

A System-Level Approach to Hardware Reconfigurable Systems*

Christian Haubelt, Stephan Otto, Cornelia Grabbe, and Jürgen Teich
 Hardware-Software-Co-Design, Department of Computer Science 12
 University of Erlangen-Nuremberg, D-91058 Erlangen, Germany
 {haubelt, grabbe, teich}@cs.fau.de

Abstract— There is trend towards networked and distributed hardware reconfigurable systems, complicating the design process at the system-level. This paper will provide a solution to the problem of design space exploration for such embedded systems of the next generation. We will show the problems occurring while exploring the design space at the system-level, leading to new properties for valid implementations. The novelty of this approach lies in the support of *explicit communication modeling* and *time-multiplexed architecture modeling* in a single model. The proposed design space exploration is based on Evolutionary Algorithms and a new slack-based list scheduler.

I. INTRODUCTION AND RELATED WORK

Today, typical target architectures for embedded systems are usually heterogeneous in both its processing elements and its communication channels. A lot of research has been done in the area of design space exploration in the last decade. The main differences in all approaches is the underlying optimization strategy and the architecture template. Only a few of them consider explicit communication, i.e., data dependencies given by the input specification must be modeled by connections in the underlying architecture. All other approaches assume implicit communication working with unconnected resources and insert communication channels on demand. However, at the system-level complex communication is usually modeled by processes as well. Hence, there is a need to support explicit communication during design space exploration. Due to their ability to explore several implementations in parallel, Evolutionary Algorithms as proposed in [3, 6, 8] are most promising to solve the task of design space exploration.

Approaches with explicit communication usually work on abstract graph-based models as shown in Figure 1(a) (c.f., [3]). The process graph g_p on the left hand side models the desired functionality, whereas the architecture graph g_a on the right restricts the architecture. Both graphs are related by so-called *mapping edges* modeling the binding possibilities of the processes. The example shown in Figure 1(a) is already an implementation, since no design decisions like resource or mapping selection are left for the designer. Hence, the implementation can be represented by a so-called *Gantt chart* (Figure 1(b)).

Target architectures often include reconfigurable hardware, e.g., FPGAs. Exactly one configuration can be stored in a single FPGA at each instant of time, whereas configurations can

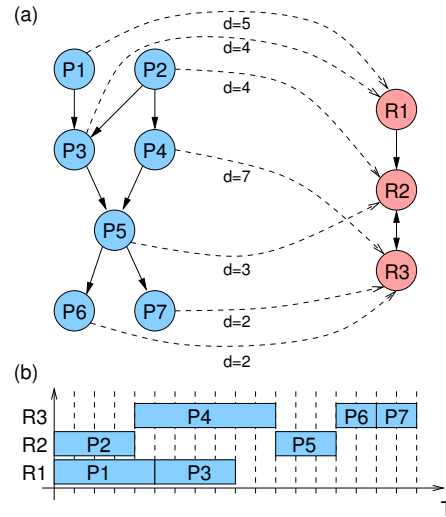


Fig. 1. (a) Model of the binding and allocation for a process graph, an architecture, and mapping edges. (b) A schedule for the binding and allocation given in (a).

change over time. Thus, configurations can be seen as *time-multiplexed* or *mutually exclusive resources*. By using hardware reconfigurable resources, a new problem known as *temporal partitioning* has to be solved [4]. This problem can be formulated as a combined partitioning and scheduling problem. The goal is the partitioning of operations into different configurations while preserving their partial order imposed by data dependencies over configuration boundaries. Approaches to solve the temporal partitioning problem are based on list scheduling, Integer Linear Programs, or network flow approaches. All these methods have in common that they assume dedicated fine grain resources. Our approach is somehow different in the way that configurations are composed of coarse grain system-level components like CPUs, busses, IP cores or again FPGAs. Furthermore, our model is capable in modeling the whole system not only a single FPGA.

As networked and distributed reconfigurable hardware platforms [6, 9] are becoming more and more important for applications in different areas of automotive, body area networks, etc., there is a need to support the design of these systems also at the system-level. First examples of coarse grain, networked reconfigurable architectures are PACT [1] and Chameleon [5] and the ReCoNet architecture [7]. Most of the system-level design space exploration tools proposed in the literature are not able to deal with mutually exclusive resources. The CORDS system is an approach to the hardware-software partitioning problem based on Evolutionary Algorithms [6]. It considers

*Supported in part by the German Science Foundation (DFG), SPP 1148 (Rekonfigurierbare Rechensysteme)

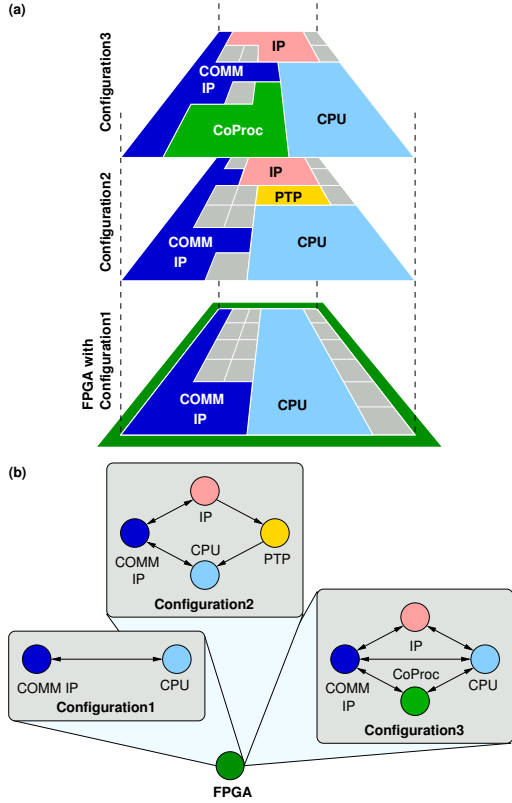


Fig. 2. (a) Three different configurations can be stored in an FPGA. (b) The corresponding graph based model of the FPGA consists of the FPGA itself and the three configurations represented as subgraphs. The subgraphs consist of the different components of the configurations.

hardware reconfigurable resources and reconfiguration times and assumes buffers for the communication implying implicit communication. The scheduling is performed with a preemptive static critical path scheduling. This approach is somehow similar to our approach but does not consider an appropriate communication model as necessary at the system-level.

To the best of our knowledge, there is no design space exploration tool which supports both, *explicit communication* and *reconfigurable hardware*. In this paper, we will close this gap by presenting such a tool based on Evolutionary Algorithms which considers both aspects. It strictly separates functionality from the architecture. In particular, it allows configurations to be composed of system-level components. We will see, that testing the feasibility of an implementation has to consider timed-multiplexed resources in a special way. In the next section, we will first discuss the problem of hardware-software partitioning for system-level applications with mutually exclusive resources. Next, Section III will focus on the feasibility of solutions found by our design space exploration tool. In Section IV, the design space exploration tool will be briefly presented. Section V will provide some conclusions.

II. PROBLEM FORMULATION

In this paper, we focus on a graph-based approach to the hardware-software partitioning problem while considering hardware reconfigurable components and explicit communication. The problem is to map a process graph onto components

in a way that data dependencies given by the process graph can be handled in the resulting implementation. But first, we will introduce a graph-based model used throughout this paper.

Definition 1 (Process Graph) A process graph $g_p = (V_p, E_p)$ consists of a finite set of vertices V_p and a finite set of directed edges $E_p \subseteq V_p \times V_p$ where vertices model processes and edges represent communication links between processes.

In this paper, we will assume the process graph to be a homogeneous dataflow model, i.e., a process can only be executed if all its predecessors finished their computations. In order to model the underlying hardware of a system, we use a so-called *architecture graph* g_a .

Definition 2 (Architecture Graph) An architecture graph $g_a = (V_a, E_a)$ consists of a finite set of vertices V_a and a finite set of directed edges $E_a \subseteq V_a \times V_a$. The vertices $v \in V_a$ of the architecture graph g_a correspond to hardware modules like CPUs, memories, FPGAs, etc. The edges $e \in E_a \subseteq V_a \times V_a$ of the architecture graph g_a correspond to connections between hardware modules modeling the topology of the architecture. Furthermore, vertices $v \in V_a$ in the architecture graph may be refined by subgraphs $\tilde{g} \in v.G$ associated with v .

In the context of reconfigurable hardware, one can see that an FPGA may be configured with different architectures, so-called *configurations*. If a set of subgraphs $G' \subseteq v.G$ is selected as a refinement of a vertex v , we are able to flatten our model. All subgraphs associated with the same vertex are mutually exclusive, i.e., at most one subgraph may be activated at each instant of time. Thus, a vertex $v \in V_a$ representing a hardware reconfigurable node may be refined by a single subgraph $g \in v.G$ modeling the selected configuration.

The basic idea of a hierarchical architecture graph is shown in Figure 2. Figure 2(a) shows an FPGA and three associated configurations. The first configuration is composed of a CPU and a communication IP whereas the second configuration uses an additional IP core. The corresponding architecture graph is shown in Figure 2(b). It consists of a single vertex modeling the FPGA and three associated subgraphs representing the configurations. Finally, mapping edges are used to restrict the search space. A mapping edge $m = (v_p, v_a) \in E_m$ models the fact that the process associated with vertex v_p can be implemented on the resource associated with vertex v_a . Furthermore, some property values, like delays, power consumption, area, etc. may be associated with the mapping edges or the resources in the architecture graph. These properties are needed for optimization purposes.

Now, the hardware-software partitioning problem can be formulated as follows: Given an acyclic process graph g_p , a hierarchical architecture graph g_a , and mapping edges E_m , find an allocation of resources and configurations as well as a corresponding feasible binding of processes to resources (c.f. Figure 3). In contrast to Figure 1, resources might be refineable by configurations. The feasibility of an allocation and a binding will be discussed in the next section.

A specification as shown in Figure 3(a) usually contains several mapping edges per process, describing possible alternative bindings of this process. Moreover, the model is parameterized in a sense that attributes are associated with resources and

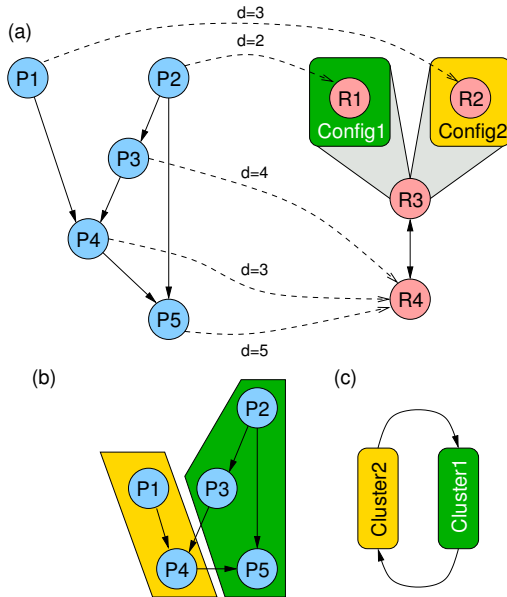


Fig. 3. (a) A process graph is mapped onto a hardware reconfigurable architecture. Resource R3 can be refined by two different configurations. (b) Corresponding cluster graph with (c) cyclic data dependencies.

mapping edges. All these attributes are used to calculate the objective values in the later design space exploration. They are usually estimated values obtained by running some synthesis or compiler tool. In the example shown in Figure 3(a) only execution times are annotated to the mapping edges.

III. FINDING FEASIBLE SOLUTIONS

In this section, we will focus on the feasibility of an allocation and a binding. As proposed by Blickle et al. [3], a feasible binding maps each process to exactly one allocated resource and communicating processes are mapped onto the same resource or adjacent resources. That way it is guaranteed that there is a communication channel for establishing the required data exchanges between communicating processes. An allocation is said to be feasible if it allows a feasible binding.

This problem is much harder when having mutually exclusive resources. A dataflow actor can fire iff all its inputs are available. So, placing predecessors in mutually exclusive configurations may invalidate a binding, since not all input data can be available at the same time. For instance, binding process P3 onto resource R1 in Figure 3(a), prohibits the execution of process P4 because process P1 and P3 cannot exist in parallel due to the mutual exclusiveness of the two configurations. Beside this simple infeasibility which can be detected locally, there is another problem regarding the schedulability. The problem comes into view when we try to schedule the process graph. A first attempt starts with process P1. By doing this, process P2 cannot be scheduled due to the mutual exclusiveness of the configurations. Hence, we cannot schedule processes P3. On the other hand, if we start with process P2, we are able to schedule process P3. But now we detect a deadlock due to process P4 which needs the input data from P1. By scheduling P1, these input data can be provided but at the same time the required input data of process P5 are deleted.

The infeasibility of such implementations can be detected formally: Connected subgraphs mapped onto the same configuration as well as their direct successors are assigned to a cluster associated with the configuration (see Figure 3(b)). With these cluster information, a so-called *cluster graph* g_c for mutually exclusive configurations can be constructed.

Definition 3 (Cluster Graph) A cluster graph $g_c = (V_c, E_c)$ is a directed graph, where vertices $v \in V_c$ are associated with the mutually exclusive configurations. For each connected subgraph mapped onto a configuration and their direct successors there is a vertex $v \in V_c$. Directed edges $e \in E_c$ exist in the cluster graph only if there are data dependencies between processes placed in different clusters.

With this model, we can prove the following theorem:

Theorem 1 If clusters associated with mutually exclusive configurations overlap or have cyclic data dependencies, it is not possible to find a valid schedule.

Proof 1 For the proof, one has to notice that in a dataflow model, an actor $v \in V_p$ can only be executed if all its predecessors $\text{pred}(v)$ have finished execution and the data are available at the outputs of the predecessors. The first part of the proof, corresponds to the case where an actor $v \in V_p$ is associated with both mutually exclusive clusters. Since the actor is mapped at most to one of the associated configurations, at least two predecessor $v_1, v_2 \in \text{pred}(v)$ of v are mapped to mutually exclusive configurations and, thus, cannot exist in parallel. Hence, the actor v can never be executed. The second part of the proof considers the case where a cyclic data dependency exists among two clusters c_1, c_2 . There is at least one actor $v_1 \in c_1$ and $v_2 \in c_2$ in each clusters c_1, c_2 depending on data of an actor \tilde{v}_1, \tilde{v}_2 in the other cluster, i.e., $(\tilde{v}_1, v_1), (\tilde{v}_2, v_2) \in E_p$ with $\tilde{v}_2, v_1 \in c_1$ and $\tilde{v}_1, v_2 \in c_2$. Hence, both clusters are unable to start executing and, as a consequence, none of these clusters will ever finish execution. \square

An example of a feasible implementation is shown in Figure 4(a). The corresponding cluster graph is given in Figure 4(c). Process P2 and its direct successors P3 and P6 are assigned to cluster Cluster1. Process P1 with its direct successor process P4 are assigned to cluster Cluster2. For the sake of completeness process P5 is assigned to an individual cluster. The Gantt chart for the implementation is shown in Figure 4(d). One can see that not only the processes are scheduled but also the associated configurations.

IV. DESIGN SPACE EXPLORATION

Our design space exploration tool MoDES (Model-based Design space exploration for Embedded Systems) is based on Multi-Objective Evolutionary Algorithms (MOEAs). It uses state-of-the-art optimization algorithms provided by the PISA framework [2]. The individuals encode all information necessary to represent an implementation and, in addition, to find new solutions. The allocation of resources and configurations is encoded in a bit string, while the binding of processes to resources is based on priority lists. We implemented a repair mechanism similar to the one proposed in [3], where resources

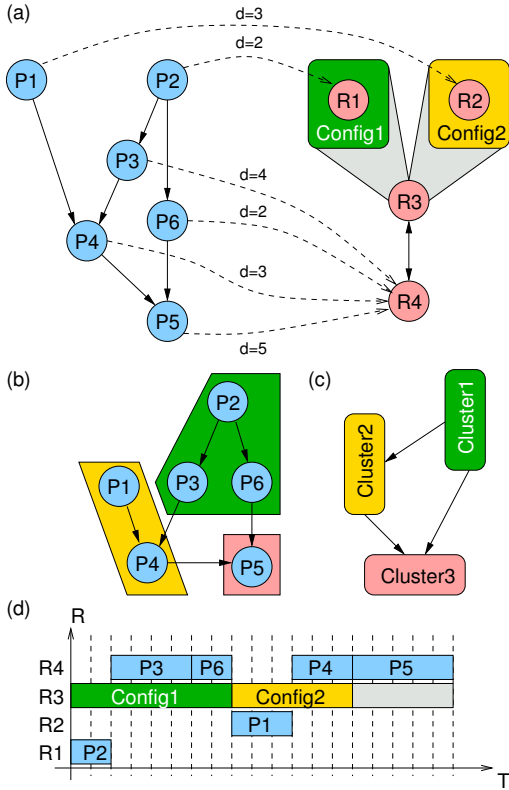


Fig. 4. (a) Feasible implementation including time-multiplexed resources. (b) Construction of the corresponding cluster graph. (c) Corresponding cluster graph. (d) Gantt chart for the implementation shown in (a).

are added on demand following a priority strategy. Implementations which are still infeasible afterwards are punished by infinite objective values. This approach differs from CORDS [6] in the way that we allow infeasible implementations in our exploration. Hence, we omit the computational intensive clustering of implementations with the same target architecture.

MoDES is adaptable in number and kind of objective values, where all objectives have to be derived from parameters given in the specification. In order to support hardware reconfigurable devices, we have implemented the feasibility check discussed in detail in Section III. Furthermore, the scheduling strategy used by MoDES must consider cluster dependencies, in order to compute feasible schedules. The implemented scheduler is a slack-based list scheduler. This scheduler is modified such that it can handle cluster dependencies, i.e., after computing the cluster graph, a schedulable operation bound on a configuration is only scheduled iff all preceding clusters are already scheduled. The scheduling algorithm is depicted in Figure 5. In a first step, the list of schedulable processes is determined. Next, each resource is tested if it is available or not. Afterwards, all processes which do not depend on the finishing of a cluster are stored in the schedule list V_k for resource k . Then, the processes with the highest priority (smallest slack) is selected for scheduling. Of course, this scheduling strategy generally produces suboptimal schedules.

We have tested the MoDES framework on an examples of an JPEG encoder mapped onto reconfigurable hardware. It should be clear that MoDES can be easily extended to consider *reconfiguration times* as proposed in [6].

LIST_SCHEDULER

```

INPUT:  $g_p, \alpha, \beta$ 
OUTPUT:  $\tau$ 
 $numsched = 0; t = 0;$ 
WHILE ( $numsched \leq |V_p|$ ) DO
   $readylist = no\_pred(V_p);$ 
  FORALL ( $k \in \alpha$ ) DO
    IF ( $not\_available(k)$ ) THEN BREAK;
     $V_k = no\_pred\_cluster(readylist, k, \beta);$ 
     $v_k = highest\_priority(V_k);$ 
     $\tau(v_k) = t$ 
  ENDFOR
   $t = t + 1;$ 
ENDWHILE
RETURN  $\tau$ 
END

```

Fig. 5. Slack-based list scheduler which can handle cluster dependencies.

V. CONCLUSIONS

Due to the trend towards networked and distributed hardware reconfigurable systems, new design space exploration methodologies are needed at the system-level. New results presented in this paper include (a) the support of explicit communication modeling and (b) the modeling of time-multiplexed architectures such as necessary to model reconfigurable hardware. It has been shown that (c) new properties for feasible solutions have to be established and tested for such systems during design space exploration. We have shown that constructing the so-called *cluster graph* helps to detect hidden deadlocks which occur from mutually exclusive resources. Based on these results, a slack-based list scheduler for time-multiplexed architectures an explicit communication was proposed.

REFERENCES

- [1] V. Baumgarten, F. May, A. Nüchel, M. Vorbach, and M. Weinhardt. PACT XPP - A Self-Reconfigurable Data Processing Architecture. In *ERSA*, 2001.
- [2] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler. PISA – A Platform and Programming Language Independent Interface for Search Algorithms. In *Proc. of EMO'03*, pages 494–508, 2003.
- [3] T. Blicke, J. Teich, and L. Thiele. System-Level Synthesis Using Evolutionary Algorithms. In R. Gupta, editor, *Design Automation for Embedded Systems*, 3, pages 23–62. Kluwer, Jan. 1998.
- [4] C. Bobda. *Synthesis of Dataflow Graphs for Reconfigurable Systems Using Temporal Partitioning and Temporal Placement*. PhD thesis, University of Paderborn, May 2003.
- [5] Chameleon Systems. *CS2000 Reconfigurable Communications Processor, Family Product Brief*, 2000.
- [6] R. P. Dick and N. K. Jha. CORDS: Hardware-Software Co-Synthesis of Reconfigurable Real-Time Distributed Embedded Systems. In *Proc. of the Int. Conf. on CAD*, pages 62–67, 1998.
- [7] C. Haubelt, D. Koch, and J. Teich. ReCoNet: Modeling and Implementation of Fault Tolerant Distributed Reconfigurable Hardware. In *Proc. of SBCCI'2003*, pages 343–348, São Paulo, Brazil, 2003.
- [8] B. Mei, P. Schaumont, and S. Vernalde. A Hardware-Software Partitioning and Scheduling Algorithm for Dynamically Reconfigurable Embedded Systems. In *Proc. of ProRISC 2000*, 2000.
- [9] I. Ouass, S. Govindarajan, V. Srinivasan, M. Kaul, and R. Vemuri. An Integrated Partitioning and Synthesis System for Dynamically Reconfigurable Multi-FPGA Architectures. In *Proc. of RAW'98*, pp.31–36, 1998.