Exploration, Partitioning and Simulation of Reconfigurable Systems

Exploration, Partitionierung und Simulation rekonfigurierbarer Systeme

Florian Dittmann, Franz Rammig, University of Paderborn,
Martin Streubühr, Christian Haubelt, University of Erlangen-Nürnberg,
Andreas Schallenberg, Wolfgang Nebel, CvO University Oldenburg

Summary Reconfigurable devices in large complex systems allow the reduction of the amount of required resources. They serve as run-time re-usable devices for performance critical data-oriented processes. However, the use of reconfigurable devices within large systems greatly increases the design complexity. The designer's task gets even harder when the goal is a resource efficient solution. Constructing a good design requires the consideration of many design alternatives. With today's complex systems and the resulting degrees of freedom the designer should be assisted by sophisticated design space exploration tools. However, all known system-level design space exploration tools do not exploit the potentials dynamic hardware reconfiguration exposes. Moreover, the implementation of selected solutions poses an additional challenge and also requires a cycle-level simulation. This paper presents a novel design methodology which is able to overcome these drawbacks by integrating state-of-the-art temporal partitioning approaches for dynamic hardware reconfiguration into system-level design space exploration.

►►► **Zusammenfassung** Dynamisch rekonfigurierbare Chips erlauben es, bei großen Systemen Ressourcen einzusparen. Sie dienen als dynamisch programmierbare Einheiten für laufzeitkritische Anwendungen. Leider erhöht der Einsatz dieser Chips die Entwurfskomplexität drastisch. Die Aufgabe wird noch schwieriger, wenn eine ressourceneffiziente Lösung gefordert ist. Um dabei eine gute Lösung zu finden, müssen viele Entwurfsalternativen untersucht werden, wobei bei der heutigen Komplexität der Systeme und der Anzahl an Freiheitsgraden der Entwickler durch Werkzeuge unterstützt werden sollte. Leider gibt es bis heute keine Explorationswerkzeuge auf Systemebene, die auch das Potential der Laufzeitrekonfigurierung ausnutzen. Der vorliegende Beitrag stellt eine neue Entwurfsmethode vor, um die genannten Aufgaben zu lösen. Dazu werden aktuelle Explorations-, Partitionierungs- und Simulationsverfahren herangezogen. Das mittels mehrerer Iterationsverfahren gewonnene Explorationsergebnis unterstützt dann den Entwickler bei der letztendlichen Implementierung.

KEYWORDS B.7.1 [Hardware: Integrated Circuits: Types and Design Styles], C.3 [Computer Systems Circuits: Special-Purpose and Application-Based Systems], I.6.2 [Computer Methodologies: Simulation and Modeling: Simulation Languages], reconfiguration, design space exploration, design methodology, simulation, partitioning, FPGA/Rekonfigurierung, Entwurfsraumexploration, Entwurfsmethodik, Simulation, Partitionierung

1 Introduction

The demanding computational requirements of today's systems is often tackled by a set of heterogeneous devices each solving a specific problem best. Such systems are composed of general-purpose CPUs, DSPs or ASICs, and increasingly reconfigurable devices like FPGAs. Being challenging even without, the design complexity of such heterogeneous systems including FPGAs in-

This work was partially funded by the Deutsche Forschungsgemeinschaft SPP 1148 *Reconfigurable Computing.*

creases, as FPGAs incorporate hardware reuse.

Several concepts targeting essential parts of the design of such systems can be found in the literature. However, integrated design methodologies from system-level to simulation and eventual generation of configurations are still subject to research, particularly if the reconfigurable FPGAs are fundamental parts of the systems. Such an integrated design methodology that spans the whole design process is proposed in this work.

1.1 Overview of the Methodology

Our proposed methodology is depicted in Fig. 1. First, an initial SystemC model is examined for potentially reconfigurable parts. This is done by inspecting data members of SystemC modules. Suitable members are encapsuled in certain C++ containers. This way an OSSS+R model (for details, see Section 4) is constructed. The model can be simulated to obtain a trace covering all assumed reconfiguration related information. From this, the execution times for the reconfigurable parts are extracted.

In parallel to this, a tool-based synthesis of the OSSS+R model to VHDL is performed. The resulting files can be synthesized with FPGA vendor specific tools, e. g., using Xilinx early access partial reconfiguration design flow [14]. It is not necessary to obtain a working model. It is sufficient to obtain estimations for reconfiguration times of the identified candidates.

In a third step, the initial SystemC model is analyzed to extract a so-called *process graph*. Next, the designer specifies the architecture template and mapping constraints resulting in a so-called *specification graph* [12]. Processes that might be mapped onto FPGAs are passed to the FPGA-level exploration which determines optimal FPGA configurations, as described below. These configurations are inserted in the architecture template and are considered during design space exploration. Using this specification graph, the system-level design space exploration phase determines an optimal selection of resources as well as an optimal mapping of processes onto these hardware resources by also considering effects from dynamic hardware reconfiguration (see Section 2). The design space exploration is based on Multi-Objective Evolutionary Algorithms (MOEAs) and is able to optimize multiple objectives, like cost, power consumption, etc., simultaneously. The result is an approximation set of Pareto-optimal solutions, from which a designer has to select a solution for implementation.

For reconfigurable FPGAs, the assigned set of tasks is partitioned by an additional phase, the FPGA-level exploration (step 5). In this phase, we optimize the sequence of configurations in order to reduce the reconfiguration overhead. This FPGA based design space exploration examines the reconfigurable nodes of the graph and proposes solutions for implementing the given set of processes. It includes the task of deriving partitions that are suitable for dynamic reconfiguration basically based on the execution time and area estimation of step 1 and 2. The key idea is to optimize configurations by reducing communication requirements, which also includes an optimized order of configuration steps. This is both done using the

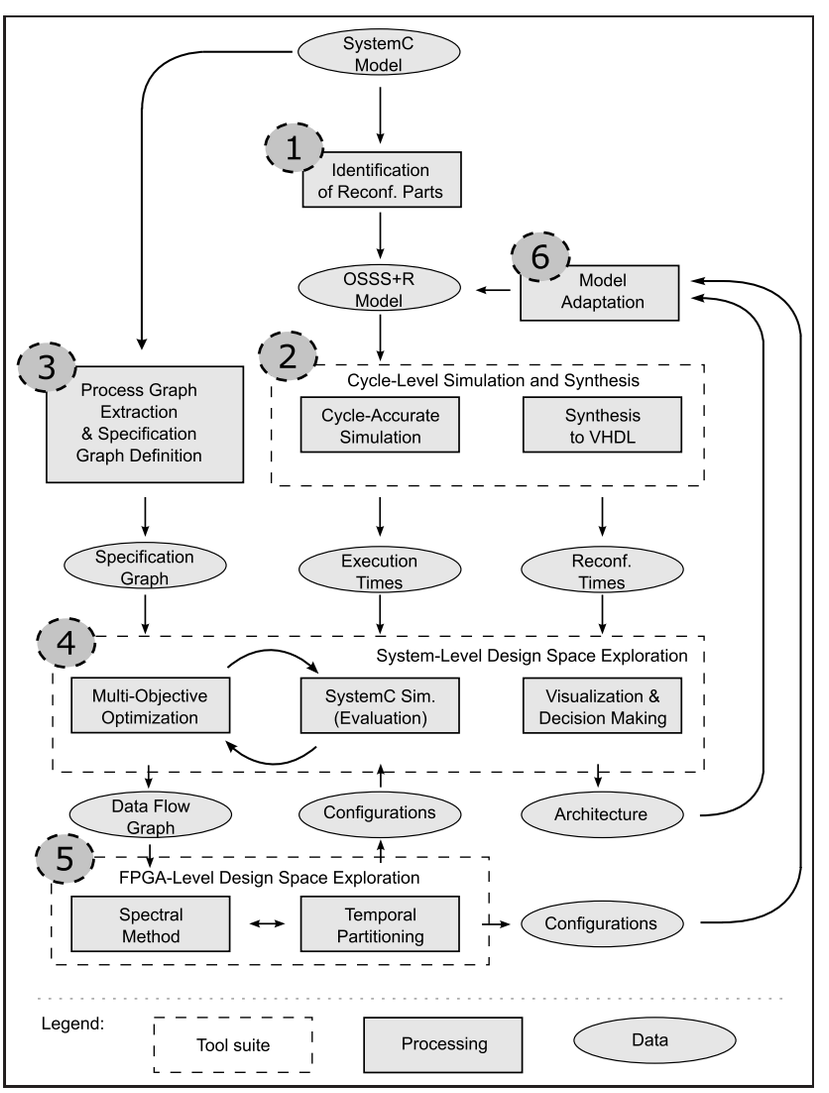

Figure 1 The proposed design flow.

spectral method (see Section 3). Finally, the results are passed back to the system-level design space exploration.

The result of the system-level design space exploration is a set of Pareto-optimal solutions. However, due to the nature of a quick exploration, the generated scoring of these solutions is not as accurate as one generated by an in-depth examination. Therefore, the generated architecture graph and the selected configurations are fed into a model adaptation phase (step 6). The resulting model is an executable specification honoring the suggested implementation advices. However, this model does not cover software parts. A further analysis by OSSS+R simulation and synthesis to bitstreams gives more detailed cost figures. Based on this information, a reliable selection among the remaining solutions can be made.

1.2 Related Work

This paper presents a novel approach for a tool-assisted design flow including system-level and FPGA-level design space exploration as well as the synthesis of complex systems with dynamically reconfigurable resources. Many publications related to each of the subtasks exist. However, to the best of our knowledge, no seamless design flow considering all these aspects exists today. Hence, we discuss related work of the phases of our design flow only.

Different system-level design space exploration tools are documented in the literature, cf. [8; 11]. Many of them use Multi-Objective Evolutionary Algorithms to perform the automatic exploration. Some of the tools also use simulation-based evaluation during exploration. However, none of them supports dynamically reconfigurable hardware in the exploration.

Temporal and spatial partitioning approaches focusing the execution of the partitioned algorithms on FPGAs can be found, e. g., in [1; 4; 7; 13; 15]. These are all sophisticated approaches to explore the runtime reconfiguration capabilities of modern FPGAs. However, only [4; 13] focus on optimization of communication between partitions, i. e., reducing FPGA routing complexity. In this work, we enhance the approach of [4] by specifically considering precedence relations.

There have been several approaches to model reconfigurable hardware. For example, both simulation and synthesis are supported by JHDL [2], a Java based language which allows a structural design description. The level of abstraction is rather low, which is a disadvantage during system-level design.

Due to the increasing popularity of SystemC, we base our methodology on this modeling framework. SystemC is a C++ based simulation library. Basically, a SystemC model is a C++ program that can be compiled using standard C++ compilers and linked with the simulation library. Executing the binary results in a simulation of the system.

In the following, we will present the three phases of our methodology in more detail, before applying the methodology to an example.

2 System-Level Design Space Exploration

The goal in Design Space Exploration (DSE) is to find an optimized allocation of hardware resources and an optimized binding of processes onto these resources. The hardware resources to be considered at system-level are typically processors, ASICs, coprocessors, busses, FPGAs with configuration bit streams, etc. A very promising approach to automatic DSE is to start from a specification including a model of the application as well as a model of the architecture template. Moreover, mapping constraints must be modeled to specify that a process might be implemented on a given resource. Such a system model is called a *specification graph* in the following. The specification graph consists of a so-called *process graph* that models the application to be implemented by means of communicating processes, the so-called *architecture graph* which models the architecture template by means of connected resources, and the *mapping edges* which model that a given process can be implemented on the connected resource (see [3]). This specification graph is the input to the automatic design space exploration as depicted in Fig. 1. The process graph is usually an abstraction from some executable application description. In the following, we will assume that the process graph represents an application given in the system description language SystemC, which is composed of modules connected by channels. From such a SystemC description, a process graph can be extracted automatically. To define an appropriate architecture template and mapping constraints, the designer usually defines some processors and additional hardware IP cores which are connected using a specific communication structure. Note that the communication infrastructure in general depends on the selected processor modules. From a synthesis point of view, it is mandatory to resort to specific platforms, i. e., specific processors and communication media. Moreover, hardware IPs should be an one-to-one transformation of SystemC modules. Such transformation can be done manually or automatically using high-level synthesis tools. Note that defining the mapping constraints requires either a synthesis step or an estimation of properties such as area and power consumption, latency, throughput, etc. A more detailed discussion on defining the specification graph and performing automatic synthesis is provided in [12].

Given a specification graph, the task of system-level design space exploration can be formally defined as the task of selecting nodes from the architecture graph, i. e., resources are allocated, and map-

ping edges are selected such that (1) each process is connected to exactly one resource through a mapping edge and (2) communicating processes are bound to the same or to directly connected resources. The latter requirement ensures that the communication demanded by the application can be handled by the allocated architecture. As there exists more than a single possible solution to this underlying decision problem, we have to optimize the solutions with respect to some objectives as area, power consumption, etc.

As typically more than a single objective function has to be optimized during system design, there does not exist a single optimal solution in general. Thus, to be precise, our goal in DSE is to find the set of *Pareto-optimal solutions*. In order to perform automatic system-level DSE exploration, so-called *Multi-Objective Evolutionary Algorithms* (MOEAs) have been proven to perform well. In our proposed design flow, we use an automatic DSE framework called SystemCoDesigner [11]. SystemCoDesigner uses so-called *hierarchical architecture graphs* to model the mutual exclusion of different configurations loaded on an FPGA. An example of an FPGA with three associated configurations is shown in Fig. 2(a). The corresponding hierarchical architecture graph is shown in Fig. 2(b).

The DSE using MOEAs can be sketched as follows: A so-called *population* of *individuals* is optimized iteratively during exploration. Each individual represents a solution and is encoded as a so-called *chromosome*. In our case, the chromosome consists of two parts. The encoding of the allocation of resources and configurations is done using bit strings. Each bit corresponds to a resource or a configuration indicating its allocation (1) or deallocation (0). The binding is encoded in a set of priority lists. To each process an ordered list is associated which sorts the outgoing mapping edges. Mapping edges are selected from these lists according to (i) their priority (the position in the list) and (ii) their ability to contribute to a feasible solution. Both, the bit string and the priority lists, are changed from iteration to iteration through genetic operations, i. e., mutation and crossover. Only the fittest solutions according to a given fitness function are placed in the next population (iteration).

In Multi-Objective Optimization, the fitness function depends on the objective values of a solution as well as the objective values of other solutions in order to generate Pareto-optimal and highly diverse solutions. In particular, to determine the objective values like latency or throughput, in our design methodology, we integrate a SystemC simulation. Here, our VPC-approach (Virtual Processing Components) is used to simulate the timing behavior of a complex system at the system-level with a task accurate granularity [17]. In later design phases (model adaptation phase), we will use the OSSS+R library for a cycle accurate simulation of the dynamic hardware reconfiguration. In turn, this simulation will provide more accurate parameters for the system-level model.

However, to be useful in a seamless design flow, a designer needs to know good FPGA configurations to be considered during system-level design space exploration. For this purpose, we propose the integration of temporal partitioning algorithms into the design space exploration. This can be done by extracting the subgraph of the process graph possibly mapped onto an FPGA and construct optimized configurations for this partial application. This issue is discussed next.

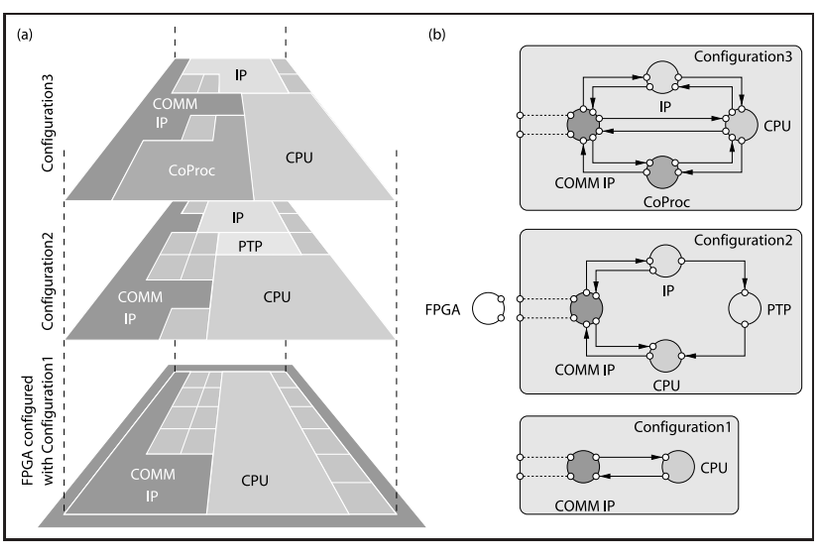

Figure 2 (a) FPGA with three associated configurations and (b) corresponding hierarchical architecture graph.

3 FPGA-Level Design Space Exploration

Run-time reconfiguration of FPGAs allows to reuse the same hardware for multiple processes, which are dynamically loaded as bitstreams on FPGAs. Moreover, algorithms divided into mutually exclusive configurations allow their execution even if the total area requirements exceed the available area.

An essential means for exploiting run-time reconfiguration is the adequate derivation of the configurations. For this purpose, we need a suitable method to partition the process graphs introduced in Section 2 into configurations. We identify the communication between the partitions as important costs, as we often have only a limited amount of communication resources passing

the partition boundaries. Additionally, storing of intermediate data should be reduced. Thus, we use a method to partition the data flow graphs with respect to communication requirements. This task of deriving optimal configurations is called the FPGA-level design space exploration (see Fig. 1). The methods applied are based on the spectral analysis of the input graphs as proposed in [10] for VLSI design. The method re-arranges graphs in space respecting the communication by a quadratic objective function of the distances between nodes (wire length model). Its output is a communication optimized placement suggestion.

For spectral placement, the input graph must be given as the Laplacian matrix B, derived from the connection matrix C and the degree matrix D ($B = D - C$). Our goal is to minimize the sum of squared distances between the nodes. Therefore, we apply the Lagrange multiplier method with the k Lagrange multipliers $\lambda_1, \lambda_2, \ldots, \lambda_k$. The solution are B's Eigenvectors associated to the k smallest non zero Eigenvalues. These Eigenvectors place the vertices in space, whose communication is now optimized concerning the wire length.

We use this result for clustering the graphs. Such clusters are meaningful references for the mutual exclusive configurations, as they comprise few external communication. Additionally, the location of the clusters in space helps to arrange the partitions on the FPGA. However, as the spectral method ignores the direction of edges and therefore does not respect precedence constraints, we use a combination of the spectral method and temporal scheduling in order to derive mutual exclusive configurations that can be loaded in sequence onto FPGAs.

Therefore, we have applied two different partitioning methods. While the first improves an idea presented for coarse grain reconfigurable devices [6], the second one requires an intermediate clustering step done via biologically inspired algorithms. Both use temporal scheduling methods to finally derive the partitions.

In detail, the first method uses the first two non zero Eigenvectors to derive the x and y coordinates of a geometrical arrangement of the tasks. As in [4], we use the third Eigenvector as reference for our temporal partitioning. Therefore, we iteratively derive bisections of the graph based on the ordering given by this Eigenvector. The data flow between each newly generated pair of subgraphs is homogenized by means of applying a modified version of the Kernighan Lin algorithm, which prefers to move tasks that reduce the amount of wrongly directed edges. Finally, two adjacent partitions will have edges in one direction only and a schedule using temporal algorithms can be derived.

In contrast, the second method is based on two Eigenvectors and a biologically inspired clustering methodology. The spectral analysis serves as basis for a biologically inspired intermediate clustering, which assigns the nodes to be clusterhead or member of cluster according to division of labor ants. The stimulus of deciding the membership is calculated by virtue of the distance of nodes in the spectral arrangement. Resource limitations help us to reduce the search space. After deriving the intermediate clusters, we decide on the final partitioning by referring to the temporal ordering of the task set. Clusters will become valid, if their precedence constraints can be fulfilled by already scheduled clusters. Otherwise, the violating tasks will be removed and added to future clusters by respecting resource constraints.

To conclude, we partition by referring to spacial (spectral method) and temporal (temporal algorithm) information. The configurations are mutually exclusive and can be distributed to the system-level DSE. In order to finally derive the best size for the configurations, we rely on the iterative behavior of our overall design methodology. After simulating and synthesizing, we gain trace or gate count information valuable for the next iteration within our design flow (ref. to Fig. 1). This approach of nested intervals allows to approximate the solution.

4 Simulation and Synthesis

In our design flow, OSSS+R serves as a representation of a given solution and is the base for evaluation (see Fig. 1). OSSS (Oldenburg System Synthesis Subset) [9] is a modeling library for SystemC.

Traditionally, hardware description languages only support static design elements. In short, hardware does not change over time. Additionally using the OSSS+R [16] library extends the set of available primitives already available in OSSS by further ones suitable for dynamic components. Using these primitives enables simulation of reconfigurable aspects (hence the +R in the name) at a high level of abstraction.

As soon as dynamic partial reconfiguration is used in a non-trivial way, resource conflicts are likely to occur. A device's reconfiguration port cannot be used to load multiple bitstreams on the device in parallel and the reconfigurable areas cannot contain multiple logic configurations in parallel. Therefore they form limited resources which may be used mutually exclusive only. OSSS+R has dedicated primitives for such resources. Borrowed from its C++ roots, it has class datatypes and therefore treats the resources as objects. Additionally, these special objects provide built-in scheduling abilities.

We model the tasks to be executed as *contexts*. Contexts are always members of SystemC modules with an infinite lifetime. From a functional point of view, they behave like ordinary C++ objects. The designer may express access to these objects anywhere in his code without restriction. During runtime however, only a subset of contexts is available per instant. This discrepancy is hidden to the designer by

automatically deciding which contexts to disable and which to enable while the system operates. Therefore, online scheduling is done by a distributed set of arbiters, which are inferred automatically. It is possible to select among a set of predefined scheduling algorithms or specify custom ones, e. g., to implement specifications as given by the FPGA-level exploration.

As a result of this approach, the programmer targets a virtual device which seems to be larger than the real device the functionality is mapped to. The device appears to provide all contexts in parallel but in fact switches between them on the fly. This is very much similar to the concept of virtual memory for microprocessors.

Note that this simulation is different from that in the system-level design space exploration phase. The former is more abstract and less timing-accurate in order to be fast. On the other hand, the OSSS+R simulation may even be cycle-accurate when given precise timing information for the reconfiguration phase. The precision is required for the optimizations in the FPGA-level design space exploration.

5 Simulated Architecture

We start with a given ordered set of partitions. For each task within the partition we annotated a core execution time and a reconfiguration time. Since the partitions are chosen in a way that there are no inter-partition dependencies between tasks, we completely configure all tasks of a given partition to the device. Once done, all tasks are started. Whenever a set of tasks is being executed, the next partition is concurrently configured to another device. As soon as both task execution and configuration are finished, the two FPGAs switch execution and configuration steps.

Configuration and execution of tasks are controlled from within the FPGA. This is where the simulation is done with a greater degree of detail than the simulation used in the exploration step. Here we consider task-interactions for scheduling and handshaking at a cycle-accurate level.

The smallest considered reconfigurable area on each FPGA is capable of containing one task. Associated with it is a control process requesting the demand for reconfiguration whenever a new task has to be prepared. This request covers a handshaking with a per-device infrastructure, that in turn performs arbitration and reconfiguration. Since there are multiple processes that may concurrently request reconfiguration, that infrastructure performs arbitration of requests, too.

After configuration, each control process uses the single bus to fetch its input tokens. The bus is shared, and therefore has an arbiter, too. The data tokens calculated by a task are consumed by other tasks. Since the tokens get destroyed by reconfiguration, they must have been fetched prior to this. This is possible, if the consumer is located in the same or subsequent partition. For consumers in later partitions, the tokens are stored in a dedicated location for that purpose. In that case both storing and fetching data involve a bus transaction and therefore consume time.

6 Results

The integration of the above proposed evaluation and simulation steps is done on an exemplary design (see Fig. 3). In order to test the integration, we apply generated process graphs using the TGFF framework [5] and define manually architecture graphs. Estimations for execution and reconfiguration times are provided again by the TGFF framework. The resulting specification graph is used as input to the system-level de-

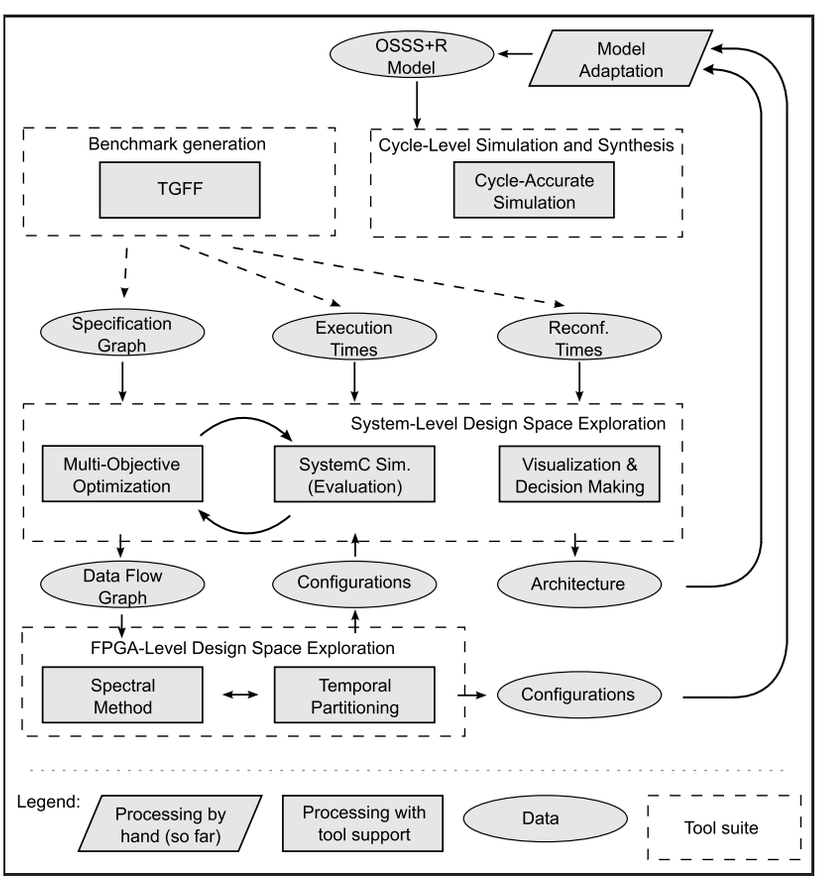

Figure 3 Parts of design flow tested in this work.

sign space exploration where in a first step, optimized configurations are determined by applying the spectral method. Selected solutions from the system-level design space exploration are assessed using OSSS+R where reconfiguration is simulated on a cycle-accurate basis. In the following, we present first results which are promising and motivate us to further work on our integrated design methodology.

In detail, we used an example with the following parameters: Initially, a process graph was generated by the TGFF framework consisting of 60 processes. Additionally, TGFF provides a randomly constructed arc set representing data dependencies, which are modeled by 80 communication processes in the process graph. Our architecture model uses 491 nodes to represent three busses connecting 8 FPGAs where each FPGA holds 60 hardware modules for task execution distributed over up to 9 configurations. The processes are annotated by TGFF with types for which in turn execution and configuration times are provided. Additionally, each data dependency consumes time. 720 mapping edges are contained in this particular example resulting in 10^{92} possible bindings which prohibits exact methods in design space exploration.

The system-level design space exploration using SystemCoDesigner was performed using the following parameters: 100 children are created from 100 parents in each iteration. The population size is 400 individuals. Three objectives, namely area, power consumption, and latency have been minimized. After 400 iterations, we obtained 14 non-dominated individuals. The exploration required about 12 hours on a Linux workstation with a 3200 MHz Pentium™ 4 Processor and 1 GB of RAM.

Thereby, the FPGA-level design space exploration was used to generate configurations for the FPGA nodes. Both techniques described in Section 3 produced reasonable results that could be fed back to the system-level design space exploration.

Finally, OSSS+R benchmarks for the most promising individuals were generated. Their detailed timing analysis returns two values each. The first value represents an initial iteration, assuming completely invalidated reconfigurable areas. The second value describes subsequent iterations. These iterations show a lower latency since some partial configuration steps may be omitted. This is the case when task types for one location match in the first and the last step of an iteration.

7 Conclusion

In this paper we presented a concept for a tool-assisted design space exploration approach for systems containing dynamically hardware reconfigurable resources. We start with a system-level description. The flow assists the designer by distributing the tasks onto multiple devices. Additionally, the configurations designated to be implemented on reconfigurable devices are enhanced to make use of the reconfiguration capability. The final output is an implementation proposal and aids the designer during synthesis.

References

[1] K. Bazargan, R. Kastner, and M. Sarrafzadeh. Fast Template Placement for Reconfigurable Computing Systems. In: *IEEE Design & Test of Computers – Special Issue on Reconfigurable Computing*, Jan–Mar 2000.

[2] P. Bellows and B. Hutchings. JHDL – An HDL for Reconfigurable Systems. In: *IEEE Symp. on FPGAs for Custom Computing Machines*, 1998.

[3] T. Blickle, J. Teich, and L. Thiele. System-Level Synthesis Using Evolutionary Algorithms. In: *Design Automation for Embedded Systems*. Kluwer, Jan. 1998.

[4] C. Bobda. *Synthesis of Dataflow Graphs for Reconfigurable Systems using Temporal Partitioning and Temporal Placement*. PhD thesis, University Paderborn, Heinz Nixdorf Institute, 2003.

[5] R. Dick, D. Rhodes, and W. Wolf. TGFF: task graphs for free. In: *Proc. Int'l Workshop Hardware/Software Codesign*, 1998.

[6] F. Dittmann and C. Bobda. Temporal placement on mesh-based coarse grain reconfigurable systems using the spectral method. In: *From Spec. to Embedded Sys. Application, Proc. of the IESS*. Kluwer, Aug. 2005.

[7] S.P. Fekete, E. Köhler, and J. Teich. Optimal FPGA module placement with temporal precedence constraints. In: *Proc. DATE 2001, Design, Automation and Test in Europe*, March 2001.

[8] M. Gries. Methods for Evaluating and Covering the Design Space during Early Design Development. In: *VLSI Jour.*, 38(2):131–183, 2004.

[9] E. Grimpe, B. Timmermann, T. Fandrey, R. Biniasch, and F. Oppenheimer. SystemC Object-Oriented Extensions and Synthesis Features. In: *Forum on Design Languages FDL '02*, Sept. 2002.

[10] K.M. Hall. An *r*-dimensional Quadratic Placement Algorithm. In: *Managment Science*, 17(3), Nov. 1970.

[11] C. Haubelt. *Automatic Model-Based Design Space Exploration for Embedded Systems – A System Level Approach*. PhD thesis, University Erlangen-Nuremberg, Germany, July 2005.

[12] C. Haubelt, J. Falk, J. Keinert, T. Schlichter, M. Streubühr, A. Deyhle, A. Hadert, and J. Teich. A SystemC-based Design Methodology for Digital Signal Processing Systems. In: *EURASIP Journal on Embedded Systems, Special Issue on Embedded Digital Signal Processing Systems*, 2007.

[13] M. Kaul and R. Vemuri. Optimal temporal partitioning and synthesis for reconfigurable architectures. In: *Proc. of Design, Automation and Test in Europe*, 1998.

[14] P. Lysaght, B. Blodget, J. Mason, J. Young, and B. Bridgeford. Enhanced Architectures, Design Methodologies and CAD Tools for Dynamic Reconfiguration on XILINX FPGAs. In: *Proc. of the FPL 2006*, Madrid, Spain, 2006.

[15] K.M.G. Purna and D. Bhatia. Temporal Partitioning and Scheduling Data Flow Graphs for Reconfigurable Computers. In: *IEEE Trans. Comput.*, 48(6), 1999.

[16] A. Schallenberg, F. Oppenheimer, and W. Nebel. OSSS+R: Modelling and Simulating Self-Reconfigurable Systems. In: *Proc. of Int'l Conf. on Field Programmable Logic and Applications*, Aug. 2006.

[17] M. Streubühr, J. Falk, C. Haubelt, J. Teich, R. Dorsch, and T. Schlipf. Task-Accurate Performance Modeling in SystemC for Real-Time Multi-Processor Architectures. In: *Proc. of Design, Automation and Test in Europe*, Mar. 2006.

1 2

3 4

5 6

1 Dipl. Inform. Florian Dittmann is research assistant at the Design of Distributed Embedded Systems research group of the Heinz Nixdorf Institute at the University of Paderborn.
Address: Heinz Nixdorf Institute, University of Paderborn, Fürstenallee 11, 33102 Paderborn, Germany, Tel.: +49-5251-60-6492, Fax: +49-5251-60-6502,
E-Mail: roichen@upb.de

2 Prof. Dr. rer. nat. Franz Rammig is the head of the Design of Distributed Embedded Systems research group of the Heinz Nixdorf Institute at the University of Paderborn.
Address: Heinz Nixdorf Institute, University of Paderborn, Fürstenallee 11, 33102 Paderborn, Germany, Tel.: +49-5251-60-6500, Fax: +49-5251-60-6502,
E-Mail: franz@upb.de

3 Dipl.-Inf. Martin Streubühr is research assistant at the System-level Design Automation group in the Department of Hardware-Software-Co-Design at the University of Erlangen-Nuremberg.
Address: University of Erlangen-Nuremberg, Am Weichselgarten 3, 91058 Erlangen, Germany, Tel.: +49-9131-85-25155, Fax: +49-9131-85-25149,
E-Mail: streubuehr@cs.fau.de

4 Dr.-Ing. Christian Haubelt leads the System-level Design Automation group in the Department of Hardware-Software-Co-Design at the University of Erlangen-Nuremberg.
Address: University of Erlangen-Nuremberg, Am Weichselgarten 3, 91058 Erlangen, Germany, Tel.: +49-9131-85-25154, Fax: +49-9131-85-25149,
E-Mail: haubelt@cs.fau.de

5 Prof. Dr.-Ing. Wolfgang Nebel is professor for VLSI design at the Computing Science Department of Oldenburg University. Additionally he is chairman of the OFFIS research institute.
Address: CvO University Oldenburg, Ammerländer Heerstrasse 114–118, 26121 Oldenburg, Germany, Tel.: +49-441-9722-280, Fax: +49-441-9722-282,
E-Mail: Wolfgang.Nebel@Informatik.Uni-Oldenburg.de

6 Dipl. Inform. Andreas Schallenberg is a member of the University's System Design Methodology research group at the Department of Computing Science of Oldenburg University.
Address: CvO University Oldenburg, Ammerländer Heerstrasse 114–118, 26121 Oldenburg, Germany, Tel.: +49-441-9722-166, Fax: +49-441-9722-282, E-Mail: Andreas.Schallenberg@Uni-Oldenburg.de