System-Level Modelling for Reconfigurable SoCs

I. Benkermi*, A. Benkhelifa†, D. Chillet*, S. Pillement*, J.C. Prévotet†, F. Verdier†
*ENSSAT – Université de Rennes I,
IRISA, 6, rue de Kerampont - 22300 Lannion - France
†ETIS – UMR 8051 – Université de Cergy-Pontoise,
ENSEA, 6, Avenue du Ponceau - 95014 Cergy-Pontoise - France

***Abstract*— The integration of dynamically reconfigurable modules into systems-on-chip ensures a certain degree of flexibility. In fact, it allows systems-on-chip to adapt to variable computation loads, due to the beginning of new tasks or to data dependent processings, for example. In order to get the most advantages of these reconfigurable modules, the operating system must provide the different tasks placement on available targets. This operation has to be performed on-line and must take into account the heterogeneousness of these different targets (software and hardware). In addition, a validation phase is necessary due to the complexity of these applications and systems. This validation can be done on a prototyping platform taking into account the entire system component set. To do so, a general simulation model must be available to evaluate performance application on the chip.**

The goal of this paper is to propose a general model of a system-on-chip based platform which includes dynamically reconfigurable modules. This model is essential prior to the implementation phase of an application and aims at providing a real simulation framework. The proposed model includes the entire system, that is, the applicative level, the middleware level and the architectural level. The model notably describes the interactions between the operating system and the reconfigurable modules and defines specific services according to the originality of the considered platform. A UML formalism has been chosen for the model description, in order to take advantage of the object oriented framework.

I. Introduction

Nowadays, algorithmic complexity of applications tends to increase in many domains such as signal, image processing or control. In parallel, embedded applications require a significant power of calculation in order to satisfy demanding real time constraints. This leads designers to opt for hardware architectures composed of heterogeneous, optimized computation units operating in parallel. Hardware components in SOC (System on Chip) may implement programmable computation units, dedicated or reconfigurable units, or even dedicated datapaths.

In particular, reconfigurable units (e.g FPGA, DART [3]), denoted here as Dynamically Reconfigurable Accelerators (DRA), allow an architecture to adapt to various incoming tasks or to various computation charge due to data dependencies treatments. The *Platform-Based Design* methodology [6] may thus be fully respected, adding flexibility to an architecture towards modifications of its functionality or even the environment (changing standards, mission statements, etc.). This flexibility is provided by the dynamic allocation of different and dedicated processing operators within the DRA. On the other hand, such heterogeneous architectures need even more complex management and control.

In this context, the utilization of an RTOS (Real Time Operating System) is more and more required to furnish services such as communications, memory management, task scheduling and placement. These services have to be fulfilled in real time according to the application constraints. Moreover, such an operating system also provides a complete design framework which is independent of the technology and of the hardware architecture, drastically reducing the time to market.

Embedded RTOS for SOCs has been of great interest and has led to several significant studies. In the context of reconfigurable architectures, a study of [4] has determined and classified the different services that operating systems should provide to handle reconfigurability. Today, two different approaches have emerged. The first consists in utilizing a pre-existent standard RTOS (RTAI [14], RTLinux, VxWorks, ...) and in adding functionnalities dedicated to the management of the reconfigurable resources [10], [9]. The second is to develop a specific RTOS from scratch by implementing the necessary functionalities devoted to the management of the reconfigurable part [19], [16], [17].

To study the interaction between operating system and SOC architecture, and particulary with dynamically reconfigurable modules, we propose to develop a platform model. The goal of this paper is to present our RSoC (Reconfigurable System on Chip) platform model, that includes the whole set of software and hardware elements and their interactions. This article first aims at identifying the different software and hardware components operating in these architectures. Specifically, its purpose is to define the role and the organization of the middleware layer. The main goal is to obtain formal criteria to develop an operating system deeply adapted to this type of architecture. This allows the software and hardware designers that are interested in RSoC to work on the same basis and to focus on the specificity of reconfigurable systems. The concern of delivering the most generic model as possible, supporting a wide varieties of platforms, has led us to adopt a formalism based on UML.

Section II of this article introduces the context of application deployments on a RSoC. In the section III the global model is proposed and the different levels of representation are defined. Some OS services definitions for RSoC management are also listed in this section. Finally, section IV concludes and draws the perspectives of the current work.

II. Mapping applications on RSoC

In this section the global methodology is presented. This methodology aims at deploying an application on a RSoC architecture. First, we introduce the hardware context (§II-A). §II-B deals with the hardware/software partitionning of an application. How such a partitionned application can be mapped on a RSoC is then presented (§II-C).

A. Architectural description of the platform

It is assumed that SoC architectures are heterogeneous. The wide variety of hardware elements of such architectures is depicted figure 1.

The hardware context of such RSoC can be summarized as follows:

- One or several general purpose processors (GPP). One processor is dedicated to executing the main part of the operating system.
- One or several specialized processors (RISC, DSP, micro-controller).
- Hardware accelerators or processing IPs such as filters, convolvers, VITERBI decoders, etc.
- Memory resources and input / output interfaces.
- One or several DRAs as described later.
- A set of communication resources (busses, hierarchical busses, communication network). A specific network for the transport of configuration information may also be necessary for parallelization between data and configuration transfers.

It is also assumed that the processing operators have different execution models (sequential, parallel, pipeline, data-flow, etc).

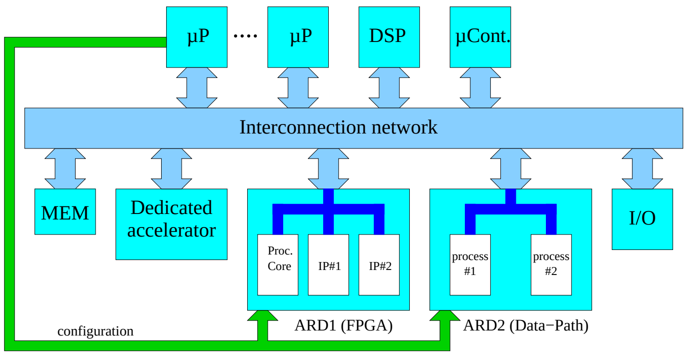

Fig. 1. Example of a highly heterogeneous reconfigurable SoC platform

The main difference between a classical SoC architecture and a reconfigurable SoC platform resides on the existence of reconfigurable areas. Recent evolutions of both reconfigurable technologies and platform-based design concepts have ensured the availability of such synthesizable reconfigurable IPs. Basic structures and grain of these IPs depend on the application:

- Fine-grain reconfigurable areas (FPGA-like) for control and bit oriented processes (ATMEL FPSLIC structures [1] for example).
- Coarse-grain areas for multimedia oriented applications mainly implemented with arithmetic computations (PACT XPP [15] architecture).
- Mixed solutions associating FPGA areas and reconfigurable data-paths (DART architecture [3] for example).

The support of dynamic allocation of multiple dedicated processes on these reconfigurable accelerators is the common characteristic of the considered architectures. In this context, dynamic allocation means that the starting time of processes is unknown at compile time. Generally, the ratio between the execution time and the configuration time of these processes is an important criteria. It is assumed that an order of magnitude leads to efficient dynamic allocations. Readers could refer to [2] where the notion of *remanence* is presented and defined for various reconfigurable platforms.

B. Application partitioning

Due to their complexity (multimedia, telecommunication, networking), applications are decomposed into multiple elementary tasks. These *applicative tasks* represent some computation without any knowledge about the effective execution targets.

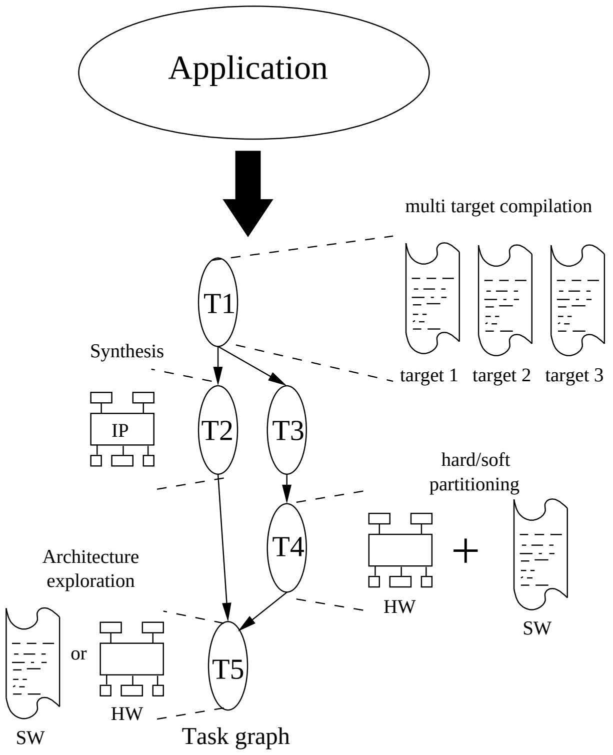

Fig. 2. The RSoC model supports different implementation schemes for applicative tasks (synthesis of hardware modules, hard/soft partitioning and multiple software compilations).

In order to take into account the heterogeneous structure of the target architecture we propose a general model based on the implementation of each applicative tasks into software and/or hardware tasks (see figure 2). A software task is a portion of code (instructions) that can be executed by one of the RSoC processors. This processor may be one of the physically available processing nodes (GPP, RISC, DSP, etc) or a soft-processor core previously configured within a DRA. A hardware task is a dedicated hardwired process directly supported by a physical IP or a hardware module configured within a DRA.

C. Application mapping on the RSoC

The global software architecture is organized around the operating system. One of the OS sevices is responsible for mapping (i.e. allocating and placing) the different software and hardware tasks onto the available targets. Following the previous architectural and software descriptions, different tasks mapping levels are defined on the platform. In the example illustrated in figure 3, a minimal RSoC platform composed of a GPP associated with a DRA is under concerned.

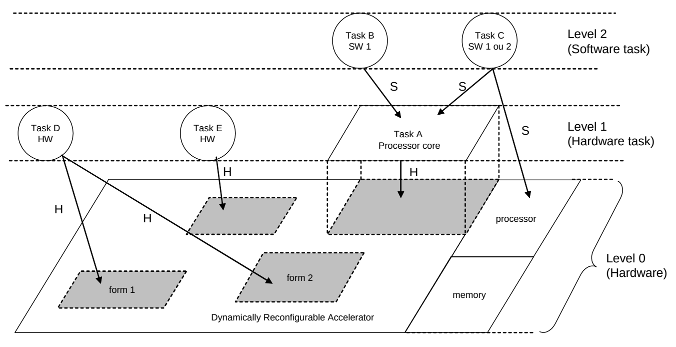

Fig. 3. The different mapping levels depend on the different processing entities

The different mapping levels are decomposed as follows:

- The physical level (level 0) corresponds to the physical architecture of the RSoC. The set of targets contains the DRA, the general purpose processor, a communication channel and memories. This level is composed of resources (memory, bandwidth, reconfigurable and CPU time) that can be allocated to the applicative tasks.
- The level 1 corresponds to the hardware tasks utilizing the physical level resources. In particular, all of the hardware objects to be configured on the DRA constitute tasks belonging to this level (dedicated synthesized IPs, processor cores, filters). Configuring a hardware task within the DRA requires a specific download of configuration data (the H arrow in figure 3). The model features multiple configurations for a unique hardware task offering this task different forms and performances.
- The level 2 contains the set of applicative software tasks to deploy on the GPP or on the processing cores within the DRA. Allocating a task on such processor is considered as a context switching (the S arrow). When the target is a processor within the DRA, an additional hardware configuration step may be necessary (it correponds to the configuration of the processor core itself). As for the hardware tasks, a software task may come in a variety of compiled codes according to the different processor targets. By doing so, heterogeneous migration of tasks is feasible.

III. General model

A graphical model based on UML has been chosen for a sake of clarity in the description of all the elements in the RSoC architecture: the applicative tasks, the hardware and software elements and the interactions between them. The corresponding UML class diagram is depicted in figure 4. The presented model guaranties the independance between the application specification and its hardware implementation. The model is divided into three layers.

A. The application layer

The application layer describes the applicative tasks (*ApplicativeTask* class). In this layer, software designers represent applications without taking into considerations the hardware implementation. This representation implies that the designer has previously broken up the application into tasks and is able to describe each one of them along with their dependences (cf. figure 2). This problem is out of the scope of this paper and we assume in the following that this decomposition has already been performed [5], [7].

An applicative task is specified through the typical parameters used in the real-time context: identification, priority, period, termination time, ready-time and whether the task is preemptible or not.

Control and data dependencies are also specified in this layer. In particular, tasks can produce and consume data to share information and to store internal temporary variables as well. These tasks properties are explicitely described in this model (*DataAccess* class). The amount of memory needed by each task is indicated in the *Data* class.

In this layer the designer may also specify the Quality of Services (*QoS* class) associated with each tasks. QoS can be defined as the purpose of matching some constraints (real-time, bandwidth or power consumption) and must be adapted by the designer to a particular application. Obviously, these constraints can be respected only if lower layers elements can support particular execution schemes. QoS specification is out of the scope of this paper.

B. The architecture layer

The architecture layer contains all the hardware objects that are physically implemented in the platform (level 0 of figure 3). It includes the processing targets, memory and communication resources.

1) Processing Targets: They (*ProcessingTarget* class) are divided into two parts, the processor-like elements (*SoftwareTarget* class) and the hardware targets (*HardwareTarget* class). These two parts may be defined by different supply voltages, power consumptions and frequencies. The representation of these parameters allows to take into account techniques such as the dynamic scaling of the supply voltage (*Dynamic Voltage Scaling*).

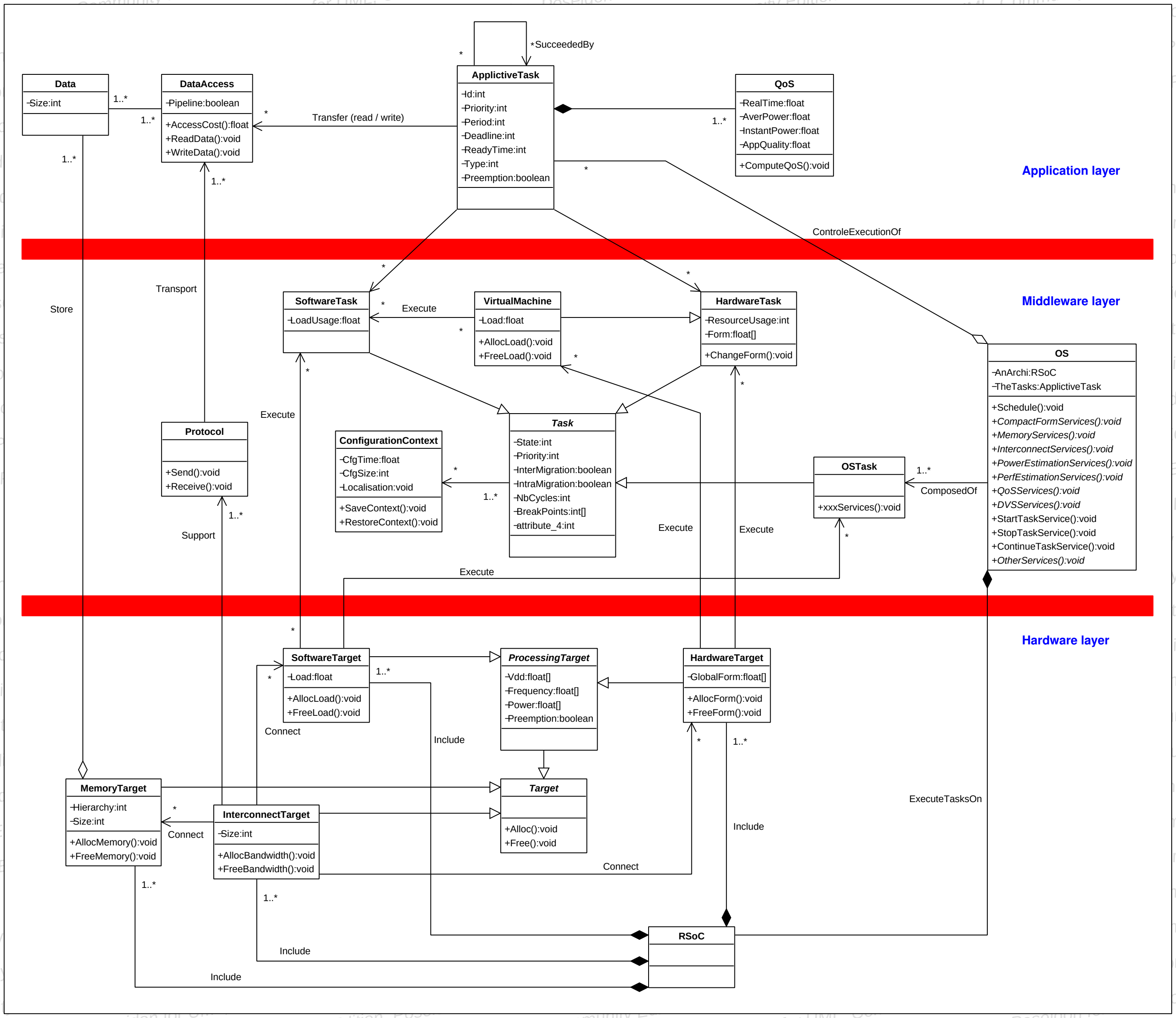

Fig. 4. General model class diagram

a) SoftwareTarget class: In the current model, the RSoC architecture features a main processor at least. The other processors (auxiliary ones) do not play a particular role in the platform management. The main processor is the main processing target of the RSoC operating system and is physically linked to all elements in the platform. It also supports the execution of software tasks. The auxiliary processors group contains all other software processing targets (for example general or dedicated architectures such as DSPs, micro-controllers, etc.). These targets also support sofware tasks and may sometimes require a micro-kernel for multitasking.

b) HardwareTarget class: The hardware processing targets are divided into two groups: the DRA and the dedicated modules. A DRA consists in a dynamically reconfigurable area, the grain of which corresponds to the structure of the available operators. The proposed model allows to consider all forms of granularity and furnishes a generic overview of reconfigurable areas. Each DRA may implement several hardware tasks according to its available resources. The dedicated modules are optimized for a given functionality, have access to communication channels within the chip but cannot be reconfigured. Examples of dedicated modules are typically predefined IP cores.

2) Memory resources: The architecture layer model also takes into account the memory organization (*MemoryTarget* class). Notably, memory hierarchy (caches) constitutes a key challenge in RSoCs [12]. Several studies are currently made in order to handle these aspects and should lead to a significative improvement of the corresponding part in the model, in a near future.

3) Communication resources: In the architecture layer, the role of communication resources is to define the hardware support for transactions between the different elements within the platform. A first type of interconnect deals with the communication between the main processor, the dedicated modules and the DRA. This type of interconnect is static and optimized during the design phase. The second type concerns interconnections between hardware modules within a DRA. It constitutes a real challenge and one of the most delicate aspects of the model. Indeed, the interconnections must structurally and functionally adapt according to the different configurations and consequently guarantee flexibility. This key point has motivated several studies on the implementation of NoC (Networks on Chip) for Reconfigurable SoCs [8], [11], that provide flexibility and quality of services.

C. Middleware layer

The middleware layer contains all the objects which are scheduled by the OS i.e. all the tasks and their characteristics related to the hardware. At this level, the interactions between an application and an implementation on a reconfigurable platform are specified. This level also describes the different configuration tasks, the communication protocols that could be used and the OS which manages all these objects.

1) Tasks and their execution supports: Tasks (*Task* class) are the main elements of the middleware layer. In this level a task may be considered in three different ways: (*SoftwareTask* class, *HardwareTask* class) and a third type to define the OS tasks (*OSTask* class). Software tasks (*SoftwareTask* class) run on RSoC processors (*SoftwareTarget* class). They are classical tasks scheduled by the OS on these targets. Hardware tasks (*HardwareTask* class) can either run on the DRA (*HardwareTarget* class) or on the dedicated modules that are present in the platform. These tasks are mainly efficient implementations of applicative tasks. They utilize some resources (*ResourceUsage* class) and occupy areas within the DRA (*Form* class).

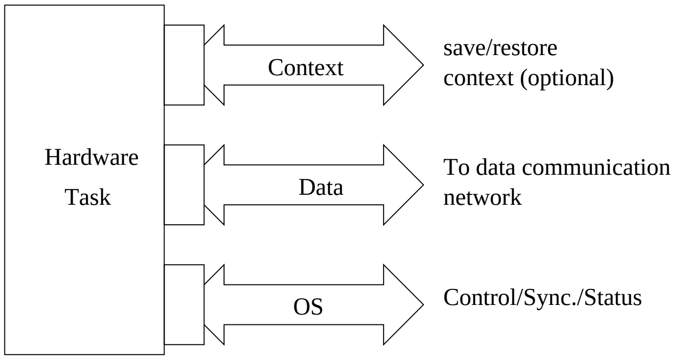

Fig. 5. Hardware task and its interfaces

A hardware task, as shown in figure 5, is able to communicate with other RSoC resources via three interfaces: the context interface allows to save and restore the state of the hardware task, the data interface interconnects the hardware task with the communication medium. Finally, the control interface helps to communicate with the operating system which manages the tasks and their scheduling.

In addition, some particular hardware tasks can execute software tasks. Examples are processor cores configured within the DRA (Altera NIOS, Xilinx MicroBlaze...) or programmable data-paths. They are denoted as virtual machines (*VirtualMachine* class).

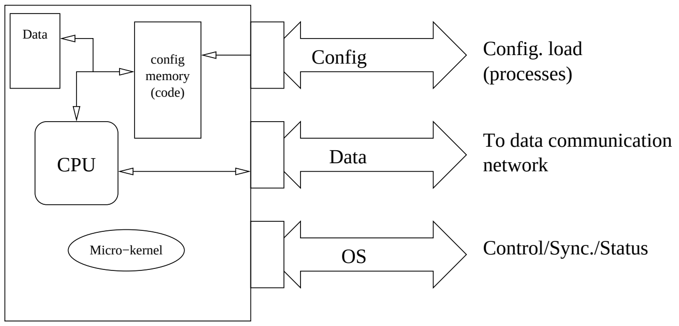

Fig. 6. Overview of a virtual machine

As presented in figure 6, a virtual machine has a code interface enabling the OS to load a task, a communication and a control interface. The virtual machine, at least, contains a micro-kernel devoted to manage the machine, to start the execution of the loaded tasks and to indicate their state. This micro-kernel may be straightforward (a state machine for small virtual machines) or complex (a local RTOS layer for software processors). In the second case, this local kernel is also in charge of the scheduling and the management of the local resources.

Moreover, each hardware or software tasks require a configuration context step (*ConfigurationContext* class) in order to be executed on a target. When a task executes on a processor, this context is similar to the context of a classical software task (program counter, stack pointer...). In the case of a hardware task, the context can be more complex and contains the whole configuration sequence of the hardware target (e.g. a FPGA bitstream).

2) Communication protocols: The middleware layer exhibits the link between the data transfers and the hardware target (*Protocol* class). The data are transferred via memory accesses which are managed by the protocol.

3) OS: The operating system (OS) with its set of services belongs to this layer. Each service is represented by a particular system task (*OSTask* class). The OS services related to the management of the platform have been identified. Some of them are slightly modified due to the presence of DRA. Others are exclusively dedicated to the management of the reconfigurable areas.

In the following, we outline the main services of an OS for RSoCs:

a) Tasks selection: The proposed model permits to specify the execution support (hardware or software) of some applicative tasks. The OS scheduler dynamically selects an execution support according to the RSoC operating state (CPU charge, timing constraints, amount of free resources on a DRA, etc).

b) Tasks migration: A task may be authorized to migrate from a software or a hardware target to an equivalent target (*IntraMigration*) or to a target of a different type (*InterMigration*). In this case, it is assumed that preemption of software and hardware tasks is possible. Sofware preemption is straightforward. However, preemption of hardware tasks involves defining behavioral breakpoints (with the corresponding context storage) [9].

c) Tasks placement and configuration: If the choice of the OS has led to the placement of a task on a DRA, a specific service is invoked. For example, if a software task has to be executed on a virtual machine, then the configuration phase of this machine may be avoided if it is already present on the DRA.

d) Resource management: As for a classical processor, an OS for a RSoC must maintain a resources occupation table. Resources are dynamic in a RSoC due to the fact that they evolve according to the performed reconfigurations. Additional details have to be foreseen on the amount, the shape and the location of the occupied DRA resources.

e) Tasks communication: This major service is essential. Two different types of communications must be handled: global RSoC communications through the static interconnection network and local reconfigurable communications within the DRAs. Indeed, the dynamic reconfiguration of operators within the DRA implies both the utilization of a flexible interconnect and the presence of dedicated interfaces. Moreover, some of these interfaces are already standardized and all the communication mechanisms are rigourously specified in several protocols (c.f. VCI [18], OCP [13], etc).

IV. Conclusion

This article proposes a general system-level model for reconfigurable SoCs (RSoCs). This layered model constitutes a first approach of formalization of a reconfigurable embedded platform and allows the software developers to deploy applications under a common framework. Software and hardware components within a RSoC have been modelized and specific services have been identified. Moreover, interactions between components have also been formalized.

The perspective of the work is to study and develop an original real-time operating system, based on the proposed model, and to implement specific applications on the described platform.

References

[1] ATMEL. FPSLIC (AVR with FPGA). `http://www.atmel.com/products/FPSLIC/`.

[2] Pascal Benoit, Gilles Sassatelli, Lionel Torres, Didier Demigny, Michel Robert, and Gaston Cambon. Metrics for reconfigurable architectures characterization: Remanence and scalability. In *SAMOS'03: Systems, Architecture, Modeling and Simulation*, Samos, Grèce, july 2003.

[3] R. David, D. Chillet, S. Pillement, and O. Sentieys. Dart : A dynamically reconfigurable architecture dealing with next generation telecommunications constraints. *9th IEEE Reconfigurable Architecture Workshop RAW*, April 2002.

[4] O. Diessel and G. Wigley. Opportunities for Operating Systems Research in Reconfigurable Computing. Technical Report ACRC-99-018, Advance Computing Research Center, School of Computer and Information Science, Univ. of South Australia, August 1999.

[5] H. Gomaa. *Software Design Methods for Concurrent and Real-Time Systems*. Addison Wesley, 1993.

[6] Kurt Keutzer, Sharad Malik, A. Richard Newton, Jan M. Rabaey, and A. Sangiovanni-Vincentelli. System-Level Design: Orthogonalization of Concerns and Platform-Based Design. *IEEE Trans. on Computer-Aided Design of Intergrated Circuits and Systems*, 19(12):1523–1543, December 2000.

[7] H. Kopetz and H. Obermaisser. Temporal composability. *Computing and Control Engineering Journal*, pages 156–162, August 2002.

[8] T. Marescaux, J-Y. Mignolet, A. Bartic, W. Moffat, D. Verkest, S. Vernalde, and R. Lauwereins. Networks on Chip as Hardware Components of an OS for Reconfigurable Systems. In *Field Programmable Logic and Application: 13th International Conference, FPL*, pages 595–605, September 2003.

[9] V. Nollet, P. Coene, D. Verkest, S. Vernalde, and R. Lauwereins. Designing an Operating System for a Heterogeneous Reconfigurable SoC. In *Reconfigurable Architectures Workshop*, Nice, France, April 2003.

[10] V. Nollet, J-Y. Mignolet, T.A. Bartic, D. Verkest, S. Vernalde, and R. Lauwereins. Hierarchical Run-Time Reconfiguration Managed by an Operating System for Reconfigurable Systems. In *International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, pages 81–87, Las Vegas, USA, June 2003.

[11] Vicent Nollet, Théodore Marescaux, and Diederik Verkest. Operating-System Controlled Network on Chip. In *Design Automation Conference, DAC*, pages 256–259, June 2004.

[12] I. Ouaiss. *Hierarchical Memory Synthesis in Reconfigurable Computers*. PhD thesis, University of Cincinnati / OhioLINK, 2002.

[13] Open Core Protocol International Partnership. Ocp-ip. `http://www.ocpip.org/`.

[14] RTAI. `http://www.rtai.org/`.

[15] H. Schueler. Smart media processing with XPP, white paper. `http://www.pactcorp.com/xneu/px_SMeXPP.html`, April 2003.

[16] C. Steiger, H. Walder, and M. Platzner. Operating Systems for Reconfigurable Embedded Platforms: Online Scheduling of Real-time Tasks. *IEEE Transaction on Computers*, 53(11):1392–1407, November 2004.

[17] M. Ullmann, M. Hübne, B. Grimm, and J. Becker. On-Demand FPGA Run-Time System for Dynamical Reconfiguration with Adaptive Priorities. In *Field Programmable Logic and Application: 14th International Conference, FPL*, pages 454–463. Springer-Verlag Heidelberg, August 2004.

[18] VSIA. `http://www.vsia.org/`.

[19] H. Walder and M. Platzner. Reconfigurable Hardware Operating Systems: From Design Concepts to Realizations. In *Proceedings of the 3rd International Conference on Engineering of Reconfigurable Systems and Architectures (ERSA)*, pages 284–287. CSREA Press, June 2003.