rSesame - A Generic System-Level Runtime Simulation Framework for Reconfigurable Architectures

Kamana Sigdel[†] Mark Thompson[‡] Carlo Galuzzi[†] Andy D. Pimentel[‡] Koen Bertels[†]

[†] Computer Engineering Laboratory
EEMCS, Delft University of Technology
The Netherlands
Email: {K.Sigdel, C.Galuzzi, K.L.M.Bertels}@tudelft.nl

[‡] Computer Systems Architecture Group
University of Amsterdam
The Netherlands
Email: {M.Thompson, A.D.Pimentel}@uva.nl

***Abstract*—As reconfigurable architectures are gaining an increasing research and industrial attention, there is a significant need for intelligent tools and methodologies to assist designers with exploration and performance evaluation of such architectures. Towards this goal, we make a first attempt to present a generic system-level modeling and simulation framework which can explore and evaluate reconfigurable systems at design-time as well as at runtime. In this paper, we show the methodology behind the framework and study its key features.**

I. Introduction

Reconfigurable architectures are becoming increasingly popular as they bear a promise of combining the flexibility of software with the performance of hardware. These architectures are typically formed with a combination of a General Purpose Processor (GPP) and a Reconfigurable hardware e.g. an FPGA. Computational intensive application fragments can be mapped onto the FPGA to speed-up their execution while control intensive application fragments can be mapped onto the GPP. As reconfigurable architectures are gaining popularity in acedemia and industry, there is a significant need for intelligent tools and methodologies to assist the designers with the exploration and performance evaluation of such architectures.

Design Space Exploration (DSE) tools and methodologies help designers to systematically explore trade-offs between various design choices for hardware/software partitioning, application-to-architecture mapping, task scheduling, task allocation, etc. The current state-of-the-art shows a great variety of tools and methodologies used for carrying out DSE for dynamically reconfigurable systems. These methodologies can loosely be categorized based on two aspects: 1) the approach used to evaluate a design candidate and 2) the stage at which this evaluation is done (see Figure 1). We identify two main evaluation approaches for DSE: the algorithmic ad-hoc approach and the framework-based modeling/simulation approach. In the former approach, algorithms of various computational complexity are used to quickly evaluate a set of criteria for a set of alternative designs. This typically consists of a custom evaluator (analytical) together with a (partial) simulator. These approaches may obtain the required results quickly. However, these results are difficult to reproduce in case of comparison and/or re-evaluation. Such algorithms include: dynamic programming, branch and bound, integer linear programming, graph partitioning, simulated annealing, genetic algorithm, ant colony optimization [1], [2], [3], [4], [5], etc. Similarly, in the framework-based modeling and simulation approach, a standardized framework is used to model, simulate and explore reconfigurable system behavior at various design stages. Such a framework offers modeling methodologies and simulation tools to evaluate different design criteria. Advantages of such approaches include higher model (component) re-usability and easy customization of the design according to the system requirements. This provides a standardized framework for developing benchmarks standards which 1) can perform fast and easy comparison between various competing alternatives, and 2) can provide easy evaluation and testing of various model components. Examples of such approaches can be found in [9], [10], [11] and [12].

[0]This research has been funded by the hArtes project EU-IST-035143, the Morpheus project EU-IST- 027342 and the Rcosy Progress project DES-6392.

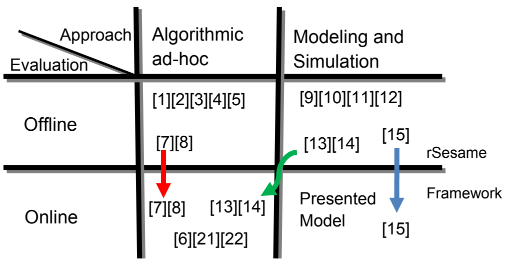

Fig. 1. Classification of DSE Tools and Methodologies

The approaches used to evaluate a design candidate are further categorized based on the evaluation time: offline and online. Offline evaluation refers to the condition where a design candidate is evaluated for fixed system constraints. Offline evaluation is often performed at design-time where no changes in the system (application, architecture and/or environment) are given as feedback to the evaluation process. Examples of such evaluation can be found in [1], [2], [3], [5] and [4]. Alternatively, online evaluation refers to runtime evaluation of the design candidates where a design candidate is evaluated for (dynamically) varying system constraints. Any changes in the system (application, architecture and/or environment) are given as a feedback to the evaluation process. As a consequence, the design parameters are adjusted during the evaluation based on the changes encountered by the system. In [6], the authors present a simple approach for online evaluation of the task mapping in which a mapping module evaluates the most frequently executed tasks at runtime and maps them onto reconfigurable hardware. This work [6], however, focuses on the low design level and targets only loop kernels. Similar approaches for high-level runtime evaluation of application mapping are presented in [7], [8], [20] and [21]. Offline evaluation is generally faster, however, it can be less accurate as the runtime behavior of a system is mostly captured by offline (static) estimations. On the other hand, the online evaluation of the design candidates can be more accurate but it is typically harder due to the enormous size of the search space generated by the runtime system parameters.

This classification shows that the trend of most current DSE research efforts focuses either on the top two quadrants or the left two quadrants in Figure 1. Even though there is no absolute separation between these quadrants and some efforts overlap each other (the arrows in the classification Figure 1). For example, the approaches in [13] and [14] combine a design time exploration together with runtime management to trade-off faster exploration with accuracy. There is still a huge gap for tools and methodologies that fit the bottom-right quadrant in the classification diagram. To the best of our knowledge, there is no existing standardized modeling and simulation infrastructure for DSE which allows designers to model and evaluate reconfigurable systems' behavior at runtime. As a consequence, various research groups rely on their custom-built proprietary models/simulators for evaluating architectures and algorithms. This results in the following problems: a) the evaluation procedure is immensely complex and b) the comparison between different evaluations is extremely hard. To address these problems, it is crucial to have a standard modeling and simulation infrastructure to evaluate reconfigurable systems, which can be re-used between different research groups. Such an infrastructure could provide a standard platform allowing easy comparison between various evaluations and hence it can also be used as a reference tool for future research. We believe, providing such a standardized framework will provide an excellent vehicle for research by adding a tremendous breakthrough in this particular research domain. At the same time, it could also provide invaluable insight to future generations of (partially) dynamically reconfigurable systems from the academic research perspective. A parallel example can be found in the field of micro-architecture with the revolution brought by the Simplescalar [19] infrastructure in the academics as well as the industrial research practice.

Towards this goal, we make a first attempt towards this goal and presented a generic modeling and simulation infrastructure which can explore and evaluate reconfigurable systems at both design-time and runtime [15]. In [16], we presented a model for system-level DSE, which can perform rapid exploration of different reconfigurable design alternatives at design time. Following up this approach, in [15], we introduced a modeling and simulation technique for two level mapping exploration of reconfigurable systems, which can explore a system at design time as well as at run time. In this paper, we extend and generalize the technique from [15] to form a generic modeling and simulation framework. Furthermore, we demonstrate the methodology behind the framework by instantiating an example model for a generic reconfigurable architecture . We also describe the behavior of the model and study its key features. The main contributions of this paper are as follows:

- Extension of the modeling and simulation framework presented in [15] with detailed elaboration of the runtime mapping manager in order to make mapping decisions of reconfigurable systems at runtime;
- Instantiation of an example model instantiated from the framework for a generic reconfigurable architecture and the study of its key features.

II. Modeling Runtime Mapping Behavior

The modeling of a system that allows runtime mapping of tasks to processing resources can be divided in two parts: the *spatial* and the *temporal* mapping behavior. *Spatial* mapping is the process of identifying which part of the application can be implemented on the reconfigurable hardware (HW tasks) and which part should be executed as software (SW tasks). Not all HW tasks may fit on the reconfigurable device at the same time. Therefore, a logical *configuration* has to be defined for a set of HW tasks for which the functional logic has been loaded on the reconfigurable device at a given moment. *Temporal* mapping is the process of determining a sequence of these configurations at runtime, such that all HW tasks can run efficiently. In summary, spatial mapping determines *where* to map a task and temporal mapping determines *when* to map a task.

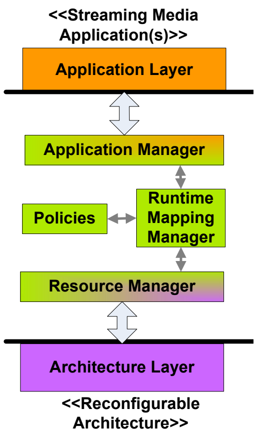

Fig. 2. The runtime mapping manager incorporated within rSesame framework.

With the rSesame framework, a task can be modeled either as a HW task or as a SW task. A task tagged as HW is always mapped onto the hardware component of the architecture and a task tagged as SW is always mapped onto its software counterpart. Task assignment for the SW and HW categories is done at design time. At runtime, these tasks are mapped onto their corresponding resources based on time, architectural resources and conditions of the system. If more than one task is mapped onto the RP and if all of them do not fit on the RP at once, they are divided into logical configurations and mapped sequentially. In this way the temporal mapping is addressed at runtime with the rSesame framework.

In order to model the spatial mapping behavior at runtime, the framework supports a third task type: *pageable*. Unlike HW and SW tasks, a task tagged as pageable does not have a fixed spatial mapping. Basically, a pageable task can be both a HW task and/or a SW task and it can be mapped onto any of the resources. For pageable tasks, the spatial mapping decision is made at runtime. These tasks are mapped either as HW or SW depending on the runtime conditions of the system (e.g. resource availability).

For HW and SW tasks, only the decision of *when* to map is made at runtime and for the pageable tasks the decisions of *when* and *where* to map are made at runtime. In other words: without pageable tasks, only the temporal mapping behavior takes place at runtime, whereas with pageable tasks, both temporal and spatial mapping take place at runtime. Designers can play with these task sets depending on the system requirements and their evaluation purpose (online and offline).

III. Runtime Mapping Manager

In any kind of reconfigurable system, in order to perform mapping of application tasks onto architectural components at runtime, there is a need for a runtime decision making entity. Within our framework we defined the *Runtime Mapping Manager* (RMM) component to make *intelligent* mapping decisions based on the dynamic system conditions and according to a specified policy. Most importantly, it has to identify whether the current mapping is sufficient to satisfy the given system constraints or not. If not, then it should also be able to identify for which tasks to change the mapping. Note that such a mapping change may in turn affect the spatial and temporal mapping of the entire application. Therefore, it is crucial for the RMM to understand both the application requirements and the architecture behavior. Figure 2 provides an abstract outline of the structure of an rSesame system model and shows the location and interaction of the RMM. This structure is the basis for our proposed framework, which we believe to be generic enough to match many other industrial and academic platforms with runtime mapping management, thereby enabling wide range deployment of our framework. Consider for example the case that the runtime manager is placed between the application and the architecture (see again Figure 2). In the actual system implementation the RMM entity may be part of the application, middleware, operating system or even implemented as a hardware component, without loss of generality of the model created within the rSesame framework. In the following part, we present a general description of a each component involved in the runtime mapping management and their respective roles and responsibilities:

- Application Manager : interacts with the application layer and monitors any changes in the application user requirement, arrival of sporadic tasks, priority of tasks, real time constraints, etc. It deals with these task requirements and administers QoS management e.g. in case to ensure the execution at a particular frame rate to maintain required image quality. The application manager is a platform independent component and is only responsible for dealing with applications. In case of any changes in the application, it notifies the RMM for optimizing the mapping decision.
- Resource Manager: is a platform dependent component and gathers information about the architecture platform. It administers the architecture behavior and provides various architectural information to the mapping manager such as free resources, timing information, etc. In case of any changes in the architecture (e.g. component failure, power safe mode), it reflects these changes to the RMM for optimizing the mapping decision.
- Runtime Mapping Manager: is responsible for making actual mapping decisions. Based on application information from the application manager and architecture information from the resource manager, the mapping manager finds the “best” mapping at runtime. The mapping manager can be designed to learn from its historical data, predict future requirements and can even employ several mapping policies to optimize the mapping. These mapping policies are implemented as a modular component, and as a result, a variety of policies can be easily plugged in and out of the system.

IV. rSesame Framework

rSesame is a generic framework in the sense that it is not restricted to modeling one type or class of reconfigurable systems. Instead it can be deployed to model and evaluate any kind of reconfigurable architecture running a wide set of streaming applications from the multimedia domain. Using rSesame, a designer can instantiate a model for their architecture and any additional architectural specifics can be augmented in the model as required. Figure 3 presents a scenario where several models are instantiated from the rSesame framework for different types of reconfigurable architectures running different sets of applications. In Section V an example (shown in Figure 4) will be given of one such instance for a specific architecture running a particular application.

The rSesame framework employs the Sesame [17] framework as a modeling and simulation platform for system level DSE.

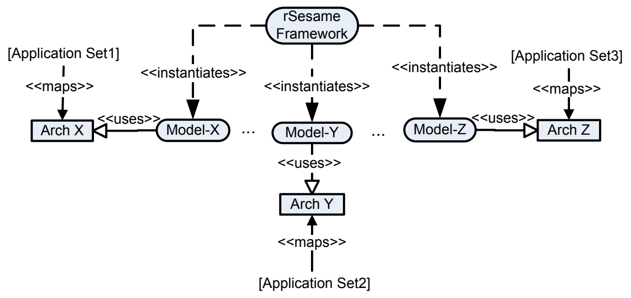

Fig. 3. Instantiation of the rSesame framework for various architectures

Sesame adheres to a transparent simulation methodology where the concerns of application and architecture modeling are separated via a mapping layer, and as a result, applications and architectures can be independently modeled. For the application modeling, we use Kahn Process Networks (KPNs) [18] at the granularity of coarse-grain tasks. A KPN consists of concurrent processes with explicit communication over FIFO channels. KPNs are deterministic and can capture the parallel and dynamic nature of streaming applications in the multimedia domain, which is our target domain. These processes contain functional application code together with annotations that generate events (Read, Write and Execute) describing the actions of the process. These events are collected as traces and are forwarded onto architecture layer using an intermediate mapping layer.

The mapping layer consists of Virtual Processors (VPs) that schedule the event traces from the application processes onto the architectural components. These VPs can be considered as a distributed form of the application manager as they bring information about the application to the RMM. The RMM is modeled as a separate mapping layer component (Figure 4). Before forwarding events from the application to the architecture, a VP asks the RMM on which processor to execute the event. Based on the current system conditions and the policy implemented, the RMM returns a target processor identifier (either the CCU or the GPP) and the VP forwards events accordingly. In this way, the RMM can model spatial and temporal mapping decisions at runtime. To support its decision making process, the RMM may request additional information about the architecture from the Resource Manager (RM) or application information from the VPs. The RMM can employ an arbitrary set of user-defined policies for runtime mapping. These policies can be simply plugged in and out of the RMM, making the framework flexible to evaluate such policies. The RM can be modeled as an architectural specific component and can reside on the architecture layer. Based on the architecture modeled, the RM may have several specific functions defined and it may provide different architecture information.

The architecture layer in the framework models the architectural resources and constraints. These architectural components are constructed from generic building blocks provided by a library which contains components for processors, memories, on-chip networks components and so on. Thus, any kind of reconfigurable architecture can be constructed from these generic components. Besides the regular parameters such as computation and communication delays, other architectural parameters like reconfiguration delay and area for the reconfigurable architecture, can be provided as extra information to these components.

V. Model Instantiation

Figure 4 depicts an example of a model that can perform runtime task mapping for a dynamic reconfigurable architecture. This architecture consists (in this case) of a GPP and a dynamically Reconfigurable Processor (RP). Application tasks can be executed

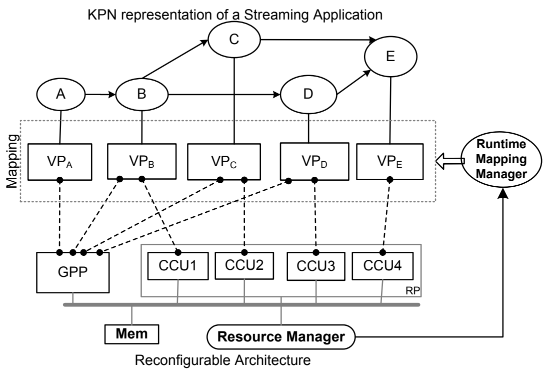

Fig. 4. Instantiation of the rSesame for a generic reconfigurable architecture

on the GPP, on the RP or on both of them. The RP can contain one or more computing units. Tasks to be accelerated on the RP are mapped onto these units. Tasks run on the GPP as regular (compiled) microprocessor code or on the RP as a hardware IP core.

In the example model, the GPP and the RP are modeled as generic processor components with some additional parameters such as area occupancy and reconfiguration delay (in case of the RP). A CCU represents an RP's custom computing unit (see Figure 4). The RMM resides between the VPs and the architecture layer. It collaborates with the RM for performing runtime mapping decisions. The RM is modeled as part of the architecture layer and it keeps track of the resource information (e.g. available area). It does this by monitoring which custom units are *configured* on the RP at any given time. A CCU can only process events when it has been configured. If it is not configured, it requests the RM to be scheduled for configuration. The RM configures CCUs according to the policy implemented (eg. as soon as area becomes available).

In general we can summarize the key features of a model created with the rSesame framework as follows:

- **Flexibility by Modular Design** : In the presented model, an application model is independent of the architectural specifics. As a result, application and/or architecture models can be re-used and altered without influencing each other. This separation of concerns makes it easier to accommodate any kind of modification to the model, permitting design variations and even completely different architectures to be modeled with ease. Moreover, a well defined structure of the RM and the RMM makes the flexible model. Therefore, any kind of mapping policy can be plugged into the model and can be evaluated without affecting other components.
- **Performance by Abstraction** : The applications are modeled at the granularity of tasks where application behavior is abstracted as read, write and execute events. The model operates based on discrete-event simulation of these events leaving out all the minute details that might otherwise hinder the model's performance. Therefore, we provide easy construction of the models and fast simulation. There is always a trade-off between detailed modeling and fast model performance and, in this case, we compromise details for performance. This is a fair trade-off for a system-level model which is targeted at very early design stages. At this level, where the design space is enormous, quick exploration is more vital than detailed exploration.
- **Ease of Use** : The modular design and the higher abstraction level together facilitate the ease of use of the model. In case of application modeling, the Kahn application models can be either automatically converted or manually derived from sequential C/C++ code. Similarly, any kind of architecture model can be constructed from generic building blocks provided by a library, which contains templates for processors, memories, on-chip networks and so on. Moreover, any policy can be implemented with a minimal effort and without having detailed knowledge of all parts of the model. Thus, the learning curve is rather moderate.
- **Input/Output** : The input given to the model consists of various architecture parameters such as hardware and software latencies, area and reconfiguration delay. The model simulates the application characteristics, architecture responses and the runtime spatial and temporal mapping behavior. As a result, it produces various system evaluation attributes such as performance, number of reconfigurations, mapping behavior of a task, area utilization, number of HW and SW tasks, percentage of HW and SW execution, etc.

VI. Conclusion and Future Work

In this paper, we presented a generic modeling and simulation framework for runtime task mapping for reconfigurable architectures. We instantiated an example model from the framework for a generic reconfigurable architecture and explained the methodologies behind the framework. We discussed the key features of the model and studied its benefits such as flexibility, performance and ease of use, etc. In future work, we will evaluate these key features of the framework by simulating and by comparing a range of different run-time heuristics from various domains.

References

[1] D. N. Rakhmatov et al., "Hardware-software bipartitioning for dynamically reconfigurable systems", *Proc. of CODES02*, 2002, pp. 145–150

[2] K. S. Chatha et al., "An iterative algorithm for partitioning and scheduling of area constrained HW-SW systems", *Proc. of IFIP99*, 1999, pp. 134–139.

[3] K. B. Chehida et al., "HW/SW partitioning approach for reconfigurable system design", *Proc. of CASES02*, 2002, pp. 247–251.

[4] K. M., G. Purna, and D. Bhatia, "Temporal partitioning and scheduling data flow graphs for reconfigurable computers", *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 48, no. 6, pp. 579–590, 1999.

[5] Y.-C. Jiang et al., "Temporal partitioning data flow graphs for dynamically reconfigurable computers", *IEEE Trans. on VLSI Systems*, vol. 15, no. 12, pp. 1351–1361, 2007.

[6] G. Still et ak, "Dynamic hardware/software partitioning: A first approach", *Proc. of DATE03*, 2003.

[7] A. Kumar et al., "Resource manager for non-preemptive heterogeneous multiprocessor system-on-chip", *Proc. of ESTIMedia06*, 2006, pp. 33–38.

[8] O. Moreira et al., "Online resource management in a multiprocessor with a network-on-chip", *Proc. of SAC07*, 2007, pp. 1557–1564.

[9] J. Noguera et al., "System-level power-performance trade-offs in task scheduling for dynamically reconfigurable architectures", *Proc. of CASES03*, 2003, pp. 73–83.

[10] P.-A. Hsiung et al., "Perfecto: A systemc-based design-space exploration framework for dynamically reconfigurable architectures," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 1, no. 3, pp. 1–30, 2008.

[11] T. Rissa et al., "System-level modelling and implementation technique for run-time reconfigurable systems" *Proc. of FCCM02*, 2002, pp. 295.

[12] Y. Qu et al., "Systemc-based design methodology for reconfigurable system-on-chip", *Proc. of Euromicro Conference on DSD05*, 2005, pp. 364–371.

[13] C. Ykman-Couvreur et al., "Design-time application exploration for mp-soc customized run-time management", in *Proc. of SoC05*, 2005, pp. 66–69.

[14] V. Nollet et al., "Run-time management of a mpsoc containing fpga fabric tiles," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 16, no. 1, pp. 24–33, 2008.

[15] K. Sigdel et al., "System-level runtime mapping exploration of reconfigurable architectures", *Proc. of RAW09*, May 2009.

[16] K. Sigdel et al., "System-level design space exploration of dynamic reconfigurable architectures," in *Proc. of SAMOS08*, July 2008, pp. 279–288.

[17] A. D. Pimentel et al., "A systematic approach to exploring embedded system architectures at multiple abstraction levels," *IEEE Trans. Comput.*, vol. 55, no. 2, pp. 99–112, 2006.

[18] G. Kahn, "The semantics of a simple language for parallel programming," *Proc. of IFIP74*, 1974.

[19] http://pages.cs.wisc.edu/˜mscalar/simplescalar.html

[20] C. Huang and F. Vahid, "Dynamic coprocessor management for fpga-enhanced compute platforms," in *Proc. of CASES08*, 2008, pp. 71–78.

[21] W. Fu and K. Compton, "An execution environment for reconfigurable computing," in *Proc. of FCCM05*, 2005, pp. 149–158.