This is the authors' version of the work. The definitive work was published in *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis (CODES+ISSS '11), Pages 129-138, 2011.*

Symbolic Design Space Exploration for Multi-Mode Reconfigurable Systems

Stefan Wildermann, Felix Reimann,Daniel Ziener, and Jürgen Teich
University of Erlangen-Nuremberg, Germany
{stefan.wildermann, felix.reimann, daniel.ziener,teich}@cs.fau.de

***Abstract*—In today's complex embedded systems not all applications are running all the time, but depend on the operational mode. By incorporating knowledge about the temporal behavior of such *multi-mode* systems, it is possible to share hardware by means of partial reconfiguration, and thus, reduce costs and improve performance. In this paper, we specify the temporal behavior of the functionality by applying known models based on state machines. In addition, we introduce an architectural model that allows to express the characteristics of nowadays partially reconfigurable architectures, focusing on FPGAs. We develop a symbolic encoding of this novel system specification, which allows to perform a unified system synthesis for allocation, binding, placement of partially reconfigurable modules, and routing the on-chip communication. The proposed encoding enables the use of sophisticated optimization techniques, coupling a SAT solver with a Multi-objective Evolutionary Algorithm. The proposed methodology is highly applicable for building multi-mode systems on advanced reconfigurable technology. We demonstrate this by experiments on test-cases from the image processing domain applying state-of-the-art technology. The results show the superiority of the presented approach in terms of run-time and quality of the found solutions compared to existing system synthesis approaches.**

I. Introduction

In the past, embedded systems were mostly designed to execute only one application which results in static implementations. Through the ongoing technological improvements, it is possible to build more complex and also more flexible systems which permit the execution of many applications on the same device. The embedded system is able to change its functionality depending on external events. Here, we are speaking of *multi-mode systems*. They can for example be found in a typical "smart phone" scenario [1] where the operational mode depends, e. g., on the radio link quality or on the interaction with the user. Other examples are "smart cameras" which have internal adaptation mechanisms that switch between appropriate signal processing algorithms, cf. [2], thus effectively changing from one to another operational mode. Multi-mode embedded systems are also deployed in database back-ends [3], where module configurations are dynamically exchanged to accelerate database transactions of different query types. All possible constellations of query types represent the operational modes.

Today, many embedded systems are designed as MPSoCs (Multiprocessor System-on-Chip) consisting of processors, buses, memories, and dedicated hardware accelerator cores. By incorporating knowledge about the temporal behavior of *multi-mode* systems into the system design process, it is possible to utilize the existing processors more efficiently by scheduling the tasks of different modes separately. Moreover, hardware resources may be shared between modules of different modes by means of dynamic partial reconfiguration. This allows to reduce monetary costs and improve performance at the same time.

The usage of modern reconfigurable technologies, such as *field-programmable gate arrays* (FPGAs), enables the design of flexible hardware accelerators. By using FPGAs which support partial reconfiguration, several accelerator modules can be instantiated in one device and independently replaced during run-time without interfering other active modules. This technology allows to switch between operational modes by exchanging so-called *partially reconfigurable modules* (PR modules) at run-time. This type of resource sharing reduces size, costs, and the power consumption of the embedded system. Reconfigurable technology may thus become a key enabler to build this kind of systems.

To be able to use reconfigurable sub-systems efficiently, it is not only necessary to decide which tasks to implement by PR modules. It is moreover necessary to determine in which area to place the task within the region dedicated for partial reconfiguration (PR region). Sophisticated hardware synthesis flows, e.g., [4], [5], [6], support the synthesis of PR modules onto dynamically reconfigurable devices. They allow the designer to select the location and shape of PR modules within these regions of the architecture, and synthesize the system accordingly. A major drawback is that there exist many placement constraints, since modern reconfigurable fabrics have a heterogeneous structure of different cell types. Furthermore, a system-wide communication has to be ensured that also supports partial reconfiguration. The designer therefore has to find efficient feasible placements for all the PR modules by hand, as well as ensure that concurrently running modules do not interfere. At the same time, it has to be guaranteed that feasible routings of the communication between the PR modules and to other static hardware resources in the system exist.

In this paper, we propose an approach to relieve the designer from these tedious tasks. Our method follows the Y-chart approach [7], so that the designer only has to provide a proper model of the multi-mode application, as well as a model of the architecture which we introduce in this paper. Furthermore, a

design space exploration (DSE) methodology is proposed that maps the functionality to the architecture. It takes into account the needs of the different operational modules and performs on-chip routing to satisfy the accordant data dependencies. Due to the multitude of constraints, we transform the problem of finding a feasible solution to a Boolean satisfiability problem (SAT). By combining a SAT solver with a Multi-objective Evolutionary Algorithm (MOEA), the applied DSE technique is able to (1) efficiently find feasible implementations and (2) searching them for high-quality implementations in terms of hardware and energy consumption as well as reconfiguration time. The designer can then choose one implementation from the found Pareto-front that best fits its needs and perform the subsequent synthesis steps.

II. Related Work

Recently, a system-level design methodology for multi-mode heterogeneous systems has been proposed in [8]. The paper proposes to perform an exploration of every mode independently, i. e., without considering the implementations of the other modes. First, the authors perform Simulated Annealing to obtain a set of non-dominated *mode implementations*. In a second step, the overall multi-mode system implementation is built by selecting those mode implementations that optimize a number of given objectives. This approach does not consider on-chip communication and the challenges of partial reconfiguration. It may even be possible that good solutions regarding reconfiguration time are simply discarded when performing the optimization for each operational mode independently. In contrast, [1] proposes a methodology to perform an exploration of the multi-mode system as a whole. The approach also considers the use of field-programmable components. However, the work does not take into account module placement on FPGAs as well as the issue of on-chip communication. [1] provides several key contributions for the design of multi-mode systems, in particular, the description of the system behavior and its modes by means of finite state machines.

All the above approaches have problems finding feasible implementations in the complex search spaces of real-world applications. They either have to drop infeasible solutions [8] or they have to apply repair mechanisms and penalty strategies [1]. Especially in the case of stringent constraints, as given when performing partial reconfiguration, they spent a huge amount of the computation time dealing with infeasible solutions, instead of advancing in the optimization process.

The approach presented in this paper is able to traverse the search space efficiently and find high-quality solutions without needing to neglect the complexity of real-world hardware. To achieve this, the SAT-decoding approach [9] is incorporated which is a heuristic using symbolic representations of the constraints which have to be met by an implementation to be feasible. SAT-decoding has been used for the design space exploration for single-mode systems, thus, we significantly enhance it as our approach will be able to cope with multi-mode systems as well as partial reconfiguration.

A second major enhancement is that our symbolic approach also considers all the details that partially reconfigurable hardware imposes on PR module placement. Several tool flows have been provided that support mapping applications onto reconfigurable architectures. Automated partitioning and synthesis is, for example, enabled by Synopsys' *Certify* or the flow reported in [10]. Their result, however, is a static, single-mode embedded system and, thus, they are not able to exploit the flexibility of dynamically reconfigurable hardware at run-time.

There exist several tool flows to synthesize multi-mode systems that also perform partial run-time reconfiguration, such as Xilinx' *PlanAhead* [4], the *INDRA* tool flow [5], or the *ReCoBus builder* [6]. In particular, the flows provide CAD tools for the hardware synthesis: the designer has to select a *synthesis region* for each PR module, the PR module is synthesized into this region and can then be dynamically configured into any area within the reconfigurable fabric which has the same underlying structure. Thus, above tools enable to place PR modules flexibly on the reconfigurable hardware. However, they lack an automated partitioning into software and PR modules. In addition, support for deciding where and when the PR modules should be placed during run-time to generate a high-quality system is also missing.

However, automatic methods for computing placements for PR modules have been proposed, e. g., in [11], [12], which calculate the locations for modules with given schedule in reconfigurable regions. They transform the task of placement into a 3-dimensional packing problem, with the third dimension being the time. De facto, sophisticated reconfigurable devices have an increasing inhomogeneity due to cells of various types included into the fabric. Therefore, only a subset of all possible 2-dimensional locations is possible for placing the modules. Moreover, these approaches may hardly be applied for multi-mode systems on MPSoCs, since they require a given schedule. Multi-mode systems may, however, switch arbitrarily between different configurations.

Besides partitioning and placement, a challenge is the layout of the underlying partially reconfigurable sub-system. Here, it is necessary to provide a flexible communication infrastructure that allows system-wide communication and supports partial reconfiguration. There are several technologies providing communication macros that can be placed statically into partially reconfigurable regions. The techniques can be classified into the following principles:

- *On-chip buses* are the most common way of linking communicating modules. For example, [13] and [14] present on-chip buses for FPGAs. They are provided as macros and allow partial reconfiguration at run-time, even during bus transactions. These techniques implement the communication infrastructure within the FPGA routing fabric resulting in a low logic overhead.
- *Circuit switching* is a technique where physically wired links are established between two or more modules as, e. g., presented in [15] and [6]. Circuit switching techniques are used for streaming data and establishing

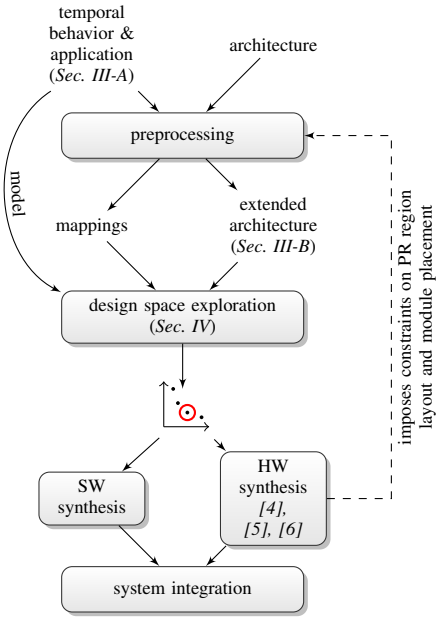


Fig. 1. The proposed design flow for the optimization of reconfigurable multi-mode systems following the Y-chart approach.

point-to-point connections between hardware modules. Sophisticated approaches, e. g., *I/O bars* [6], allow to implement this behavior with a low overhead within the FPGA routing fabric itself. A module is able to read the incoming data, modify it if necessary, and pass it further to the next module via dedicated communication interfaces. Hence, modules occupy signals of the I/O bar to send data to successive modules.

III. Design Flow & Models

Figure 1 illustrates the proposed design flow. It follows the Y-chart approach [7] which separates design behavior (application), design structure (architecture), and, as a result, the physical design (system integration). To be able to deal with multi-mode systems the application model is augmented according to [1] by modeling the temporal behavior, i. e., the operational modes and their possible transitions.

The *preprocessing* augments the application and the architecture by means of high-level synthesis, profiling, simulation, and tracing techniques. The goal is to find possible *mappings*, i. e., for each *process* those *resources* which are able to execute, and extract important performance parameters, such as throughput, required hardware resources (CLBs, BRAMs, DSPs, etc.), and execution time for processes, as well as bandwidth or number of signals for the communication. For being able to facilitate the design of multi-mode systems which employ partial reconfiguration, the resources provided by partial reconfiguration, like PR regions and buses, are added to the architectural model. The proposed architectural model is compatible with all available hardware synthesis flows, while being flexible enough to respect all hardware constraints imposed by real-world reconfigurable technology.

The resulting system specification enables to perform the design space exploration by unifying the following tasks:

- Deciding which tasks to run in software and which in hardware in each mode.
- Placement of partially reconfigurable modules of hardware tasks.
- Allocation of architecture resources.
- On-chip routing of the communication of data-dependent tasks via on-chip buses or circuit switching techniques.
- Multi-objective optimization to find high-quality implementations regarding several objectives, like reconfiguration time, power consumption, etc.

Afterward, the designer can choose one of the found alternative high-quality implementations that best fits its needs and perform the subsequent synthesis steps.

A. Application Model

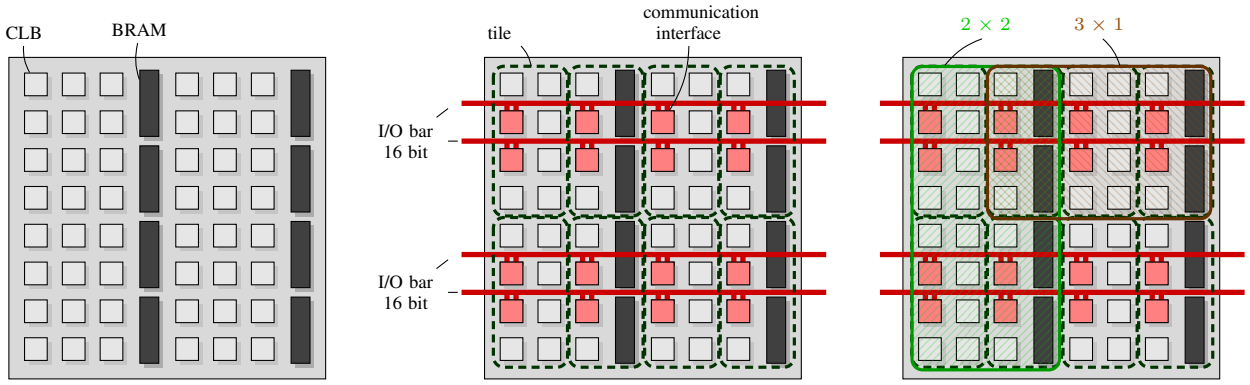
The functionality of the system is described by the *application graph* $G_T(V_T, E_T)$ containing the functional components of all operational modes. The set V_T consists of tasks $t \in T \subseteq V_T$ that represent high-level operations of the application, and communication nodes $c \in C \subset V_T$ denoting messages, signals, and data being exchanged between tasks. Each communication node has exactly one predecessor, the *sender*, and may have several successors, the *receivers*. Hence, it is possible to model multi-cast communication.

Modes and possible transitions between them are specified by an *operational mode state machine* (OMSM) $G_O(O, E_O)$, as it is proposed in [1]. Each state $o \in O$ refers to an operational mode, and each edge $e \in E_O$ refers to a transition between two different modes. In some cases, the maximal transition time for switching between two modes is bounded. This bound is specified as d_e^{max}, $e = (o, o') \in E_O$. A system with multiple operational modes is characterized by the property that different algorithms may run in each mode. Although the modes are executed mutually exclusive, applications and, consequently, their constituent tasks may be part of several modes. By using this application model instead of providing a separate description for each mode, we do not lose information about tasks being part of multiple modes.

We denote the set of tasks being active in mode o as $T_o \subseteq T$ and the communication nodes as $C_o \subseteq T$. Based on the configuration, the application graph of each mode $G_T[T_o \cup C_o]$ can be induced from the application graph $G_T(T, E_T)$. Section V-A provides an example for the application model.

B. Architectural Model

An architecture is formally defined as a *resource graph* $G_R(R, E_R)$, cf. [16]. A resource $r \in R$ can stand for any entity of the heterogeneous system architecture, like microprocessors, application-specific processors, co-processors, but also communication resources as on-chip buses. The directed edges E_R indicate the communication links between them. In this work, we concentrate on how to model the partially reconfigurable sub-systems.



(a) PR region (b) tiling (c) allocation areas

Fig. 2. Example of a PR region and a possible tiling.

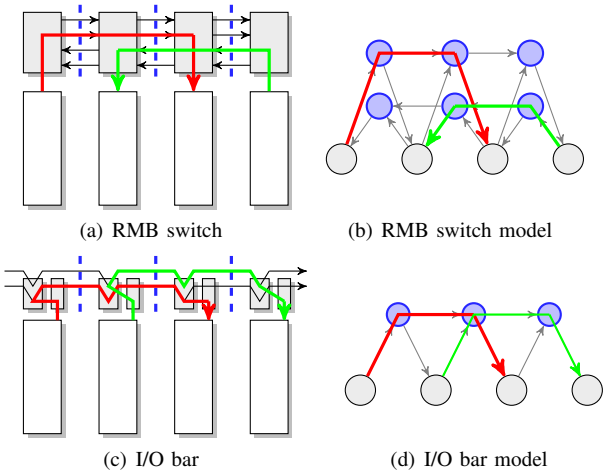
1) Tiling Reconfigurable Regions: Reconfigurable systems are built by partitioning the reconfigurable chip area into static regions and partially reconfigurable regions (PR regions). Logic residing in the static part is needed throughout system operation, like connection logic for the communication interfaces. PR regions are provided for run-time reconfiguration. Hardware modules may be dynamically loaded into appropriate areas within these regions.

Generally, when designing the layout of a PR region, the goal is to achieve the most flexible usage. To achieve this, so-called *tiling* is performed, where the PR region is divided into *reconfigurable tiles* (cf. [13]). Reconfigurable tiles serve as the smallest atomic unit available for partial reconfiguration. By combining tiles, bigger areas may be built for loading hardware modules. Modern reconfigurable fabrics consist of arrays of cells of different types, like logic blocks (CLBs), block RAMs (BRAMs), or multiply-accumulate units (DSPs), as shown in Fig. 2(a). Consequently, a PR region may contain various types of tiles which consist of different cell types according to the underlying cell structure.

The basic idea behind tiling is that each tile offers interfaces to the chosen communication macros. Therefore, the granularity of tiling depends on the chosen communication technique. Koch et al. give a rule how to calculate the size of tiles in [17], depending on the applied communication technique as well as the set of tasks which are candidates for being placed as PR modules. This enables the tiling of the PR region before starting system-level synthesis.

2) Communication in PR Regions: A possible tiling of a PR region is shown in Fig. 2(b). The figure also illustrates that each tile contains dedicated cells which provide the connection logic to access the communication macro. In the proposed model, all of these *communication interfaces* are included into the resource graph as a subset $R_{interface} \subset R$.

Thus, how to model the communication macro depends on the underlying technology. For example, an on-chip bus $r \in R_{bus} \subset R$ is a shared medium which provides mutually exclusive access. This means that all communication interfaces $r \in R_{interface}$ that enable access to a bus r' are connected with r' and vice-versa. However, communication based on



(a) RMB switch (b) RMB switch model

(c) I/O bar (d) I/O bar model

Fig. 3. Examples of circuit switching techniques for (a) RMB and (c) I/O bar. Connections between adjacent communication interfaces cross the same cuts (dashed blue line), thus inducing routing resources (blue) in the resource graphs (b) and (d) for each cut and routing direction.

circuit switching happens via dedicated wires. For abstracting from the physical wires used for circuit switching, routing resources $R_{switch} \subset R$ are introduced. Each node $r \in R_{switch}$ represents a cut between the wires between two adjacent communication interfaces, for each direction in case of directed switches. Therefore, each element $r \in R_{switch}$ represents the set of wires accessible in the same routing direction between these interfaces. Figure 3 illustrates this model for common circuit switching techniques: Figure 3(a) illustrates the physical implementation of the *RMB technique* [15] and Figure 3(b) the corresponding model containing routing resources for each cut in each routing direction. Similarly, Figure 3(c) illustrates the I/O bar technique as presented in [6] with one streaming direction. The corresponding model is shown in Figure 3(d), containing resource nodes for each cut between adjacent communication interfaces. The result is a chain of resource nodes $r \in R_{switch}$. This model allows to express that signals may be read and written via a communication interface or, alternatively, may pass a communication interface without

being accessed.

3) Extracting Partial Resources: The basic idea of our reconfigurable multi-mode system model is to identify all those areas within the PR region which may be occupied by tasks implemented as PR modules, denoted *allocation area* in the following. We denote the formal representation of these allocation areas as *partial resources*, which are included into the resource graph. As already stated before, an allocation area is built by combining several contiguous tiles. Therefore, the number of additional partial resources may potentially grow exponentially with the number of reconfigurable tiles. However, only those partial resources are included that represent areas which may actually be used for efficiently implementing a set of given tasks as hardware modules, i. e., the set of (a) feasible and (b) efficient allocation areas.

- An allocation area for implementing a task as PR module is called *feasible* if it provides a sufficient number of cell types to meet the tasks requirements. It is therefore necessary to determine these requirements in terms of required CLBs, BRAMs, DSPs, etc. This can be achieved by logic synthesis of a high-level description of the task.
- A feasible allocation area A of a task is called *efficient* only if the task has no other feasible allocation area B such that area A encloses area B.

Example: Consider a task which requires 10 *CLBs and* 4 *BRAMs, and assume that the synthesis flow only supports the synthesis of modules with rectangular allocation areas. Now, Fig. 2(c) shows two feasible and efficient allocation areas for implementing the task as PR module in the given PR region. Altogether,* 5 *of these areas exist.*

An automatic approach to extract possible shapes and positions of such areas for given tasks is presented in [18]. All identified areas are added as partial resources R_{PR} and linked with those communication interfaces lying within the corresponding area.

4) Mappings and Conflict Sets: Mappings $m = (t, r) \in M \subseteq T \times R$ relate tasks with those resources on which they can be implemented. Due to the proposed architectural model, this representation also allows to encode the placement of PR modules by mappings $m = (t, r')$ with $r' \in R_{PR}$. Now, it has to be ensured that the areas of modules which are mapped concurrently do not overlap. Therefore, *conflict sets* are specified where each conflict set $R_\cap$ contains all these partial resources which corresponding areas overlap. This means that all areas represented by the resources $r \in R_\cap$ use at least one common reconfigurable tile. The set $\mathcal{R}_\cap = \{R_\cap^{(1)}, R_\cap^{(2)}, \ldots\}$ denotes the set containing all conflict sets.

IV. DSE of Partially reconfigurable Multi-mode systems

This section describes how to perform DSE for dynamically reconfigurable multi-mode systems that are specified according to the given models. The basic task is to (a) find feasible implementations that (b) are optimal regarding multiple objectives. The design space exploration basically builds new implementations according to the specification, evaluates them, and, as a result, returns the best found implementations to the designer. Creating a single feasible implementation is called *system synthesis*, which has to compute the *allocation*, the *binding*, and the *routing* to build an implementation:

- The *allocation* determines the used hardware.
- A *binding* has to be determined for each operational mode $o \in O$ by selecting mappings and, thus, assigning the tasks to allocated resources. Each task $t \in T_o$ which is active in mode o must be bound in the corresponding bindingexactly once.
- *Routing* has to be performed for each operational mode. The task of routing is to determine a valid route for each communication $c \in C_o$ over the allocated routing resources of the resource graph that connects the sender with all receivers.

System synthesis itself is known to be NP-complete. Thus, solving DSE precisely fails for real-world problems, while common iterative heuristics have problems finding feasible solutions at all or need simplified problems. To cope with this problem, we use a SAT solver to find feasible implementations and an evolutionary algorithm to traverse the search space. Therefore, we provide a symbolic encoding of feasible implementations in Section IV-A, which then can be used as input for the sophisticated SAT-based DSE techniques, as described in Section IV-B.

A. Symbolic Encoding of Implementations

The encoding of feasible implementations consists of the following binary variables:

- $\mathbf{m_o}$: one binary variable for each mapping $m \in M$, indicating whether to choose m in mode o (1) or not (0).
- $\mathbf{r}$: binary variable indicating whether to allocate resource r (1) or not (0).
- $\mathbf{c_{o,r}}$: binary variable indicating whether communication c is routed over resource r in mode o (1) or not (0).
- $\mathbf{c_{o,(r,\tilde{r})}}$: binary variable indicating whether communication c is routed over link $(r, \tilde{r})$ in mode o (1) or not (0).

With these variables, we present in the following the constraints for a feasible binding of the application in Section IV-A1, a feasible routing of the communication between the bound tasks in Section IV-A2, the corresponding feasible allocation of resources in Section IV-A3, as well as additional optimization constraints in Section IV-A4 which further speed up the DSE.

1) Binding: We ensure that in each operational mode, exactly one binding for each task running in this mode has to be chosen,
$\forall o \in O$, $\forall t \in T_o$:

$$\sum_{\forall m=(t,r)\in M} \mathbf{m_o} = 1. \tag{1}$$

Also for tasks not part of the current mode, a placement may be chosen. By means of this *pre-loading*, it is possible to reduce the reconfiguration time when leaving the current mode since modules of following modes are already loaded.

This optional selection is encoded according to
$\forall o \in O$, $\forall t \in T \setminus T_o$:

$$\sum_{\forall m=(t,r)\in M} \mathbf{m_o} \leq 1. \tag{2}$$

In many application domains, like signal processing, tasks are typically periodically activated. In this case, bindings onto software-programmable resources (denoted R_{SW}) have to pass the schedulability test. For this, only the tasks actively running in a mode have to be considered in a multi-mode system:
$\forall o \in O$, $\forall r \in R_{SW}$:

$$\sum_{\substack{m\in M \\ m=(t,r)}} \mathbf{m_o} \cdot \left(\frac{exec(m)}{period(m)} \cdot hp_o \right) \leq hp_o, \tag{3}$$

where $exec(m)$ and $period(m)$ are the execution time and the period of mapping m, and hp_o is the *hyper-period* of all tasks. The hyperperiod is used to get integer coefficients.

Furthermore, it has to be avoided that the allocation areas of those tasks overlap that are placed as partial modules in the same operational mode. This would else impose a conflict since one module would overwrite the logic of another also placed module[1]. $\mathcal{R}_\cap$ contains the sets of all conflict sets according to Section III-B4.. This means that at most one mapping onto one resource of a conflict set may be chosen. Again, only the mappings of the same mode are considered:
$\forall o \in O$, $\forall R_\cap \in \mathcal{R}_\cap$:

$$\sum_{\substack{\forall m=(t,r)\in M \\ r\in R_\cap}} \mathbf{m_o} \leq 1. \tag{4}$$

Mode transitions may have *reconfiguration time constraints* given as bound $d^{max}_{(o,o')}$. Reconfiguration time depends on how many dynamic modules have to be loaded when switching to the new mode. Given the time $D_{load}(m)$ for loading a mapping m, the constraint is given as:

$$\sum_{\forall r\in R} \sum_{\forall t\in T(o')} \sum_{\forall m=(t,r)\in M} \Delta(o,o',m) \cdot D_{load}(m) \leq d^{max}_{(o,o')}. \tag{5}$$

This means that all dynamic components that are part of the new mode o' have to be loaded if they were not already loaded in the previous step. In this case, $\Delta(o,o',m) = \mathbf{m_{o'}} \cdot \overline{\mathbf{m_o}}$ has value 1, meaning the dynamic mapping is chosen for the current mode o' but not the previous one. Note that $\Delta(o,o',m)$ is a non-linear construct, which, however, can be linearized[2].

2) Routing: An encoding for a feasible on-chip routing is given in [19]. It is applicable for complex networks with irregular multi-hop topologies and uses a decision variable for each combination of tasks, resource, and hop. However, since every possible hop is considered, this encoding implies a tremendous overhead for circuit switching: the main problem is that circuit switching is represented as a chain of resource nodes (see Sec. III-B2). The length of such a chain determines the number of hops. Consequently, above encoding grows quadratically with the length of such chains of circuit switching resources. We therefore propose the following novel formulation of feasible routings which is adequate for architectures tackled by our approach and grows linearly with the chain length of routing resources. Here, $M_{send,c}$ denotes the set of all mapping options of the unique sender of communication c, and $M_{recv,c}$ the set of all mapping options of all receivers. Let R_{comm} be the set of routing resources, then the routing constraints may be formulated as follows,
$\forall o \in O, \forall c \in C(o)$:
$\forall m{=}(t,r) \in M_{send,c}$

$$-\mathbf{m_o} + \mathbf{c_{o,r}} \geq 0 \tag{6a}$$

$\forall m{=}(t,r) \in M_{recv,c}$

$$-\mathbf{m_o} + \mathbf{c_{o,r}} \geq 0 \tag{6b}$$

$\forall r \in R$

$$-\mathbf{c_{o,r}} + \sum_{\forall m=(t,r)\in M_{send,c}} \mathbf{m}_o + \sum_{\forall(\tilde{r},r)\in E_R} \mathbf{c_{o,(\tilde{r},r)}} \geq 0 \tag{6c}$$

$$-\mathbf{c_{o,r}} + \sum_{\forall m=(t,r)\in M_{recv,c}} \mathbf{m}_o + \sum_{\forall(r,\tilde{r})\in E_R} \mathbf{c_{o,(r,\tilde{r})}} \geq 0 \tag{6d}$$

$\forall e \in E_R$, $e = (r,\tilde{r}) \vee (\tilde{r},r)$

$$-\mathbf{c_{o,e}} + \mathbf{c_{o,r}} \geq 0 \tag{6e}$$

$\forall$ cycles $L = (e_1, e_2, ..., e_k)$ in $G_R(R, E_R)$ with $e_i \in E_R$

$$\overline{\bigwedge_{e\in L} \mathbf{c_{o,e}}} = \bigvee_{e\in L} \overline{\mathbf{c_{o,e}}} \tag{6f}$$

Eq. (6a) implies[3] that the route contains the communication interface to that the sender is bound, and Eq. (6b) ensures that the communication is routed on the resources containing receivers. The following constraints then ensure that the route connects these resources. Eq. (6c) implies that, if c is routed on r, either the sender is bound on r or the message arrives via an input link of r. Eq. (6d) states that if c is routed on r then either a receiver is mapped onto this resource or it leaves via an output link. Eq. (6e) ensures to route c on the actual resource when it is also routed on its input or output links. Finally, Eq. (6f) ensures acyclic routing by avoiding cycles in the resource graph $G_R(R, E_R)$.

One important limiting factor in the design of embedded systems is the communication infrastructure since bandwidth on buses and wires for circuit switching are restricted. With a message having a bandwidth requirement of $bw(c)$ and a bus providing a bandwidth of $bw(r)$, the *bandwidth constraint* is

[1]Note that this contraint may also be adapted for coarse-grained reconfigurable architectures (CRGAs). Here, the partial resource $r \in R_{PR}$ also contains the processing elements (PEs) $\pi \in r$ that are affected when the module is loaded onto the CRGA. This allows to formulate resource restriction constraints for all PEs π according to the specific workload of the modules assigned to r in each mode.

[2]Expression $c := a \cdot \overline{b}$ can be linearized by $c \leq a$, $c \leq 1 - b$, $c \geq a - b$.

[3]An implication of the form $a \rightarrow b$ is formulated as linear constraint $-a + b \geq 0$.

formulated,
$\forall o \in O$, $\forall r \in R_{bus}$:

$$\sum_{\forall c \in C_o} \mathbf{c_{o,r}} \cdot bw(c) \leq bw(r), \tag{7}$$

meaning that the accumulated bandwidth requirements of all communications routed over a routing resource $r \in R_{bus}$ in the operational mode o must not exceed the maximal bandwidth of r. Equivalently, with a communication c requiring a certain number of wires $wires(c)$ of the wires provided for circuit switching $wires(r)$ by the routing resource $r \in R_{switch}$, the *circuit switching constraint* is formulated accordingly,
$\forall o \in O$, $\forall r \in R_{switch}$:

$$\sum_{\forall c \in C_o} \mathbf{c_{o,r}} \cdot wires(c) \leq wires(r) \tag{8}$$

3) Allocation: The allocation is encoded for all resources not constituting routing resource according to,
$\forall r \in R \setminus R_{comm}$:

$$-\mathbf{r} + \sum_{o \in O} \sum_{m=(p,r) \in M} \mathbf{m_o} \geq 0 \tag{9}$$

$\forall o \in O$, $\forall m = (t, r) \in M$:

$$-\mathbf{m_o} + \mathbf{r} \geq 0 \tag{10}$$

ensuring that a resource without any mapping is not allocated. The encoding for the routing resources $r \in R_{comm}$ is done accordingly, using the routing decisions $\mathbf{c}_{o,r}$ instead of the mappings.

4) Optimization Constraints: When considering partial reconfigurable sub-systems, it is a design problem to decide which parts of the reconfigurable fabric should be used dynamically, and which parts are kept static throughout system operation. We include additional constraints, that describe whether hardware modules of tasks should be implemented as static parts of the reconfigurable sub-system, instead of being considered as partially replacable components. In this way, it might be possible to reduce reconfiguration times. However, other mappings into overlapping area can no longer be selected.

We encode this design option by introducing following additional binary variables:

- $\mathbf{m_s}$: one binary variable for each mapping option in $m \in M$, indicating whether to choose m as static mapping (1) or not (0).
- $\mathbf{m_{d,o}}$: one binary variable for each mapping option in $m \in M$, indicating whether to choose m as a dynamic mapping in mode o (1) or not (0).

The following formulation states that when a mapping is chosen in a certain mode, it has also be decided whether it becomes a static or a dynamic part of the system:
$\forall o \in O$, $\forall m \in M$:

$$-\mathbf{m_o} + \mathbf{m_s} + \mathbf{m_{d,o}} = 0. \tag{11}$$

Note,that this decision is already included in Equations (1) and (2). By modeling this decision explicitly, it is, however, possible to reach solutions faster, which are better regarding the reconfiguration time.

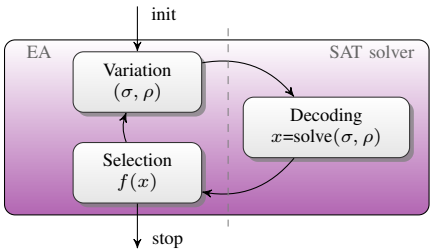


Fig. 4. *SAT decoding* optimization approach by coupling an Evolutionary Algorithm and SAT solver, cf. [9].

B. SAT Decoding for DSE

Sophisticated SAT-based optimization techniques can be applied for DSE. Modern SAT solvers are very powerful tools to solve highly constrained problems. They can also be applied to solve optimization problems which are formulated as SAT and have a single linear objective function. However, most design spaces have multiple, often non-linear objectives. For being able to apply SAT solvers efficiently to multi-objective optimization problems, Lukasiewycz et al. propose *SAT decoding* [9], which couples a Multi-objective Evolutionary Algorithm (MOEA) with a SAT solver, as illustrated in Fig. 4. Instead of modifying the implementation, the EA modifies the SAT parameters (σ, ρ) that are used by the SAT solver to determine feasible solutions. Each parameter modification leads to a different strategy and, in most cases, to a different solution provided by the SAT solver. By iteratively performing these steps, better feasible solutions can be found over time. The MOEA thus enables to approximate the Pareto-front of feasible solutions. Still, the MOEA is a heuristic. Not necessarily all solutions may be found.

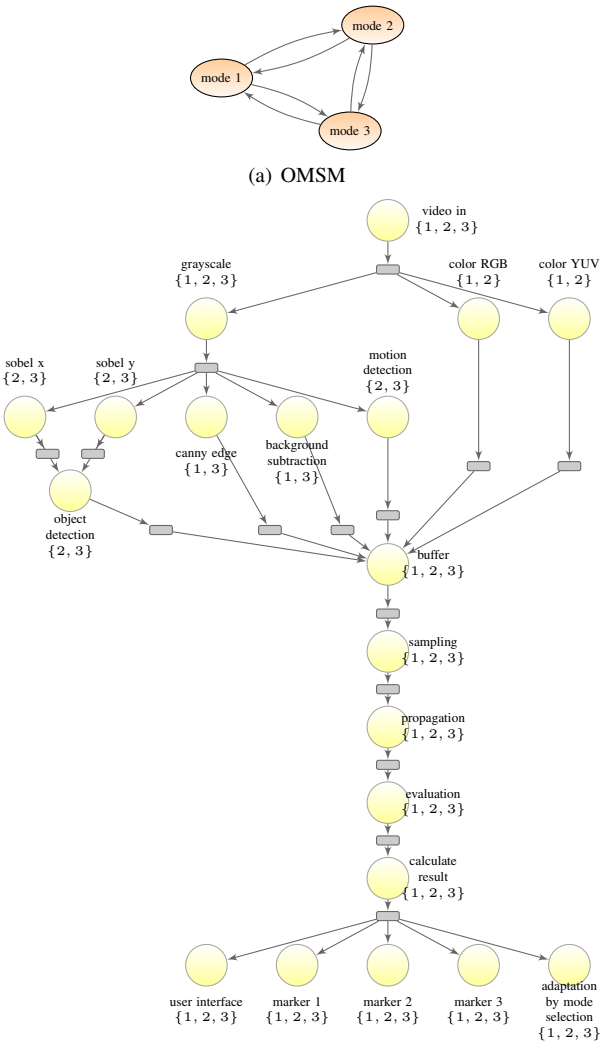
V. Experimental Results

This section presents the results of applying the proposed system-synthesis methodology to a case study from the image processing domain. After introducing the case study, the proposed approach is evaluated and compared to other encoding variants.

A. Case Study

We use a smart-camera scenario as case study for our approach. The chosen *video surveillance application* for object tracking works on HD 720p video streams with a frame rate of 30 fps. The application can switch between modes which run different image processing algorithms to deal with varying environmental conditions. Fig. 5 shows an example of a corresponding application with three modes.

The *heterogeneous architecture* consists of 5 processor variants, with four RISC processors and one application-specific signal processor. Furthermore, partially reconfigurable regions are provided to run PR modules. We have chosen the publicly available ReCoBus synthesis flow [6] as state-of-the-art technology, since it offers several features: first, communication macros for an on-chip bus which supports



(a) OMSM

(b) application

Fig. 5. Multi-mode application from tc2. Tasks are annotated with braces listing the modes where they are active.

partial reconfiguration. Second, a circuit-switching communication macro, called I/O bar, which allows to stream the video signal through the reconfigurable area. We use a 32 bit wide I/O bar instance. Our architecture, based on the Xilinx Virtex technology and the layout of [20], provides two 2-dimensional reconfigurable regions, each consisting of 28×2 tiles (Fig. 6). Tiles have a width of one CLB column and a height of 16 CLB rows. Each tile provides access to the on-chip bus and the communication interfaces of the I/O bar. The tiles consist either of CLBs or BRAMs, with cells comprised of the latter not providing communication interfaces. The on-chip bus is connected to the communication interface via a bridge.

We have based the case study on the architecture and results from [20] where we have quantified the performance values, reconfiguration times, and resource requirements of the tasks by analyzing the high-level descriptions in C for software variants, and in VHDL for hardware variants on the implemented architecture. The feasible and efficient allocation areas of the

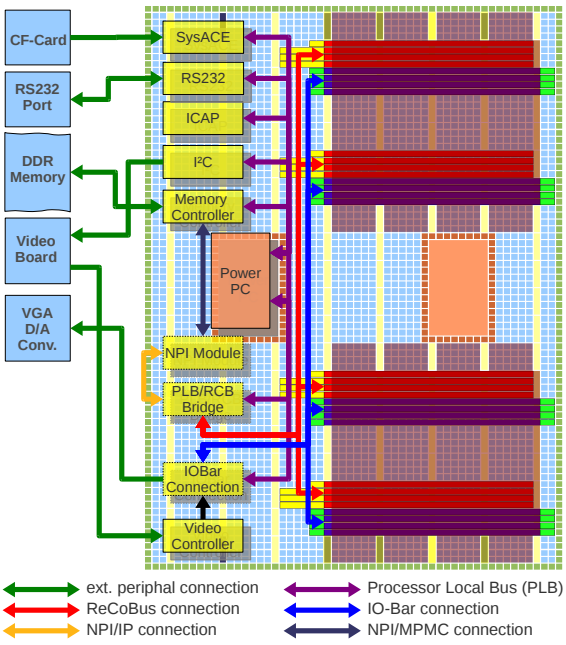


Fig. 6. Layout of the reconfigurable sub-system from [20], providing two 2-dimensional reconfigurable regions, each consisting of 28×2 tiles. Tiles have a width of one CLB column and a height of 16 CLB rows. Partial modules may possess different sizes and be placed in a two-dimensional manner.

TABLE I

DESCRIPTION OF THE TEST-CASES. EACH MODE IS GIVEN AS A TRIPLE OF (#TASKS/#COMMUNICATION, EXECUTION PROBABILITY).

test case	modes
tc1	(16/11, 0.50), (18/13, 0.50)
tc2	(16/11, 0.36), (18/13, 0.36), (18/13, 0.28)
tc3	(16/11, 0.28), (18/13, 0.28), (18/13, 0.22), (18/12, 0.22)
tc4	(16/11, 0.20), (18/13, 0.20), (18/13, 0.15), (18/12, 0.15), (18/13, 0.2)

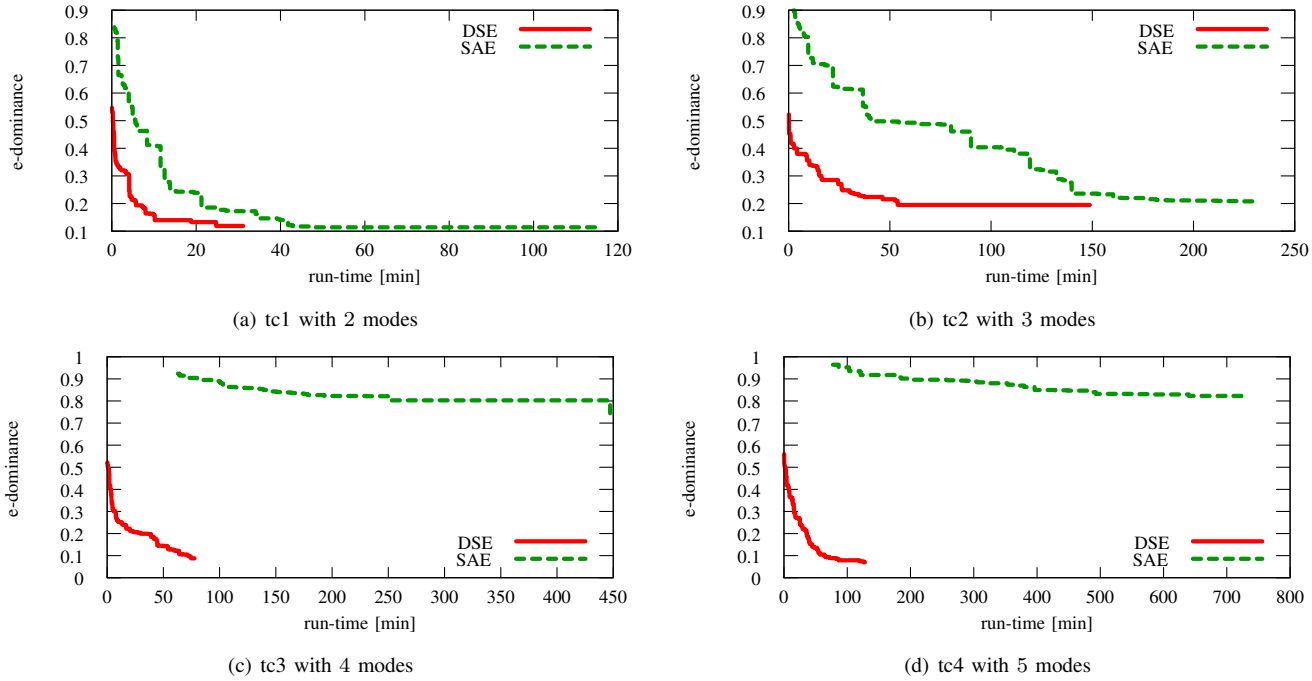
tasks are extracted by using the algorithm presented in [18] to derive minimal synthesis regions, and then determining the feasible positions for found regions. From this specification, we have generated four test-cases which are summarized in Table I. As in [1], [8], we have assigned each mode with an execution probability, giving the probability that the system is in the corresponding mode.

The presented approach is implemented using the publicly available meta-heuristic framework OPT4J [21] and the DSE tool from [19]. All experiments are carried out on a Core 2 Quad 3.00 GHz. The chosen objectives are:

- *costs* evaluating the chip size of the chosen allocation. Processors are assigned with fix hardware costs. Moreover, the costs for the required PR regions are calculated based on the chosen placements.
- *power consumption* is calculated using analytical models for multi-mode systems as described in [1].
- *average reconfiguration time* is used as additional objective to allow fast transitions between modes.

B. Comparison

To compare our proposed symbolic system-level synthesis to the state-of-the-art, we have also implemented an approach



(a) tc1 with 2 modes

(b) tc2 with 3 modes

(c) tc3 with 4 modes

(d) tc4 with 5 modes

Fig. 7. Average ϵ-dominance over time for the compared approaches and test-cases.

based on the methodology given in [1]. We denote this approach as *SAE*. Here, *binding* is performed by encoding mapping as a *multi-mode task mapping string*, which is evolved by an Evolutionary algorithm. The task of PR module placement is not detailed in [1]. We therefore use our architectural model which provides an encoding of the placement. This also shows the flexibility of the proposed model. Moreover, if the placement of a solution from SAE is not valid due to overlappings, we apply a randomized first-fit placement heuristic as repair strategy. Still, the task mapping encoding of SAE does not guarantee that all found solutions are actually feasible. Therefore, [1] proposes to apply a penalty on the fitness function when placement and scheduling constraints are violated. Moreover, a repair strategy is deployed when violating placements for several successive evaluations. Here, a repair step remaps hardware tasks onto processors. In our implementation of SAE, we initiate one repair step after 100 successive area-infeasible evaluations. The authors propose to use an inner loop for *communication mapping and scheduling*. Since no details on this step are given, we test the found solutions by applying our symbolic encoding, and checking, whether there exists at least one feasible routing for the chosen binding. If not, a constant penalty is applied on the fitness function. For also performing the *architecture exploration* with *SAE*, all resources with at least one active mapping are allocated.

We have applied our proposed approach, denoted DSE, and SAE onto the case studies and performed 3 runs per test case for each approach. For both approaches, the population size of the MOEA was set to 100. DSE was run for 2000 generations, and SAE for 10000. The results of the experiments are depicted in Fig. 7, which gives the average ϵ-dominance over the run-time. The ϵ-dominance is used to compare the Pareto-fronts of certain iterations by a scalar value. To determine this metric, the effective Pareto-points are selected by combining the results of all test-runs and all algorithms. This global Pareto-front is denoted as set $\mathcal{PF}^*$. Then, the average ϵ-dominance is calculated for the points of the Pareto-front $\mathcal{PF}$ of each algorithm for each iteration and each test-run. The ϵ-value measures the distance of a point $u \in \mathcal{PF}$ to the closest point in the global Pareto-front $v \in \mathcal{PF}^*$. It is calculated as $(1+\epsilon) \cdot u_i \geq v_i$ over all objectives i. Now, $\epsilon = 0$ means that u is on the Pareto-front and, since we normalize the objectives, $\epsilon = 1$ means that u is farthest away from the Pareto-front.

Test-cases tc1 and tc2 reveal that both approaches are able to find high-quality solutions for the small test-cases. The search strategy of SAE basically has a high repair rate when starting the optimization approach. Repair remaps tasks onto the processors, so that in the initial phase hardly any tasks are mapped into the reconfigurable hardware. Over time, solutions are generated that also contain partial hardware modules, so that eventually high-quality solutions are generated. In contrast, the proposed DSE advances much faster to these high-quality solutions.

As soon as the test-cases get more complex in terms of number of modes, number of tasks, and number of communication, SAE fails to find good solutions in reasonable time. The problem is that the approach spends too much time for finding feasible solutions instead of performing the optimization. Figures 7(c) and (d) show that SAE needs over an hour of time in average to find feasible solutions at all. In comparison, our DSE approach is able to optimize from the beginning finding high-quality solutions in reasonable time.

Fig. 8 shows the Pareto-fronts of DSE and SAE for test-case

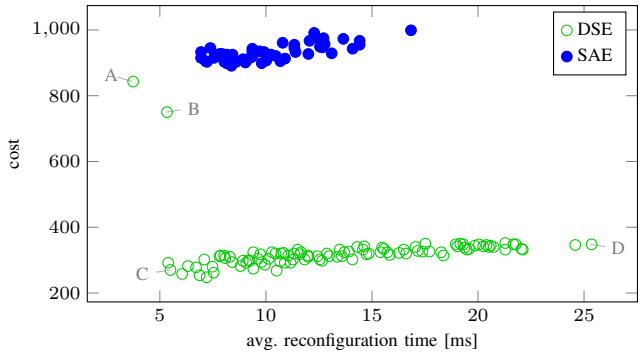


Fig. 8. Pareto-fronts for tc4.

tc4. Depicted are the found design points with their reconfiguration time and cost. Note that all solutions found by DSE have a lower power consumption than those of SAE. The plot reveals that, for SAE, the solutions have high costs stemming from the fact that all 5 processor variants are allocated. In contrast, DSE finds two solutions A and B allocating only 4 and 3 processor variants, respectively. Nonetheless, both implementations utilize the allocated resources much better, allowing to significantly reduce both, power consumption and reconfiguration time.

Moreover, there are several DSE solutions with an increased use of reconfigurable hardware. In this solutions, either one medium-performance processor (e.g., design point C), or one medium-performance processors together with a cheap low-performance processor (e.g., design point D) are allocated. The spread of found solutions stems from outsourcing more functionality into the reconfigurable hardware, the reconfiguration time increases and with that the cost since more reconfigurable area is required. However, hardware modules run more efficiently regarding the power consumption.

VI. Conclusion

In this work, we propose a design space exploration for multi-mode reconfigurable embedded systems. By respecting the different operational modes of an application and, thus, exploiting the knowledge that not all tasks in the system are running concurrently all the time, embedded system can be designed which utilize the hardware more efficiently. We therefore include the temporal behavior of the system using known models. In addition, we introduce a novel model that allows to specify the characteristics of partial reconfiguration on the architecture level. This makes it possible that we can perform allocation, binding, partial module placement and on-chip routing for multi-mode systems in a holistic approach without the need of cascading optimization loops. Moreover, our proposed symbolic encoding scheme allows to specify the constraints for feasible implementations. Using this encoding, we are able to apply sophisticated SAT-based optimization techniques.

We have compared our approach to the de facto state-of-the-art of performing system-level synthesis of multi-mode systems that also respects reconfigurable sub-systems. The experimental results show that our novel mapping strategy performs significantly better when dealing with complex test-cases. This is due to the fact, that we do not have to apply penalty and repair strategies to deal with infeasible solutions.

Moreover, we have presented a design flow that incorporates our system-level synthesis. The flow complies with the state-of-the-art synthesis tools for partially reconfigurable systems, as well as sophisticated communication technologies for reconfigurable systems. Still, our proposed specification model allows to abstract from the underlying synthesis flows. This makes the proposed methodology highly applicable for building multi-mode systems on advanced reconfigurable technology.

References

[1] M. Schmitz, B. Al-Hashimi, and P. Eles, "Cosynthesis of energy-efficient multimode embedded systems with consideration of mode-execution probabilities," *IEEE TCAD*, vol. 24, no. 2, pp. 153–169, 2005.

[2] S. Wildermann, A. Oetken, J. Teich, and Z. Salcic, "Self-organizing computer vision for robust object tracking in smart cameras," in *Proc. of ATC*, 2010, pp. 1–16.

[3] R. Müller and J. Teubner, "FPGA: what's in it for a database?" in *SIGMOD*, 2009, pp. 999–1004.

[4] P. Lysaght, B. Blodget, J. Mason, J. Young, and B. Bridgford, "Invited paper: Enhanced architectures, design methodologies and CAD tools for dynamic reconfiguration of Xilinx FPGAs," in *Proc. of FPL*, 2006, pp. 1–6.

[5] J. Hagemeyer, B. Kettelhoit, M. Koester, and M. Porrmann, "INDRA – integrated design flow for reconfigurable architectures," in *Proc. of DATE*, 2007.

[6] D. Koch, C. Beckhoff, and J. Teich, "ReCoBus-Builder – a Novel Tool and Technique to Build Statically and Dynamically Reconfigurable Systems for FPGAs," in *Proc. of FPL*, Heidelberg, Germany, Sep. 2008, pp. 119–124.

[7] B. Kienhuis, E. Deprettere, K. Vissers, and P. Van Der Wolf, "An approach for quantitative analysis of application-specific dataflow architectures," in *Proc. of ASAP*, 1997, pp. 338–349.

[8] L. Huang and Q. Xu, "Energy-efficient task allocation and scheduling for multi-mode MPSoCs under lifetime reliability constraint," in *Proc. of DATE*, 2010, pp. 1584–1589.

[9] M. Lukasiewycz, M. Glaß, C. Haubelt, and J. Teich, "Efficient symbolic multi-objective design space exploration," in *Proc. of ASP-DAC*, 2008, pp. 691–696.

[10] D. Göhringer, M. Hübner, M. Benz, and J. Becker, "A design methodology for application partitioning and architecture development of reconfigurable multiprocessor systems-on-chip," in *Proc. of FCCM*, 2010, pp. 259–262.

[11] S. Fekete, E. Köhler, and J. Teich, "Optimal FPGA module placement with temporal precedence constraints," in *Proc. of DATE*, 2001, pp. 658–667.

[12] K. Danne and S. Stühmeier, "Off-line placement of tasks onto reconfigurable hardware considering geometrical task variants," in *From Specification to Embedded Systems Application*. Springer, 2005, pp. 311–311.

[13] J. Hagemeyer, B. Kettelhoit, M. Koester, and M. Porrmann, "Design of homogeneous communication infrastructures for partially reconfigurable FPGAs," in *Proc. of ERSA*, 2007, pp. 238–247.

[14] D. Koch, C. Haubelt, and J. Teich, "Efficient reconfigurable on-chip buses for FPGAs," in *Proc of FCCM*, 2008, pp. 287–290.

[15] H. ElGindy, H. Schroder, A. Spray, A. Somani, and H. Schmeck, "RMB – A reconfigurable multiple bus network," in *Proc. of HPCA*, 1996, pp. 108–117.

[16] T. Blickle, J. Teich, and L. Thiele, "System-level synthesis using evolutionary algorithms," *Design Automation for Embedded Systems*, vol. 3, pp. 23–58, 1998.

[17] D. Koch, C. Beckhoff, and J. Teich, "Minimizing internal fragmentation by fine-grained two-dimensional module placement for runtime reconfigurable systems," in *Proc. of FCCM*, 2009, pp. 251–254.

[18] M. Koester, W. Luk, J. Hagemeyer, M. Porrmann, and U. Ruckert, "Design optimizations for tiled partially reconfigurable systems," *IEEE TVLSI*, no. 99, pp. 1–14, 2010.

[19] M. Lukasiewycz, M. Streubühr, M. Glaß, C. Haubelt, and J. Teich, "Combined system synthesis and communication architecture exploration for MPSoCs," in *Proc. of DATE*, 2009, pp. 472–477.
[20] A. Oetken, S. Wildermann, J. Teich, and D. Koch, "A bus-based SoC architecture for flexible module placement on reconfigurable FPGAs," in *Proc. of FPL*, 2010, pp. 234–239.
[21] M. Lukasiewycz, M. Glaß, F. Reimann, and J. Teich, "Opt4J - a modular framework for meta-heuristic optimization," in *Proc. of GECCO*, 2011.