Design Optimizations for Tiled Partially Reconfigurable Systems

Markus Koester, Wayne Luk, *Fellow, IEEE*, Jens Hagemeyer, Mario Porrmann, *Member, IEEE*, and Ulrich Rückert, *Member, IEEE*

***Abstract*—In partially reconfigurable architectures, system components can be dynamically loaded and unloaded allowing resources to be shared over time. Dynamic system components are represented by partial reconfiguration (PR) modules. In comparison to a static system, the design of a partially reconfigurable system requires additional design steps, such as partitioning the device resources into static and dynamic regions. We present the concept of tiled PR regions, which enables a flexible online-placement of PR modules. Dynamic reconfiguration requires a suitable communication infrastructure to interconnect the static and dynamic system components. We present an embedded communication macro, a communication infrastructure that interconnects PR modules in a tiled PR region. Efficient online-placement of PR modules depends not only on the placement algorithm, but also on design-time aspects such as the chosen synthesis regions of the PR modules. We propose a design method for selecting suitable synthesis regions for the PR modules aiming to optimize their placement at run-time.**

***Index Terms*—Communication macro, design automation, field-programmable gate arrays (FPGAs), overlap graph, reconfigurable architectures.**

I. Introduction

THE reconfigurability of field-programmable gate arrays (FPGAs) is a promising approach to enhance the flexibility of a given architecture resulting in adaptable and customizable hardware systems. The partial reconfigurability of the FPGA enables part of the device to be changed or adapted without reconfiguring the whole chip. Thus, the system component, which requires being reconfigured, can be replaced by a new one, while the remaining components continue to operate without interruption.

A reconfigurable system typically comprises an area for static system components (base region) and one or more partially reconfigurable regions (PR regions) for dynamic system components. The dynamic system components are represented by partial reconfiguration modules (PR modules). The placement of a PR module is done by configuring a predefined area in a PR region of the FPGA with the corresponding configuration data. A PR module must be able to communicate to the rest of the system from any of its feasible positions. In order to avoid restricting the number of feasible positions of the PR modules, it is necessary to design a homogeneous communication infrastructure, which does not introduce any further degree of heterogeneity. The amount of additional resources, which are required to achieve the required homogeneity, should be low to provide a large number of free resources within the PR region.

Most realizations of dynamically reconfigurable systems use simple approaches that are based on fixed module slots. The placement flexibility of these implementations is different from the flexibility assumed and analyzed in the theoretical research work. In this paper we try to help close this gap by showing how today's heterogeneous FPGAs can be used for dynamic reconfiguration with free module placement, varying module sizes, and multiple instances of modules.

An important aspect in the design of a partially reconfigurable system is the implementation of the communication infrastructure. In [1], a tool flow for generating suitable homogeneous communication infrastructures is introduced, which supports Xilinx Virtex-II FPGAs in a one-dimensional partially reconfigurable system. In this paper, we present refinements of this concept with the following focus.

- The communication infrastructure introduced in [1] is enhanced to support 2-D partial reconfiguration. The presented *embedded communication macro* is optimized for Virtex-4, Virtex-5 and Virtex-6 FPGAs.
- The embedded communication macro is evaluated and compared to alternative communication macros.

The proposed communication infrastructure enables the placement of PR modules in a tiled partially reconfigurable system and avoids restricting the number of feasible positions of the PR modules. As suggested in [2], the placeability of a PR module can be further enhanced at design-time by optimizing the area the PR module is initially synthesized for. This area is referred to as *synthesis region*. In this paper we take up the idea of improving the placeability and extend it to cover the following aspects.

- A method to derive the set of minimal synthesis regions is introduced.
- The concept of *subregions* is analyzed and compared to single-module PR regions and tiled PR regions.
- The design-time optimization presented in [2] uses exhaustive search and is therefore only suitable for small problem sizes. For a larger set of PR modules, a heuristic approach is presented, which is based on a genetic algorithm.

Manuscript received June 18, 2009; revised September 15, 2009 and February 26, 2010. This work was supported by the German Research Council (DFG) and the U.K. Engineering and Physical Sciences Research Council (EPSRC).

M. Koester and W. Luk are with the Department of Computing, Imperial College London, London SW7 2AZ, U.K. (e-mail: mkoester@doc.ic.ac.uk; wl@doc.ic.ac.uk).

J. Hagemeyer and M. Porrmann are with the System and Circuit Technology, University of Paderborn, 33102 Paderborn, Germany (e-mail: jenze@hni.uni-paderborn.de; porrmann@hni.uni-paderborn.de).

U. Rückert is with Cognitronics and Sensor Systems, Bielefeld University, 33615 Bielefeld, Germany (e-mail: rueckert@techfak.uni-bielefeld.de).

Digital Object Identifier 10.1109/TVLSI.2010.2044902

The rest of this paper is organized as follows. In Section II related work to the topic of partial reconfiguration is summarized. Section III gives an overview of tiled partially reconfigurable systems. The concept of subregions for tiled PR regions is introduced. Section IV classifies suitable communication infrastructures for PR regions and describes the concept of homogeneous communication macros. The resource requirements of the communication infrastructures are compared for a tiled PR region implementation in Xilinx Virtex FPGAs. Section V introduces the overlap graph as a novel methodology to quantify the placeability of the PR modules. A design method is described, which selects suitable synthesis regions for PR modules aiming to optimize their placement at run-time. The design method is demonstrated for a tiled PR region using the homogeneous communication macro. Section VI concludes this paper.

II. Related Work

One of the first system approaches for partially reconfigurable FPGAs is the swappable logic unit (SLU) presented by Brebner [3]. An SLU is a square-shaped fixed sized partially reconfigurable region with standard interfaces at each side. It is designed for the Xilinx XC6200 FPGA, which has a homogeneous array of logic resources. The placement of a PR module is done by allocating it to any available SLU. Modules can only be interconnected by placing them next to each other. In [4] and [5] a module-based design flow for partial reconfiguration is introduced, which is supported by the current Xilinx design tools. The design flow enables partial reconfiguration at the granularity of a PR region. A PR module always occupies a whole PR region—even if not all of the resources of the region are used. Every PR module uses a predefined set of routing resources to interconnect with the system outside the PR region. In [5] these routing resources are referred to as *bus macros*.

Since the upper bound of dynamic system components that can operate in parallel is equal to the number of PR regions, this type of PR region can be classified as a *single-module PR region*. The main drawbacks of this approach are as follows. The size of a PR module is limited by the size of the PR region. In systems with multiple PR regions this restriction can have a huge impact on the maximum size of a PR module. Apart from the size limit, small PR modules occupy the resources of the whole PR region causing low resource utilization. A solution to this problem is to combine multiple small modules to a larger one. However, such an approach requires additional memory for the configuration data for all combinations of these modules. Furthermore, the combined modules share the communication interface of the PR region. Consequently, the bandwidth of each individual module is limited, especially if several of these modules communicate with the rest of the system at the same time.

To overcome this drawback, a PR region can be designed to support a flexible placement of multiple modules within the region. In such a scenario placement algorithms are required, which aim at maximizing the resource utilization by minimizing the degree of fragmentation of the free resources inside the PR region. If the scheduling information and the execution times of the PR modules are known at design-time, the placement can be computed using offline methods such as those by Fekete [6] and Danne [7].

If the scheduling of the PR modules cannot be computed at design-time, the placement of a module must be performed at run-time. In this paper, we target applications that require online placement of PR modules. In this case, dynamic reconfiguration is triggered by external events such as changes in the environment or changes in the objectives of the application. Typical application scenarios can be found in the areas of software-defined radio (SDR) [8], signal and image processing [9], or adaptive control systems [10]. Bazarghan *et al.* [11] describe the problem of placing a PR module as an online bin-packing problem. Steiger *et al.* [12] discuss the problem of online placement and scheduling of hard real-time tasks for partially reconfigurable devices. Handa *et al.* [13] introduce a placement algorithm aiming at reducing the degree of fragmentation. In [14] a placement approach is introduced that considers the routing costs to existing instances of PR modules. Lu *et al.* [15] introduce an algorithm for PR module placement, which prepartitions the area of the PR region into blocks of different sizes. After the placement of a PR module the blocks are split and merged to maintain contiguous free resources.

The above methods assume a homogeneous PR region and neglect the fact that FPGAs are heterogeneous architectures. The heterogeneity of the resources significantly limits the placement of the PR modules within the PR region. A suitable placement algorithm to handle the heterogeneity is introduced in [16].

When placing a PR module at a desired position, the communication infrastructure requires to be designed to allow interconnection of the PR module with the rest of the system. This can be realized by adapting the existing communication infrastructure using online routing. JRoute [17] and ADB [18] are tools that support online routing for Xilinx Virtex FPGAs. Both operate directly on the configuration data using the JBits API [19]. Raaijmakers and Wong [20] introduce a similar concept, which enables online routing to disconnect and connect arbitrarily shaped PR modules. The modification of the configuration data is done by the Xilinx design tools. In [21] and [22] online routing concepts are proposed, which are based on predefined routing primitives for horizontal and vertical routing. Online routing is performed by interconnecting these primitives.

While online routing solves the interconnection problem for partially reconfigurable regions, it requires a processing unit which performs the online routing on every placement or removal of a PR module. With an increasing number of PR modules the total reconfiguration time increases as well. Apart from the reconfiguration process of the PR module, an additional reconfiguration process is needed for the dynamic routing channels. After a PR module is removed the previously utilized routing channels should be released as well, which requires a further reconfiguration process. Thus, online routing introduces a larger overhead with respect to reconfiguration time and additional resources.

In this paper, we focus on a different approach based on communication macros. These macros implement static routing channels passing through the partially reconfigurable region. Dynamically configured PR modules can connect to these routing channels via predefined connection points to establish the communication to the rest of the system. In [8], Sedcole *et al.* demonstrate the concept of static routing channels in PR regions. In [5] bus macros, which support reserving a static

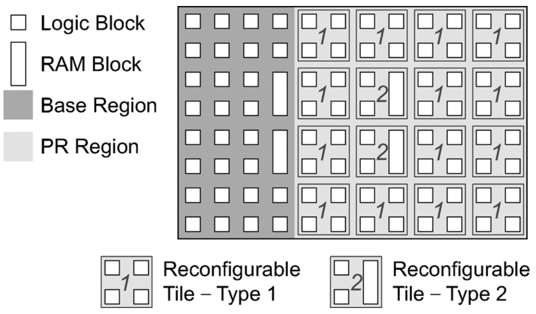

Fig. 1. Example of a partitioning scheme using a PR region with reconfigurable tiles.

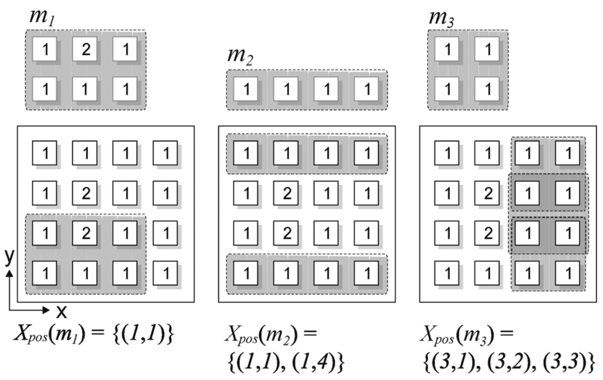
m_1
m_2
m_3
y
x
$X_{pos}(m_1) = \{(1,1)\}$
$X_{pos}(m_2) = \{(1,1), (1,4)\}$
$X_{pos}(m_3) = \{(3,1), (3,2), (3,3)\}$

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

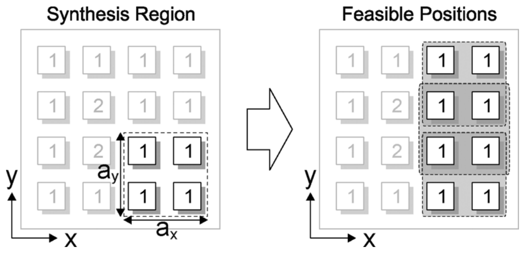

Fig. 3. Synthesis region of PR module m_3 and the corresponding feasible positions.

link between the static part of the system and a PR region, are described. In [1] embedded communication macros are introduced. These allow advanced communication infrastructures, such as buses or networks to be built. All the previously described methods have been designed for Xilinx Virtex-II FPGAs, which only allow a 1-D partially reconfigurability. In this paper, we focus on communication schemes, which are suited to 2-D partial reconfigurability as supported by Xilinx Virtex-4 up to Virtex-6 FPGAs.

III. Tiled Partially Reconfigurable Systems

In a tiled partially reconfigurable system as described in [1] the partially reconfigurable region is subdivided into reconfigurable tiles. Tiled partitioning allows for the placement of multiple PR modules with various sizes 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 tile sizes may vary according to the different resource types within each tile. In this context we define $r_{\text{tile}}(x,y)$ as the *resource availability* of the reconfigurable tile at position (x,y) in the PR region. The resource availability $r_{\text{tile}}(x,y)$ is an n-tuple, where n is the number of different resource types. For instance, $r_{\text{tile}}(x,y) = (\#Slices, \#BRAMs, \#DSP48s)$ describes the resource availability for tiles of FPGAs with Slices, BlockRAMs, and DSPs. 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 *communication macros*. Fig. 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. Systems using tiled PR regions can be designed to be open or closed, such that PR modules can be added or removed from the set of PR modules.

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* $X_{\text{pos}}(m) = \{(x_1,y_1),(x_2,y_2),\ldots\}$. Without loss of generality, $(x_i,y_i) \in X_{\text{pos}}(m)$ denotes 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. Fig. 2 shows the PR region of Fig. 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.

The synthesis of a PR module for a tiled PR region differs from the one for single-module PR regions. Firstly, the PR region is partitioned into reconfigurable tiles. Then the embedded communication macro is placed to interconnect the tiles. The PR module of the dynamic system component is synthesized within a predefined *synthesis region* using the preplaced embedded communication macro. The number of available resources within the synthesis region must be large enough to satisfy the resource requirements of the dynamic system component. Finally, the partial configuration data of the PR module is generated, which only targets the tiles of the chosen synthesis region. By manipulating the configuration data as described in [23] and [24], a PR module can be placed at any position with the same arrangement as the types of tiles from which it is built. Therefore, the location and size of the synthesis region defines the set of feasible positions of the PR module as indicated in Fig. 3.

In any reconfigurable system, the number and the size of the concurrently executable PR modules is 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 $(n_{\text{alloc}}-1)$ 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 rejected.

In this case, the synthesis regions and the corresponding feasible positions need to be selected to obtain the required allocation width. In the context of tiled PR regions, the PR region can be partitioned into disjoint *subregions*. A subregion is an area within a tiled PR region that typically is composed of a few tiles. It offers enough resources to allow the placement of an instance of any PR module. Within the subregion modules can be placed according to their feasible positions. Compared to an unconstrained tiled PR region, the tiles occupied by an instance of any PR module must be located completely within a subregion. This ensures that at any time as many PR modules can be executed in parallel as given by the number of disjoint subregions. As a consequence the number of subregions corresponds to the allocation width.

Tiled subregions and single-module PR regions both provide a fixed allocation width. The advantage of subregions over single-module PR regions is that small modules do not waste the resources of a whole PR region, but only occupy the required number of tiles leaving the remaining tiles within the subregion for the placement of additional PR modules. In the context of subregions, the allocation width only describes a lower bound on the number of PR modules that can be executed in parallel.

IV. Communication Infrastructure

Partially reconfigurable systems require a communication infrastructure to interconnect the static and the dynamic system components. In general, a communication infrastructure comprises unidirectional and bidirectional signals. With respect to tiled PR regions, unidirectional signals can be driven by either a static component in the base region or by a PR module in the PR region. Bidirectional signals can be driven by multiple drivers, i.e., any static or dynamic system component. Furthermore, a communication infrastructure comprises shared signals and dedicated signals. Shared signals are multipoint connections, e.g., in a bus-based communication infrastructure they are typically used to transmit data and address information, while control and arbitration often require dedicated signals. In the context of partially reconfigurable systems the shared and dedicated signals from the base region to the PR modules are realized by static communication channels. These channels are implemented by predefined logic cells and corresponding routing resources.

An approach for establishing the communication between modules and the base region by means of bus macros is introduced in [4], [5]. The bus macro reserves a fixed set of connection points that are used for passing signals between a PR module and the base region. Within each module the connection points are located at the same relative position as shown in Fig. 4. Although the term suggests a bus macro to be a macro for implementing bus structures, it is typically not used for this purpose. Rather, this type of connection is commonly used in single-module PR regions to implement a

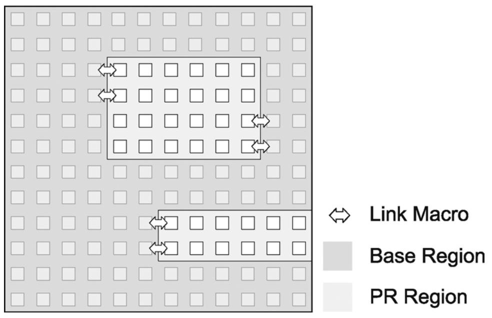

Fig. 4. Example of a link macro implementation for single-module PR regions.

communication link between a PR module and the base region. We refer to this type of connection as a *link macro*.

A. Communication Infrastructures Using Modified Link Macros

Extensions to link macros are necessary for a more advanced partitioning of the reconfigurable area, e.g., for tiles that cannot have a direct connection to the base region because all adjacent resources are occupied by other tiles.

Link macros for single-module PR regions can be adapted to realize *link macros between tiles* (LMBT). In this communication infrastructure, link macros are used to interconnect neighboring tiles. The main advantage of this approach is its simplicity. It can be easily realized for 1-D as well as for 2-D arrangements of tiles, as shown in Fig. 5(a). However, this simple approach has various disadvantages. Link macros are only used to establish the connection from one tile to another. Therefore, the bandwidth suffers from a potentially large number of hops across a sequence of tiles. Moreover, the connection of the link macros within a tile, which forms the actual communication infrastructure, must be realized inside the PR modules. This means that the actual routing is module-dependent and will be interrupted and changed during the reconfiguration process of a tile. For the same reason, the implementation of the communication for one tile depends on the communication infrastructure of the surrounding tiles. Each new module placement will require changes to all other modules that are involved in its communication causing additional reconfiguration overhead. These problems are somewhat related to online-routing approaches.

A simple approach to circumvent the restrictions of the approach described above is to ensure that each tile can be directly connected to the base region. To achieve this, *link macros with communication channels* can be used. An example of this concept is illustrated in Fig. 5(b) for a 2-D partitioning of the PR region. The communication channels are part of the base region, and the link macros are placed between this region and the partially reconfigurable region. The partitioning illustrated in Fig. 5(b) shows that the PR region is now split into multiple PR region segments since the area of the communication channels can only be used for communication and static logic. Since the PR region segments are separated by the communication channel, the maximum size of a PR module is limited to the size of a segment.

Wormhole routing [25] is a method to realize a communication infrastructure that circumvents the limitations of the approaches described above. Here, communication macros

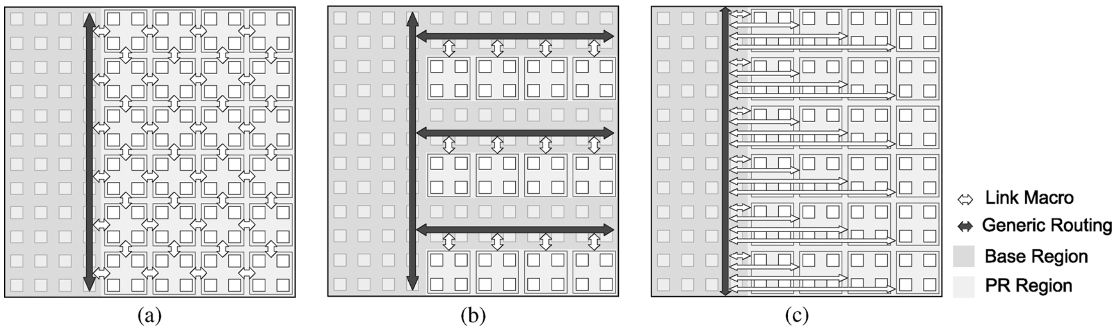

Fig. 5. Example of communication infrastructure using modified link macros. (a) Link macros between tiles (LMBT); (b) link macros in combination with communication channels; (c) wormhole routing.

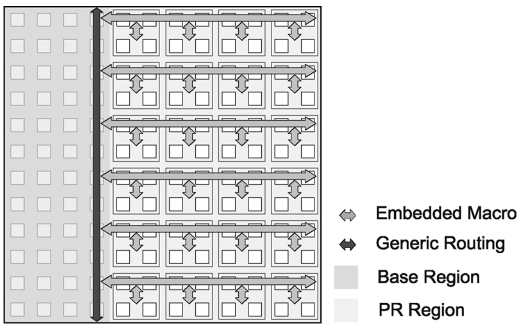

Fig. 6. Example of communication infrastructure using embedded macros.

are used that span over one or more complete tiles, if the destination tile is not adjacent to the base region. These adapted link macros are referred to as wormhole macros or wormhole routing [25]. When the reconfigurable area comprises a large number of tiles, this approach consumes a considerable amount of routing resources, as can be seen in Fig. 5(c). Therefore the routing resources that are available for PR modules linearly decrease from right to left. Since only point-to-point connections between a tile and the base region are used, every tile requires an exclusive set of communication lines.

B. Embedded Macros

The term *embedded macro* describes a method of using a communication infrastructure, which is embedded into the tile itself. The embedded macro is not a point-to-point connection. Instead of using multiple instances of simple link macros, one monolithic macro is used as shown in Fig. 6. The macro connects all tiles and the base region in a homogeneous manner. It combines the advantages of the three link macro variants described above, without sharing their drawbacks.

Fig. 7 shows the realization of a 1-bit bidirectional multipoint connection in the embedded macro. The multipoint connection is realized by horizontally interconnected AND gates. Each tile has a connection point for input data (D_{in}) and for output data (D_{out}). The enable signal is used to grant access to the multipoint connection. Since every PR module uses the same resources for the embedded macro, the macro is not modified at the run-time reconfiguration of a PR module. This ensures that the communication is not interrupted during the reconfiguration process. The macro is fully embedded in the tiles, such that there is no need for additional routing channels outside the tiles.

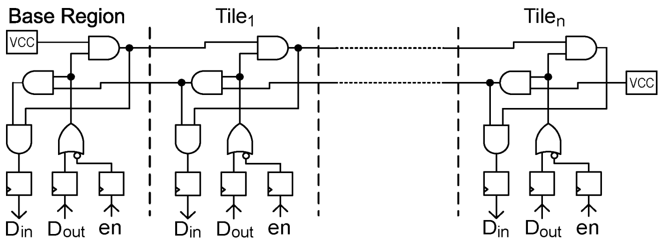

Fig. 7. 1-Bit bidirectional multipoint connection in the embedded macro.

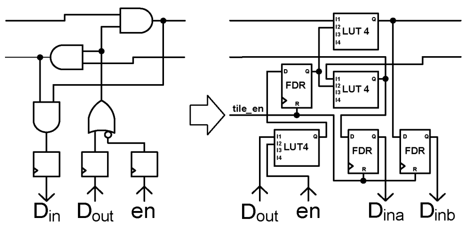

Fig. 8. Xilinx Virtex-4 implementation of a bidirectional multipoint connection in the embedded macro.

The implementation of the embedded macro can be optimized according to the given resources of the target device. Fig. 8 shows an implementation of the multipoint connection, which is optimized for the slice architecture of Xilinx Virtex-4 FPGAs. The implementation efficiently utilizes the resources of a Virtex-4 slice and requires only three lookup table (LUTs) and three registers. Additionally, the $tile\_en$ input can be used to individually enable or disable the data input/output (I/O). The data input signal D_{in} is divided into D_{ina} and D_{inb} to allow for a unidirectional and bidirectional communication.

Besides the Virtex-4 FPGA, we have optimized the embedded macro for Virtex-5 and Virtex-6 FPGA families. Table I shows a summary of the resource requirements for the optimized implementations. All connections to and from the communication infrastructure are registered, thus enabling high communication speeds. Furthermore, the registered implementation avoids problems in the PR flow related to asynchronous paths between modules and base region. The resource requirements cannot be directly compared between the different FPGA families since the routing and slice architecture are different in all device families, as indicated in the row “Resources per Slice” in Table I.

Table I shows five different types of signals, which usually occur in a communication infrastructure. Bidirectional shared

TABLE I
RESOURCES PER TILE FOR GENERIC COMMUNICATION STRUCTURES ON DIFFERENT FPGA-ARCHITECTURES. ALL SIGNALS ARE REGISTERED. N DENOTES THE TOTAL NUMBER OF COMMUNICATION TILES IN THE EFFECTIVE COMMUNICATION STRUCTURE. THE TERM *slices* REFERS TO THE SIZE OF A SLICE IN THE SPECIFIC ARCHITECTURE

	Virtex-4 Spartan-3 Virtex-II	Virtex-5	Virtex-6 Spartan-6
Resources per Slice (specific per architecture)	2 (4-Input)-LUTs, 2 FFs	4 (6-Input)-LUTs, 4 FFs	4 (6-Input)-LUTs, 8 FFs
Bidirectional Shared Signals (individual enable)	$1.5\ \frac{\text{V4-SLICES}}{\text{SIGNAL}}$	$0.75\ \frac{\text{V5-SLICES}}{\text{SIGNAL}}$	$0.5\ \frac{\text{V6-SLICES}}{\text{SIGNAL}}$
Bidirectional Shared Signals (common enable)	$1.0\ \frac{\text{V4-SLICES}}{\text{SIGNAL}}$	$0.5\ \frac{\text{V5-SLICES}}{\text{SIGNAL}}$	$0.5\ \frac{\text{V6-SLICES}}{\text{SIGNAL}}$
Unidirectional Shared Signals (driven by one point)	$0.5\ \frac{\text{V4-SLICES}}{\text{SIGNAL}}$	$0.25\ \frac{\text{V5-SLICES}}{\text{SIGNAL}}$	$0.125\ \frac{\text{V6-SLICES}}{\text{SIGNAL}}$
Dedicated Signals (binary encoding)	$\left(\left\lceil\frac{\log_2(N)}{2}\right\rceil+1\right)\frac{\text{V4-SLICES}}{\text{SIGNAL}}$	$\left(\left\lceil\frac{\log_2(N)}{4}\right\rceil+1\right)\frac{\text{V5-SLICES}}{\text{SIGNAL}}$	$\left(\left\lceil\frac{\log_2(N)}{8}\right\rceil+1\right)\frac{\text{V6-SLICES}}{\text{SIGNAL}}$
Dedicated Signals (one-hot encoding)	$\left\lceil\frac{N}{2}\right\rceil\frac{\text{V4-SLICES}}{\text{SIGNAL}}$	$\left\lceil\frac{N}{4}\right\rceil\frac{\text{V5-SLICES}}{\text{SIGNAL}}$	$\left\lceil\frac{N}{8}\right\rceil\frac{\text{V6-SLICES}}{\text{SIGNAL}}$

signals with common enable use a single enable signal. An example for these type of signals are the data lines of a bus-based communication infrastructure. Typically, many of these signals are used together for the same bus (e.g., 32 or 64 signals). Bidirectional shared signals with individual enable can be used to implement non-dedicated control signals, such as R/W signals. Unidirectional shared signals are driven by one component, e.g., by the base region or by a certain tile within the PR region. They can be used, e.g., for address signals in a single master system.

In addition to shared signals, a communication infrastructure typically uses dedicated signals, enabling access to individual modules. Dedicated signals are necessary for control and arbitration. A straightforward implementation of dedicated point-to-point connections causes an inhomogeneity of the PR region. An inhomogeneous communication infrastructure can significantly reduce the number of feasible positions of the PR modules. In order to maintain the homogeneity of the communication infrastructures, the implementation of dedicated signals requires special design concepts introduced in [1] and [27].

Fig. 9 shows the realization of dedicated signals in the embedded macro. All dedicated signals utilize the same routing channels, such that the homogeneity of the macro is preserved. The separation of the signals is achieved by bit masking. Each tile has its unique address stored in dedicated address registers. The states of these registers must be preserved during runtime reconfiguration. Table I shows the corresponding resource requirements for binary-encoded dedicated signals as well as one-hot encoded dedicated signals. Fig. 10 shows the implementation of binary-encoded dedicated signals optimized for a Xilinx Virtex-4 FPGA. This implementation can be realized with three Virtex-4 slices using the fast dedicated carry chains of the CLB. This allows us to obtain a realization with a short delay even for large number of tiles. The address registers are initialized during the reconfiguration process by using the mechanisms described in [1].

C. Comparison

This section presents a comparison of the previously described embedded communication macro and the alternative communication macros for partially reconfigurable systems.

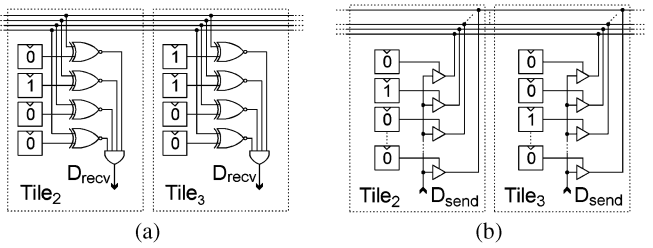

Fig. 9. Implementation of dedicated signals in the embedded macro. (a) Binary-encoded receive signal; (b) one-hot-encoded send signal.

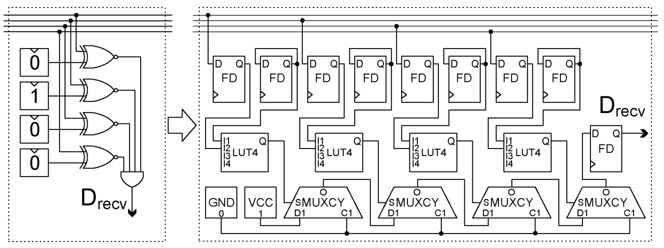

Fig. 10. Xilinx Virtex-4 implementation of dedicated signals in the embedded macro.

Since the implementation of the link macros with communication channels [see Fig. 5(b)] is located in the static region, we concentrate on the approaches, which implement the routing within the PR region, such as the LMBT approach [see Fig. 5(a)], the wormhole routing approach [see Fig. 5(c)] and the embedded macro [see Fig. 6]. Table II shows the resource requirements of a single-master implementation of an embedded macro. For dedicated signals, a maximum length of eight consecutive tiles (effective length $N \leq 8$) is assumed.

In a link macro or embedded macro implementation registers can be inserted into each tile to reduce communication delay and subsequently improve the clock frequency. However, using a pipelined communication may require additional adaptations of the protocol since the number of clock cycles for the communication depends on the distance of the communicating tiles. In the embedded macro we decided to add registers to all inputs and outputs of the bus subscriber as shown in Fig. 7. This guarantees a high clock rate with a pipelined structure and a fixed

TABLE II
RESOURCE REQUIREMENTS FOR A SINGLE-MASTER EMBEDDED COMMUNICATION MACRO

Signal Class	Signal Type	V4-Slices / Tile	V5-Slices / Tile	V6-Slices / Tile
Bidir. Sh. Signals (individual enable)	ACK, ADDR_ACK	3	2	1
Bidir. Sh. Signals (common enable)	32 Bit DATA	32	16	16
Unidir. Sh. Signals (driven by base region)	16 Bit ADDR, 4 Bit BE, R/W, BURST, RST, STB	12	6	3
Dedicated Signal (binary encoding)	CS	3	2	2
	Total:	*50*	*26*	*22*

TABLE III
COMPARISON OF RESOURCE REQUIREMENTS FOR THE SINGLE-MASTER IMPLEMENTATION WITH N CONSECUTIVE TILES, REALIZED USING DIFFERENT MACRO APPROACHES

	Embedded Macro	LMBT	Wormhole Macro
V4-Slices / Tile	50	94 [47]	47
V4-Slices / Base Region	50	47	$N \cdot 47$
V4-Slices / Design	$(N+1) \cdot 50$	$N \cdot 94$	$N \cdot 94$

latency of two clock cycles. To prevent a bandwidth reduction caused by the bus latency, the generic bus architecture supports fast back-to-back transactions as well as separate address- and data-acknowledges. Thus, in most cases it is possible to transfer one data word per clock cycle, which results in a near to optimal transfer rate.

Table III compares the resource requirements of the single-master implementation with the resource requirements of a comparable wormhole routing and link macro implementation on a Virtex-4 FPGA. The link macro implementation is realized by macros between tiles as shown in Fig. 5(a). All link-macro-based tiles require 94 slices for the communication, except the rightmost tiles, which only need 47 slices. For a communication infrastructure between the base region and 3 tiles (effective length $N = 3$) the embedded macro requires 200 slices, while the link macro approach and the wormhole routing approach require 282 slices. Even for a low number of consecutive tiles the embedded macro needs less resources than the other implementations. While embedded macro primitives can implement different types of signals and topologies, link-macro-primitives suffer from the fact that they only allow unidirectional point-to-point connections. As a consequence, two unidirectional signals (link macro primitives) are necessary to implement one bidirectional signal. In the wormhole routing approach the number of communication resources within each tile is lower than that in the embedded macro implementation. In contrast to the embedded macro implementation, the number of resources in the base region scales linearly with the number of tiles, such that in total the wormhole routing requires as many slices as the link macro implementation.

TABLE IV
FEATURE COMPARISON OF DIFFERENT COMMUNICATION MACRO APPROACHES

	Embedded Macro	LMBT	Wormhole Macro
Shared Signals	yes	yes	no
Dedicated Signals	yes	no	yes
Homogeneity	yes	yes	no
Interrupt-Free Reconfiguration	yes	no	yes
Clock Frequency	o	-	+
Resources	+	o	-

Table IV shows a feature comparison of the different communication macro approaches. As discussed earlier, the embedded macro supports the implementation of shared signals. Using LMBT, the problem of realizing shared signals is shifted from the macro to the module implementation. Shared signals can only be realized if the module supports forwarding of input and output signals to its neighboring modules. The wormhole routing macro only allows for direct point-to-point connection from a tile to the base region, such that a shared signal cannot be implemented directly. Dedicated signals can be implemented by embedded macros and wormhole macros. Since LMBT does not allow a point-to-point connection from any tile to the base region, they cannot be used for dedicated signals. When configuring a PR module in a PR region with LMBT, the communication is interrupted for the time of the configuration process. Since the routing channels of the embedded macro and of the wormhole macro are fixed, the communication is not interrupted during a reconfiguration process.

In general the delay of all presented communication approaches increases with the number of consecutive tiles. Among the presented approaches, the wormhole macro allows the fastest clock frequency because the base region and the specific tile are connected directly using fast routing resources. However, the dedicated routing resources from the base region to each tile will cause an excessive consumption of routing resources, and a large resource requirement in the base region (cf. Table III). The link macro between tiles approach requires less routing resources than the wormhole macro approach, while the overall slice requirement is comparable. Furthermore, this macro does not require the enormous amount of slices in the base region, as the wormhole routing did. The clock frequency of the link macro between tiles approach will be much slower than the wormhole approach, since most of the communication infrastructure is realized in the module. Thus, the bandwidth suffers from short hops across the sequence of tiles. In contrast to the LMBT approach, the embedded macro supports optimized homogeneous routing within each tile. Thus, the maximum delay is smaller than in a comparable LMBT approach, where the routing depends on the module implementation.

With respect to the placeability of PR modules, it is important to avoid introducing additional heterogeneity in the PR region, since this would further restrict the number of feasible positions of a PR module. Therefore, the communication infrastructure should be as homogeneous as possible. Wormhole routing is heterogeneous, since various routing channels are used to interconnect the tiles with the base region. The embedded macro and the

LMBT approaches are homogeneous, such that the number of feasible positions for the corresponding PR modules is greater than that for wormhole routing.

The placeability of the PR modules can be further improved if various synthesis options are considered, which define the feasible positions of the PR modules. In the following section, we describe an optimization method to improve the placeability of the PR modules.

V. Design-Time Optimization of PR Modules

With respect to run-time placement, the PR modules vary according to their resource requirements, their shape, and their feasible positions. Each feasible position of a PR module can have a different degree of overlap with the feasible positions of the other PR modules in the system. The degree of overlap has an impact on the placeability of the PR module. Those feasible positions that overlap with many other feasible positions are likely to be blocked by a previously placed instance of another PR module. Thus a reasonable online placement policy is to always select the free position with the least degree of overlap as discussed in [16]. Besides maintaining a large number of free positions at run-time, it is also possible to optimize the placeability of PR modules at design-time. This is done by minimizing the degree of overlap of the feasible positions of the given PR modules. At design-time the set of feasible positions of a PR module is defined by the shape and position of the synthesis region. The optimization of the placeability is done by selecting the synthesis regions of the PR modules that allow the best possible placement at run-time.

A. Placeability of PR Modules

In order to optimize the placeability of the PR modules a metric is required, which quantifies the degree of overlap of the feasible positions. 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 $(x,y)\in X_{\text{pos}}(m)$ of the PR module $m\in M$. The set of all vertices is defined as

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

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). Fig. 11 shows the overlap graph for the PR modules of the example in Fig. 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*

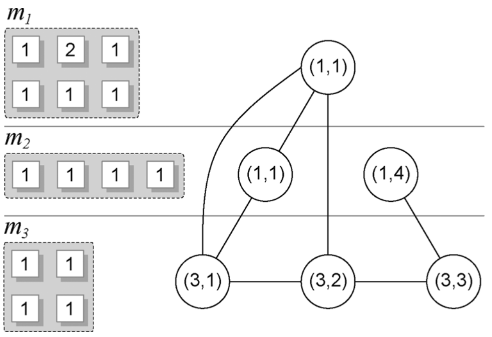

Fig. 11. Example of an overlap graph.

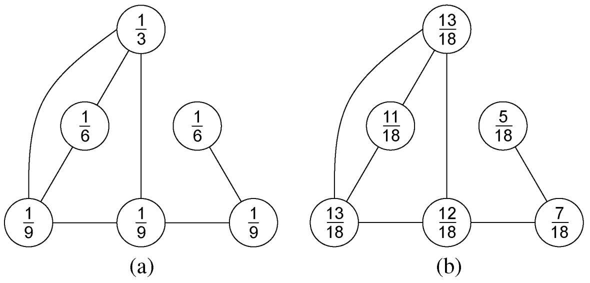

Fig. 12. Probability and the position weights. (a) Probability weights $w_p(v)$; (b) position weights $w_{\text{pos}}(v)$.

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

are computed for each vertex $v=(m,x,y)\in V$, where $p_{\text{alloc}}(m)$ 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. Fig. 12(a) shows the corresponding probability weights for the PR modules shown in Fig. 2 with a fixed $p_{\text{alloc}}(m)=1/|M|$. With 3 PR modules, the allocation probability is $\forall m\in M: p_{\text{alloc}}(m)=1/3$ and PR module m_2 has 2 feasible positions, such that the probability weight for its feasible positions is $w_p(v)=1/(3\cdot 2)=1/6$.

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

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

and the resulting position weight is calculated by

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

The position weights for the PR modules of Fig. 2 are shown in Fig. 12(b). The position weights reflect the degree of overlap. For example, 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).

Apart from the design-time aspects the position weight can also be used for the placement of PR modules at run-time. The placement is done by selecting the available position with the least position weight. This ensures maintaining a large number of available positions for future placements.

A metric to evaluate the degree of overlap of all feasible positions is to generate a weighted sum 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 when randomly placing a module, the overlap weight of all PR modules is defined as follows:

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

The weighted mean 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 $w_{\text{ovr}}(V)$. A small $w_{\text{ovr}}(V)$ 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. Thus the overlap weight is a metric for the placeability of all PR modules.

B. Optimization of the Overlap Weight

A dynamic system component can be represented by different PR module variants. Each PR module variant has 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. For simplicity reasons we assume that each dynamic system component is represented by one PR module in the following.

By using the overlap graph it is possible to optimize the feasible positions of these PR modules at design-time aiming to improve the placeability at run-time. The optimized design flow is shown in Fig. 13 and involves the following steps:

1) synthesis regions of at least one PR module variant for each dynamic system component are selected;
2) resulting PR modules and their feasible positions are computed;
3) overlap graph is generated based on the feasible positions of all PR modules;
4) overlap weight is derived based on the overlap graph.

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 synthesis region whose feasible positions cause the minimum overlap weight.

The design space of a PR module is defined by its set of possible synthesis regions. In the following we describe an algorithm to determine all synthesis regions for a rectangular PR module. Let us assume that for a synthesis region $s = (x, y, a_x, a_y)$ the dimensions in number of tiles are (a_x, a_y) and the position within the PR region is (x, y). D is the set of dynamic system components of the target application. To generate a PR module for a dynamic system component $d \in D$, a synthesis region needs to be defined offering enough resources to satisfy the resource requirement $r(d) = (\#Slices, \#BRAMs, \#DSP48s)$. If $r_{\text{tile}}(x, y) = (\#Slices, \#BRAMs, \#DSP48s)$ denotes the resource availability of the tile at position (x, y), then the resource constraint for generating a PR module at the synthesis region $s = (x, y, a_x, a_y)$ is

$$\sum_{i=0}^{a_x-1}\sum_{j=0}^{a_y-1} r_{\text{tile}}(x+i, y+j) \geq r(d). \tag{6}$$

A synthesis region is minimal if it cannot be reduced in any dimension without violating the resource constraint. The set of all minimal synthesis regions $S_{\min}(d)$ for a dynamic system component d can be derived by Algorithm 1.

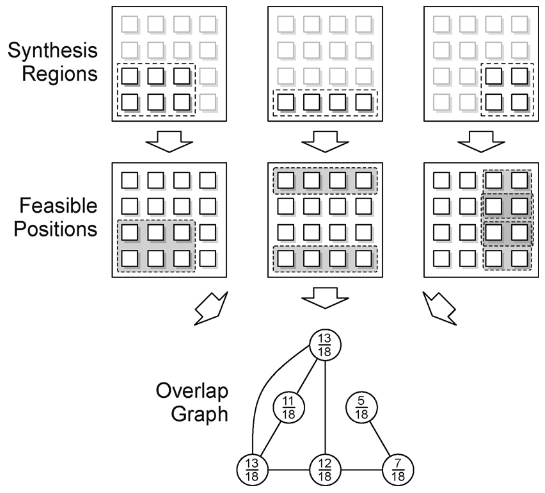

Fig. 13. Design steps from the selection of the synthesis region to the computation of the overlap weight considering only one PR module for each dynamic system component.

Alg. 1: Method to derive the set of minimal synthesis regions $S_{\min}(d)$ for dynamic system component $d \in D$.

Inputs: Dynamic system component d, resource requirement $r(d)$, Minimum width a_{xmin}, minimum height a_{ymin}. maximum width a_{xmax}, maximum height a_{ymax}, size of PR region (A_x, A_y), resource availability $r_{tile}(x, y)$.

Output: Set of minimal synthesis regions $S_{min}(d)$ for dynamic system component d.

```
(1) $a_x := a_{xmax}$
(2) $a_y := a_{ymin}$
(3) $S_{min}(d) := \{\}$
(4) while $a_y \leq a_{ymax}$ and $a_x \geq a_{xmin}$
(5)     for $y := 0, \ldots, A_y - a_y$
(6)         $x := 0$
(7)         $S_{cand} := \{\}$
(8)         while $x \leq A_x - a_x$
(9)             $(S_{cand}, a_x) = \mathbf{minreg}(d, S_{cand}, x, y, a_x, a_y)$
(10)            $x := x + 1$
(11)        end while
```

```
(12)          if $S_{cand} \neq \{\}$
(13)              $a_x := a_x - 1$
(14)              $S_{min}(d) := S_{min}(d) \cup S_{cand}$
(15)          end if
(16)      end for
(17)      $a_y = a_y + 1$
(18) end while
```

At the beginning the width a_x of the synthesis region is initialized with a predefined maximum width value $a_{x\,\max}$. By default this should be the width of the PR region in terms of number of tiles (A_x). To avoid very large and very small aspect ratios of the synthesis region, which would affect the performance of the resulting PR module, the value can be reduced. The height of the synthesis region is initialized with a predefined minimum value $a_{y\,\min}$. By default this should be 1, but it can also be increased to avoid large aspect ratios. The maximum height $a_{y\,\max}$ $(\mathrm{default} = A_y)$ and the minimum width $a_{x\,\min}$ $(\mathrm{default} = 1)$ can be specified, respectively. In every iteration of the loop in line (4), the height a_y is increased by 1 and the width a_x is adjusted according to the selected position and the resource requirements of the system component. The loop in line (5) iterates the vertical position y and the loop in line (8) iterates the horizontal position x. The set S_{cand} contains all found synthesis region candidates. Based on the candidates S_{cand}, the position (x, y), and the size (a_x, a_y) the function $(S_{\mathrm{cand}}, a_x) = \mathrm{minreg}(d, S_{\mathrm{cand}}, x, y, a_x, a_y)$ performs the following steps.

1) If the synthesis region (x, y, a_x, a_y) does not satisfy (6), then minreg does not change the candidates S_{cand} and the width a_x.
2) If the synthesis region (x, y, a_x, a_y) satisfies (6) and $(x, y, a_x - 1, a_y)$ does not, then (x, y, a_x, a_y) is a new candidate. If there is no other previously found candidate with the same size, which causes an identical set of feasible positions, then (x, y, a_x, a_y) is added to the set of candidates S_{cand}. The width a_x is not changed.
3) If the synthesis region $(x, y, a_x - 1, a_y)$ satisfies (6), then the previously found synthesis region candidates in S_{cand} are not minimal and can be discarded. The width a_x is reduced until $(x, y, a_x - 1, a_y)$ does not satisfy (6). The candidates are set to $S_{\mathrm{cand}} = \{(x, y, a_x, a_y)\}$.

For each dynamic system component $d \in D$ the set of minimal synthesis regions $S_{\min}(d)$ is computed. If only one PR module is generated for each system component, then the search space for all combinations of PR module variants is

$$\prod_{i=1}^{n} |S_{\min}(d_i)|, \quad n = |D|.$$

If $s_i \in S_{\min}(d_i)$, then the objective of the design-time optimization is to find the combination of synthesis regions $(s_1, s_2, \ldots, s_n)$, which results in an overlap graph that has the smallest overlap weight.

If a small number of PR modules (e.g., $|M| \leq 10$) is required to be added to the system, the set of PR modules with a minimum overlap weight can be determined by exhaustive search. To deal with 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.

Genetic algorithms are known to be a suitable approach for solving complex combinatorial optimization problems [28]. To deal with a large number of dynamic system components, we have implemented a genetic algorithm with the following specifications. The chromosome representation of an individual is done by a vector of synthesis regions $(s_1, s_2, \ldots, s_n)$, where $s_i \in S_{\min}(d_i)$. The fitness function is the inverse of the resulting overlap weight. Recombination is performed by selecting two parent individuals and swapping the genes (synthesis regions) up to a randomly selected crossover point. The selection of the parent individuals is done by roulette wheel selection, i.e., the probability of the individual of being selected depends on its fitness value. The mutation of an individual is achieved by randomly changing a gene.

In the following section, we demonstrate the impact of the design optimization for an example application.

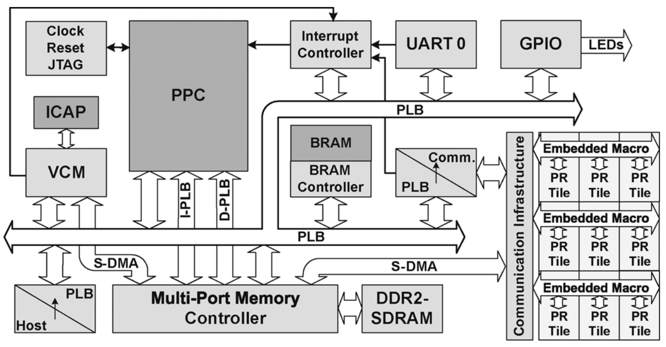

Fig. 14. Overview of the system implementation.

C. Example Implementation

For evaluating the design-time optimization, we consider an application in the context of embedded systems with a general purpose processor connected to a PR region. Partial reconfiguration is used to dynamically load IP cores, to allow hardware acceleration and improve system performance. An overview of the system implementation is shown in Fig. 14. The static system components are interconnected using a Processor Local Bus (PLB). The multiport memory controller allows for fast memory access to externally connected DDR2-SDRAM. The reconfiguration is controlled by the Virtex Configuration Manager (VCM) [26], which enables partial reconfiguration at a maximum speed of the ICAP interface (100 MHz, 32-bit). A Linux-based operating system is used, which supports run-time management of PR modules [29]. The communication infrastructure, which interconnects the PR region and the static system components, is based on the embedded communication macro as described in Section IV-B. The embedded macro preserves the homogeneity of the tiled PR region and ensures a large number of feasible positions for the PR modules.

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

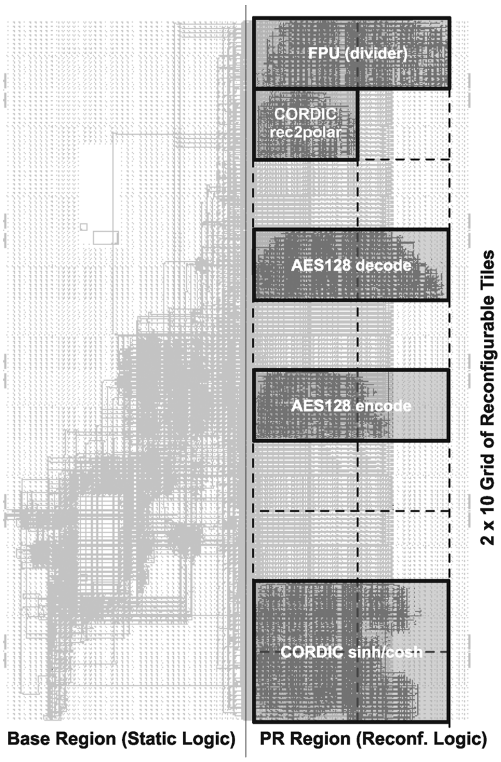
FPU (divider)
CORDIC rec2polar
AES128 decode
AES128 encode
CORDIC sinh/cosh
2 x 10 Grid of Reconfigurable Tiles
Base Region (Static Logic)
PR Region (Reconf. Logic)

Fig. 15. Partitioning of the Xilinx Virtex-4 FX100 FPGA using (2 × 10) tiles.

TABLE V
SPECIFICATION OF THE TILES FOR THE DIFFERENT PR REGIONS

PR Region	Tile Type	Slices	BRAMs	DSPs	peak Bandwidth	Slices for Comm.
(1x10)	center	1536	16	4	12.8 GBit/s	3,26%
(2x10)	left	768	8	4	10.7 GBit/s	6,51%
	right	768	8	-		
(3x10)	left	512	4	4	9.41 GBit/s	9,77%
	center	512	8	-		
	right	512	4	-		

TABLE VI
EXAMPLE SET OF DYNAMIC SYSTEM COMPONENTS

Application	Component	Slices	BRAMs	DSPs	Config. Data Size [kByte]
AES128	decryption	2450	4	-	127
	encryption	2195	-	-	102
CORDIC	arctan	2282	-	-	213
	rec2polar	748	-	-	63
	polar2rec	495	-	-	62
	sinh/cosh	2339	-	-	213
FPU	universal	1931	-	12	124
	add/sub	1149	-	-	63
	divider	1490	-	-	94
	mulitplier	1019	-	8	58

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. Fig. 15 shows an example of the partitioning for a Virtex-4 FX100 FPGA.

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 not be too small. The number of resources of the different tiles are shown in Table V. In this example the interconnection of the tiles is realized by a single-master embedded communication macro, which supports 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 [1]. The chosen communication infrastructure requires 56 slices per tile. As shown in Table V, the percentage of slices used for communication increases with reducing tile size. For each partitioning we have measured the register to register delay from the base region to the rightmost tile by using the timing analyzer tool of the Xilinx design tools. Based on the delay the peak bandwidth for a 32-bit communication is shown in Table V. With an increasing number of tiles the critical path of the embedded macro increases as well, such that the resulting peak bandwidth decreases.

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 direction. This homogeneity can be used to define subregions based on the following procedure. 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 VI. The components are considered to be hardware accelerators, which are dynamically loaded to enhance the functionality of the PowerPC.

Until now commonly accepted benchmarks for comparing dynamically reconfigurable systems have not been established. In order to test the placement, 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. Compared to a random removal of placed instances, this policy maintains comparability between the simulated benchmarks. Each test sequence contains 10 000 placement requests. The placement is done by selecting the free feasible position with the least position weight w_{pos}. A

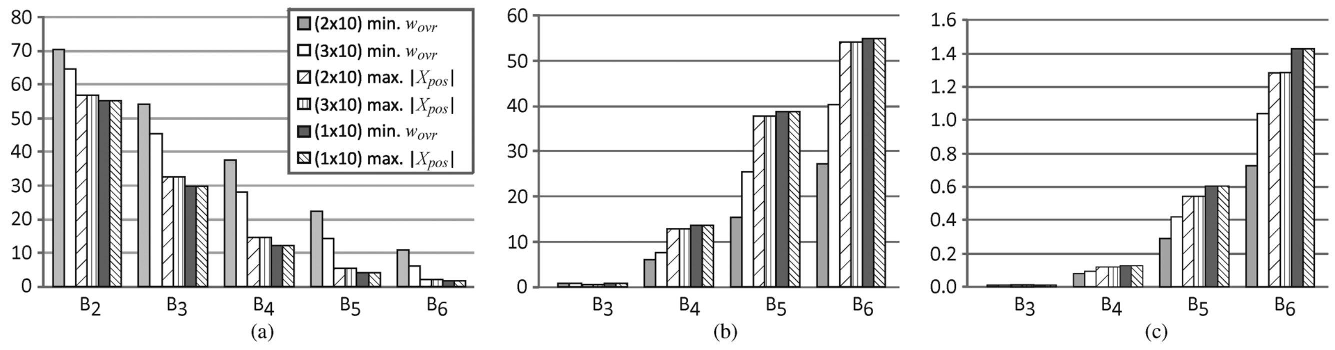

Fig. 16. Example simulation using the FPU components and the PR region partitionings (1 × 10), (2 × 10), and (3 × 10). (a) Average available positions [%]; (b) placement violations [%]; (c) average placement requests in queue.

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. We consider two approaches to handle placement violations. The first approach is a *reject policy*, where the unsuccessful placement request is rejected and not repeated in the future. This covers real-time systems, which do not allow delaying placements. The second approach is a *queuing policy* that queues unsuccessful placement requests and performs the placement as soon as enough resources become available.

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 the ones optimized with respect to the maximum number of feasible positions (max. $|X_{\text{pos}}|$).

A metric to determine the placement quality is to consider the percentage of available positions, which is determined by $(\sum_{m \in M} |X_{\text{free}}(m)|)/(\sum_{m \in M} |X_{\text{pos}}(m)|)$, where $X_{\text{free}}(m)$ is the set of available feasible positions for PR module m. Fig. 16(a) 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.

Fig. 16(b) shows the placement violations considering the reject policy. The benchmark B_2 did not cause any placement violations and is therefore not shown. In B_4, B_5, and B_6 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 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. For a queuing policy the results are equivalent, as shown in Fig. 16(c). It shows the average number of placement requests in the queue.

TABLE VII
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

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 tiles have a different resource availability and the dynamic system components use different types of resources, like in the case of the FPU components.

Table VII shows the percentage of placement violations using all dynamic system components and a reject policy. It focuses on the impact of using subregions in the PR region. In the simulation the subregions span the whole width of the PR region. The subregions for the tiled PR regions (2 × 10) must have a size of at least (2 × 3). A height of 3 is necessary since the component *FPU universal* requires 3 of the left tiles. By using subregions an allocation width of $n_{\text{alloc}} = 3$ can be achieved, while the tiled PR regions (2 × 10) and (3 × 10) without subregions cause placement violations in the benchmark B_3.

The previously shown results are based on the assumption that each dynamic system component has only one PR module variant. In the following we focus on the impact of using n different PR module variants for each dynamic system component.

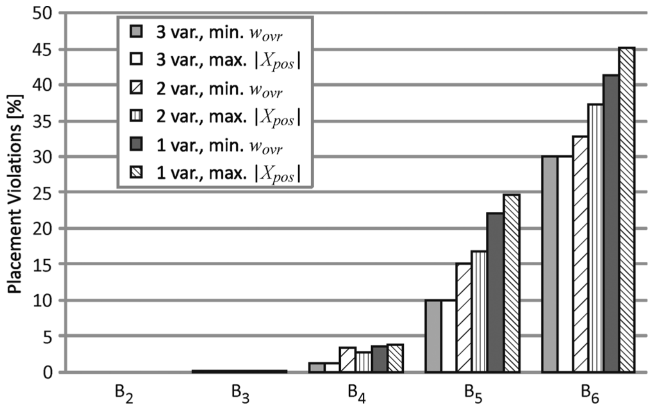

Fig. 17. Percentage of placement violations using all components and different numbers of PR module variants for (2 × 10) tiles.

Due to the homogeneity in vertical direction, the tiled PR region (2 × 10) allows a PR module to be made of either left tiles only, or right tiles only, or both type of tiles. Therefore, a dynamic system component can have at the most $n = 3$ PR module variants. The optimization of the overlap weight is done by the genetic algorithm as specified in Section V-B with a population size of 200 individuals. The best solution after 50 generations is used to generate the PR modules. The set of overlap weight optimized PR modules is generated by selecting the n PR module variants with the best w_{ovr} for each dynamic system component. In the case of the PR modules optimized for the maximum number of feasible positions, for each dynamic system component the n PR module variants with the largest number of feasible positions are added to the set of PR modules. Fig. 17 shows the placement violations for the tiled PR region (2 × 10).

The simulations with one PR module for each dynamic system component show that the overlap weight optimized PR modules cause fewer placement violations than the PR modules optimized for the maximum number of feasible positions. In the simulations with three PR module variants, the set of overlap weight optimized PR modules is the same as the set of PR modules with the maximum number of feasible positions. The placement violations are reduced by up to 55.4% for the benchmark B_5. With an increasing number of PR module variants, the number of placement violations decreases, but the memory for storing the equivalent configuration data increases as well. Therefore, the overlap weight optimization is particularly useful for embedded systems with a limited amount of configuration data memory that allow storing only one or two PR module variants per dynamic system component.

VI. Conclusion

In this paper, we discuss concepts to implement the on-chip communication infrastructure for interconnecting partial reconfiguration modules. Currently, Xilinx bus macros are the only semi-automated approach for generating communication links between two adjacent single-module PR regions. For tiled PR regions, which offer a higher flexibility in placing a PR module, these communication links are not suitable. In heterogeneous FPGA architectures, it is important to design a homogeneous communication infrastructure to avoid further restricting the placement of PR modules by introducing an additional degree of heterogeneity. The utilized communication resources in each tile should be the same, such that PR modules can connect at a predefined connection point without causing any glitches or interrupting ongoing communication. The presented embedded communication macro fulfills these requirements and can easily be adapted to any partially reconfigurable architecture. In comparison with link macros or wormhole routing approaches, the embedded communication macro has the best resource utilization.

In tiled PR regions, the placement quality of PR modules depends not only on the placement algorithm, but also 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 a PR module 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 for deriving the position weights. As shown by experiments, the overlap optimization can significantly improve the placement of PR modules. In addition to overlap optimization, the paper introduces the concept of subregions in a tiled PR region. A tiled PR region with subregions and multiple single-module PR regions share the property of supporting a fixed allocation width, which is necessary for applications that are not able to handle placement violations. In a single-module PR region approach the packing of several small modules into a combined PR module requires sharing the communication interface. Additionally, the synthesis of different PR modules for the various combinations is required. These drawbacks are overcome in the tiled subregion approach. The use of subregions allows placing additional PR modules if a reasonable amount of free contiguous resources is available within the subregion.

Although the presented design concepts are demonstrated for Xilinx Virtex FPGAs, they can be applied to any partially reconfigurable device. Future work includes refining the design method to support automated search for the number and the sizes of the tiles, and extending our approach to cover a wide range of applications.

References

[1] J. Hagemeyer, B. Kettelhoit, M. Koester, and M. Porrmann, “Design of homogeneous communication infrastructures for partially reconfigurable FPGAs,” in *Proc. Int. Conf. Eng. Reconfigurable Syst. Algorithms (ERSA)*, 2007, pp. 238–247.

[2] M. Koester, W. Luk, J. Hagemeyer, and M. Porrmann, “Design optimizations to improve placeability of partial reconfiguration modules,” in *Proc. Int. Conf. Des., Autom. Test Eur. (DATE)*, 2009, pp. 976–981.

[3] G. Brebner, “The swappable logic unit: A paradigm for virtual hardware,” in *Proc. 5th IEEE Symp. FPGA-Based Custom Comput. Mach. (FCCM)*, Washington, DC, 1997, pp. 77–86.

[4] Xilinx Inc., San Jose, CA, “Two flows for partial re-configuration: Module based or small bit manipulations,” Appl. Notes 290, 2002.

[5] 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 *Proc. 16th Int. Conf. Field Program. Logic Appl.*, 2006, pp. 12–17.

[6] S. Fekete, E. Köhler, and J. Teich, “Optimal FPGA module placement with temporal precedence constraints,” in *Proc. Conf. Des., Autom. Test Eur.*, Piscataway, NJ, 2001, pp. 658–667.

[7] K. Danne and S. Stühmeier, “Off-line placement of tasks onto reconfigurable hardware considering geometrical task variants,” in *Proc. Int. Embed. Syst. Symp. (IESS)*, 2005, pp. 15–17.

[8] P. Sedcole, B. Blodget, J. Anderson, P. Lysaght, and T. Becker, "Modular partial reconfiguration in Virtex FPGAs," in *Proc. 15th Int. Conf. Field Program. Logic Appl. (FPL)*, 2005, pp. 211–216.

[9] P. Manet, D. Maufroid, L. Tosi, G. Gailliard, O. Mulertt, M. Di Ciano, J.-D. Legat, D. Aulagnier, C. Gamrat, R. Liberati, V. La Barba, P. Cuvelier, B. Rousseau, and P. Gelineau, "An evaluation of dynamic partial reconfiguration for signal and image processing in professional electronics applications," *EURASIP J. Embed. Syst.*, vol. 2008, pp. 1–11, 2008.

[10] S. Toscher, T. Reinemann, and R. Kasper, "An adaptive FPGA-based mechatronic control system supporting partial reconfiguration of controller functionalities," in *Proc. NASA/ESA Conf. Adapt. Hardw. Syst.*, 2006, pp. 225–228.

[11] K. Bazargan, R. Kastner, and M. Sarrafzadeh, "Fast template placement for reconfigurable computing systems," *IEEE Des. Test Comput.*, vol. 17, no. 1, pp. 68–83, Jan. 2000.

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

[13] M. Handa and R. Vemuri, "Area fragmentation in reconfigurable operating systems," in *Proc. Int. Conf. Eng. Reconfigurable Syst. Algorithms*, 2004, pp. 77–83.

[14] A. Ahmadinia, C. Bobda, M. Bednara, and J. Teich, "A new approach for on-line placement on reconfigurable devices," in *Proc. 18th Int. Parallel Distrib. Process. Symp. (IPDPS)*, 2004, pp. 134–140.

[15] 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 *Proc. 22nd Ann. Int. Parallel Distrib. Process. Symp. (IPDPS)—RAW*, 2008, pp. 1–8.

[16] M. Koester, H. Kalte, and M. Porrmann, "Task placement for heterogeneous reconfigurable architectures," in *Proc. IEEE Conf. Field-Program. Technol. (FPT)*, 2005, pp. 43–50.

[17] E. Keller, "Jroute: A run-time routing API for FPGA hardware," in *Proc. 15th Int. Parallel Distrib. Process. Symp. (IPDPS)—Workshops Parallel Distrib. Process.*, 2000, pp. 874–881.

[18] N. Steiner and P. Athanas, "An alternate wire database for Xilinx FPGAs," in *Proc. 12th Ann. IEEE Symp. Field-Program. Custom Comput. Mach.*, 2004, pp. 336–337.

[19] S. A. Guccione and D. Levi, "XBI: A java-based interface to FPGA hardware," in *Proc. SPIE Conf. Configur. Comput.: Technol. Appl.*, 1998, vol. 3526, pp. 97–102.

[20] S. Raaijmakers and S. Wong, "Run-time partial reconfiguration for removal, placement and routing on the Virtex-II-Pro," in *Proc. Int. Conf. Field Program. Logic Appl.*, K. Bertels, W. A. Najjar, A. J. van Genderen, and S. Vassiliadis, Eds., 2007, pp. 679–683.

[21] M. Huebner, C. Schuck, and J. Becker, "Elementary block based 2-dimensional dynamic and partial reconfiguration for virtex-II FPGAs," presented at the 20th Int. Parallel Distrib. Process. Symp. (IPDPS), Rhodes Island, Greece, 2006.

[22] P. Athanas, J. Bowen, T. Dunham, C. Patterson, J. Rice, M. Shelburne, and J. Suris, "Wires on demand: Run-time communication synthesis for reconfigurable computing," in *Proc. Int. Conf. Field Program. Logic Appl.*, 2008, pp. 513–516.

[23] H. Kalte and M. Porrmann, "REPLICA2Pro: Task relocation by bitstream manipulation in Virtex-II/Pro FPGAs," in *Proc. ACM Int. Conf. Comput. Frontiers*, 2006, pp. 403–412.

[24] T. Becker, W. Luk, and P. Cheung, "Enhancing relocatability of partial bitstreams for run-time reconfiguration," in *Proc. IEEE Symp. Field-Program. Custom Comput. Mach. (FCCM)*, 2007, pp. 35–44.

[25] T. Marescaux, A. Bartic, D. Verkest, S. Vernalde, and R. Lauwereins, "Interconnection networks enable fine-grain dynamic multi-tasking on FPGAs," in *Proc. 12th Int. Conf. Field-Program. Logic Appl. (FPL)*, 2002, pp. 795–805.

[26] J. Hagemeyer, B. Kettelhoit, M. Koester, and M. Porrmann, "A design methhodology for communication infrastructures on partially reconfigurable FPGAs," in *Proc. 17th Int. Conf. Field Program. Logic Appl. (FPL)*, 2007, pp. 331–338.

[27] J. Hagemeyer, B. Kettelhoit, and M. Porrmann, "Dedicated module access in dynamically reconfigurable systems," presented at the 20th Int. Parallel Distrib. Process. Symp. (IPDPS), Rhodes Island, Greece, 2006.

[28] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley, Jan. 1989.

[29] M. D. Santambrogio, V. Rana, and D. Sciuto, "Operating system support for online partial dynamic reconfiguration management," in *Proc. Int. Conf. Field Program. Logic Appl.*, 2008, pp. 455–458.

Markus Koester received the Ph.D. degree from the University of Paderborn, Paderborn, Germany, in 2007, for his work on resource management for partially reconfigurable architectures.

In October 2007, he received a Post-Doctoral scholarship from the German Research Council (DFG) and continued his research on partial reconfiguration in the "Custom Computing" Research Group, Imperial College London, London, U.K. His works were focused on design optimizations for FPGA-based reconfigurable systems. In April 2009, he returned to the University of Paderborn and started to work as a Research Assistant with the "System and Circuit Technology" Group. He is currently conducting research in the area of reconfigurable computing in embedded systems.

Wayne Luk (F'09) received the M.A., M.Sc., and D.Phil. degrees in engineering and computing science from the University of Oxford, Oxford, U.K.

He is a Professor of computer engineering with the Department of Computing, Imperial College London, London, U.K., and was a Visiting Professor with Stanford University, Stanford, CA, and with Queen's University Belfast, Belfast, U.K. His research interests include theory and practice of customizing hardware and software for specific application domains, such as multimedia, financial modelling, and medical computing. Much of his current work involves high-level compilation techniques and tools for high-performance computers and embedded systems, particularly those containing accelerators such as field-programmable gate arrays and graphics processing units.

Jens Hagemeyer studied electrical engineering combined with computer science at the University of Paderborn, Paderborn, Germany. He received the diploma degree from the University of Paderborn, in 2006, where he is currently pursuing the Ph.D. degree in the area of FPGA-centric research topics, especially run-time reconfiguration of single-FPGA and multi-FPGA systems.

He is working with the Research Group "System and Circuit Technology", University of Paderborn, as a Research Assistant. He has designed several components for the RAPTOR rapid prototyping system, using state of the art HDI-PCB technology. Since 2004, he is active in the area of FPGA research.

Mario Porrmann (M'08) received the Diploma degree in electrical engineering from the University of Dortmund, Dortmund, Germany, and the Dr.-Ing. degree from the University of Paderborn, Paderborrn, Germany, in 1994 and 2001, respectively.

From 2001 to 2009, he was an Assistant Professor (Akademischer Oberrat) in the Research Group System and Circuit Technology, Heinz Nixdorf Institute, University of Paderborn. Currently, he is an Acting Professor of the System and Circuit Technology Group. His research is focused on the development and analysis of on-chip multiprocessor systems, dynamically reconfigurable microelectronic systems, and resource-efficient architectures for network components.

Ulrich Rückert (M'90) received the Diploma degree in computer science and the Dr.-Ing. degree in electrical engineering from the University of Dortmund, Dortmund, Germany, in 1984 and 1989, respectively.

From 1985 to 1992, he worked on microelectronic implementation of neural networks at the Faculty of Electrical Engineering, University of Dortmund. From 1993 to 1994, he was a Professor with the Technical University of Hamburg-Harburg, Germany, heading a research group on Microelectronics. In 1995, he joined the Heinz Nixdorf Institute, University of Paderborn, Paderborn, Germany. As a Full Professor, he held the chair in System and Circuit Technology. In 2001, he was appointed Adjunct Professor with the Department of Information Technology, Queensland University of Technology, Brisbane, Australia. Since 2009, he has been a Professor with Bielefeld University, Bielefeld, Germany. His research group Cognitronics and Sensor Systems is a member of the Cognitive Interaction Technology Cluster of Excellence. His main research interests are bio-inspired architectures for nanotechnologies and cognitive robotics.