all modules in order to extract the resource requirement information for all modes of all modules. (Alternatively resource estimation techniques can also be used [12]. If IP cores are used for some modules, usage information is often available up front.)
2) Resource requirements are passed to the equation solver with a list of valid configurations. It uses these inputs with the formulation discussed in the next section, and determines the optimal region configuration.
3) Once a region allocation has been decided, wrapper modules are created that represent different configurations for each region.
4) A netlist for each wrapped configuration is then automatically generated using the vendor synthesis tools.
5) Determining the floorplanning of the static and reconfigurable regions must then be done. A number of floorplanners are available that are tailored for partial reconfiguration [5], and these may be used. Alternatively, a manual approach using Xilinx *PlanAhead* is possible. We are also currently working on a more efficient floorplanner, targeting the uneven distribution of heterogeneous resources on modern FPGAs.
6) The area constraints generated by the floorplanner along with the timing requirements are used to generate the

User Constraints File, which directs the place and route tool to achieve design goals. The netlists generated, as well as the *User Constraints File*, are passed to *PlanAhead*, which performs the placement and routing operations.

C. Mathematical Formulation

To solve the allocation problem, we represent it mathematically using an objective function and a number of constraints. Based on the previous analysis, the problem of finding the optimal number of regions and the region that each module should be assigned to can be described using the objective functions:

1) Minimise the total FPGA resource requirement,
2) minimise average reconfiguration time.

Subject to the conditions:

1) All modules in the design being implemented,
2) all required configurations being implemented,
3) each module being implemented only once,
4) the design fitting in the given device,
5) the number of PR regions being greater than or equal to 1 and less than or equal to the total number Reconfigurable Modules.

We denote the variables used in the formulation as shown in Table I.

TABLE I
NOTATION USED IN FORMULATION.

Notation	Meaning
F_i	Total amount of resource type i in the FPGA (types can be Slice, BRAM, DSP)
N	Total number of reconfigurable modules
C	Set of Configurations
R_s	Reconfiguration speed
d_{uq}	Decision variable – 1 if module u is present in reconfigurable region q otherwise 0
d_{umc}	Decision variable – 1 if module u is present in mode m in configuration c otherwise 0
R_{uiMAX}	Maximum number of resource type i used by module u
R_{umi}	Number of resource type i used by module u in mode m
R_{di}	Total requirement of resource type i in the partition scheme
R_{qic}	Number of resource type i used in region q in configuration c
R_{qi}	Maximum number of resource type i consumed by region q
A_q	Area of region q in normalized units
W_i	Area weighing factor for resource type i
W_{fi}	Number of frames in resource type i
t_q	Reconfiguration time for region q
t_c	Reconfiguration time for configuration c
t_w	Worst case reconfiguration time
t_a	Average reconfiguration time

The maximum number of resource type i for module u is given by the maximum usage of i in different modes of u:

$$R_{uiMAX} = \max_m(R_{umi}), \tag{1}$$

Since each module should be implemented only once, the sum of allocation decision variables should be 1:

$$\sum d_{uq} = 1. \tag{2}$$

If multiple reconfigurable modules are merged into a single region, the area required for resource type i for each configuration c is determined as follows. The area required for each mode of module u is taken into account only if the mode exists in the current configuration c. The area required for each module is summed over all modules present in region q. The partition method can vary from using a single region to using a separate region for each module.

$$R_{qic} = \sum_u R_{umi} * d_{umc} * d_{uq}; c = 1, 2, ...C; q = 1, 2....N \tag{3}$$

From the set of resource requirements for different configurations, the maximum resource requirement for type i is determined, which is the required result:

$$R_{qi} = \max_c(R_{qic}) \tag{4}$$

The total amount of resource type i required for the whole design is the sum of resource type i among all the regions:

$$R_{di} = \sum_q R_{qi} \tag{5}$$

In order for the design to fit into a particular FPGA, for each resource type i, the total resources required should be less than or equal to resource type i present in the FPGA:

$$F_i - R_{di} \geq 0 \tag{6}$$

The total area cost of region q is given by:

$$A_q = \sum_i W_i * R_{qi} \tag{7}$$

where W_i is the weighing factor for resource i calculated as the ratio of total resources to resources of type i.

Now consider reconfiguration time. Reconfiguration time for region q can be determined by dividing the area of q by the speed of configuration.

$$t_q = \sum_i W_{fi} * R_{qi}/R_s, \tag{8}$$

where W_{fi} is the weighing factor determined by the number of reconfigurable frames required for resource type i. When modules are merged, reconfiguration of any of the modules in the region will lead to the reconfiguration of the whole region. The frequency of reconfiguration is application dependent. Total configuration time for the system when it changes from configuration c_i to configuration c_j is calculated as the sum of the configuration time for regions, whose modules change their modes.

$$t_c = \sum_q t_q * d_{cq}, \tag{9}$$

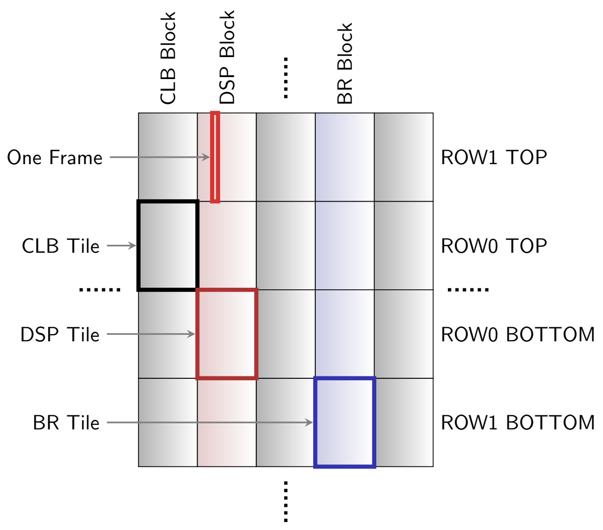

Fig. 4. Virtex 5 FPGA architecture.

where $d_{cq} = 1$ if, for any $d_{uq} = 1$, $d_{umc_i} \neq d_{umc_j}$ else 0. Average reconfiguration time is the average of all possible configuration times.

$$t_a = \bar{t_c} \tag{10}$$

Worst case reconfiguration time (t_w) for a partition scheme is calculated as the maximum reconfiguration time among all possible configuration transitions.

$$t_w = \max(t_c) \tag{11}$$

In order to improve the overall system performance, average reconfiguration time is selected as the minimisation objective. For applications where a strict reconfiguration time limit must be met, worst case reconfiguration time can be selected as the objective function. The solutions to the equations are generated using an equation solver written in C. The average reconfiguration time results are then plotted against resource requirements and the Pareto-Optimal points determined.

V. Optimal Region Allocation

A. Architecture Analysis

In Xilinx Virtex FPGAs, the smallest reconfigurable unit is a *frame*. In Virtex-5 FPGAs, the height of a *frame* spans an entire device row [13]. The number of rows[1] in a device depends upon the size of the device. In the Virtex-5 FX70T, which contains 11,200 Slices, 128 DSP Slices and 296 BlockRAMs, there are eight rows. A row crosses columns of different resource types, including Configurable Logic Blocks (CLBs), DSP Slices and Block RAMs. These columns extend the full height of the device and are referred to as *blocks*. A tile is one row high and one block wide, and contains a single type of resource as shown in Fig. 4. One CLB tile contains 20 CLBs, one DSP tile contains 8 DSP Slices and one BRAM tile contains 4 Block RAMs. For accurate calculation of efficient partial reconfiguration schemes, the reconfigurable regions should be considered in terms of these basic tiles since configuration must occur on a per-tile basis. Hence in all equations, the resource i represents one tile of resource type i (Slice, DSP, BRAM, etc). The XC5VFX70T device contains 280 CLB tiles, 16 DSP tiles and 74 Block RAM tiles. A general measure of area cost for each region is found by normalising the area for CLB tiles, DSP tiles and BRAM tiles. The normalisation factor used is 1:18:4 for CLB, DSP and BRAM tiles respectively, based on the number of tiles in the device, corresponding to W_i in the formulation. One CLB tile comprises 36 configuration frames, a DSP tile 28 frames and BRAM tile 30 frames, which corresponds to W_{fi} in the formulation.

[1]Note "rows" here refers to a Xilinx term used in their architecture descriptions.

B. Application Results

Here, we apply our region allocation approach to an example design implemented on a Virtex-5 FX70T FPGA. The design has one static region and five reconfigurable modules. The design is a wireless video receiver chain with blocks used from existing designs and vendor IP. The system can operate in various modes, and adapts to channel conditions and user requirements at runtime. The resource utilisation for each reconfigurable module and mode is as shown in Table II.

TABLE II
Resource utilisation for reconfigurable modules.

Module	Mode	Slices	BR	DSP
Matched Filt (F)	1. Filter1	818	0	28
	2. Filter2	500	0	34
Recovery (R)	1. Fine	318	1	13
	2. Coarse1	195	1	5
	3. Coarse2	123	0	8
	4. None	0	0	0
Demodulator (M)	1. BPSK	50	0	2
	2. QPSK	97	0	4
Decoder (D)	1. Viterbi	630	2	0
	2. Turbo	748	15	4
	3. DPC	234	2	0
Decoder (V)	1. MPEG4	4700	40	65
	2. MPEG2	4558	16	32
	3. JPEG	2780	6	9

A sample of configurations used is shown here:
$S \rightarrow F_1 \rightarrow R_3 \rightarrow M_1 \rightarrow D_1 \rightarrow V_1$
$S \rightarrow F_1 \rightarrow R_3 \rightarrow M_1 \rightarrow D_1 \rightarrow V_2$
$S \rightarrow F_1 \rightarrow R_3 \rightarrow M_1 \rightarrow D_1 \rightarrow V_3$
...

This system description was formulated as in the previous section, and the solver used to find the optimal solution. The transition between configurations is modelled in a random fashion with each configuration transiting to at least 5 other configurations.

A plot of average reconfiguration time against total resource requirements is shown in Fig. 5. Reconfiguration speed is taken as 234 MB/s [14]. Several of the solutions are infeasible due to a lack of resources in the chosen device. Using a one-region-per-module scheme, this design will not fit into the XC5VFX70T device, since that scheme requires 18 DSP tiles

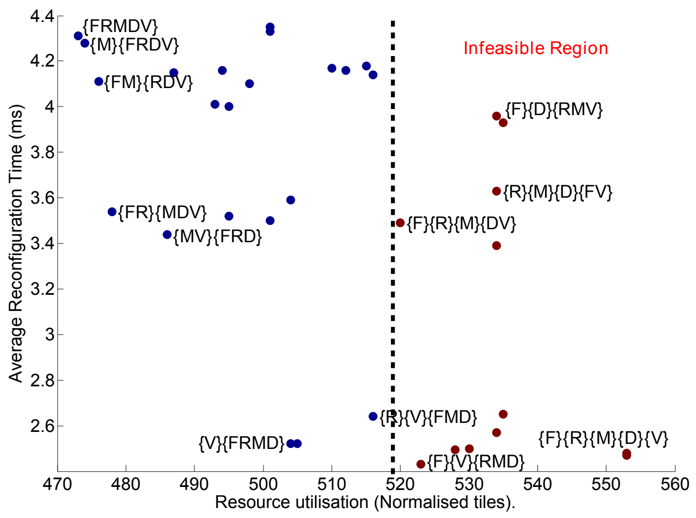

Fig. 5. Resource requirement and configuration time.

and the device has only 16. Using a larger FPGA would increase system cost significantly. The partition scheme in which all the modules are implemented together (labelled {FRMDV}) gives the lowest resource utilisation of 473 tiles, but the average reconfiguration time for this scheme is considerably higher at 4.32 ms. Using the proposed partitioning method, we can find schemes that fit the design into the FX70T device with lower reconfiguration time. This is one of the attractions of using this analytical method. There are six Pareto optimal points in the feasible region of the plot. The configuration in which the decoder (V) is implemented in a single region and all other modules are implemented together in another region, labelled {V},{FRMD} in the plot, gives the lowest average reconfiguration time. This scheme uses 504 normalised tiles and has a average reconfiguration time of 2.52 ms. The scheme in which the filter (F) and recovery (R) modules are combined in a single region and other modules are implemented together (labelled {FR},{MDV} in the plot) lies closest to the origin and hence is the optimal partitioning. This scheme uses 478 normalised tiles and the average reconfiguration time is 3.54 ms and hence gives 13.5% area improvement compared to one-module-per-region partitioning. Worst case reconfiguration time for the optimal scheme is 4.36 ms, that for one-region-per-module partitioning is 4.69 ms. The bitstream storage requirement for these partitions was also calculated. The optimal solution {FR},{MDV} requires 53 Mbits storage while implementing all modules in a single region requires 81 Mbits to store bitstreams. These results depend significantly on the configurations defined by the application. The upper bound area consumption will be that of using separate regions for each module and the lower bound is a single-region scheme.

VI. Conclusion and Future Work

Determining the number of partial reconfiguration regions and the allocation of reconfigurable modules to regions is not always trivial, but this choice can impact FPGA resource utilisation, reconfiguration time and the storage requirement for configuration bitstreams. In this paper we showed how to determine the tradeoff between the number of reconfigurable regions and reconfiguration overhead. A new technique for determining the optimal number of regions and the assignment of reconfigurable modules into these regions has been introduced, which can be incorporated into the existing vendor-supported partial reconfiguration tool flow. It is demonstrated that efficient reconfiguration schemes improve resource consumption and reduce reconfiguration time. The method presented here will help designers with less FPGA architecture knowledge to generating efficient PR implementations, allowing designs to fit within limited device constraints.

We are currently working on an efficient floorplanner to manage the placement of the optimal regions. The floorplanner, solver along with vendor supplied synthesis and place and route tools will bring us one step close to achieving the ultimate goal of enabling fully automated partial reconfiguration design for adaptive applications.

References

[1] *UG702: Partial Reconfiguration User Guide*, Xilinx Inc., 2010.

[2] C. Bieser, M. Bahlinger, M. Heinz, C. Stops, and K. D. Mueller-Glaser, "A novel partial bitstream merging methodology accelerating Xilinx Virtex-II FPGA based PR system setup," in *Proceedings of Int. Conf. on Field Programmable Logic and Applications (FPL)*, 2006.

[3] P. Sedcole, B. Blodget, T. Becker, J. Anderson, and P. Lysaght, "Modular dynamic reconfiguration in Virtex FPGAs," *IEE Proceedings of Computers and Digital Techniques*, vol. 153, no. 3, pp. 157–164, 2006.

[4] C. Conger, R. Hymel, M. Rewak, A. D. George, and H. Lam, "FPGA design framework for dynamic partial reconfiguration," in *Proceedings of Reconfigurable Architectures Workshop (RAW)*, 2008.

[5] L. Singhal and E. Bozorgzadeh, "Multi-layer floorplanning on a sequence of reconfigurable designs," in *Proceedings of Int. Conf. on Field Programmable Logic and Applications (FPL)*, 2006.

[6] P. Banerjee, M.Sangtani, and S.Sur-Kolay, "Floorplanning for partial reconfiguration in FPGAs," in *Proceedings of International Conference on VLSI Design*, 2009.

[7] V. Rana, S. Murali, D. Atienza, M. D. Santambrogio, L. Benini, and D. Sciuto, "Minimization of the reconfiguration latency for the mapping of applications on FPGA-based systems," in *IEEE/ACM Int. Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2009.

[8] S. Fahmy, J. Lotze, J. Noguera, L. Doyle, and R. Esser, "Generic software framework for adaptive applications on FPGAs," in *Proceedings of IEEE Symp. on Field-Programmable Custom Computing Machines (FCCM)*, 2009.

[9] E. El-Araby, I. Gonzalez, and T. El-Ghazawi, "Performance bounds of partial run-time reconfiguration in high-performance reconfigurable computing," in *Proceedings of Int. Works. on High-performance Reconfigurable Computing Technology and Applications (HPRCTA)*, 2007.

[10] S. Liu, R. Pittman, A. Forin, and J. Gaudiot, "On energy efficiency of reconfigurable systems with run-time partial reconfiguration," in *Proceedings of IEEE Int. Conf. on Application-specific Systems Architectures and Processors (ASAP)*, 2010.

[11] D. Koch, C. Beckhoff, and J. Torrison, "Fine-grained partial runtime reconfiguration on Virtex-5 FPGAs," in *Proceedings of IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2010.

[12] P. Schumacher and P. Jha, "Fast and accurate resource estimation of RTL-based designs targeting FPGAs," in *Proceedings of Int. Conf. on Field Programmable Logic and Applications (FPL)*, 2008.

[13] *UG191: Virtex-5 FPGA Configuration User Guide*, Xilinx Inc., 2010.

[14] M. Liu, W. Kuehn, Z. Lu, and A. Jantsch, "Run-time partial reconfiguration speed investigation and architectural design space exploration," in *Proceedings of Int. Conf. on Field Programmable Logic and Applications (FPL)*, 2009.