Design Optimizations to Improve Placeability of Partial Reconfiguration Modules

Markus Koester and Wayne Luk
Department of Computing,
Imperial College London, UK
Email: {mkoester, wl}@doc.ic.ac.uk

Jens Hagemeyer and Mario Porrmann
System and Circuit Technology,
University of Paderborn, Germany
Email: {jenze, porrmann}@hni.uni-paderborn.de

***Abstract*—In partially reconfigurable architectures, system components can be dynamically loaded and unloaded allowing resources to be shared over time. This paper focuses on the relation between the design options of partial reconfiguration modules and their placement at run-time. For a set of dynamic system components, we propose a design method that optimizes the feasible positions of the resulting partial reconfiguration modules to minimize position overlaps. We introduce the concept of subregions, which guarantees the parallel execution of a certain number of partial reconfiguration modules for tiled reconfigurable systems. Experimental results, which are based on a Xilinx Virtex-4 implementation, show that at run-time the average number of available positions can be increased up to 6.4 times and the number of placement violations can be reduced up to 60.6%.**

I. Introduction

The reconfigurability of FPGAs allows the implementation of adaptable and customizable hardware systems. Partially reconfigurable FPGAs allow a part of the system to be changed or adapted without the necessity of reconfiguring the whole chip. Thus, the system component, which requires to be reconfigured, can be replaced by a new one, while the remaining components continue to operate without interruption. In such systems, dynamic system components are represented by partial reconfiguration modules (PR modules). The ability of dynamically loading and unloading PR modules allows FPGA resources to be shared over time. In this paper we focus on a system with a partially reconfigurable region (PR region) that consists of an array of reconfigurable tiles. A reconfigurable tile is the atomic unit for partial reconfiguration.

The placement of a PR module is achieved by allocating a reasonable amount of contiguous reconfigurable tiles. In the context of reconfigurable computing various methods have been proposed on how to place a PR module at run-time. In [1] Bazarghan et al. describe the problem of placing a PR module as an online bin-packing problem. Steiger et al. [2] discuss the problem of online placement and scheduling of hard real-time tasks for partially reconfigurable devices. Handa et al. [3] introduce a placement algorithm aiming at reducing the degree of fragmentation. In [4] a placement approach is introduced, that considers the routing costs to existing instances of PR modules. Lu et al. [5] introduce an algorithm for PR module placement, which pre-partitions the area of the PR region into blocks of different sizes. On the placement of a PR module the blocks are split and merged to maintain contiguous free resources.

All previously described methods assume a homogeneous PR region and neglect the fact, that FPGAs are heterogeneous architectures. A partially reconfigurable region typically comprises different types of resources, such as logic blocks, memory blocks, and DSP blocks. The heterogeneity of the resources significantly limits the placement of the PR modules within the PR region. Placement approaches such as bin-packing cannot be used unless the packing rule is adapted to handle placement constraints caused by the heterogeneity. As described in [6], [7] the configuration data (partial bitstream) of a PR module can be placed at any position with the same arrangement as the types of tiles from which it is built. Thus, the placement of a PR module is restricted to a set of feasible positions, which depends on the synthesis region the PR module is generated from. In [8] a placement algorithm is described, which considers the heterogeneity of the PR region. The authors show that each PR module can have a different set of feasible positions, causing a different resource utilization of the reconfigurable tiles in the PR region.

In this paper we focus on design-time issues for heterogeneous partially reconfigurable regions. The synthesis region of a PR module is the area in the device, that is used to generate the configuration data (partial bitstream) from. It defines the set of feasible positions for the PR module and affects the placeability of the PR module at run-time. Hence, the synthesis regions of the PR modules can be optimized in such a way as to minimize the degree of overlap of the corresponding feasible positions. We introduce a concept to evaluate the degree of overlap at design-time by using an overlap graph. Furthermore, we introduce the partitioning of the reconfigurable tiles of the PR region into subregions, which can be used to guarantee that a certain number of PR modules can be placed in parallel. The proposed concepts are demonstrated in an example implementation using a Xilinx Virtex-4 FPGA.

The rest of the paper is organized as follows: Section II summarizes the concept of tiled PR regions. Section III describes the design-time evaluation of the degree of overlap by using the overlap graph and the optimization of the placeability of PR modules. Section IV provides an example of how to apply the proposed optimization method to a set of dynamic system

978-3-9810801-5-5/DATE09 © 2009 EDAA

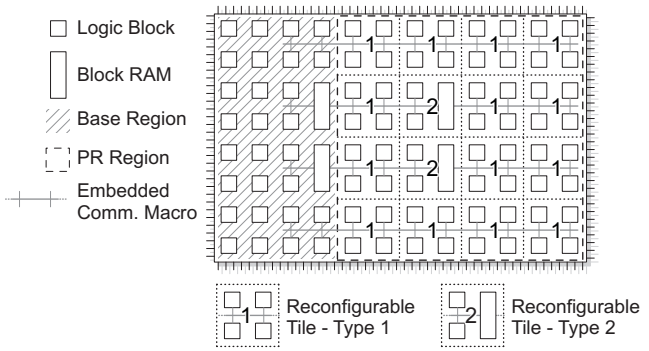


Fig. 1. Example of a partitioning scheme using a PR region with reconfigurable tiles.

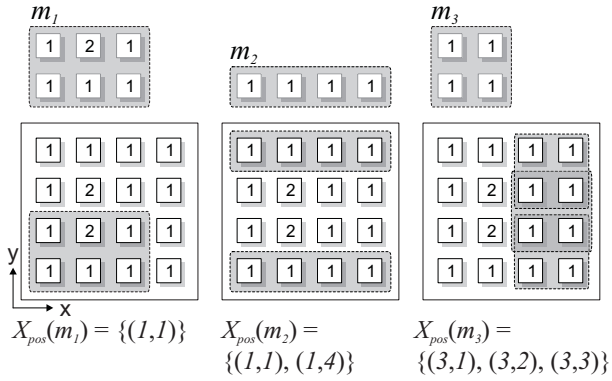


Fig. 2. Example of a set of PR modules and their feasible positions.

components on a Xilinx Virtex-4 FPGA. Experimental results show the benefit of the proposed design-time optimizations. Section V summarizes the paper.

II. Tiled PR Regions

In [9] a module-based design flow for partial reconfiguration is introduced, which is supported by the Xilinx design tools. The design flow enables partial reconfiguration at the granularity level of a PR region. This type of PR Region can be classified as a *single-module PR region*, where the number of PR regions describes the upper bound of dynamic system components that can execute in parallel. The drawbacks of this approach are that the size of a PR module is limited by the size of the PR region, and that small PR modules occupy the resources of the whole PR region.

An alternative to the single-module PR Region is the *tiled PR region* described by Hagemeyer et al. [10]. In this approach the partially reconfigurable region is subdivided into reconfigurable tiles. Such tiled partitioning allows placement of multiple PR modules in a PR region. A reconfigurable tile can be considered as an atomic unit of partial reconfiguration. A PR region may contain several different types of tiles offering different amounts of available resources. The tiles of the same type are built identically using the same number and arrangement of resources. All static components of the system are located in the so called base region. The communication between the PR modules and the static system components is realized by so called embedded communication macros [10]. In each tile a certain amount of resources are reserved for the embedded communication macro. Every tile is connected to its neighboring tile using the same type of resources. Figure 1 shows an example with a base region and a PR region, which is partitioned into an area of 4×4 reconfigurable tiles. The PR region in the example is heterogeneous, since two different types of tiles are used. At run-time an instance of a PR module is mapped to one or several contiguously aligned tiles. This is done by partially reconfiguring the selected tiles using the equivalent configuration data (partial bitstream) of the PR module.

By manipulating the configuration data as described in [6], [7], a PR module can be placed at any position with the same arrangement as the types of tiles from which it is built. If M is the set of all PR modules in the system, then all placement options of a PR module $m \in M$ can be described by the set of feasible positions $X_{pos}(m) = \{(x_1, y_1), (x_2, y_2), \ldots\}$. Without loss of generality, $(x_i, y_i) \in X_{pos}(m)$ describes the position of the tile at the lower left corner of the PR module m with respect to the lower left corner of the PR region. A PR module can take up any size from a single tile to all tiles of the PR region. Figure 2 shows the PR region of Figure 1 and an example of a set of PR modules with the corresponding feasible positions. The values in each tile indicate the type of tile.

In any reconfigurable system, the number and the size of the concurrently executable PR modules are restricted by the number of available resources of the reconfigurable unit. In this context we introduce the term *allocation width*, which can be described as follows. A PR system has an allocation width of n_{alloc}, if for any possible configuration of $(n_{alloc}-1)$ instances of PR modules, it is still possible to place one instance of any PR module. In a tiled PR region a fixed allocation width cannot be guaranteed in any case, since the placement is subject to the current configuration of PR modules and the degree of fragmentation. However, an application can require a reconfigurable unit with a certain allocation width, since the execution of essential dynamic system components should not be delayed or reject.

Applications that do not demand a particular allocation width should be designed to be able to handle *placement violations*. A placement violation is the situation when the application is requesting the placement of a PR module, but in the current configuration there are not enough free contiguous resources available to place an instance of the requested PR module. There are three basic policies to handle placement violations. The first option rejects the placement and the application is forced to execute an alternative system component. This can be a software or hardware component using different types of resources or a different algorithm implementation. The second option delays the placement until the execution of one of the existing instances of PR modules is finished, and a suitable number of resources are released. The third option rearranges the existing instances of PR modules

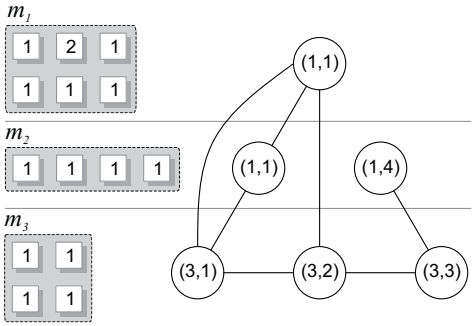
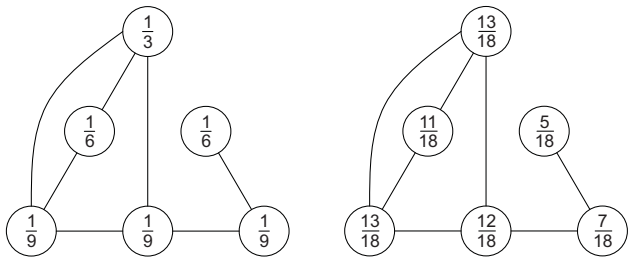


Fig. 3. Example of an overlap graph



(a) Probability weights $w_p(v)$ (b) Position weights $w_{pos}(v)$

Fig. 4. Probability and the position weights.

aiming to reduce the fragmentation to be able to place the requested PR module. In [11], [12] suitable defragmentation methods are introduced. However, such defragmentation brings overhead such as lengthening the reconfiguration time to relocate existing instances of PR modules.

III. Placeability of PR Modules

In this section we describe a scheme to evaluate the degree of position overlap and the corresponding placeability of PR modules at design-time. With respect to run-time placement, the PR modules vary according to their resource requirements, their shape, and their feasible positions. The feasible positions of the PR modules are known at design-time and depend on the selected synthesis region. If a PR module is placed at one of its feasible positions, it affects the availability of the feasible positions of the other PR modules. Those feasible positions become unavailable that utilize at least one tile of the selected position. The overlap graph $G = (V, E)$ is an undirected graph that enables visualizing these resource dependencies. It shows which of the feasible positions of the PR modules overlap with each other. The graph can be used with arbitrarily shaped PR modules. For simplicity we will focus on rectangular PR modules in the following. A vertex $v = (m, x, y) \in V$ represents a feasible position $(x, y) \in X_{pos}(m)$ of the PR module $m \in M$. The set of all vertices is defined as

$$V = \bigcup_{m \in M} \{(m, x, y) \mid (x, y) \in X_{pos}(m)\}. \tag{1}$$

Hence, the number of vertices is the same as the sum of feasible positions of all PR modules. For a vertex $v_1 = (m_1, x_1, y_1) \in V$ and a vertex $v_2 = (m_2, x_2, y_2) \in V$ an edge (v_1, v_2) is created, if $v_1 \neq v_2$ and the area of PR module m_1 at position (x_1, y_1) overlaps with the area of PR module m_2 at position (x_2, y_2). Figure 3 shows the overlap graph for the PR modules of the example in Figure 2. The overlap graph can be further adapted to take into account the temporal relations between PR modules if they are known at design-time. In this case the edges between PR modules, which are never used at the same time, can be removed.

With the overlap graph we can evaluate the degree of overlap for each feasible position of the PR modules. For this purpose we introduce the *position weight*. Using the overlap graph the computation of the position weights is done in two steps. First the probability weights

$$w_p(v) = p_{alloc}(m)/|X_{pos}(m)| \tag{2}$$

are computed for each vertex $v = (m, x, y) \in V$, where $p_{alloc}(m)$ denotes the probability of an allocation of the PR module m. The probability weight $w_p(v)$ indicates the probability of a feasible position to be chosen, if all tiles in the PR region are available and a random placement is applied. Figure 4(a) shows the corresponding probability weights for the PR modules shown in Figure 2 with a fixed $p_{alloc}(m) = 1/|M|$. In the example of the 3 PR modules the allocation probability is $\forall m \in M : p_{alloc}(m) = 1/3$ and PR module m_2 has 2 feasible positions, such that the probability weight for its feasible positions is $w_p(v) = 1/(3 * 2) = 1/6$.

The position weight $w_{pos}(v)$ of a feasible position is computed by summing up the probability weights of the adjacent vertices. The set of adjacent vertices V_{adj} is defined as

$$V_{adj}(v) = \{v_{adj} \mid (v, v_{adj}) \in E\}, \tag{3}$$

and the resulting position weight is calculated by

$$w_{pos}(v) = w_p(v) + \sum_{v_{adj} \in V_{adj}(v)} w_p(v_{adj}). \tag{4}$$

The position weights for the PR modules of Figure 2 are shown in Figure 4(b). The position weights reflect the degree of overlap. E.g., the placement of an instance of m_2 at position $(1, 4)$ only blocks the position $(3, 3)$ of m_3, while the placement of an instance of m_2 at position $(1, 1)$ blocks the positions $(1, 1)$ of m_1 and $(3, 1)$ of m_3. Therefore, the position weight $5/18$ of position $(1, 4)$ of m_2 is lower than the one from position $(1, 1)$.

A metric to evaluate the degree of overlap of all feasible positions is to generate a sort of average value of the position weights of all feasible positions. As the probability weight $w_p(v)$ reflects the probability of a feasible position to be selected at a random placement, the overlap weight of all PR modules is defined as follows:

$$w_{ovr}(V) = \frac{1}{|V|} \sum_{v \in V} w_{pos}(v) \cdot w_p(v) \tag{5}$$

The average value of the position weights is divided by the total number of feasible positions $|V|$ to balance the degree

of overlap and the number of feasible positions. The synthesis regions of the given PR modules can be selected in such a way as to minimize $w_{ovr}(V)$. A small $w_{ovr}(V)$ indicates that the overlaps of feasible positions of the PR modules are small. Minimizing the overlap weight aims at maximizing the number of available positions after placement of a PR module at run-time.

A PR module of a dynamic system component can be synthesized in various options. Hence, a dynamic system component can be represented by different PR module variants. Each PR module variant can have a different shape or location of the synthesis region resulting in a different set of feasible positions. But with the increasing number of PR module variants the amount of memory for storing the corresponding configuration data increases as well. Therefore, it is necessary to limit the number of PR module variants or even consider only one PR module for each dynamic system component. An objective at design-time is to select a suitable number of PR module variants for each dynamic system component aiming to optimize the placeability at run-time. The selected PR modules can be evaluated by computing the overlap weight. If a PR module of a new dynamic system component requires to be added to an existing system, the degree of overlap with the existing PR modules can be minimized by selecting the PR module variant causing the minimum overlap weight. If a small number of PR modules (e.g. $\mid M \mid \leq 10$) is required to be added to the system, the set of PR modules with a minimum overlap weight can be determined by exhaustive search. For a large number of new dynamic system components heuristic approaches are required, since the search space needs to cover all combinations of PR module variants.

For those applications that are not tolerant of placement violations, the system is required to guarantee that a certain number of PR modules can be executed in parallel at all time. Therefore, the selection of PR module variants can be performed to obtain a certain allocation width. In the context of tiled PR regions, the PR region can be partitioned into disjoint subregions. If $r(m) = (\#Slices, \#BRAMs, ...)$ is the resource requirement for the PR module $m \in M$ and r_{sub} is the number of available resources in a subregion, then r_{sub} has to satisfy

$$\forall m \in M : r(m) \leq r_{sub}. \tag{6}$$

The resources occupied by an instance of any PR module are located completely within a subregion, such that at any time as many PR modules can be executed in parallel as given by the number of disjoint subregions. Therefore, the number of subregions corresponds to the allocation width.

IV. Example Implementation

In order to implement a tiled PR region using Xilinx Virtex-4 FPGAs, a suitable region has to be selected. In the Virtex-4 architecture a column of a clock region, which is referred to as a frame, is the smallest partially reconfigurable unit. Hence, the area of a PR region should be multiples of a frame, and the smallest possible height for a reconfigurable tile is the height of a frame. In the example implementation we vertically divide the FPGA, such that the resources located left of the center column are dedicated to static system components (base region), and the resources located right of the center column are considered for the tiled PR region. Figure 5 shows an example of the partitioning for a Virtex-4 FX100 FPGA.

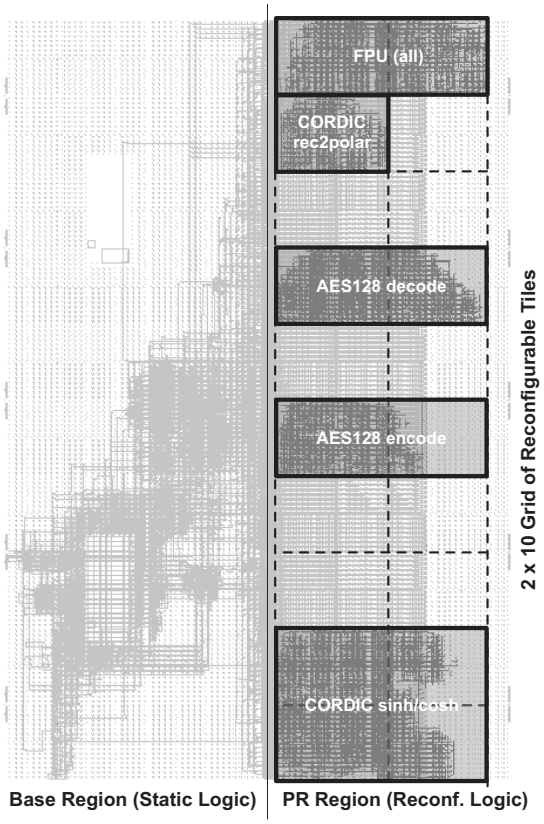


Fig. 5. Partitioning of the Xilinx Virtex-4 FX100 FPGA using (2×10) tiles.

We consider three different partitionings for the tiled PR region with an area of (1×10), (2×10), and (3×10) tiles. The tile sizes are chosen with respect to the ratio between the resources needed for the communication infrastructure and the resources available for the partially reconfigurable logic. Inside a tile, the number of resources for interconnecting the tiles and the base region does not depend on the tile size but on the specification of the communication infrastructure. To maintain a certain degree of available resources for partially reconfigurable logic within each tile, the tile size should therefore not be too small. The number of available resources of the different tiles are shown in Table I. The interconnection of the tiles is realized by an embedded communication macro supporting a shared bus for 32-bit data width, and 14-bit address width. The bus structure features 16-bit dedicated control signals with 2-bit dedicated signals, which are implemented by using the techniques described in [10].

Since the various resource types (Slices, BRAMs, DSPs) of Virtex-4 FPGAs are arranged in columns, the reconfigurable fabric can be considered as homogeneous in the vertical

TABLE I
AVAILABLE RESOURCES OF THE TILES.

PR Region	Tile Type	Slices	BRAMs	DSPs
(1x10)	center	1536	16	4
(2x10)	left	768	8	4
	right	768	8	-
(3x10)	left	512	4	4
	center	512	8	-
	right	512	4	-

TABLE II
EXAMPLE SET OF DYNAMIC SYSTEM COMPONENTS.

Application	Component	Slices	BRAMs	DSPs
AES128	decryption	1382	13	-
	encryption	1009	9	-
CORDIC	arctan	1985	-	-
	rec2polar	728	-	-
	polar2rec	501	-	-
	sinh/cosh	2162	-	-
FPU	all	1435	-	12
	add/sub	557	-	-
	divider	922	-	-
	mulitplier	338	-	8

direction. This homogeneity can be used to define subregions based on the following principle. Starting from the bottom tile row of the PR region, the area of the subregion is gradually expanded until it contains enough available resources so that each single PR module can be placed. The process repeats with the next subregion, which is located on top of the preceding subregion, until the top row of the tiles in the PR region is reached.

The test system comprises the embedded PowerPC of the Virtex-4 FX100 FPGA and the dynamic system components listed in Table II. As a benchmark we have generated different random load and unload sequences $B_2, B_3, \ldots, B_6$ for the selected dynamic system components. For the benchmark B_n the value of n reflects the number of instances of PR modules that are executed in parallel. Each benchmark removes the earliest placed instance of PR module and places a new randomly chosen PR module. In the case of a placement violation, the placement is repeated with a new randomly selected PR module until it succeeds. Each sequence contains 10000 placement requests. The placement is done by selecting the free feasible position with the least position weight w_{pos}.

In the following example we only select the FPU components and generate one PR module for each component using the PR regions (1×10), (2×10), and (3×10) without subregions. For each PR region we generate two sets of PR modules. One set contains the PR modules with feasible positions optimized with respect to a minimal overlap weight (min. w_{ovr}). The set is derived by computing the overlap weight for every possible combination of synthesis regions for the PR modules and selecting the combination with the least overlap weight. The second set of PR modules contains

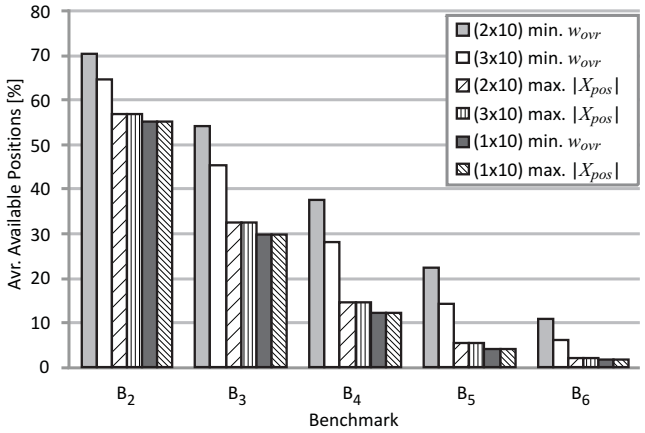


Fig. 6. Percentage of available positions of the FPU PR modules.

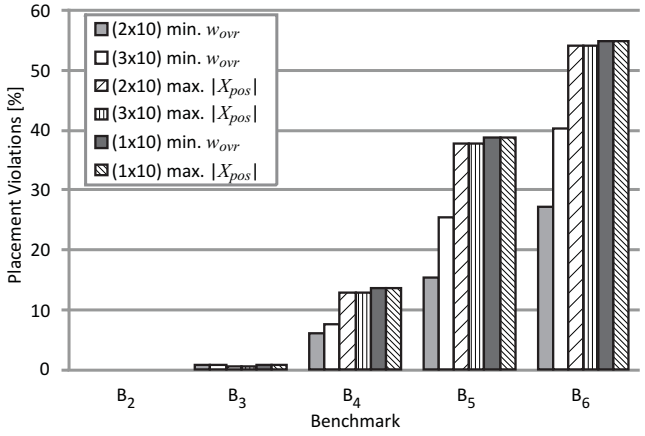


Fig. 7. Percentage of placement violations using the FPU components.

the ones optimized with respect to the maximum number of feasible positions (max. $|X_{pos}|$). Figure 6 shows the average percentage of available positions of the PR modules for each benchmark. A larger number of available positions indicates a lower degree of fragmentation and a better placeability of additional instances of PR modules. As the number of concurrently placed instances of PR modules increases, the number of available positions decreases. When comparing the different sets of PR modules, we find that the *overlap weight optimized PR modules* using the (2×10) PR region offer the most available positions, followed by the overlap weight optimized PR modules using the (3×10) PR region. The average number of available positions of the overlap weight optimized PR modules is up to 6.4 times larger (benchmark B_5) than one from the PR modules optimized with respect to the maximum number of feasible positions.

Figure 7 confirms the result, as the percentage of placement violations of the set of overlap weight optimized PR modules is significantly lower than the percentage of placement violations of the other sets of PR modules. A lower number of placement violations suggests a large degree of resource utilization of the PR region. By using the overlap weight optimized PR modules the placement violations can

TABLE III
PERCENTAGE OF PLACEMENT VIOLATIONS USING ALL COMPONENTS.

PR Region		Benchmark				
		B_2	B_3	B_4	B_5	B_6
(2x10)	tiled	0.0	0.1	3.7	22.2	41.2
	tiled subregion	0.0	0.0	7.3	22.4	39.8
(3x10)	tiled	0.0	0.1	3.2	22.9	43.7
	tiled subregion	0.0	0.0	8.3	23.6	41.8
single-module		0.0	0.0	100.0	100.0	100.0

be reduced by up to 60.6% (benchmark B_5). For the chosen FPU components, the (2×10) PR region has the best run-time behavior, although the (3×10) PR region offers a larger degree of placement options.

When only the CORDIC components are used, the set of overlap weight optimized PR modules is identical to the set of PR modules optimized for the maximum number of feasible positions. This is due to the fact that all tiles offer the same amount of slices, and the CORDIC components use slices only. So the overlap weight optimization only has an impact on placement if the dynamic system components use different types of resources, like in the case of the FPU components.

Table III shows the percentage of placement violations using all dynamic system components. It focuses on the impact of using subregions in the PR region. By using subregions an allocation width of $n_{alloc} = 3$ can be achieved, while the tiled PR regions (2×10) and (3×10) without subregions cause placement violations in the benchmark B_3 and therefore only have an allocation width of $n_{alloc} = 2$. For single-module PR regions the benchmarks B_4, B_5, and B_6 cannot accommodate the requested PR modules and they are therefore not suitable. Using tiled PR regions, more than 90% of the requested PR modules of the benchmark B_4 can be placed. This shows the increased placeability of PR modules of tiled PR regions over single-module PR regions especially for applications that are able to handle placement violations.

V. CONCLUSION

When using tiled PR regions in heterogeneous reconfigurable architectures, the placement algorithm should be able to deal with the limited feasible positions of a PR module. The placement quality does not only depend on the placement algorithm, but on design-time aspects such as the chosen synthesis regions of the PR modules and their corresponding feasible positions. In this paper we propose a design method for selecting a synthesis region for PR modules aiming to optimize placement at run-time. The key idea is to compute a position weight for each feasible position of every PR module and to generate an overlap weight, which quantifies the degree of position overlaps. The overlap graph is a data structure to derive the position weights. As shown by experiments, the overlap optimization can significantly improve the placement of PR modules. In addition to the overlap optimization, the paper introduces the concept of subregions in a tiled PR region. The use of subregions allows a fixed allocation width and is therefore particularly suitable for applications that are not able to handle placement violations.

Currently, the minimization of the overlap weight is done by exhaustive search, which only allows a limited number of PR modules to be added to the system. In future work, a heuristic search algorithm will be developed that is able to find a near optimum overlap weight for a large number of PR modules. Furthermore, the proposed approach will be refined based on further applications such as software defined radio [7], and extended to cover a variety of tiled partially reconfigurable devices.

ACKNOWLEDGMENT

This work was supported by the German Research Council (DFG) and the UK Engineering and Physical Sciences Research Council (EPSRC).

REFERENCES

[1] K. Bazargan, R. Kastner, and M. Sarrafzadeh, "Fast template placement for reconfigurable computing systems," *IEEE Design and Test of Computers*, vol. Vol. 17, No. 1, pp. 68–83, 2000.

[2] C. Steiger, H. Walder, and M. Platzner, "Operating systems for reconfigurable embedded platforms: Online scheduling of real-time tasks," *IEEE Transactions on Computers*, vol. 53, no. 11, pp. 1393–1407, 2004.

[3] M. Handa and R. Vemuri, "Area fragmentation in reconfigurable operating systems," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms*. CSREA Press, 2004, pp. 77–83.

[4] A. Ahmadinia, C. Bobda, M. Bednara, and J. Teich, "A new approach for on-line placement on reconfigurable devices," in *18th International Parallel and Distributed Processing Symposium (IPDPS 2004)*. IEEE Computer Society, 2004.

[5] Y. Lu, T. Marconi, G. N. Gaydadjiev, K. Bertels, and R. J. Meeuws, "A self-adaptive on-line task placement algorithm for partially reconfigurable systems," in *Proceedings of the 22nd Annual International Parallel and Distributed Processing Symposium (IPDPS 2008) - RAW2008*, April 2008, pp. 1–8.

[6] H. Kalte and M. Porrmann, "REPLICA2Pro: Task relocation by bitstream manipulation in Virtex-II/Pro FPGAs," in *Proceedings of the ACM International Conference on Computing Frontiers*, 2006.

[7] T. Becker, W. Luk, and P. Cheung, "Enhancing relocatability of partial bitstreams for run-time reconfiguration," in *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE Computer Society Press, 2007, pp. 35–44.

[8] M. Koester, H. Kalte, and M. Porrmann, "Task placement for heterogeneous reconfigurable architectures," in *Proceedings of the IEEE 2005 Conference on Field-Programmable Technology (FPT'05)*. IEEE Computer Society, 2005, pp. 43–50.

[9] P. Lysaght, B. Blodget, J. Mason, B. Bridgford, and J. Young, "Enhanced architectures, design methodologies and CAD tools for dynamic reconfiguration of Xilinx FPGAs," in *16th International Conference on Field Programmable Logic and Applications*, 2006, pp. 12–17.

[10] J. Hagemeyer, B. Kettelhoit, M. Koester, and M. Porrmann, "Design of homogeneous communication infrastructures for partially reconfigurable FPGAs," in *Proc. of the Int. Conf. on Engineering of Reconfigurable Systems and Algorithms (ERSA '07)*. CSREA Press, 2007.

[11] M. Koester, H. Kalte, M. Porrmann, and U. Rückert, "Defragmentation algorithms for partially reconfigurable hardware," *IFIP International Federation for Information Processing Series*, vol. 240, pp. 41–53, 2007.

[12] J. Angermeier, S. Fekete, T. Kamphans, D. Koch, N. Schweer, J. van der Veen, and J. Teich, "No-break dynamic defragmentation of reconfigurable devices," in *Proceedings of International Conference on Field-Programmable Logic and Applications (FPL 08)*, Heidelberg, Germany, Sep. 2008.