HAL
open science

Targeting Reconfigurable FPGA based SoCs using the MARTE UML profile: from high abstraction levels to code generation

Imran Rafiq Quadri, Huafeng Yu, Abdoulaye Gamatié, Samy Meftali, Jean-Luc Dekeyser, Eric Rutten

To cite this version:

Imran Rafiq Quadri, Huafeng Yu, Abdoulaye Gamatié, Samy Meftali, Jean-Luc Dekeyser, et al.. Targeting Reconfigurable FPGA based SoCs using the MARTE UML profile: from high abstraction levels to code generation. International Journal of Embedded Systems, 2010, 18 p. inria-00525015v2

HAL Id: inria-00525015

https://inria.hal.science/inria-00525015v2

Submitted on 10 Nov 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Targeting Reconfigurable FPGA based SoCs using the UML MARTE profile: from high abstraction levels to code generation

Imran Rafiq Quadri, Abdoulaye Gamatié, Samy Meftali, Jean-Luc Dekeyser

INRIA Lille Nord Europe - LIFL - USTL - CNRS, FRANCE
E-mail: {firstname.lastname}@lifl.fr

Huafeng Yu*

IRISA/INRIA Rennes - Bretagne Atlantique, FRANCE
E-mail: Huafeng.Yu@inria.fr
* Corresponding author

Éric Rutten

INRIA Grenoble Rhône-Alpes, FRANCE
E-mail: Eric.Rutten@inria.fr

Abstract:
As SoC design complexity is escalating to new heights, there is a critical need to find adequate approaches and tools for handling SoC co-design aspects. Additionally, modern reconfigurable SoCs offer advantages over classical SoCs as they integrate adaptivity features to cope with mutable design requirements and environment needs. This paper presents a novel approach for addressing system adaptivity and reconfigurability. A generic model of reactive control is presented in a SoC co-design framework: Gaspard2. Afterwards, control integration at different levels of the framework is illustrated for both functional specification and FPGA synthesis. The presented works are based on Model-Driven Engineering and the UML MARTE profile proposed by Object Management Group, for modeling and analysis of real-time embedded systems. Our contributions thus relate to presenting a complete design flow to move from high level MARTE models to automatic code generation, for implementation of dynamically reconfigurable SoCs.

Keywords: Intensive Signal Processing; UML; MARTE; Model-Driven Engineering; SoC; Reconfigurability; FPGAs.

Biographical notes: Imran Rafiq Quadri is currently finishing his PhD thesis in Computer Science from the University of Lille 1. He is equally also a research and teaching assistant at University of Lille 1. He achieved his Master's degree from University of Lille 1 in the field of embedded systems in 2006, and is currently involved in the DaRT project at INRIA Lille-Nord Europe; working in the field of partial dynamic reconfiguration and FPGA synthesis.

Huafeng Yu is currently an expert research engineer at IRISA/INRIA Rennes, France. He received his PhD degree in Computer Science from University of Lille 1 in France in 2008. His research interests concentrate on safe design of embedded systems via formal methods.

Abdoulaye Gamatié received a PhD degree in Computer Science in 2004 from University of Rennes 1, where he also held an assistant professor position for two years. In 2005, he was a postdoctoral fellow at INRIA Futurs. Since 2006, he is a research scientist at CNRS, in France. His research interests include methodologies and tools for the reliable design of embedded systems in general.

Éric Rutten, PhD 1990 and Hab. 1999 at University of Rennes, France, works at INRIA in Grenoble, in the field of reactive embedded systems. He is currently working on model-based control of adaptive and reconfigurable computing systems, using control techniques; particularly discrete controller synthesis.

Samy Meftali obtained his PhD in Computer Science from University of Grenoble 1 in 2001. Since then, he is an associate professor at University of Lille 1 in France. His research interests are mainly modeling, simulation and FPGA implementation of intensive signal processing embedded systems.

Jean-Luc Dekeyser received his PhD degree in computer science from the University of Lille 1 in 1986. After a few years at the Supercomputing Computation Research Institute in Florida State University, he joined University of Lille 1 as an assistant professor. He is currently a professor at University of Lille 1 and is also heading the DaRT project at INRIA Lille-Nord Europe.

Copyright © 2008 Inderscience Enterprises Ltd.

1 Introduction

Since the early 2000s, Systems-on-Chip or SoC has emerged as a new methodology for embedded systems design. In a SoC, the computing units, e.g., programmable processors, memories and I/O devices, are all integrated into a single chip. These SoCs are generally dedicated to applications like multimedia video codecs, software-defined radio and radar/sonar detection systems, that require intensive data-parallel computations. Unlike general parallel applications that focus on code parallelization, data-parallel applications concentrate on regular data partitioning, distribution and their access to data.

1.1 Motivations

As the computational power increases for SoCs in accordance with Moore's law (Moore 1965), more functionalities are expected to be integrated into the system. As a result, more complex software applications and hardware architectures are integrated, leading to a *system complexity* issue which is one of the main hurdles faced by SoC designers. The consequence of this complexity is that the system design, particularly software design, does not evolve at the same pace as that of hardware. This has become a critical issue and has finally led to the *design productivity gap*.

Adaptivity and *reconfigurability* are also critical issues for SoCs which must be able to cope with end user environment and requirements. For instance, mode-based control plays an important role in modern embedded systems by permitting description of Quality-of-Service or QoS choices: 1) changes in executing functionalities, e.g., color or black and white picture modes for modern digital cameras; 2) changes due to resource constraints of targeted hardware/platforms, for instance switching from a high memory consumption mode to a smaller one; or 3) changes due to other environmental criteria such as communication quality and energy consumption. A suitable control model must be generic enough to be applied to both software and hardware design aspects.

For implementing dynamically reconfigurable SoCs, *Field Programmable Gate Arrays* or FPGAs are considered as an ideal solution, due to their inherent reconfigurable nature. Designers can initially implement, and afterwards, reconfigure a complete FPGA based SoC for the required customized solution. Thus FPGAs offer a migration path for final *Application Specific Integrated Circuit* or ASIC implementation. State of the art FPGAs can also change their functionality at *runtime*, known as *Partial Dynamic Reconfiguration* or PDR (Lysaght et al. 2006). This feature allows to modify specific regions of an FPGA on the fly, with the advantage of time-sharing the available hardware resources for executing multiple mutually exclusive tasks. It permits context switching depending upon application needs, hardware limitations and QoS requirements. Currently only Xilinx FPGAs fully integrate partial dynamic reconfiguration. These FPGAs also support internal self dynamic reconfiguration, in which an internal controller, e.g., a *hardcore*/*softcore* embedded processor, manages the reconfiguration aspects.

1.2 Elevation of design abstraction levels

An effective solution to SoC co-design problem consists of raising design abstraction levels. The important challenge is to find efficient design methodologies that raise design abstraction levels to reduce overall complexity. These methods must effectively handle issues like accurate expression of inherent system parallelism, such as application loops and hierarchy. They should also be able to express the control at higher abstraction levels to integrate adaptivity and reconfigurability features in modern embedded systems.

Unified design approach is an emerging research topic for addressing the various issues related to SoC co-design. High level SoC co-modeling design approaches have been developed such as Model-Driven Engineering or MDE (OMG 2007*b*). MDE enables high level system modeling of both software and hardware, with the possibility of integrating heterogeneous components into the system. *Model transformations* (S. Sendall and W. Kozaczynski 2003) can then be carried out to generate executable models from the high level models. MDE is supported by several standards and tools.

Gaspard2 (INRIA DaRT team 2009, Gamatié et al. 2010) is an MDE-based SoC co-design framework dedicated to specification of parallel hardware and software. It is based on the standard UML MARTE profile (OMG 2008); and allows to move from high level MARTE models to different execution platforms. It exploits the inherent *parallelism* included in repetitive constructions of hardware elements or regular constructions, such as application loops. The applications targeted by Gaspard2 also focus on a specific application domain, that of intensive data-parallel computation applications.

1.3 Our contribution

In this paper, we present a generic control semantics for the specification of system adaptivity and specially dynamic reconfigurability in SoCs. The introduced control semantics are integrated in Gaspard2 and are specified at a high abstraction level. This control semantics can be integrated at different SoC design levels, with an example of the application level. However, for integrating aspects of dynamic reconfigurability, we propose integration at a design level that links the basic building blocks of applications/architectures to their *Intellectual Properties* or IPs. Integration at the IP deployment level focuses on FPGA synthesis and is specially oriented towards partial dynamic reconfiguration. Our design flow generates two key concepts related to a dynamically reconfigurable FPGA based SoC. Firstly, we generate the code for a dynamically reconfigurable region, which relates to a high level application model, translated into a hardware functionality, e.g., a hardware accelerator and its different implementations, by means of model transformations.

Secondly, the control semantics are utilized for the generation of the source code related to a reconfiguration controller, that manages the different implementations related to the hardware accelerator. Thus our design flow is mainly application-driven in nature.

Finally, a case study related to a dynamically reconfigurable correlation module application is presented in the context of an anti-collision radar detection system, to validate our design methodology.

The rest of this paper is organized as follows. Related works are detailed in Section 2. An overview of the MDE-based Gaspard2 framework is provided in Section 3. Section 4 describes the control model in software applications, while Section 5 presents the control model for IP deployment and FPGA. Section 6 presents our case study. Control models used at different levels are compared in Section 7. Finally, Section 8 gives the conclusion of the paper.

2 Related works

In this section, we detail some works in the domain of dynamically reconfigurable SoCs. We particularly focus on fine grain reconfigurable FPGA based SoCs, as compared to coarse grain reconfigurable architectures, of which numerous examples exist both in academic research and industry. Works related to reconfigurable SoCs can be categorized in several families: some works try to elevate design abstraction levels, such as providing specifications in system level languages like SystemC[1]; for decreasing the complexity related to creation of dynamically reconfigurable systems. Others deal with optimizations directly at the *Register Transfer Level* or RTL by introducing new tools and methodologies.

2.1 Elevation of design abstraction levels

The MoPCoM project (Koudri et al. 2008) aims at modeling and code generation of dynamically reconfigurable embedded systems using the UML MARTE profile for SoC co-design (Vidal et al. 2009). However, the targeted applications are relatively simple unlike those considered in the SoC industry. While the authors claim that they are capable of creating a complete SoC co-design framework, in reality, the high level application model is converted into an equivalent hardware design, with each application task transformed into a hardware accelerator in a target FPGA. Additionally, while the project permits modeling of the targeted FPGA architecture at the UML level as inspired from the works presented in (Quadri, Meftali & Dekeyser 2009*a*, Quadri, Muller, Meftali & Dekeyser 2009), they are only capable of generating the *microprocessor hardware specification* file for input in Xilinx EDK tool, for manual manipulation of the partial dynamic reconfiguration flow. Moreover, IP reuse is not possible with this methodology.

In the OverSoC project (Pillement & Chillet 2009), the authors also provide a high level modeling methodology for implementing dynamically reconfigurable architectures. They integrate an operating system or OS, for providing and handling the reconfiguration mechanism. The global platform is conceptually divided into *active* and *reactive* components representing the reconfigurable architecture (an FPGA) and the OS respectively. The OS is executed on a general purpose processor interfacing with the FPGA. The active component is further composed of several sub components that represent the computation and reconfiguration components. The former relates to FPGA resources such as CLBs and LUTs, while the latter corresponds to an FPGA internal hardware reconfiguration core responsible for the actual switching. Finally, SystemC is used for simulation and verification of the OS for managing the reconfigurable aspects. However, final implementation on FPGAs has not been carried out. It is up to the OS to determine whether an application task should be executed on the general purpose processor or the FPGA, depending upon the required resources.

A more complex OS is presented in (Bergmann et al. 2003), as an embedded uCLinux is used for managing partial dynamic reconfiguration. A customized device driver has been created to manage the hardware reconfigurable core, allowing users to carry out dynamic configuration in traditional Linux shell programs. However, the bitstreams are generated manually using the FPGA editor tool, raising chances of design errors.

(Brito et al. 2007) use a SystemC based design flow for implementing partial dynamic reconfiguration. The SystemC kernel is modified for the integration of reconfiguration operations for activation/disactivation of reconfigurable modules. Initial simulation is carried out using a SystemC model, which is then converted into a *Hardware Description Language* or HDL RTL model for actual implementation and comparison. The drawback of this approach is that the reconfiguration time related to module is predetermined by the designers. Additionally, the system only provides *on-off* functionality for the modules resulting in a simplified design with respect to partial dynamic reconfiguration.

In (Nezami et al. 2008), HandleC is used to implement partial dynamic reconfiguration for Software defined Radio, however, they only provide the design methodology and no actual implementation is carried out. In (F. Berthelot and F. Nouvel and D. Houzet 2008), a SynDEx based design flow is presented to manage SoC reconfigurability via implementation in FPGAs, with the application and architecture parts modeled as components.

2.2 Targeting coarse grain architectures

There also exists a large number of research works like Chameleon (Salefski & Caglar 2001), Rapid (Ebeling et al. 1996) and projects such as Morpheus (Morpheus 2010) that deal with coarse grain reconfigurable architectures. However these projects and works normally consider a lower abstraction level.

SPEEDS! or *Speculative and Exploratory Design in Systems Engineering* (SPEEDS! 2009) is also an European project for embedded systems development based on SysML and AUTOSAR. Equally, the EPICURE project (J.P. Diguet and G. Gogniat and J.-L. Philippe et al

2006) defines a design methodology in order to bridge the gap between high abstract specifications and heterogeneous reconfigurable architectures. The framework is based on Esterel design technologies and provides verification and synthesis capabilities. However, one of the existing drawbacks of this framework is the lack of available support for a high abstraction level design methodology, in order to reduce design complexity. Additionally, works such as Molen (Panainte et al. 2007) propose efficient compilers for reconfigurable architectures. The proposed Molen compiler was implemented on a Virtex II FPGA, and takes into account details such as related to synthesis placement conflicts as well as reconfiguration latencies.

2.3 *High level control methodologies*

In (Latella et al. 1999, Schäfer et al. 2001), the authors concentrate on control based modeling and verification of real-time embedded systems in which the control is specified at a high abstraction level via UML state machines and collaborations; by using model checking. A similar approach has been presented in (Faugere et al. 2007). However, control methodologies vary in nature as they can be expressed via different forms such as Petri Nets (Nascimento et al. 2004), or other formalism such as mode automata (Maraninchi & Rémond 2003, Talpin et al. 2006).

Mode automata extend synchronous dataflow languages with an imperative style, but without many modifications of language style and structure. They are a simplified version of Statecharts (Harel 1987) in syntax, which have been well adopted for the specification of control oriented reactive systems. Mode automata have a clear and precise semantics, which makes the inference of system behavior possible, and are supported by formal verification tools.

2.4 *Methodologies of partial dynamic reconfiguration*

For implementing partial dynamic reconfiguration in modern FPGAs, Xilinx initially proposed several design flows, which were not very effective leading to new alternatives. (Sedcole et al. 2005) presented an effective modular approach for 2-dimensional reconfigurable modules. Similarly, (Becker et al. 2003) implemented 1-dimensional modular reconfiguration using a horizontal slice based bus macro to connect the static and partially dynamic regions. They enhanced their works by placing arbitrary 2-dimensional rectangular shaped modules using routing primitives (Schuck et al. 2008).

In 2006, Xilinx introduced the *Early Access Partial Reconfiguration Design Flow* (Xilinx 2006) that integrated concepts introduced earlier in works such as (Sedcole et al. 2005) and (Becker et al. 2003). Researches such as (Bayar & Yurdakul 2008) and (Paulsson et al. 2007) focus on implementing softcore internal configuration cores on Xilinx FPGAs such as Spartan-3, that do not have the hardware internal reconfiguration cores, for effective implementation of PDR. Finally in (Koch et al. 2006), this reconfigurable core is connected with Network-on-Chip based FPGAs.

In comparison to the above related works, our proposition takes into account the following domains: SoC co-design, data intensive parallel computation applications, control/data flow, MDE, the UML MARTE profile, SoC adaptivity and PDR for fine grain reconfigurable FPGAs; which is the novelty of our design framework.

3 Gaspard2: a SoC co-design framework

Gaspard2 (INRIA DaRT team 2009, Gamatié et al. 2010) is an MDE oriented and MARTE compliant SoC design framework as shown in Figure 1, providing an *Integrated Development Environment* or IDE dedicated to the visual co-design of embedded systems. The framework enables fast design and code generation with the help of UML graphical tools and technologies, such as Papyrus[2] and Eclipse Modeling Framework[3].

The Gaspard2 framework is based on a *repetitive model of computation* or MoC that relies on the Array-OL specification language (Boulet 2007, 2008). The MoC describes the *potential parallelism* present in a system; and describes repetitive data intensive multidimensional computations; with the help of repetitive data dependencies. The manipulated data are in the form of multidimensional arrays, which have at most one possible *infinite dimension*. The arrays can be specified with a certain type specification, such as an array *shape*. The spatial and temporal dimensions are treated in the same manner, in the form of arrays. Particularly, time is expanded as one dimension of arrays. Additionally, access to data is carried out in the form of sub-arrays called *patterns*. In turn, in Gaspard2, data are also manipulated in the form of multidimensional arrays.

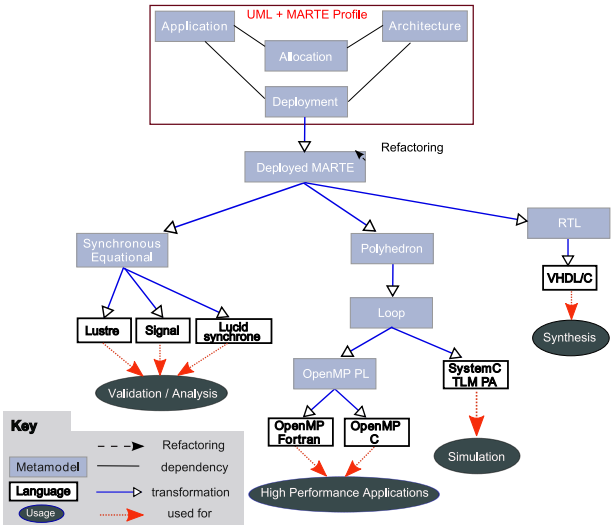

Figure 1: A global view of the Gaspard2 framework

3.1 *Basic repetitive modeling concepts*

Gaspard2 has also contributed to the conception of the UML MARTE profile. One of the key MARTE packages, the

Repetitive Structure Modeling or RSM package is inspired from Gaspard2 and its model of computation. Additionally, some other packages such as the *Allocation* package and *Hardware Resource Modeling* package have also benefited from existing Gaspard2 concepts.

RSM enables the possibility to specify the shape of a repetition, by a multidimensionality, and also permits to represent a collection of potential links such as a multidimensional array. This repetition can be specified for an instance or a port of a component. The advantage is double fold: For hardware modeling, RSM presents a clear mechanism for expressing the links in a topology, as well as increasing the expression power of the mechanism for describing these complex topologies (Quadri et al. 2008). Complex regular, repetitive structures such as cubes and grids can be modeled easily via RSM, in a compact manner. Similarly for application aspects, RSM helps to determine different types of parallelism such as task and data parallelism.

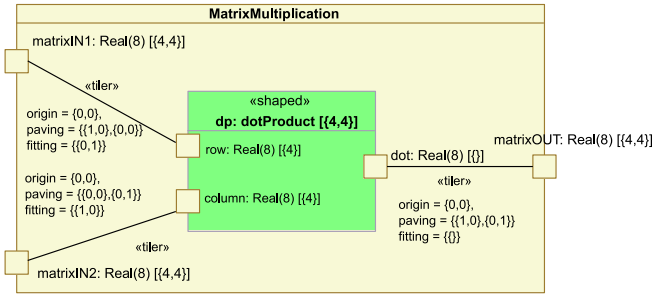

Figure 2: Representing *Data Parallelism* in Gaspard2 with the MARTE profile: The repetitive `MatrixMultiplication` component is composed of a repeated task: the `dP` instance of a `dotProduct` component. This repeated task represents the computing task, which takes one row and column from two input matrices; and produces one element in the final produced matrix. This task is elementary in nature and is thus represented differently from other tasks; and can be henceforth deployed, as discussed in Section 5.1

A `repetitive component` such as shown in Figure 2 expresses the data parallelism in an application: in the form of sets of input and output `patterns` consumed and produced by the repetitions of the interior `part`. On the other hand, a `hierarchical` component such as illustrated in Figure 12 contains several `parts`; and defines a complex functionality in a modular manner. This concept thus provides a structural aspect of the application: specifically, task parallelism can be described using such a component.

The shape of a pattern is described according to a `tiler` connector which describes the tiling of produced and consumed arrays. The `reshape` connector allows to represent complex link topologies in which the elements of a multidimensional array are redistributed in another array.

An `interRepetition` dependency is used to specify an acyclic dependency among the repetitions of the same component, compared to a tiler, that describes the dependency between the repeated component and its owner component. Particularly, this dependency specification leads to the sequential execution of repetitions of the repeated part. A `defaultLink` connector provides a default value for the part repetitions that are linked with an `interRepetition` dependency, with the condition that the source of the dependency is absent. These last two RSM concepts have been illustrated in Figure 15 discussed later on in the paper, and are essential for control modeling.

3.2 Model transformations

Models in MDE are not only used for communication and comprehension but using model transformations (S. Sendall and W. Kozaczynski 2003), produce concrete results such as executable source code. With the help of metamodel(s) that define the concepts of their respective models, and to which these models conform to; models can be recognized by machines. As a result, they can be processed, i.e., a model is taken as input/source and then some models/targets are generated. This process is called a model transformation.

For the purpose of automatic code generation from high level models, Gaspard2 adopts MDE model transformations towards different execution platforms, such as targeted towards synchronous domain for validation and analysis purposes (Gamatié, Rutten, Yu, Boulet & Dekeyser 2008, Yu et al. 2008, Yu 2008); or FPGA synthesis (Le Beux 2007, Quadri, Meftali & Dekeyser 2009*a*), as shown in Figure 1. Model transformation chains permit moving from high abstraction levels to low enriched levels. Usually, the initial high level models contain only domain-specific concepts, while technological concepts are introduced seamlessly in the intermediate levels, by means of intermediate metamodels and respective models.

There exists a large number of transformation languages and tools such as ATLAS Transformation Language (INRIA Atlas Project n.d.), Kermeta (INRIA Triskell Project n.d.) among others; that support the *Meta-Object Facility Query/View/Transformation* or MOF QVT standard (OMG 2005) for model query and transformations. However, few of the QVT transformation tools are capable to execute large complex transformations such as present in the Gaspard2 framework. Also none of these engines is fully compliant with the QVT standard. Nevertheless, some new tools such as QVTO (OMG 2007*a*) have recently emerged that implement the QVT Operational language, and are effective for handling the complex Gaspard2 model transformations.

For model transformations, Gaspard2 adapts QVTO as the de-facto tool for model transformations. It should be made evident that current model transformations are only uni-directional in nature. Similarly, Gaspard2 has adopted Acceleo[4], a code generation tool that is compliant with the MOF2Text standard[5].

Finally, Figure 3 shows the global overview of the model transformation chain related to implementing partial dynamic reconfiguration, as discussed in this paper. Initially a Gaspard2 application is modeled and deployed, along with the associated control aspects in the Gaspard2 environment with the Papyrus modeling tool; conforming to an extended version of the UML MARTE profile. This modeling is

independent from any implementation details until the deployment phase.

Afterwards two *model-to-model transformations*, namely the *UML2MARTE* and *MARTE2RTL* transformations help to create an intermediate model, corresponding to its own metamodel, with concepts nearly equivalent to the *Register Transfer level* or RTL. This model is considered as a low abstraction level with details nearly corresponding to electronic RTL. The model provides details related to the hardware accelerators and the control features which can be used for eventual code generation. Finally using a *model-to-text transformation*, we generate the code related to different implementations of a hardware accelerator and the state machine for the reconfiguration controller. These aspects are detailed later on in the paper. As the model transformation rules are not trivial in nature and are about the size of several thousand lines of code, it is not possible here to give a generalized summary of the transformation rules present in our design flow.

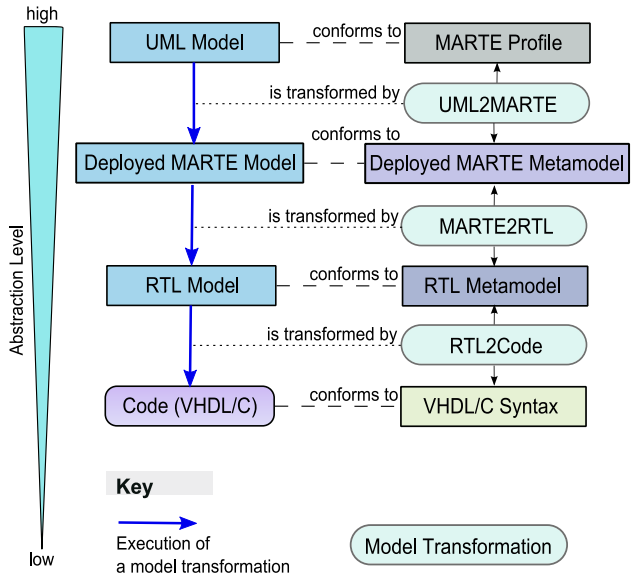

Figure 3: An abstract overview of the model transformation chain. Several intermediate metamodels help to bridge the gap between high level modeled UML diagrams and the final RTL code generation

3.3 Reactive control modeling

This section provides the initial hypothesis related to the generic control semantics for expressing system reconfigurability. Several basic control concepts, such as `Mode Switch Component` and `State Graphs` are presented first. Then a basic composition of these concepts, which helps in eventual building of the mode automata, is discussed. This modeling derives from mode conception in mode automata. The notion of exclusion among modes helps to separate different computations. As a result, programs are well structured and fault risk is reduced.

3.3.1 Modes and Mode switch component

A *mode* is a distinct method of operation that produces different results depending upon the user inputs. A `Mode Switch Component` in Gaspard2 contains at least one mode; and offers a switch functionality that chooses execution of one mode, among several alternative present modes (Labbani et al. 2005). The mode switch component in Figure 4 illustrates such a component having a *window* with multiple tabs and interfaces. For instance, it has a mode value input port m, as well as several data input and output ports, i.e., i_d and o_d respectively. The switch between the different modes is carried out according to the *mode value* received through m.

The modes, M_1, ..., M_n, in the mode switch component are identified by the mode values: m_1, ..., m_n. Each mode can be hierarchical, repetitive or elementary in nature; and transforms the input data i_d into the output data o_d. All modes have the same interface (i.e. i_d and o_d ports). All the input and outputs share the same time dimension, ensuring correct one-on-one correspondence between the inputs/outputs. The activation of a mode relies on the reception of mode value m_k by the mode switch component through m. For any received mode value m_k, the mode runs exclusively. It should be noted that only mode value ports, i.e., m; are compulsory for creation of a mode switch component, as illustrated in Figure 4. Other type of ports, such as input/output ports are not always necessary and are thus represented with dashed lines.

3.3.2 State graphs

A state graph in Gaspard2 is similar to Statecharts (Harel 1987), which are used to model the system behavior using a state-based approach. We term these state graphs as *Gaspard state graphs*. A state graph can be expressed as a graphical representation of transition functions, as discussed in (Gamatié, Rutten & Yu 2008). A state graph is composed of a set of vertices called *states*. A state connects with other states through directed edges which are called *transitions*. Transitions can be conditioned by some *events* or Boolean expressions. A special label *all*, on a transition outgoing from state s, indicates any other events that do not satisfy the conditions on other outgoing transitions from s. Each state is associated with some mode value specifications that provide mode values for the state. Formal definitions of Gaspard state graphs have been presented in (Yu 2008). State graphs, can also be either parallely composed, or composed in a hierarchy; as illustrated in (Yu 2008).

The main difference between our state graphs and Harel's Statecharts is that the former are transition functions in which neither initial or final states are defined. This is not the case in Statecharts or more generally in automata. The way transitions are fired in our state graphs and the information related to current state of the state graph is determined by using `interRepetition` dependencies. Details related to this aspect can be found in Section 3.4.

A state graph in Gaspard2 is associated with a `Gaspard State Graph Component`, as shown in Figure 4. Thus

a state graph determines the internal behavior of a Gaspard state graph component. A Gaspard state graph component determines the mode value definition by means of its associated state graph. The mode values allow to activate different exclusive computations or modes in the related mode switch components. Thus, Gaspard state graph components are ideal complements of mode switch components, with mode values being the relation between the two concepts. A Gaspard state graph component can be viewed as a controller component, while the mode switch component switches between the modes according to the present controller.

Similarly to the mode switch component, a Gaspard state graph component has its interfaces. These interfaces include event inputs from the environment, source state inputs, target state outputs and mode outputs. Event inputs are used to trigger transitions present in the associated Gaspard state graph. The source state inputs determine the states from which the transitions take place, while target state outputs determine the destination states of the fired transitions. The mode outputs are associated with a mode switch component in order to select the correct mode for execution.

3.3.3 Combining modes and state graphs

Once mode switch components and Gaspard state graph components are introduced, a `Macro Component` can be used to compose them together. An abstract representation of the macro component in Figure 4 illustrates one possible composition; and represents a complete Gaspard2 control structure. In the macro, the Gaspard state graph component produces mode values and sends them to the mode switch component. The latter switches the modes accordingly. Some data dependencies between these components are not always necessary, for example, the data dependency between I_d and i_d, and these dependencies are drawn with dashed lines in Figure 4. The illustrated figure is used as a basic composition, however, other compositions are also possible, for instance, one Gaspard state graph component can control several mode switch components (Quadri, Meftali & Dekeyser 2009*b*). In order to simplify the illustration, events `e1`, `e2` and `e3` are only shown as a single event I_e.

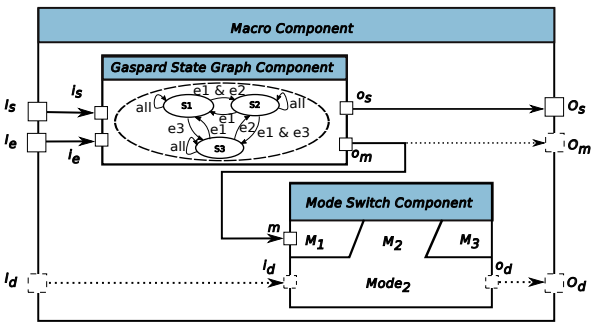

Figure 4: An example of a macro structure

3.4 MARTE concepts for constructing mode automata

We now present the utilization of some MARTE concepts which aid in the modeling of mode automata. The basic concepts of Gaspard2 control have already been presented earlier in Section 3.3, but its complete semantics have not been detailed. For this purpose, we propose to integrate mode automata semantics with the control aspects. This choice is made to remove design ambiguity, enable desired properties, to enhance design correctness and verifiability. In addition to previously mentioned control concepts, we make use of three additional MARTE concepts, as present earlier in Section 3.1; namely: `interRepetition` dependency, `tiler` and `defaultLink` connectors, which are essential for building mode automata.

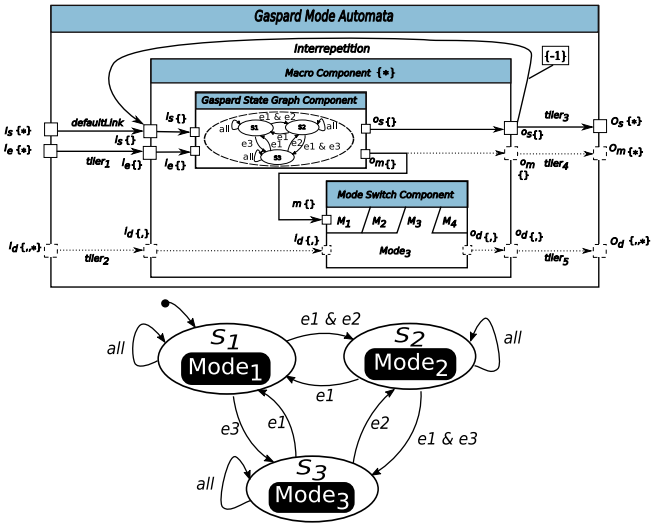

Figure 5: Abstract representation of a generic Gaspard2 mode automata (illustration on the top). The illustration on the bottom of the figure is a simplified explanation of the one on the top

Hence, it is possible to establish mode automata from Gaspard2 control model, which requires two subsequent steps. First, the internal structure of a generic `Gaspard Mode Automata` is presented by the `Macro component` illustrated in Figure 4. The Gaspard state graph component in the macro acts as a state-based controller and the mode switch component achieves the mode switch function. Secondly, `interRepetition` dependency specifications should be specified for the macro component and it should be placed in a repetitive component context.

The reasons are as follows: a macro component represents only one single transition function (one map) from a source state to a target state, where as an automata has continuous transitions which form an execution trace. In order to execute continuous transitions as present in a typical automata, the macro should be repeated to have multiple transitions. This functionality is determined by the `interRepetition` dependency.

A vector associated to an `interRepetition` dependency expresses the dependencies between the

repetitions inside the repetition context, i.e., the `Gaspard Mode Automata` component. Thus an `interRepetition` dependency serializes the repetitions and data can be conveyed between these repetitions. An `interRepetition` dependency sends the target state of one repetition as the source state to the next repetition. This permits the construction of mode automata which can be then executed. Figure 5 illustrates an example of this construction.

If a dependent repetition is not defined in the repetition space, a default value is selected. The `defaultLink` provides default value for repetitions whose dependency for the input is absent. Additionally, this concept helps to give the initial state value for the first repetition of the macro component. While in a graphical modeling approach, the initial state of a state machine/Statechart can be determined by an initial pseudostate, a Gaspard state graph does not contains an initial state, as explained earlier.

Thus this mechanism bridges the gap between a graphical representation and the actual semantics. It thus creates an equivalency between a state graph without no initial state and an automaton with an initial defined state. Finally, the `tiler` connectors help in interconnecting a repetitive component to the multiple repetitions of its interior repeated task.

An infinite dimension is present on the input and output state, events ports of the Gaspard mode automata component to account for continuous control/data flows. Similarly the non obligatory mode output ports, input and output data ports also have an infinite dimension in addition to other possible dimensions. Since the macro component represents one single transition, its respective ports have shape values equal to $\{\}$, accounting for one value in the dataflow at an instant of time t.

Similarly, the internal sub components of the macro component also share the same shape values. Finally, the shape value of $\{*\}$ on the macro component represents its multiple, possible infinite, dimensions. The macro component is repeated in a sequential temporal dimension by means of the `interRepetition` dependency. Thus as compared to traditional Statecharts, no final state is necessary. If the repetition of the macro component is not set to an infinite value, the macro will stop repeating when it reaches the end of its repetition space, causing termination of the mode automata.

It should be noted that parallel and hierarchical mode automata can also be built using our approach. In the parallel automata, several automata can be executed in parallel, for example, automata can be related to the application aspects while another can be linked to the architecture. Thus both application and architecture switches can be carried out simultaneously, provided that no conflicts arise by the simultaneous switching. Additionally, in case of control at a SoC design level such as application modeling, the switching between the states or modes can be instantaneous in nature, and is regarded as a change in the functionality. However when this semantics is applied to an execution platform, a stabilization phase may be required for switching between two states.

While this intermediate phase can be modeled using high level modeling semantics, this step has not been undertaken in our approach, in order to present a generic semantics applicable to all SoC design levels. Finally, we refer the reader to (Gamatié, Rutten & Yu 2008) for the detailed formal semantics related to Gaspard mode automata.

4 Adaptivity at application level

The previous section described an abstract control model for integrating dynamic aspects in a given system. Similarly, these control mechanisms can be integrated in different levels in a SoC co-design framework, with the advantage of introducing dynamic aspects in the targeted SoCs. A detailed analysis related to control integration at different SoC design levels in the particular case of the Gaspard2 framework has been presented in (Quadri, Meftali & Dekeyser 2009*b*). In the context of this article, we present the integration at the application and deployment modeling levels in Gaspard2.

4.1 Control example at application level

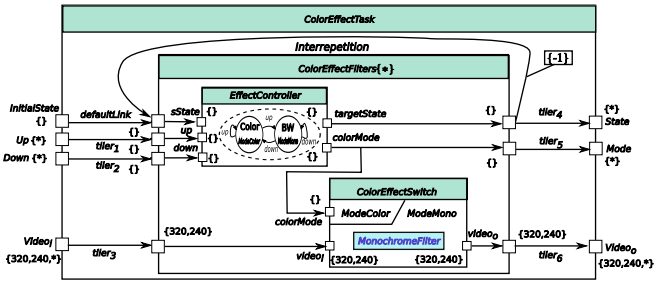

Figure 6: An example of color style filter in a smart phone modeled with the Gaspard2 mode automata

The control model enables the specification of system adaptivity at the application level (Yu 2008). Integration of control model and the construction of a mode automata at application level is very similar to the generic Gaspard mode automata shown in Figure 5. Figure 6 represents the mode automata at the application level by illustrating an example of color effect processing module `ColorEffectTask` used in typical smart phones. This module is used to manage the color effects of a video clip and provides two possible options: color or monochrome/black&white modes, which are implemented by `ColorFilter` and `MonochromeFilter` respectively. These two filters are elementary tasks at the application modeling level, which should be deployed to their respective IPs. The changes between these two filters are achieved by `ColorEffectSwitch` upon receiving mode values through its mode port `colorMode`. The mode values are determined by `EffectController`, whose behavior is demonstrated by its associated state graph.

The `ColorEffectFilters` can be treated as a macro component; and is composed of `EffectController` and `ColorEffectSwitch` components. `ColorEffectFilters`

executes the processing of one frame of the video clip, which should be repeated. In the example, `ColorEffectTask` provides the repetition context for `ColorEffectFilters`. An `interRepetition` dependency is also defined, which connects the different repetitions of the `ColorEffectFilters` component. It has an associated vector with a value of {-1}. Simply put, the source state of one `ColorEffectFilters` repetition relies on the target state of the previous `ColorEffectFilters` repetition. The data computations inside a mode are set in the mode switch component `ColorEffectSwitch`.

Each mode in the switch can produce different end results with regards to environmental or platform requirements. Each mode can have a different demand of memory, CPU load, etc. Thus environmental changes/platform requirements are captured as events; and taken as inputs for the control.

For the application level, the Gaspard2 control model has been implemented with UML state machines and collaborations in (Yu 2008). A model transformation chain from high level MARTE models to synchronous languages can bridge the gap between these models and targeted synchronous language code. By considering the code generated from an application model, validation techniques such as model checking can also be applied. The same code can also be used for controller synthesis to enforce relevant properties with respect to functional and non-functional requirements. All these aspects have been addressed in a case study for the design of a Gaspard2 data-parallel multimedia application (Yu 2008).

5 Adaptivity at IP deployment level

As explained before in the paper, we present an application driven approach for the design and development of dynamically reconfigurable SoCs. For this, we have focused on two main aspects related to a reconfigurable SoC.

For dynamic reconfiguration in modern SoCs, an embedded controller is essential for managing a dynamically reconfigurable region. This component is usually associated with some control semantics such as state machines, Petri nets etc. The controller normally has two functionalities: one responsible for communicating with the FPGA *Internal Configuration Access Port* hardware reconfigurable core or ICAP (Blodget et al. 2003) that handles the actual FPGA switching; and a state machine part for switching between the available configurations. The first functionality is written manually due to some low level technological details which cannot be expressed via a high level modeling approach.

The control at the deployment level is utilized to generate the second functionality automatically via model transformations. Finally the two parts can be used to implement partial dynamic reconfiguration in an FPGA that can be divided into several static/reconfigurable regions. A reconfigurable region can have several implementations, with each having the same interface, and can be viewed as a mode switch component with different modes. In our design flow, this dynamic region is generated from the high abstraction levels, i.e., a complex Gaspard2 application specified using the MARTE profile. Using the control aspects in the subsequently explained Gaspard2 deployment level, it is possible to create different configurations of the modeled application. Afterwards, using model transformations, the application can be transformed into a hardware functionality, i.e., a dynamically reconfigurable hardware accelerator, with the modeled application configurations serving as different implementations related to the hardware accelerator.

We now present integration of the control model at the deployment level. We first explain the deployment level in Gaspard and our extensions followed by the control model.

5.1 *Deployment level in* Gaspard2 *framework*

Gaspard2 defines a notion of a *Deployment* specification level (Atitallah et al. 2007) in order to generate compilable code from a SoC model. This level is related to the specification of *elementary* components: basic building blocks of all other components, having atomic functions. Although the notion of deployment is present in UML, the SoC design has special needs, not completely fulfilled by this notion. In order to generate an entire system from high level specifications, all implementation details of every elementary component have to be determined. Low level behavioral or structural details are much better described by using usual programming languages instead of graphical UML models.

Hence, Gaspard2 extends the MARTE profile to allow deploying of elementary components. The deployment level associates every elementary component to an implementation code hence facilitating IP reuse. Each elementary component ideally can have several implementations. The reason is that in SoC design, a functionality can be implemented in different ways. For example, an application functionality can either be optimized for a processor, thus written in assembler or C/C++, or implemented as a hardware accelerator using traditional HDLs like VHDL/Verilog or with SystemC. Hence the deployment level differentiates between the hardware and software functionalities; and permits moving from platform-independent high level models to platform dependent models for eventual implementation. Deployment provides IP information to model transformations to form a compilation chain, in order to transform the high abstraction level models for different domains: formal verification, simulation, high performance computing or synthesis. Hence deployment can be seen a potential extension of the MARTE profile enabling a complete flow from model conception to automatic code generation. We now present a brief overview of the deployment concepts.

A `VirtualIP` expresses the functionality of an elementary component, independently from the compilation target. For an elementary component K, it associates K with all its possible IPs. The desired IP is then selected by the SoC designer by linking it to K via an `implements` dependency. Finally, the concept of `CodeFile`, is used to specify, for a given IP, the file corresponding to the source

code and its required compilation options. This last concept is not illustrated in the paper due to space limitations. The `CodeFile` thus identifies the physical path of the source code. As compared to the deployment specified in (Atitallah et al. 2007), the current deployment level has been modified to respect the semantics of traditional UML deployment.

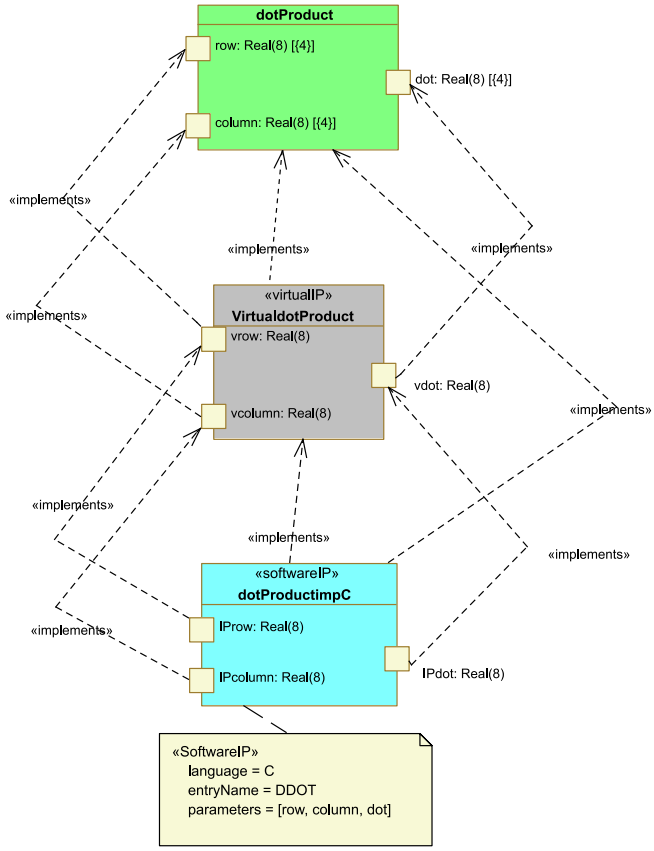

Figure 7: Deployment of an elementary `dotProduct` component in Gaspard2

5.2 *Configurations at the deployment level*

As stated before, an elementary component can be associated with only one IP among the different available choices. Thus the result of the application/architecture or the mapping of the two forming the overall system is a static one. This collective composition is termed as a *Configuration*. The current model transformations for RTL level only allow to generate one hardware accelerator from the modeled application, hence one configuration, for final FPGA execution.

Integrating control in deployment allows to create several configurations related to the modeled application for the final realization in an FPGA. Each configuration is viewed as a collection of different IPs, with each IP associated with its respective elementary component. The end result being that one application model is transformed by means of model transformations and intermediate metamodels into a dynamically reconfigurable hardware accelerator, having different implementations equivalent to the modeled application configurations.

A `Configuration` has the following attributes. The name attribute helps to clarify the configuration name given by a SoC designer. The ConfigurationID attribute permits to assign unique values to each `Configuration`, which in turn are used by the control aspects presented earlier. Theses values are used by a Gaspard state graph to produce the mode values associated with its corresponding Gaspard state graph component. These mode values are then sent to a mode switch component which matches the values with the names of its related collaborations, which are illustrated later on. If there is a match, the mode switch component switches to the required configuration. The InitialConfiguration attribute sets a Boolean value to a configuration to indicate if it is the initial configuration to be loaded onto the target FPGA. This attribute also helps to determine the initial state of the Gaspard state graph.

An elementary component can also be associated with the same IP in different configurations. This point is very relevant to the semantics of partial bitstreams, e.g., FPGA configuration files for partial dynamic reconfiguration, supporting *glitchless dynamic reconfiguration*: if a configuration bit holds the same value before and after reconfiguration, the resource controlled by that bit does not experience any discontinuity in operation. If the same IP for an elementary component is present in several configurations, that IP is not changed during reconfiguration. It is thus possible to link several IPs with a corresponding elementary component; and each link relates to a unique configuration. We apply a condition that for any n number of configurations with each having m elementary components, each elementary component of a configuration must have *at least* one IP. This allows successful creation of a complete configuration for eventual final FPGA synthesis.

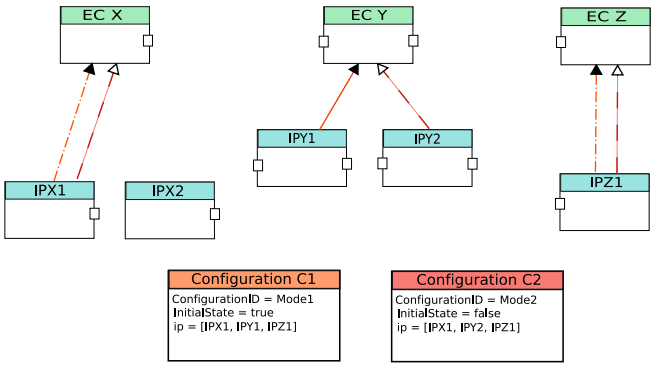

Figure 8: Abstract overview of configurations in deployment

Figure 8 represents an abstract overview of the configuration mechanism introduced at the deployment level. We consider a hypothetical Gaspard2 application having three elementary components *EC X*, *EC Y* and *EC Z*, having available implementations *IPX1*, *IPX2*, *IPY1*, *IPY2* and *IPZ1* respectively. For the sake of clarity, this abstract representation omits several modeling concepts such as `VirtualIP` and `Implements`. However, this representation is very close to UML modeling as presented earlier in the paper. A change in associated implementation of any of

these elementary components may produces a different end result related to the overall functionality, and different QoS criteria such as effectively consumed FPGA resources.

Here two configurations *Configuration C1* and *Configuration C2* are illustrated in the figure. *Configuration C1* is selected as the initial configuration and has associated IPs: *IPX1*, *IPY1* and *IPZ1*. Similarly *Configuration C2* also has its associated IPs. This figure illustrates all the possibilities: an IP can be globally or partially shared between different configurations: such as *IPX1*; or may not be included at all in a configuration, e.g., case of *IPX2*.

Once the different implementations are created by means of model transformations, each implementation is treated as a source for a partial bitstream. A bitstream contains packets of FPGA configuration control information as well as the configuration data. Each partial bitstream signifies a unique implementation, related to the reconfigurable hardware accelerator which is connected to an embedded controller. While this extension allows to create different configurations, the state machine part of the controller is created manually. For automatic generation of this functionality, the deployment extensions are not sufficient. We then use the existing control concepts presented in section 3.3 to solve these issues.

5.3 Integrating control at the deployment level

We now explain control integration at the deployment level in the context of the Gaspard2 SoC co-design framework. Control at this level provides advantages over other levels due to its independent nature. Details related to these advantages can be found in (Quadri, Meftali & Dekeyser 2009*b*).

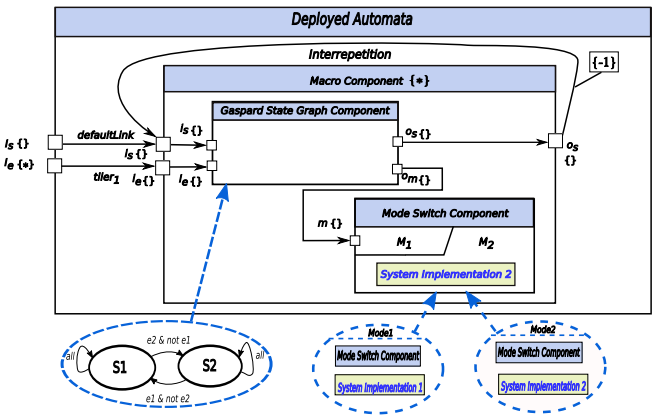

Figure 9: Integrating control at deployment level

Figure 9 shows the integration of control at the deployment level in Gaspard2. As compared to control models at other levels, e.g., such as application level, which only incorporate structural design aspects, this control model deals with behavioral aspects. The deployment level automata, termed as `Deployed Mode Automata` deals with atomic elementary components and their related implementations. These implementations are present at the lowest hierarchical level in the modeling; in order to address global system level implementations. As compared to other control models, a mode in a mode switch component represents a global system implementation of the modeled application, and is a collection of different implementations associated with their respective elementary components. Here, dataflow associated to the generic Gaspard mode automata is not explicitly expressed and input/output data ports are suppressed at all hierarchical levels in the control model at deployment level.

Also we need to address the issue related to the incoming events arriving in a deployed mode automata. In a control model at application or architecture, the events arrive either from the external environment, or the events are produced at any time instant in the application or architecture itself due to the actions of some elementary components. However in the deployment level, the incoming events are not related to the high level modeling but are basically used to represent low level user inputs depending upon the chosen execution platform. For example at the RTL level, these user events can arrive in the form of user or environment input from a camera attached to an FPGA, or inputs received via an universal asynchronous receiver/transmitter or UART terminal. A designer modeling the system at a high level is not concerned with these low level implementation details. However, in order to make this control model as flexible as possible, and to respect the semantics of the abstract control model, event ports have been added to control-deployment proposal. During the model transformations and eventual code generation, these event ports are replaced and translated into actual event values which are used during FPGA implementation phase.

Similarly, for mode automaton at an application or architecture level, its initial state is usually determined by a component that has input event ports and an output state port. Initially some events are generated and taken as input by that component in order to produce the initial state. After that, the component remains inactive due to the absence of the events arriving on its input ports. This initial state is then sent to the mode automata and serves to determine the initial state of the Gaspard state graph. However, for the deployed mode automata, structural aspects are absent and only information related to elementary components is present. Thus the initial state related to the deployed Gaspard state graph cannot be determined explicitly. This limitation has been removed by introducing new concepts at the deployment level, which help to determine the initial state of the deployed mode automata. However, the proposal retains the usage of an initial state port and the `defaultLink` concept, as they help to conform to the abstract control model; and are used in subsequent model transformations for eventual code generation.

Finally, the current control at deployment is only related to creating a state machine for a reconfigurable controller. In cases of FPGAs supporting several embedded hardcore/softcore processors; it is possible to select any one to act as a controller. However, this requires additional allocation types semantics to be linked to the deployment. Currently the code generated from our design flow is

explicitly linked to a generic controller, and it is up to the user to determine the nature and position of the controller.

6 Case study

In order to validate our design flow, we now present a case study of an anti-collision radar detection system. Vehicle based anti-collision radar detection systems are becoming increasingly popular in automotive industry as well as in research. Furthermore, these devices provide additional safety to provide collisions and fatal accidents; and could become mandatory aboard vehicles in the next years. The principle of the system is to avoid collision between the equipped vehicle and the one in front, or other kind of obstacles such as pedestrians, animals. The algorithms which form the basis of these complex systems require large amounts of regular repetitive computations. This computational necessity requires the execution of these algorithms in parallel hardware architectures, such as hardware accelerators. We first provide a general overview of these systems, followed by the modeling of their key components and eventual code generation. Finally the paper provides implementation details for integrating aspects of dynamic reconfiguration in these systems.

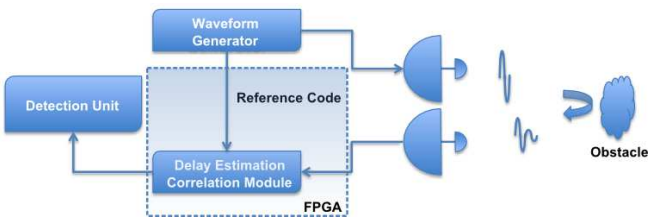

Figure 10: Block diagram of the anti-collision radar detection system

The anti-collision radar detection system is illustrated in Figure 10. The radar consists of two antennas; and emits a signal modulated with a *Pseudo Random Binary Sequence* (PRBS), resulting in formation of a reference code (Quadri, Elhillali, Meftali & Dekeyser 2009). The PRBS has interesting correlation as well intercorrelation characteristics (Douadi et al. 2008). When the transmitted wave encounters an obstacle, it is reflected and creates an echo which is captured by means of the second antenna. The echo is converted into a signal containing information related to the distance of the detected obstacle. Unfortunately, this information cannot be directly interpreted due to the presence of time delays and noise in the incoming signal. The PRBS present in the incoming signal is recognized by means of a *Delay estimation correlation module* or DECM present in the embedded system; and determines the time of flight. Thus, distance to the object and its speed can be calculated easily.

Also, it is not mandatory to exploit all the precision of the returned signal, since the information contained in the least significant bits is embedded with noise. In (Douadi et al. 2008), the authors recommend to use only 4 bits of the incoming signal, because the information contained in the 5th and the following bits are not significant.

For the radar system, the DECM can be implemented on an FPGA, as these reconfigurable SoCs allow to execute the detection algorithm that retains the necessary information present in the incoming signal. This information corresponds to the PRBS utilized in the emission of the signal. The role of the detection algorithm is to highlight the similarities between the reference code and the received signal: when the received signal corresponds with the reference code, the presence of an obstacle is detected. The inverse case means that the received signal contains little or no information related to the reference code and therefore, objects are not effectively detected.

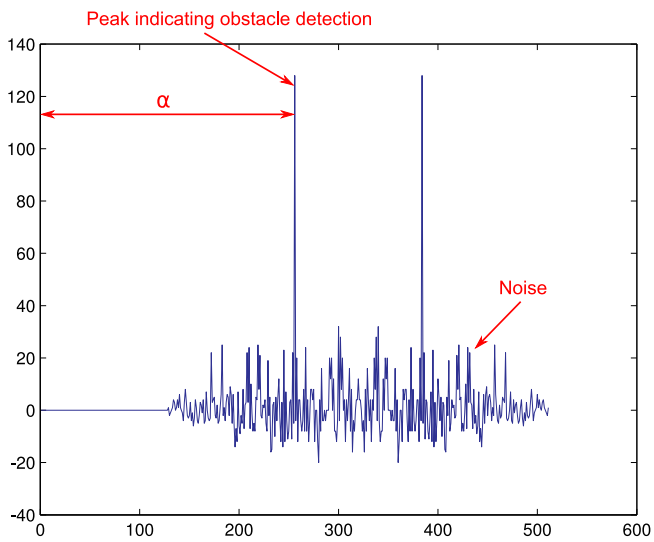

Figure 11: MATLAB result of the correlation

Figure 11 shows the result of a simulated correlation measurement in MATLAB. The outcome of a correlation between the reference code of a 127 length PRBS and the received simulated response (integrated with time delays and noise) yields a peak as indicated in the figure. A peak indicates the successful detection of an obstacle, and its position corresponds to the delay that we have introduced in the simulation. As the radar emits and receives a signal continuously in a temporal dimension, the correlation step is also repeated continuously, resulting in a peak at different intervals of time, indicating object detection at different time intervals. In the Figure, we illustrate the results of two correlations. To perform the necessary obstacle detection with the radar, the correlation peaks need to be localized, between the emitted code and its returned echo. The object which is detected by the correlation has a distance d from the radar, which is given by:

$$d = c\alpha/2 \tag{1}$$

Where c is the speed of the propagated signal (equal to 3.10^8, corresponding to the speed of light); and α is the respective time delay.

In this section, we have presented the structure of the anti-collision radar detection system. The DECM module

is the key element of this radar detection system, and the correlation computation is very time consuming especially for longer PRBSs. Our case study is mainly concerned with this functionality; details related to the modeling of the DECM module have been presented in (Quadri, Elhillali, Meftali & Dekeyser 2009, Quadri, Muller, Meftali & Dekeyser 2009), hence in the context of this paper, we only provide the top hierarchical level of our modeled application.

6.1 Delay estimation correlation module

Correlation algorithms are among the type of digital processing largely employed in DSP (digital signal processing) based systems. They offer a large applicability range such as linear phase and stability. A correlation algorithm normally takes some input data values and computes an output which is then multiplied by a set of coefficients. Afterwards the result of this multiplication is added together to produce the final output. While a software implementation can be utilized for implementing this functionality, the correlation functionality will be sequentially executed. Where as a hardware implementation allows the correlation functions to be executed in a parallel manner and thus increases the processing speed. However this implementation is not flexible for minute changes, hence a reconfigurable DECM module is an ideal solution as it offers the flexibility of a software implementation while retaining the capability to construct customized high performance computing circuits.

Figure 12 represents the top level of our modeled DECM module. The component instance `trm` of the component `TimeRepeatedMultiplicationAddition` determines the global multiplications while instance `trat` of component `TimeRepeatedAdditionTree` determines the overall sum. The `TimeRepeatedMultiplicationAddition component` itself carries out a partial sum between received elements of the reference code and the received signal at each clock cycle, which are then sent to the `TimeRepeatedAdditionTree` component to execute the overall addition operation. The instance `trdg` of component `TimeRepeatedDataGen` produces the data values for the generated incoming signal while the instance `trcg` of component `TimeRepeatedCoeffGen` produces the reference code.

We have identified four key elementary components `CoeffGen`, `DataGen`, `MultiplicationAddition` and `Addition` in our modeled application as shown in Figure 12. They are present in different levels of hierarchy in the components `TimeRepeatedCoeffGen`, `TimeRepeatedDataGen`, `TimeRepeatedMultiplicationAddition` and `TimeRepeatedAdditionTree` respectively at the top level of the application in Figure 12. Any change in the implementations of any of the elementary component directly affects the final result as well as other QoS criteria: such as reconfiguration time; consumed FPGA resources and the computation power. Deployment of these elementary components such as that of `MultiplicationAddition` can effect the overall QoS results. While it is theoretically possible to have a large number of configurations, only two have been considered for our case study. In this paper, we propose to associate two different implementations related to the `MultiplicationAddition` component, one written in a DSP like fashion, while other written using an If-then-else construct. While changing of an IP related to an elementary component might seem insignificant, it causes a global influence resulting in different QoS end results related to the DECM module.

Afterwards, the deployment phase is carried out as illustrated as abstractly represented in Figure 13 and all the elementary components are deployed. Then the modeling of the mode automata related to the DECM is initiated; with the mode automata serving to switch between the different DECM configurations.

Figures 14 and 15 illustrate the various concepts related to the construction of the mode automata. This modeling approach corresponds to the abstract control concepts introduced earlier in the paper, thus redundant explanatory information is unnecessary. Here the `DECM State Graph` contains two states `state_DECM_DSP` and `state_DECM_Ifelse` corresponding to the respective configurations modeled previously. This state graph is related to the `DECM State Graph Component` that serves as a control component. Its counterpart, the controlled mode switch component or `DECM MSC`, contains several collaborations. Each collaboration is signifying the internal behavior of the mode switch components on the basis of their interior parts and the incoming mode value on mode switch component's port. The combination of the control and controlled component forms the basis of a macro component that represents a single transition in the mode automata. For continuous transitions, the macro is placed in a repetitive context task: the `DECM Mode Automata` component along with its respective `tilers`, `interRepetition` and `defaultLink` dependencies.

As mentioned previously, one of goals of our design flow is the creation of a dynamically reconfigurable hardware accelerator with several configurations, that can be swapped dynamically in a run time reconfigurable SoC. The UML model is transformed by the various model transformations present in our design methodology, they generate the various implementations related to the hardware functionality. The control model is equivalently converted into a state machine for eventual utilization by the reconfiguration controller of the SoC in question. Figure 16 represents the Gaspard2 framework in the Eclipse environment as well as the different models present in our design flow. The *UML model* corresponds to the modeled control integrated deployed functionality and is directly generated from the UML diagram with integrated MARTE profile. The model-to-model transformations mentioned earlier permit to create several intermediate models, such as the RTL model corresponding to its own metamodel. Finally, the last step of our design flow consists of the generation of the source code related to the hardware accelerator and the configuration controller, by means of a model-to-text transformation.

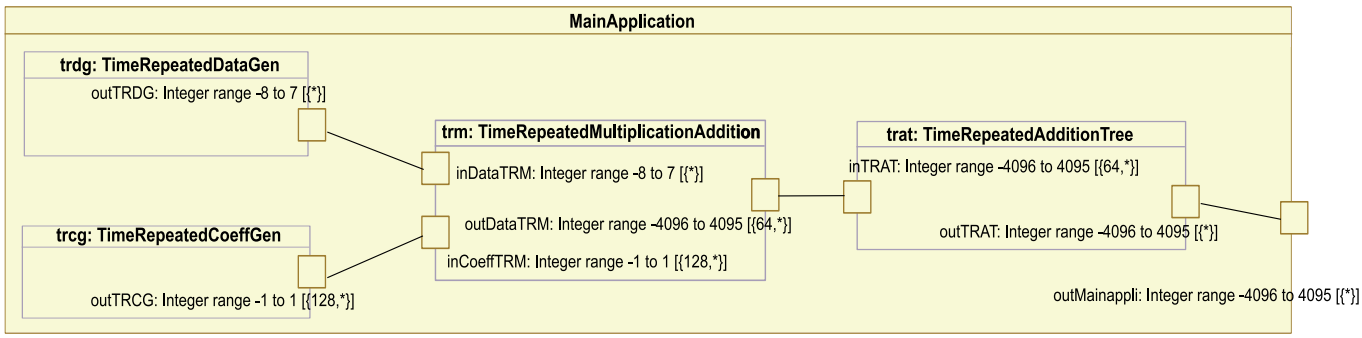

Figure 12: The top level view of the DECM

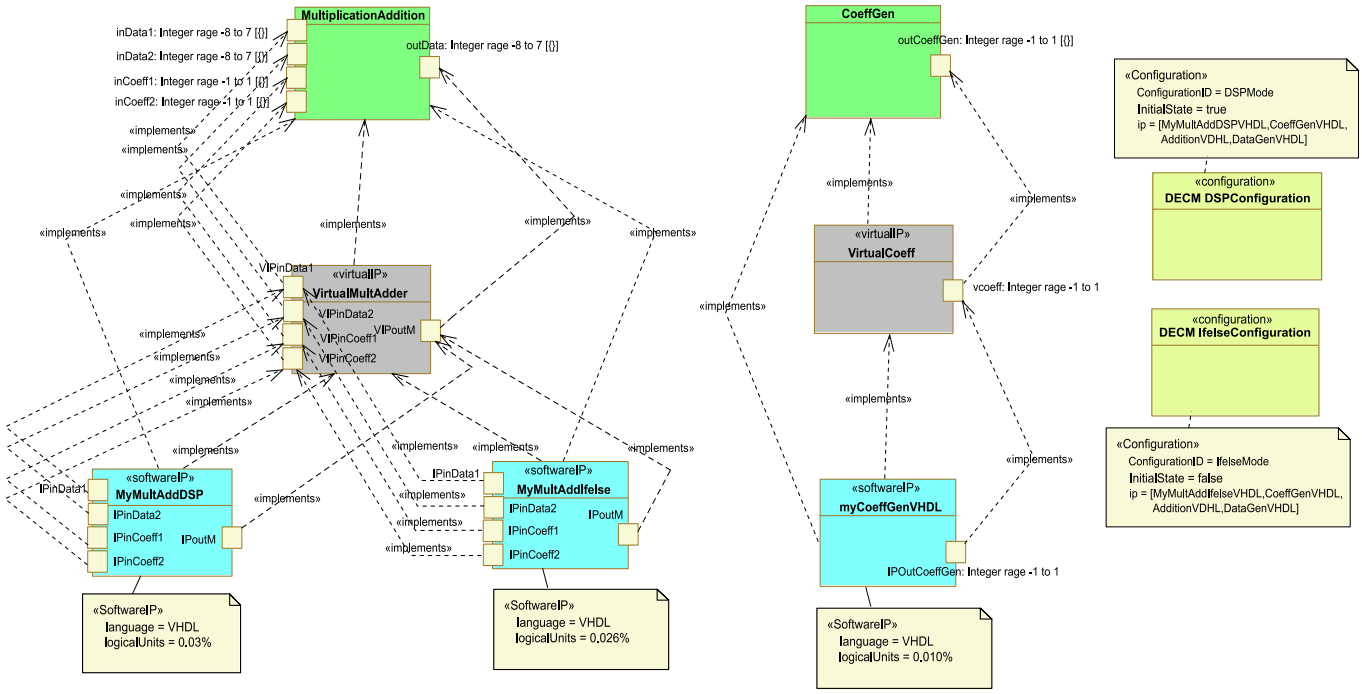

Figure 13: Deployment of the elementary components of the DECM

We now present some of the simulation results related specifically to the generated hardware functionality.

6.2 *Simulation of the modeled functionality*

The verification of the modeled application and its eventual equivalent hardware execution is first carried out by means of simulation using the industry standard ModelSim[6] simulation tool. Once the code for the various configurations has been generated from the model transformations, we move on to the simulation phase. The simulation helps to verify the correctness of the generated functionality, before moving on to the eventual FPGA synthesis. Figures 17 and 18 show the simulation results of the DSP configuration. Here in the simulation, we observe several intermediate integer values ranging between -100 to 200, between time intervals of 20 ns. These values are equivalent to the noise illustrated in the MATLAB simulation results. The simulation is set to run for about 8000 ns which is sufficient to observe the execution time of the application functionality. In the simulation, besides the intermediate noise values, we observe two significant integer values of more than 900 at two distinct time instants.

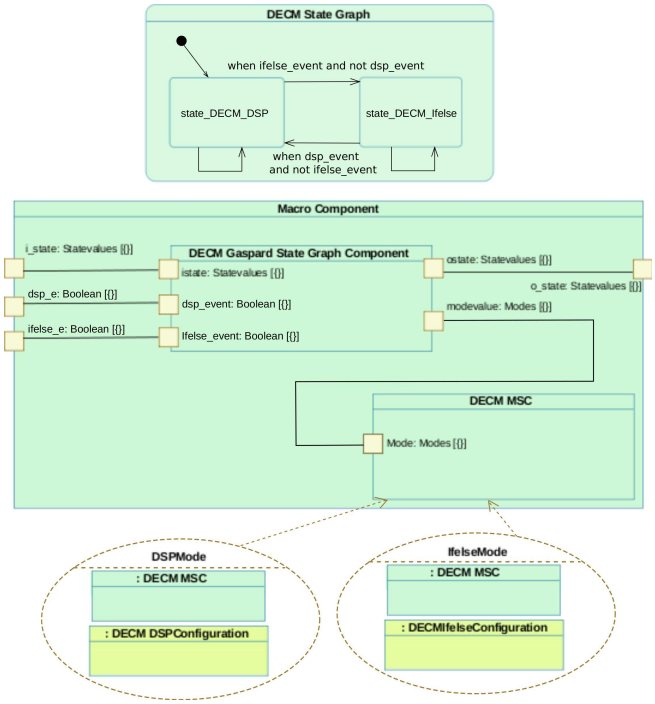

Figure 14: Mode automata concepts for the DECM: part one

These two values are equivalent to two peaks in the simulation, which correspond to the MATLAB simulation result illustrated in Figure 11. As the simulation results are a perfect match to the earlier MATLAB simulation results, the generated configuration is considered valid. The simulation results verify the functionality related to the different implementations of the high level modeled application functionality.

Currently in our design flow, we primarily make use of simulation to verify the correctness of the generated

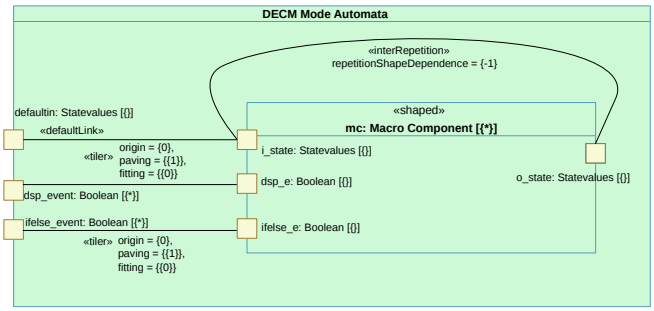

Figure 15: Mode automata concepts for the DECM: part two

code. However, it may be possible that the generated code does not produce expected results or the generated syntax was incorrect. As a solution, model based techniques such as traceability can be of extreme significance. Traceability helps to determine errors in the high level models and the corresponding model transformations. Currently this aspect is being studied in our research team for eventual integration in the Gaspard2 framework, and could be a future extension of our design flow (Aranega et al. 2009).

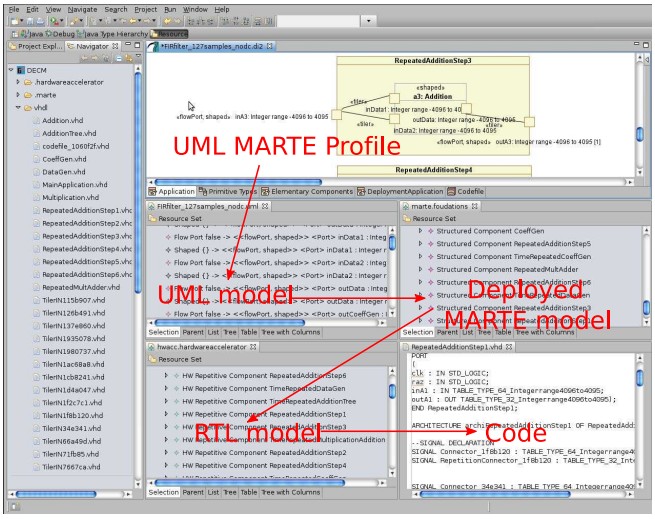

Figure 16: The transformation flow related to our design flow

Figure 17: First peak of the DSP configuration

Figure 18: Second peak of the DSP configuration

6.3 Implementing a partial dynamically reconfigurable DECM

In the previous sections, we have presented the initial details related to the application selected for this paper, along with its modeling at the MARTE profile level. Afterwards via the design flow presented during the course of this paper, the integrated model transformations generate the source code from the high abstraction level input models. Once the source code has been generated, we move onto implementing a partial dynamically reconfigurable SoC (Quadri, Muller, Meftali & Dekeyser 2009). This section deals with the implementation details and provides the validation of our design methodology.

We first investigated the architecture related to implementing partial dynamic reconfiguration in Xilinx FPGAs. In Figure 19 we present the global structure of our reconfigurable architecture that was implemented on the Xilinx Virtex-II Pro XC2VP30 FPGA on a XUP Board[7]. This particular type of structure is popular in the domain related to dynamic reconfigurable FPGAs, and various variants have been built from this classical structure, such as presented in (Claus et al. 2007, Tumeo et al. 2007). The choice of selecting the classical structure was 1) to compare our system with other existing partial dynamic reconfiguration based systems in literature, and 2) to provide the basic template for a model driven dynamically reconfigurable system that can be optimized by the domain experts, in order to generate their customized versions.

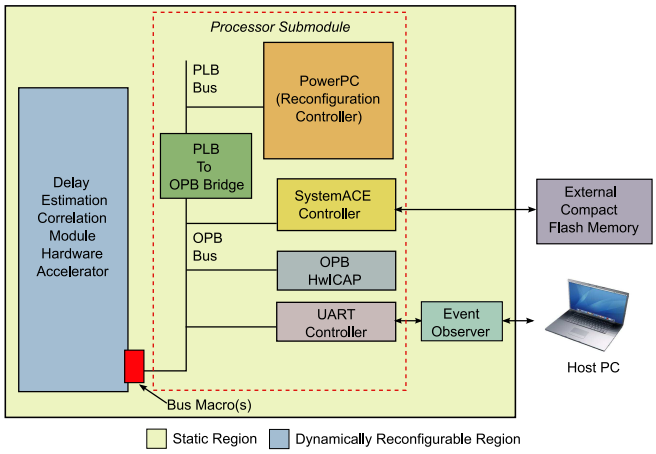

Figure 19: Block Diagram of the architecture of our reconfigurable system

In our selected system structure, we make use of the embedded hardcore PowerPCs present in the Xilinx Virtex-II Pro series FPGAs. One of the PowerPCs is selected as the reconfigurable controller and the state machine code generated from the high level control model in our design flow is executed on this processor. The partial dynamic reconfiguration system can be mainly divided into two main parts. The static region and the dynamically reconfigurable one. The static region mainly consists of a processor sub module that contains the reconfiguration controller and other necessary peripherals for dynamic reconfiguration.

The processor submodule is connected to a dynamically reconfigurable hardware accelerator via bus macros. These bus macros are communication modules that help in the communication between the static/dynamic regions. The hardware accelerator is equivalent to the hardware functionality generated from the high level modeled application in our design flow, and serves as the partially reconfigurable region in the overall system. The various implementations/partially reconfigurable modules related to the partially reconfigurable region are consistent with the modeled configurations at the deployment phase. The bus macros which are connected to the outputs of the hardware accelerator have a special enable/disable signal that permits the controller to disable the bus macros during a configuration switch to another state. Once a successful switch is carried out, the bus macros are enabled again. Thus during the switch, no output is generated from the partially reconfigurable region, causing the system to always remain in a safe state.

Finally the processor submodule system is connected to an event observer. The event observer receives the event values and relays them to the RS232 UART (universal asynchronous receiver/transmitter) controller of the processor submodule. Users can send input events, from the host PC, related to the configuration switches to the partial dynamic reconfiguration system, by means of a hyperTerminal. A program running on the hyperTerminal gives the user the choice of switching between the available configurations. Each configuration switch is related to a specific input character, that is mapped to a specific event in the executing controller program. When this value is received by the partial dynamic reconfiguration system, the associated state transition is carried out.

Once the code has been generated from our model driven design flow, we move on to the initial design partition phase of our partial dynamic reconfiguration system according to the Xilinx EAPR flow (Xilinx 2006). The processor submodule for the partial dynamic reconfiguration system is initially created by means of the Xilinx Platform studio. The source code for the controller is selected to be executed on the PowerPC 405_0, with a clock frequency of 100 MHz. The second PowerPC while present in the figure, is not connected to any clock signals and is therefore deactivated. A Processor Local Bus Block-RAM or PLB BRAM interface controller permits interfacing between the PLB and a Block-RAM of size 128 KB. This size is sufficient to store the data and instructions of the executable processor code, and on-chip-memory is not required. Using FPGA BRAMs to store the data/instructions allows the processor code and initialized variables to be written directly into the memory, when the FPGA is configured initially.

The processor subsystem is then inserted into a top level VHDL file that contains the component instantiations and port mappings related to the processor subsystem and the dynamically reconfigurable DECM module. This DECM module is connected to the static processor subsystem by means of bus macros which are also present in the top hierarchical level. Afterwards, synthesis is carried out to generate the appropriate files for eventual implementation of

Partial dynamic reconfiguration using the PlanAhead design tool (Xilinx 2006).

We now present the partial synthesis results of some of the modeled application components in our case study carried out with the Xilinx ISE on the XUP board. Figure 20 shows the global view of the synthesis of the modeled DECM application.

Once the synthesis has been carried out, we move onto generation of the partial bitstreams related to the different modeled configurations as well as the static bitstream. Figure 21 only shows the partial bitstream related to the DSP configuration, while Figure 22 shows the full initial bootup bitstream that is a merge of the static bitstream and the DSP partial bitstream.

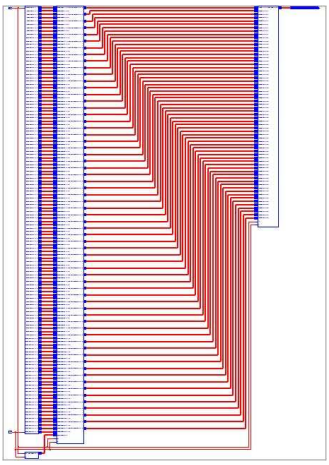

Figure 20: Synthesis result of the top level of the DECM

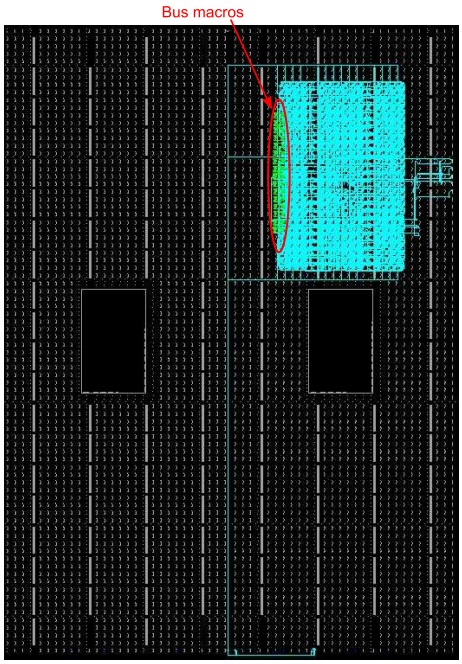

Figure 21: Partial bitstream related to the DSP configuration

Table 1 shows the results related to the two configurations. The first configuration consumes slightly more FPGA resources as compared to the second one. Additionally, the reconfiguration time for the first configuration is higher as compared to the second one. This is due to the fact that the ICAP core needs to modify several additional frames for the first configuration, as compared to the latter. While the reconfiguration time is extremely high for both configurations, this is due to the low bandwidth: 115200 bps; of the RS232 controller and the large size of the partial bitstreams. Using an external RAM memory can greatly increase the reconfiguration times, similarly various other optimizations can be carried out with respect to this implementation, such as introducing a DMA in the reconfigurable system, a customized ICAP controller or usage of a PLB ICAP core.

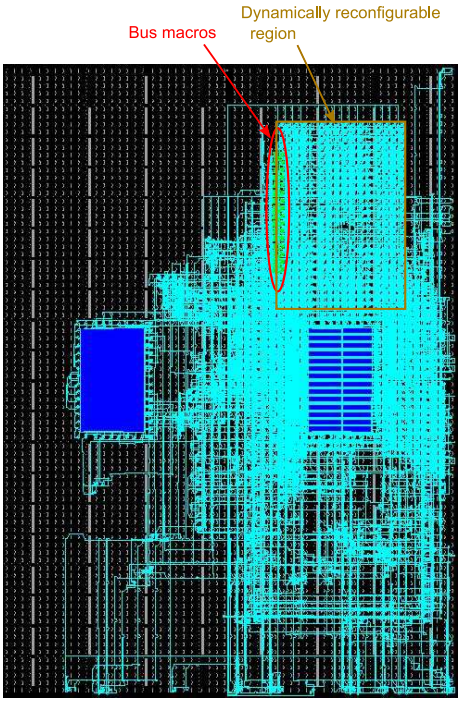

Figure 22: Full bitstream related to the partial dynamic reconfiguration system

	DSP Configuration	If-then-else Configuration
Slices	1272/13696 (9.287%)	1186/13696 (8.659%)
Slice FlipFlops	2084/27392 (7.608%)	1944/27392 (7.096%)
LUTs	1584/27392 (5.782%)	1836/27392 (6.702%)
Time (secs)	1.45	1.41

Table 1 Results related to the two configurations for the hardware accelerator. The percentage is in overview of the total FPGA resources.

7 Control models and FPGA synthesis

Many different approaches exist for expressing control semantics, such as Petri Nets (Nascimento et al. 2004); *if-then-else*, *switch* and *goto*-based semantics. However mode automata were selected as they clearly separate

control/data flow. They also adapt a state based approach facilitating seamless integration in our framework; and can be expressed at the MARTE specification levels. The same control semantics are then used throughout our framework to provide a single homogeneous approach.

With regard to partial dynamic reconfiguration, different implementations of a reconfigurable region must have the same external interface for integration with the static region at run-time. Mode automata control semantics can express the different implementations collectively via the concept of a mode switch, which can be expressed graphically at high abstraction levels using the concept of a mode switch component. Similarly a state graph component expresses the controller responsible for the context switch between the different implementations.

Both control models expressed in this paper can be used for FPGA synthesis. The first model introduces dynamic aspects in the application, which may modify the structure of an application, as different modes in a mode switch component can have different natures. For example, one mode could be elementary in nature while another can be hierarchically composed. The application can be associated and correspondingly deployed in different ways. The whole application could be allocated and deployed onto a single processor or a hardware accelerator; or split into parts. The first case while allowing reconfiguration, is not dynamically reconfigurable in nature.

In the second case, the parts can be correspondingly allocated and deployed: the state graph component onto a processor, and the mode switch components onto the reconfigurable regions. The disadvantage of this approach is that multiple allocations are required between the application and the targeted architecture. This control model can also be applied onto the architecture level of our framework. A processor can have a reconfiguration manager observing QoS criteria. If the processor heats up during execution, the manager can change the frequency of the processor. An internal controller could carry out aspects of dynamic reconfiguration. In addition, an external controller can be used to globally change the architecture.

The deployment level control model renders the application or architecture reusable. The designer can change partial functionality of the application by changing some IPs related to corresponding elementary components. This model is thus more interesting as one application functionality can be reused without changing its structure, and its overall implementation can be changed only depending upon QoS criteria and hardware resources limitations. Currently the deployment level is explicitly linked with a specific reconfigurable controller in the targeted FPGA. In case of multiple processors: one managing the configuration and the other executing some application functionality, this information must be elevated to the allocation level to correctly link the related entities. It is also possible to combine the two control models for simultaneous integration in our framework.

Additionally, some of the low level RTL details can be integrated into high level models by means of a *Design Space Exploration* strategy. For example, reconfiguration time related to a specific configuration can be determined after an initial synthesis and then can be added to models by means of UML attributes. This will allow designers to take QoS criteria into consideration when changing between the different configurations. Additionally, details related to the targeted architecture can also be modeled at the high abstraction levels. An initial contribution related to this aspect has been detailed in (Quadri, Meftali & Dekeyser 2009*a*); and could be a future extension of our design flow.

8 Conclusion

This paper presented a high level design flow for targeting reconfigurable SoCs in the context of a model-driven co-design framework, Gaspard2, which is compliant with the MARTE standard. We have selected two key points of these reconfigurable systems to be modeled at high abstraction levels. We have mainly taken into account the dynamically reconfigurable region and the semantics related to the reconfigurable controller; managing for switching between the different implementations related to the region. The control semantics is based on mode automata and is integrated at different levels of SoC co-design. In the context of partial dynamic reconfiguration, they have been integrated at the deployment level in our Gaspard2 framework. Afterwards, a case study consisting of a component in an anti-collision radar detection system has been illustrated for validating the proposed design methodology. Our high-level modeling approach enables implementation and FPGA synthesis of various system configurations rapidly in a flexible manner. It therefore helps to explore different design choices about the system, which is usually a delicate task. Finally, we provide a comparison between the control semantics at different design levels, specifically for FPGA synthesis and dynamic reconfiguration.

References

Aranega, V., Mottu, J.-M., Etien, A. & Dekeyser, J.-L. (2009), Traceability Mechanism for Error Localization in Model Transformations, *in* '4th International Conference on Software and Data Technologies (ICSOFT 2009)', Sofia, Bulgaria. To Appear.

Atitallah, R. B., Piel, E., Niar, S., Marquet, P. & Dekeyser, J.-L. (2007), Multilevel MPSoC simulation using an MDE approach, *in* 'SoCC 2007'.

Bayar, S. & Yurdakul, A. (2008), Dynamic Partial Self-Reconfiguration on Spartan-III FPGAs via a Parallel Configuration Access Port (PCAP), *in* 'HiPEAC'08 Workshop on Reconfigurable Computing'.

Becker, J., Huebner, M. & Ullmann, M. (2003), Real-Time Dynamically Run-Time Reconfigurations for Power/Cost-optimized Virtex FPGA Realizations, *in* 'VLSI'03'.

Bergmann, N., Williams, J. & Waldeck, P. (2003), Egret: A flexible platform for real-time reconfigurable system-on-chip, *in* 'Proceedings of International Conference on Engineering Reconfigurable Systems and Algorithms', pp. 300–303.

Blodget, B., McMillan, S. & Lysaght, P. (2003), A lightweight approach for embedded reconfiguration of FPGAs, *in* 'Design, Automation & Test in Europe, DATE'03'.

Boulet, P. (2007), Array-OL revisited, multidimensional intensive signal processing specification, Research Report RR-6113, INRIA, `http://hal.inria.fr/inria-00128840/en/`.

Boulet, P. (2008), Formal Semantics of Array-OL, a Domain Specific Language for Intensive Multidimensional Signal Processing, Technical Report 6467, INRIA, France. `http://hal.inria.fr/inria-00261178/en`.

Brito, A., Kuhnle, M., Hubner, M., Becker, J. & Melcher, E. (2007), Modelling and Simulation of Dynamic and Partially Reconfigurable Systems using SystemC, *in* 'Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI'07)', IEEE Computer Society, pp. 35–40.

Claus, C., Muller, F., Zeppenfeld, J. & Stechele, W. (2007), 'A new framework to accelerate Virtex-II Pro dynamic partial self-reconfiguration', *IPDPS 2007* pp. 1–7.

Douadi, L., Deloof, P. & Elhillali, Y. (2008), Real time implementation of reconfigurable correlation radar for road anticollision system, *in* 'IEEE International Conference on Industrial Technology (ICIT 2008)', pp. 1–7.

Ebeling, C., Cronquist, D. & Franklin, P. (1996), Rapid - reconfigurable pipelined datapath, *in* 'Proceedings of the 6th International Workshop on Field-Programmable Logic, Smart Applications, New Paradigms and Compilers (FPL' 96)', Springer-Verlag, pp. 126–135.

F. Berthelot and F. Nouvel and D. Houzet (2008), 'A Flexible system level design methodology targeting run-time reconfigurable FPGAs', *EURASIP Journal of Embedded Systems* **8**(3), 1–18.

Faugere, M., Bourbeau, T., Simone, R. & Sebastien, G. (2007), MARTE: Also an UML Profile for Modeling AADL Applications, *in* 'ICECCS '07: Proceedings of the 12th IEEE International Conference on Engineering Complex Computer Systems', IEEE Computer Society, pp. 359–364.

Gamatié, A., Le Beux, S., Piel, E., Ben Atitallah, R., Etien, A., Marquet, P. & Dekeyser, J.-L. (2010), 'A model driven design framework for massively parallel embedded systems', *ACM Transactions on Embedded Computing Systems (TECS)* . To appear.

Gamatié, A., Rutten, E. & Yu, H. (2008), A Model for the Mixed-Design of Data-Intensive and Control-Oriented Embedded Systems, Research Report RR-6589, INRIA, `http://hal.inria.fr/inria-00293909/fr`.

Gamatié, A., Rutten, E., Yu, H., Boulet, P. & Dekeyser, J.-L. (2008), 'Synchronous Modeling and Analysis of Data Intensive Applications', *EURASIP Journal on Embedded Systems* . Hindawi Publishing Corporation.
URL: *http://dx.doi.org/10.1155/2008/561863*

Harel, D. (1987), 'Statecharts: A Visual Formalism for Complex Systems', *Science of Computer Programming* **8**(3), 231–274.

INRIA Atlas Project (n.d.), 'ATL', `http://modelware.inria.fr/rubrique12.html`.

INRIA DaRT team (2009), 'GASPARD SoC Framework'. `http://www.gaspard2.org/`.

INRIA Triskell Project (n.d.), 'Kermeta', `http://www.kermeta.org/`.

J.P. Diguet and G. Gogniat and J.-L. Philippe et al (2006), 'EPICURE: A partitioning and co-design framework for reconfigurable computing', *Journal of Microprocessors and Microsystems* **30**(6), 367–387.

Koch, R., Pionteck, T., Albrecht, C. & Maehle, E. (2006), An adaptive system-on-chip for network applications, *in* 'IPDPS 2006'.

Koudri, A., Aulagnier, D., Vojtisek, D., Soulard, P., Moy, C., Champeau, J., Vidal, J. & Lann, J.-C. (2008), Using MARTE in the MOPCOM SoC/SoPC Co-Methodology, *in* 'MARTE Workshop at DATE'08'.

Labbani, O., Dekeyser, J.-L., Boulet, P. & Rutten, E. (2005), Introducing control in the Gaspard2 Data-Parallel MetaModel: Synchronous Approach, *in* 'Proceedings of the International Workshop MARTES: Modeling and Analysis of Real-Time and Embedded Systems'.

Latella, D., Majzik, I. & Massink, M. (1999), Automatic Verification of a Behavioral Subset of UML Statechart Diagrams Using the SPIN Model-Checker, *in* 'Formal Aspects Computing', Vol. 11, pp. 637–664.

Le Beux, S. (2007), Un flot de conception pour applications de traitement du signal systématique implémentées sur FPGA à base d'Ingénierie Dirigée par les Modèles, PhD thesis, University of Lille 1, France.

Lysaght, P., Blodget, B. & Mason, J. (2006), Invited Paper: Enhanced Architectures, Design Methodologies and CAD Tools for Dynamic Reconfiguration of Xilinx FPGAs, *in* 'FPL'06'.

Maraninchi, F. & Rémond, Y. (2003), 'Mode-automata: a new domain-specific construct for the development of safe critical systems', *Sci. Comput. Program.* **46**(3), 219–254.

Moore, G. (1965), 'Cramming more components into integrated circuits', *Electronics* **38**(8), 114–117.

Morpheus (2010), 'Multi-purpose dynamically reconfigurable platform for intensive heterogeneous processing'. `http://www.morpheus-ist.org/`.

Nascimento, B., Sérgio, P., Maciel, M., Romero, P., Lima, M., Sant'ana, R., Filho, S. & Guilhermino, A. (2004), A partial reconfigurable architecture for controllers based on Petri nets, *in* 'SBCCI '04: Proceedings of the 17th symposium on Integrated circuits and system design', pp. 16–21.

Nezami, K. G., Stephens, P. W. & Walker, S. D. (2008), Handel-C Implementation of Early-Access Partial-Reconfiguration for Software Defined Radio, *in* 'IEEE Wireless Communications and Networking Conference (WCNC'08)', pp. 1103–1108.

OMG (2005), 'MOF Query /Views/Transformations'. `http://www.omg.org/cgi-bin/doc?ptc/2005-11-01`.

OMG (2007*a*), 'M2M/Operational QVT Language'. `http://tiny.cc/tFGGx`.

OMG (2007*b*), 'Portal of the Model Driven Engineering Community'. `http://www.planetmde.org`.

OMG (2008), 'Modeling and Analysis of Real-time and Embedded systems (MARTE)', http://www.omgmarte.org/.

Panainte, E. M., Bertels, K. & Vassiliadis, S. (2007), 'The molen compiler for reconfigurable processors', *ACM Transactions in Embedded Computing Systems (TECS)* **1**.

Paulsson, K., Hubner, M., Auer, G., Dreschmann, M., Chen, L. & Becker, J. (2007), 'Implementation of a Virtual Internal Configuration Access Port (JCAP) for Enabling Partial Self-Reconfiguration on Xilinx Spartan III FPGA', *FPL 2007* pp. 351–356.

Pillement, S. & Chillet, D. (2009), High-level model of dynamically reconfigurable architectures, pp. 1–7.

Quadri, I. R., Boulet, P., Meftali, S. & Dekeyser, J.-L. (2008), Using an mde approach for modeling of interconnection networks, *in* 'The International Symposium on Parallel Architectures, Algorithms and Networks Conference (ISPAN 08)'.

Quadri, I. R., Elhillali, Y., Meftali, S. & Dekeyser, J.-L. (2009), Model based design flow for implementing an Anti-Collision Radar system, *in* '9th International IEEE Conference on ITS Telecommunications (ITS-T 2009)'.

Quadri, I.-R., Meftali, S. & Dekeyser, J.-L. (2009*a*), 'A Model Driven design flow for FPGAs supporting Partial Reconfiguration', *International Journal of Reconfigurable Computing* . Hindawi Publishing Corporation.

Quadri, I. R., Meftali, S. & Dekeyser, J.-L. (2009*b*), Integrating Mode Automata Control Models in SoC Co-Design for Dynamically Reconfigurable FPGAs, *in* 'International Conference on Design and Architectures for Signal and Image Processing (DASIP 09)'.

Quadri, I. R., Muller, A., Meftali, S. & Dekeyser, J.-L. (2009), MARTE based design flow for Partially Reconfigurable Systems-on-Chips, *in* '17th IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC 09)'.

S. Sendall and W. Kozaczynski (2003), 'Model Transformation: The Heart and Soul of Model-Driven Software Development', *IEEE Software* **20**(5), 42–45.

Salefski, B. & Caglar, L. (2001), Reconfigurable computing in wireless, *in* 'Proceedings of the 38th annual Design Automation Conference (DAC'01)', ACM, pp. 178–183.

Schäfer, T., Knapp, A. & Merz, S. (2001), Model Checking UML State Machines and Collaborations, *in* 'CAV Workshop on Software Model Checking', ENTCS 55(3).

Schuck, C., Kuhnle, M., Hubner, M. & Becker, J. (2008), A framework for dynamic 2D placement on FPGAs, *in* 'IPDPS 2008'.

Sedcole, P., Blodget, B., Anderson, J., Lysaght, P. & Becker, T. (2005), Modular Partial Reconfiguration in Virtex FPGAs, *in* 'FPL'05', pp. 211–216.

SPEEDS! (2009), 'SPEculative and Exploratory Design in Systems Engineering'. `http://www.speeds.eu.com/`.

Talpin, J.-P., Brunette, C., Gautier, T. & Gamatié, A. (2006), Polychronous Mode Automata, *in* A. Press, ed., 'Embedded Software Conference'.

Tumeo, A., Monchiero, M., Palermo, G., Ferrandi, F. & Sciuto, D. (2007), 'A Self-Reconfigurable Implementation of the JPEG Encoder', *ASAP 2007* pp. 24–29.

Vidal, J., Lamotte, F. D. & Gogniat, G. (2009), A co-design approach for embedded system modeling and code generation with UML and MARTE, *in* 'Design, Automation and Test in Europe (DATE'09)'.

Xilinx (2006), Early Access Partial Reconfigurable Flow. `http://www.xilinx.com/support/prealounge/protected/index.htm`.

Yu, H. (2008), A MARTE-Based Reactive Model for Data-Parallel Intensive Processing: Transformation toward the Synchronous Model, PhD thesis, USTL/LIFL, France.
URL: *http://sites.google.com/site/huafengyu/pub-files/thesis.pdf*

Yu, H., Gamatié, A., Rutten, E. & Dekeyser, J.-L. (2008), 'Safe Design of High-Performance Embedded Systems in a MDE framework', *Innovations in Systems and Software Engineering (ISSE)* **4**(3), 215–222.

Note

[1]http://www.systemc.org/
[2]www.papyrusuml.org/
[3]www.eclipse.org/emf/
[4]http://www.acceleo.org/pages/home/en
[5]http://www.omg.org/cgi-bin/doc?ad/2004-4-7
[6]http://www.model.com/
[7]http://www.xilinx.com/univ/xupv2p.html