

Placing Streaming Applications with Similarities on Dynamically Partially Reconfigurable Architectures

J. Angermeier, S. Wildermann, E. Sibirko, J. Teich

Hardware/Software Co-Design

University of Erlangen-Nuremberg

{angermeier, wildermann, teich}@cs.fau.de

Abstract—By means of partial reconfiguration, parts of the hardware can be dynamically exchanged during operation what allows to adapt the system to changing requirements, and even enables the implementation of self-managing systems. This however requires sophisticated system architectures as well as proper algorithmic runtime support.

In this paper, we present an algorithm for placing streaming applications at runtime. The approach considers the heterogeneity of common FPGAs, such as Block-RAMs, as well as the routing restrictions of on-chip streaming interconnections. To reduce reconfiguration time, we extend the data flow graphs by OR-nodes to describe differing parts of applications while keeping their similarities. This allows us to model systems which only reconfigure differing parts when switching between applications. The proposed algorithm is implemented as runtime support on an FPGA-based system-on-chip.

I. INTRODUCTION

Systems-on-chip (SoCs) are powerful embedded systems with a large field of applications, e.g., in signal and image processing, robotics, and multimedia applications. When used in dynamic environments, it is however necessary that such SoCs are easily adaptable to new requirements. Even the use of self-organizing mechanisms is conceivable. Here, one precondition is the reconfigurability of the computing platform. Partial reconfiguration of FPGAs allows to dynamically stop and change parts of the hardware during operation. This however requires (a) system architectures which support partial reconfiguration, (b) proper reconfiguration managers which provide algorithms for scheduling and placing applications at runtime.

This paper deals with modeling and dynamically placing streaming applications on FPGA-based platforms. The focus lies on systems where it is not predictable at design time which applications actually run and in which order they are executed. This is especially the case when implementing autonomic SoCs. For example, [1] introduces a methodology for self-organizing smart cameras which switch between configurations depending on runtime decisions. In this self-managing context, placement needs to be flexible since it is not known beforehand when and between what configurations reconfiguration happens.

A further challenge of reconfigurable computing is that the overall system performance degrades due to the additional time and power needed for partial reconfiguration as well as the sequential reconfiguration process. A solution here is to

reuse common hardware parts and only replace the differences when switching between hardware configurations.

There exist several online placement algorithms for FPGAs, e.g., [2], [3], [4]. In this paper, we present a novel algorithm for placing streaming applications. The approach considers the heterogeneity of common FPGAs, such as Block-RAM columns, as well as the routing restrictions of on-chip streaming bars. To reduce reconfiguration time, we extended the data flow graph with OR-nodes to describe similarities of multiple applications which allows us to only reconfigure differing parts of applications.

The paper is organized as follows. Section II presents related work. Section III provides the preliminaries of this work. The problem of placing streaming applications is formally defined in Section IV. Section V provides the placing algorithm. An implementation of the algorithm on an FPGA platform is described in Section VI before a conclusion is given in Section VII.

II. RELATED WORK

Partial run-time reconfiguration of FPGAs has been investigated since several years. Still, reconfiguration harbors many problems. There are of course architectural issues concerning how to support partial reconfiguration and how to establish system-wide communication. Solutions can be classified into three basic principles: *On-chip busses* are the most common way of linking together communicating modules. For example, [5] and [6] present on-chip busses for flexible module placement. Hard macros implement the communication infrastructure within the FPGA fabric. *Circuit switching* is a technique where physically wired links are established between two or more modules as, e.g., presented in [7] and [8]. [8] has a low overhead since it implements the wires directly within the routing fabric of the FPGA. *Packet switching* techniques, e.g., networks-on-chip [9], cannot be implemented without a major overhead on FPGAs and will not be further considered in this paper.

Hardware reuse is basically investigated on the logic and the system level. [10] provides a mechanism for hardware reuse on the logic level. Frames in common with the previously placed configuration are determined which will then be reused. A similar approach is followed in [11] which aims at increasing the number of common frames by properly ordering the LUT inputs to keep their content

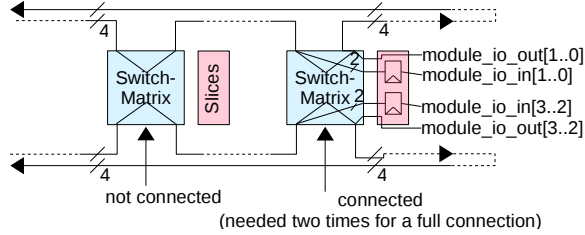


Figure 1. Streaming bar implemented through the FPGA routing fabric.

same for several configurations. Such approaches are highly dependent on the architecture and computationally intensive.

System level approaches are often performed via graph matching techniques. E.g., [12] presents an approach where common components between successive configurations are determined to minimize reconfiguration time. [13] presents an approach which is based on the *reconfiguration state graph* (RSG) which models the configurations and the reconfiguration between them. These approaches are applied during synthesis and require knowledge of the switching order between configurations. The result is the synthesized system. Hence, these methods are not applicable when extending a running system without having to perform a complete re-design, also affecting flexibility as well as maintainability and limiting the use of self-managing mechanisms.

III. PRELIMINARIES

Reconfigurable architectures are typically divided into a static and a dynamic part. The static part consists of pre-configured components, e.g., CPU, peripherals, and memory controllers. The dynamic part is designed for loading hardware modules at runtime. In this part, the fundamental task is to provide an infrastructure for system-wide communication.

In the following, the communication interfaces of the reconfigurable part, which are targeted in this work, are presented. Then, the architecture and the application models are introduced.

A. Architecture

Section II describes approaches to provide communication interfaces for dynamic placement of hardware modules. On-chip busses are suitable for dynamically integrating hardware modules into an FPGA by partial reconfiguration. Sophisticated techniques [5], [6] are provided as FPGA macros which allow partial reconfiguration at run time, even during bus transactions. Modules can be placed in *resource slot*, the smallest atomic piece of FPGA area that can be allocated by a hardware module while a module may occupy multiple resource slots.

Circuit switching techniques are used for streaming data and establishing point-to-point connections between hardware modules. Sophisticated approaches, e.g., *streaming bars* [8], allow to implement this behavior with a low overhead within the FPGA routing fabric. The basic principle is depicted in Fig. 1. In each resource slot, a module is able

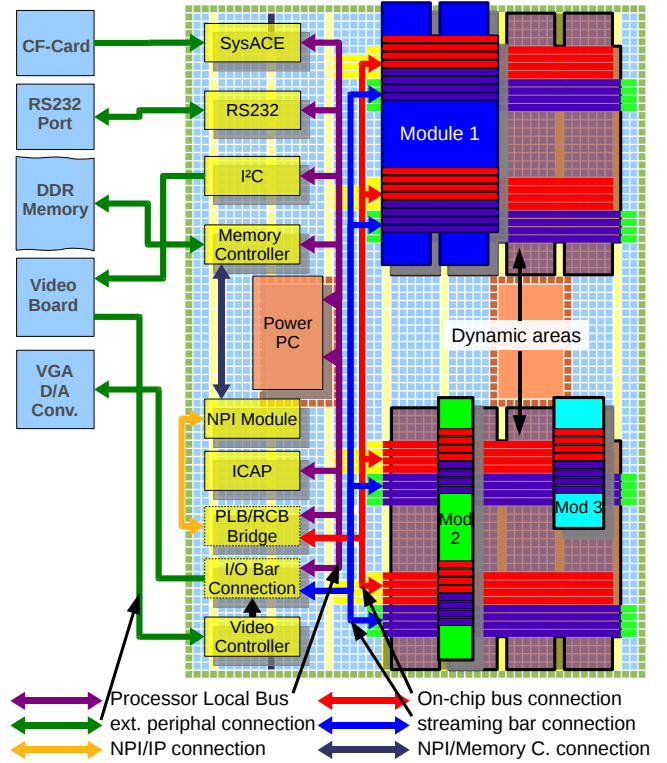


Figure 2. System overview of one possible partitioning of a heterogeneous FPGA-based SoC platform consisting of CPU sub-system and reconfigurable area from [14]. Reconfigurable modules can vary in size and be freely placed, allowing a very good exploitation of the FPGA space. The on-chip ReCoBus (red) and the streaming bar (blue) are routed through the dynamic part, allowing a system-wide communication between hardware and software modules.

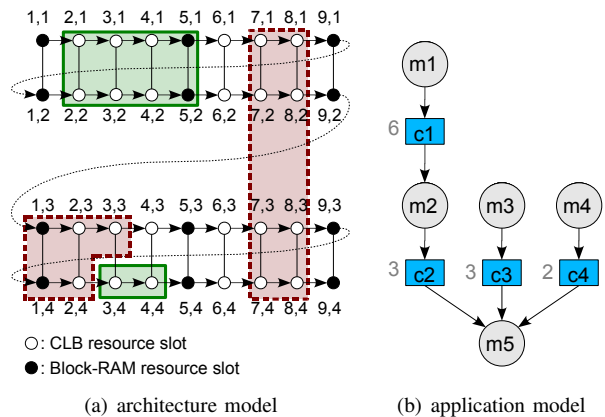


Figure 3. Example of (a) an architecture graph also containing two feasible allocation regions (green solid lines) and two infeasible regions (red dashed lines), as well as (b) a data flow graph describing a streaming application.

to read the incoming data, modify it if necessary, and pass it further to the next module. By properly configuring the switch matrix, modules can be connected to these signal by accessing the data via the flip-flops of the Configurable Logic Block (CLB) slices. Hence, modules occupy signals

of the streaming bar to send data to successive modules.

Fig. 2 illustrates an FPGA-based system-on-chip which serves as a case study. The two dark-red areas on the right top and bottom compose the dynamic part of the system. Reconfiguration is only possible in the dynamic part which contains a reconfigurable on-chip bus and streaming bars as communication primitives. Both primitives are generated by the framework ReCoBus-Builder¹ [15]. Note that the partial modules possess different sizes and that modules can be placed in a two-dimensional manner since four macros are placed in the partial part.

B. Architecture Model

The proposed architecture is formally defined as an architecture graph $G_A(V_A, E_A)$ where vertices $v_{x,y} \in V_A$ are organized in a two-dimensional $w \times h$ mesh with $1 \leq x \leq w$ and $1 \leq y \leq h$. FPGAs have typically a heterogeneous structure and consist of Configurable Logic Blocks (CLBs) and Block-RAMs. The architecture graph therefore provides vertices $V_A = L \cup B$ where a node $v_l \in L$ indicates a resource slot consisting of CLBs, and a node $v_b \in B$ indicates a resource slot consisting of Block-RAMs. The set of edges is given as $E_A = E_D \cup E_U$. Edges $e \in E_D$ indicate a directed connection of two successive resource slots via a streaming bar connection. Edges $e \in E_U$ are undirected and denote that the two connected, adjacent resource slots can be merged to implement a hardware module which requires multiple slots. A module can be placed in any *feasible allocation region* which is defined as follows.

Definition 1: A *feasible allocation region* (FAR) is a set of nodes $V_{FAR} \subset V_A$ which fulfill (a) the vertices $v \in V_{FAR}$ form a rectangular shape, and (b) two adjacent vertices $v, v' \in V_{FAR}$ are connected by an edge $(v, v') \in E_D$.

An example of an architecture graph is given in Fig. 3(a). The system consists of four macros, comparable to the case-study in Fig. 2. So, the nodes are arranged in four rows. Each row is connected by directed edges in E_D , since the streaming bar is routed horizontally. The end of each streaming bar macro is connected to the row below, technically implemented via multiplexers (cf. [14]). Superimposed nodes of the first and the second as well as the third and fourth row are connected by undirected edges in E_U , respectively. In the example, resource slots of the second and the third row cannot be connected. Fig. 3(a) also depicts examples of feasible and infeasible allocation regions within this architecture.

C. Application Model

A streaming application is defined by a directed bipartite data flow graph $G_T(V_T, E_T)$. Vertices $V_T = M \cup C$ represent processing modules M and communication nodes C . Edges E_T represent data dependencies between them. An example is given in Fig. 3(b). Each module $m_i \in M$ represents a functional IP core which can be dynamically loaded

Application

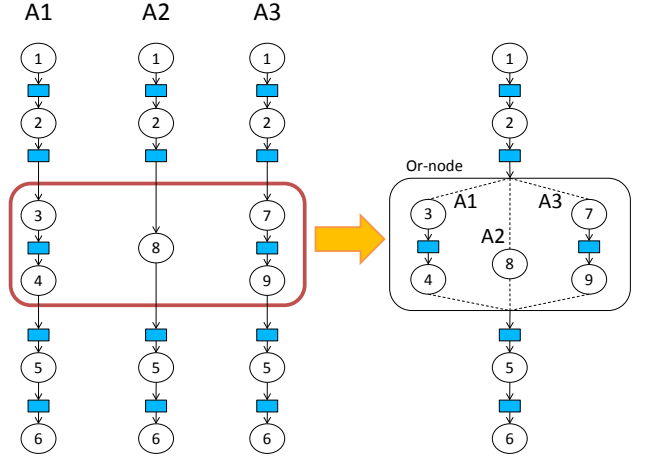


Figure 4. Multiple task graphs with strong similarities can be merged by introducing an OR-node.

and is described by a rectangular bounding box denoted by the parameters $height(m_i)$ and $width(m_i)$. The parameter $typ(m_i) \in \{Master, Slave\}$ states whether the module serves as a slave module or a master module. The latter is able to initialize bus requests. Modules may also have Block-RAM utilization. The corresponding requirements can be represented as a string to apply string matching techniques find FARs to hold the modules during placing.

C represents the set of communication nodes. Each node $c \in C$ has exactly one predecessor module, but may have several successors modules. This paper concentrates on communication of streaming applications established via the streaming bar. Nonetheless, on-chip busses influence the placement: They have a limited number of signals which are available for one macro. Thus, only a restricted number of master modules, which have to be connected to a bus arbiter, can be placed per macro.

As explained in III-A, the streaming bar provides a number of signals which can be accessed by modules. Since the signals are streamed over the architecture, this communication affects the placement. The parameter $com(c)$ denotes the module's bandwidth requirements as the number of required signals to establish the communication c between the predecessor module $pred(c)$ and its successor modules $succ(c)$, occupying this number of signals until they are consumed by the last successor module.

D. Or-nodes

When the data flow graphs of multiple applications have strong similarities, an OR-node may be used as a super-node to merge the graphs of these applications into a single data flow graph. This transformation is illustrated in Figure 4: The module nodes of the data flow graphs are annotated with the ID of their respective IP core. The differences of the task graphs of three applications are merged into

¹ReCoBus homepage: <http://www.recobus.de/>

a single OR-node with the alternatives $\{3, 4\}$, $\{8\}$, and $\{7, 9\}$. The edges connecting the in-port of the OR-node with the alternatives are annotated with the identifier of the application the alternative belongs to.

IV. PROBLEM FORMULATION

Placing tasks on a reconfigurable device is a complex problem, in which many resource constraints must be respected. In our scenario, one important limited resource consists in the communication infrastructure, e.g., the streaming bar. Typically, the amount of used connection bandwidth at each instant in time may not exceed a certain maximum number. Therefore, it may be necessary to delay the placement of one module until some more communication bandwidth is released. Furthermore, the data-dependencies must be respected, i.e., all the temporary results on which one task depends must have been already computed before it can be executed. In addition, each module may require Block-RAM resources at specific relative positions, thus the Block-RAM resource limitations must also be respected for the task placement. In a more formal manner, the placement problem can be specified as follows:

Given a task graph $G_T(V_T, E_T)$ and an architecture graph $G_A(V_A, E_A)$, find for each module $m_i \in V_T$ with outgoing communication nodes $C_i^{out} = succ(m_i)$ and incoming communication nodes $C_i^{in} = pred(m_i)$ an FAR in G_A , such that the following conditions are fulfilled:

- $height(FAR) = height(m_i)$.
- $width(FAR) = width(m_i)$.
- $ram(FAR) = ram(m_i)$.
- $\forall c \in C_i^{in} : (x(m_i) > x(pred(c)) \ \&\& \ y(m_i) == y(pred(c)) \ || \ y(m_i) > y(pred(c)))$, where x and y correspond to the coordinates of the FAR along the edges E_D and where $pred(c)$ refers to the sender of respective communication node c .
- $Master(row_y) \leq MaxMasterNbr(row_y)$, where y refers to the y position of the FAR. This takes into account the limited number of master modules allowed per macro.
- The number of required output lines $com_{out,i}$ may not exceed the number of freely available communication bandwidth, with

$$com_{out,i} = \sum_{c \in C_i^{out}} com(c). \quad (1)$$

V. ALGORITHM

The online placement is given as Algorithm 1. In each iteration, a module candidate is determined out of the list of modules to be placed. Therefore, it is checked for each module if all its predecessors are already placed. In the case that this is fulfilled, the communication bandwidth requirements, e.g., streaming bar signals, are checked. Either, there must be enough free bandwidth available, or bandwidth claimed by some predecessor can be freed, because it is no longer necessary as the data end points have already

Algorithm 1 : Placement Algorithm

Initialize list of rectangles, communication bandwidth usage;

for $i = 1$ to $|M|$ **do**

Determine set of modules for which data-dependencies are met, and still unplaced, named K_i ;

Remove modules from K_i , for which currently not enough communication bandwidth is available;

Remove modules from K_i , for which no suitable FARs can be found in G_A with a first-fit heuristic.

If more than one module is still included in K_i , then select one according to criteria such as number of descendants, module size, Block-RAM usage, etc.

Place K_i in the determined FAR and update resource usage informations;

end for

been placed. If there are multiple modules which all fulfill these requirements, then a priority based approach is used to determine the selection of the module. Usually modules with more descendants in their data flow graph receive a higher priority. Furthermore, the priorities can also depend on user defined objectives or the underlying architecture. The priorities are dynamic, meaning that they are determined new in each iteration.

An important step of the algorithm is to determine the possible FARs. Hereby, the candidate rectangles are dependent on the underlying architecture graph. As described in Section III-B, heterogeneities on the reconfigurable device partition the reconfigurable area. Based on that, an initial list of possible rectangles is created. Hereby, rectangles can have different heights, measured in resource slots, and are sorted in the order of their appearance in the architecture graph. A rectangle is determined in the lists by the *First-Fit* rule, i.e., the first rectangle found in which the module can be placed. Thus, modules are tried to be placed near the origin at coordinate $(0, 0)$ to reduce fragmentation. Data dependencies are respected by first checking for each module if all predecing modules are already placed. Special care must be taken for modules with Block-RAM requirements which are represented with a string and are checked by a string matching approach.

In the following, the proceeding of the algorithm is explained by an example. Given a data flow graph and an architecture graph as shown in Fig. 3. Assume all modules have height and width of one unit, and have no Block-RAM requirements. Each communication node in the application graph of Fig. 3 is annotated with the required bandwidth. The number of provided signals of the streaming bar has a value of 8. The algorithm first initializes the available

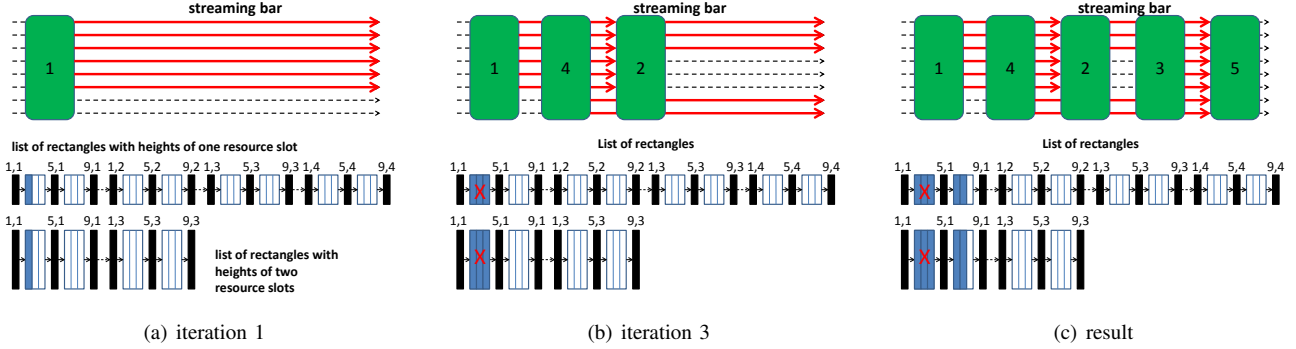


Figure 5. Example of the placement algorithm.

resources, creates a list of available rectangles, and sets back all communication bandwidth allocations.

The algorithm places one task per iteration, and thus requires 5 iterations to completely place the application. In the first iteration, only the modules $\{1, 3, 4\}$ have no unplaced predecessors, and there is also sufficient communication bandwidth available for them. Module 1 gets a higher priority, because it has two descendants while both other modules have only one descendent. Thus, 1 is selected for placement, and an FAR is determined for the candidate in the first CLB element at coordinate $(2, 1)$. The corresponding reconfigurable area is marked as unavailable and 6 communication lines are marked as used. The resulting state is illustrated in Fig. 5(a).

In the second iteration, the modules $\{2, 3, 4\}$ may be placed according to their data dependencies. Only two communication lines are freely available, however module 3 is in need of three lines, so it can't be placed. Module 2 is also in need of three communication channels, however it is also the communication end-point of module 1 such that these six lines can be freed, and enough bandwidth is available. Module 4 requires only two communication channels, and thus fulfills also the bandwidth conditions. In this case-study, module 4 is given a higher priority due to its fewer communication bandwidth requirements.

In the next iteration, the modules $\{2, 3\}$ fulfill the data dependency constraints. While there is sufficient communication bandwidth available for module 2, there isn't enough for module 3. Thus, module 2 holds a communication bandwidth of 2, and module 4 holds a bandwidth of 2. The resulting state is illustrated in Fig. 5(b).

In the fourth iteration, module 3 is the only candidate, as module 5 depends on it. There is enough communication bandwidth available, and also an FAR can be found which offers sufficient area for the module to be placed.

In the final iteration, module 5 is placed. The result of the algorithm can be seen in Fig. 5(c).

When placing a task graph with an OR-node, special care must be taken: For each module included in the OR-node hierarchy, an area must be determined which provides sufficient space such that each of these modules may be

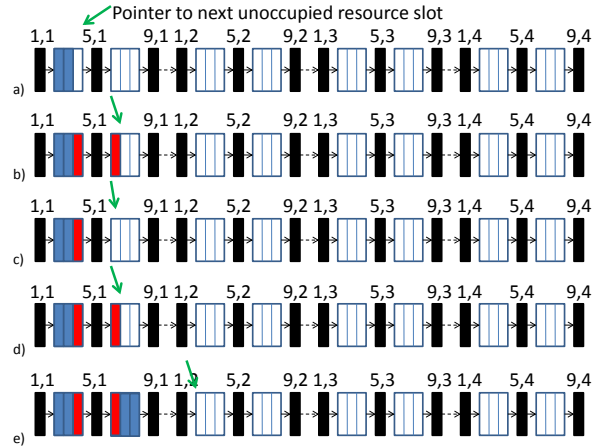


Figure 6. Example for placing the merged data flow graphs from Figure 4 containing an OR-node.

placed there exclusively, and all the Block-RAM and communication bandwidth constraints can be met altogether. This is done by determining the placement for each module individually and reserving the maximal bounding box. An example placement of an OR-node is illustrated in Figure 6. The figure shows a list of rectangles used for the placement of the applications from Fig. 4, in 6a) after the placement of the modules $\{1, 2\}$, in 6b) for $\{1, 2, 3, 4\}$ in 6c) for $\{1, 2, 8\}$, and in 6d) for $\{1, 2, 7, 9\}$. Thus, the maximal region ranging from coordinate $(4, 1)$ to $(6, 1)$ is reserved for the complete OR-node. Afterwards, the tasks $\{5, 6\}$ are placed, as can be seen in part 6e).

VI. IMPLEMENTATION

The proposed algorithm was implemented on a Xilinx Vitex-II XUP Pro board by extending the SoC in [14]. The procedure of runtime reconfiguration is illustrated in Fig. 7. It is triggered by the PowerPC. The image of the static part is loaded from a CF card into the memory (see steps 1 to 4 in Fig. 7). This is necessary, since for each partial reconfiguration also logic from the static part has to be rewritten. The CF card also contains the data flow descriptions

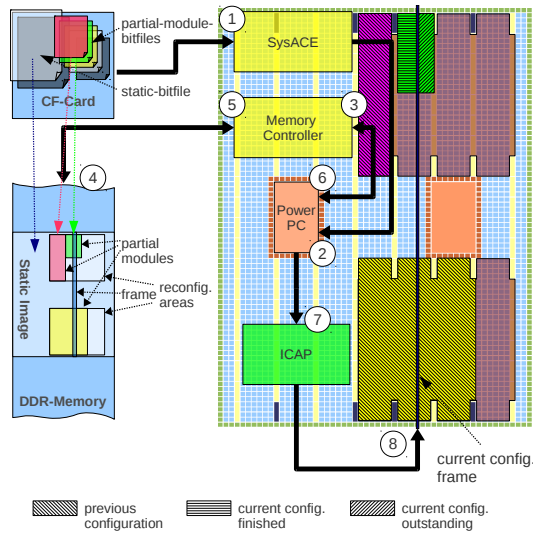


Figure 7. Reconfiguration procedure.

which are provided in XML format, as well as the images of the partial modules.

When loading a new application, the XML description is first loaded to initialize the placement algorithm. Then, candidate modules are selected and positions determined as described in the last section. When placing a module, its image is loaded into the memory (via steps 1, 2, 3, 4). Then, it is combined with the corresponding parts of the static image and other partial modules which are already configured, to generate the frames for reconfiguration. These are then loaded via steps 5 to 8 illustrated in Fig. 7.

VII. CONCLUSION

Partial reconfiguration of FPGAs allows to dynamically change the configuration of a hardware system at runtime. In this case, it can be switched between applications consisting of similar tasks without having to reconfigure the whole system, but only the differing parts. This paper deals with the placement of streaming applications by respecting the bandwidth constraints of the communicating tasks. In order to be applied in real world, the placement approach also takes into account the heterogeneities of the reconfigurable device, such as e.g., Block-RAMs and CPUs. This paper focuses especially on the modelling and placement of multiple applications with strong similarities in their data flow graphs. We introduce OR-nodes into the graphs to differentiate between differing parts of applications with several similarities. A new placement algorithm is presented for the placement of the applications on dynamically reconfigurable architectures which allows fast switching between them. It respects the constraints for the task communication imposed by the limited bandwidth of reconfigurable communication systems, as well as the placement constraints due to heterogeneities of the reconfigurable area. An implementation in

an FPGA-based SoC demonstrates the applicability of the proposed placement approach.

REFERENCES

- [1] S. Wildermann, A. Oetken, J. Teich, and Z. Salcic, "Self-organizing computer vision for robust object tracking in smart cameras," in *ATC*, to appear, ser. LNCS. Springer-Verlag, 2010.
- [2] K. Bazargan, R. Kastner, and M. Sarrafzadeh, "Fast template placement for reconfigurable computing systems," in *IEEE Design and Test of Computers*, 2000, pp. 68–83.
- [3] Y. Lu, T. Marconi, G. Gaydadjiev, and K. Bertels, "An efficient algorithm for free resources management on the FPGA," in *DATE*, 2008, pp. 1095–1098.
- [4] T.-Y. Lee, C.-C. Hu, and C.-C. Tsai, "Adaptive free space management of online placement for reconfigurable systems," in *IMECS*, 2010, pp. 322–326.
- [5] J. Hagemeyer, B. Kettelhoit, M. Koester, and M. Pormann, "Design of homogeneous communication infrastructures for partially reconfigurable FPGAs," in *ERSA*, 2007, pp. 238–247.
- [6] D. Koch, C. Haubelt, and J. Teich, "Efficient reconfigurable on-chip buses for FPGAs," in *FCCM*, April 2008, pp. 287–290.
- [7] H. ElGindy, H. Schroder, A. Spray, A. Somani, and H. Schmeck, "RMB – A reconfigurable multiple bus network," in *HPCA*, 3-7 1996, pp. 108–117.
- [8] D. Koch, C. Beckhoff, and J. Teich, "ReCoBus-Builder – A novel tool and technique to build statically and dynamically reconfigurable systems for FPGAs," in *FPL*, Sept. 2008, pp. 119–124.
- [9] C. Bobda, A. Ahmadiania, M. Majer, J. Teich, S. Fekete, and J. van der Veen, "DyNoC: A dynamic infrastructure for communication in dynamically reconfigurable devices," in *FPL*, 24-26 2005, pp. 153–158.
- [10] U. Malik and O. Diessel, "On the placement and granularity of FPGA configurations," in *FPT*, 6-8 2004, pp. 161–168.
- [11] K. Prasad Raghuraman, H. Wang, and S. Tragoudas, "A novel approach to minimizing reconfiguration cost for LUT-based FPGAs," in *VLSI Design*, 3-7 2005, pp. 673–676.
- [12] N. Shirazi, W. Luk, and P. Y. K. Cheung, "Automating production of run-time reconfigurable designs," in *FCCM*. Washington, DC, USA: IEEE Computer Society, 1998, p. 147.
- [13] M. Rullmann and R. Merker, "Design methods and tools for improved partial dynamic reconfiguration," in *Dynamically Reconfigurable Systems - Architectures, Design Methods and Applications*. Springer, Berlin, Mar 2010, pp. 147–163.
- [14] A. Oetken, S. Wildermann, J. Teich, and D. Koch, "A bus-based SoC architecture for flexible module placement on reconfigurable FPGAs," in *FPL*, to appear, Aug. 2010.
- [15] D. Koch, C. Beckhoff, and J. Teich, "A communication architecture for complex runtime reconfigurable systems and its implementation on Spartan-3 FPGAs," in *FPGA*, Feb. 2009, pp. 233–236.