Designing dynamically reconfigurable SoCs: From UML MARTE models to automatic code generation

Imran Rafiq Quadri, Samy Meftali, Jean-Luc Dekeyser

Designing dynamically reconfigurable SoCs: From UML MARTE models to automatic code generation

Imran Rafiq Quadri, Samy Meftali and Jean-Luc Dekeyser,
INRIA LILLE NORD EUROPE - LIFL - University of Lille - CNRS, Lille, France
{Imran.Quadri, Samy.Meftali, Jean-Luc.Dekeyser}@lifl.fr

***Abstract*—Due to continuous hardware/software evolution related to Systems-on-Chip (SoC) and the addition of features such as *Partial Dynamic Reconfiguration*, the complexity of SoC design and development has escalated exponentially. This has resulted in increased time to market and development costs. Without the usage of effective design tools and methodologies, large complex SoCs are becoming increasingly difficult to manage, resulting in a *productivity gap*. The design space, representing all technical decisions that need to be elaborated by the SoC design team is therefore, becoming immense and difficult to explore. Similarly, manipulation of these systems at low implementation levels such as *Register Transfer Level* (RTL) can be hindered by human interventions and the subsequent errors. This paper presents a novel design methodology that decreases the design complexity by raising the design abstraction levels. It makes use of Model-Driven Engineering and the UML MARTE profile to move from high level UML models to automatic code generation, for implementing dynamically reconfigurable SoCs.**

I. Introduction

Reconfiguration is becoming a pivotal feature of modern Systems-on-Chip (SoCs) based embedded systems, in an increasingly evolving market space. A reconfigurable SoC offers increased functional extensibility in return for lower performance. Being able to be reconfigured an arbitrary number of times, these systems offer designers the means to add new functionalities and make system modifications, after SoC fabrication. *Dynamic* or *Run-time reconfiguration*, is a special type of reconfiguration [1] that enables system modification during execution, and introduces the concept of *virtual hardware*. Hence designers can change the executing applications on these systems, depending upon *Quality-of-Service* (QoS) criteria, related to the environment or the platform: such as consumed surface area, energy consumption levels, etc. Currently, *Field Programmable Gate Array* (FPGA) based SoCs offer an ideal solution for implementing dynamic reconfiguration. Moreover, SoC application functionalities can be easily implemented as hardware designs on these reconfigurable SoCs. As compared to traditional SoCs, these dynamically reconfigurable SoCs offer advantages such as low energy consumption, increased flexibility; with the compromise of additional costs per unit. Currently only Xilinx based FPGAs support dynamic reconfiguration.

However, the evolution of SoC and features such as dynamic reconfigurable have increased the design complexity to new levels. Henceforth, designing effective SoCs is a challenging issue. Numerous methodologies and propositions have been proposed to reduce SoC design complexity. A *Platform* or *component* based approach is widely accepted in the SoC industry, enabling system conception in a compositional manner. The hierarchy related to the SoC is visible quite clearly, and designers are capable of re-utilizing components that have been developed either internally or by third parties. Other methodologies make use of high abstraction levels at different design levels, in order to elevate the low level technical details.

Unified design approach is an emerging research domain 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* [2]. MDE allows high level system modeling of both software and hardware, with the possibility of integrating heterogeneous components into the system. *Model transformations* [3] can then be carried out to generate executable models from the high level models. MDE is supported by several standards and tools.

The UML MARTE (Modeling and Analysis of Real-Time and Embedded Systems) profile is an upcoming industry standard of Object Management Group (OMG) [4], that is dedicated to model-driven development of embedded systems. MARTE extends UML, in order to model the features of software and hardware parts of a real-time embedded system and their relations, along with added extensions (for e.g. timing constraints, performance and scheduling analysis).

Gaspard2 [5], [6] is an MDE-based SoC Co-Design framework that integrates a subset of the MARTE profile for the design and development of parallel hardware/software; and allows to move from high level MARTE models to different execution platforms.

The contributions of this paper relate to presenting a MARTE oriented design flow that enables moving from high level UML models to automatic *Register Transfer Level* (RTL) code generation for implementing dynamically reconfigurable FPGA based SoCs. For this goal, we focus on two key concepts: Firstly, we generate the code for a dynamically reconfigurable region, that 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, we propose mode automata control semantics, that 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 high level design methodology.

The rest of this paper is organized as follows. Related works are detailed in section II, while an overview of Gaspard2 framework is provided in section III. Section IV describes our global design methodology, along with a brief overview of our contributions related to extension of the MARTE profile, control semantics, system configurations and model transformations in Gaspard2. Afterwards, section V presents our case study followed by a conclusion in section VI.

II. Related works

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 RTL by introducing new tools and methodologies. In this section, we only focus on the first aspect, details related to the second one can be found in [7].

The MoPCoM project [8] aims to target modeling and code generation of dynamically reconfigurable embedded systems using the MARTE profile [9]. While the authors claim that they are capable of creating a complete SoC Co-Design framework, in reality, only 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 [10], [11], 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 [12], the authors also provide a high level modeling methodology for implementing dynamically reconfigurable SoCs. They integrate an operating system (OS), for 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.

[13] use a SystemC based design flow for implementing partial dynamic reconfiguration. The SystemC kernel is modified for the integration of reconfiguration operations for activation/dis-activation 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 [14], 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 [15], a SynDEx based design flow is presented to manage SoC reconfigurability via implementation in FPGAs, with the application and architecture parts modeled as components.

There exists also large number of high level synthesis tools, that elevate the design abstraction levels. Numerous academic tools and commercial products have also been developed. On the commercial side, we found products such as CATAPULT C[2] and CODEVELOPER[3], while on the academic front, tools such as SPARK [16] and GAUT [17] are available. The drawback related to these tools is mainly that they are usually text based in nature making system hierarchy difficult to visualize and they are normally non-compatible. Hence a designer needs to become an expert in each different tool.

As compared to the above mentioned related works, we propose a design flow that provides abstraction at a level higher than those provided by high level synthesis tools. Additionally, the novelty of our approach is that we take into account aspects of complex SoC applications, control semantics, UML MARTE profile, SoC Co-Design and finally partial dynamic reconfiguration in FPGA based SoCs.

III. Gaspard2 : SoC Co-Design framework

Gaspard2 [5], [6] incorporates the UML MARTE profile for SoC specification and provides an *Integrated Development Environment* or IDE dedicated to the visual co-design of embedded systems. It is based on a repetitive model of computation that permits to express the potential parallelism in a system, enabling modeling of application loops and regular repetitive hardware structures in a compact manner. Gaspard2 integrates several MARTE packages and concepts such as *Allocation* (enabling mapping of a SoC application onto an architecture) and *Repetitive Structure Modeling* (RSM); the details to which can be found in [18].

Gaspard2 not only permits to specify the SoC *application*, targeted *architecture* and their *allocation* at the UML MARTE level, but also proposes an *Intellectual Property* (IP) deployment level that permits to link the elementary components of the application/architecture to user defined or third party IPs

[1] http://www.systemc.org/

[2] http://www.mentor.com/products/esl/high_level_synthesis/catapult_synthesis/

[3] http://www.impulseaccelerated.com/products_universal.htm

[11]. The only limitation of this level is that only a single IP can be assigned to an elementary component for final selection of an execution platform and eventual code generation. This results in a final static system implementation, which while true for normal architectures, is a critical drawback for dynamically reconfigurable SoCs.

A. Model transformations in Gaspard2

Models in MDE are not only used for communication and comprehension but using model transformations [3], 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 and technologies, such as targeted towards synchronous domain for validation and analysis purposes [19], [20]; Dynamic run-time reconfiguration in FPGAs [10], among others. Figure 1 illustrates the Gaspard2 framework and our specific contributions which are detailed later on in the paper. The model transformation chains permit moving from high abstraction levels in Gaspard2 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.

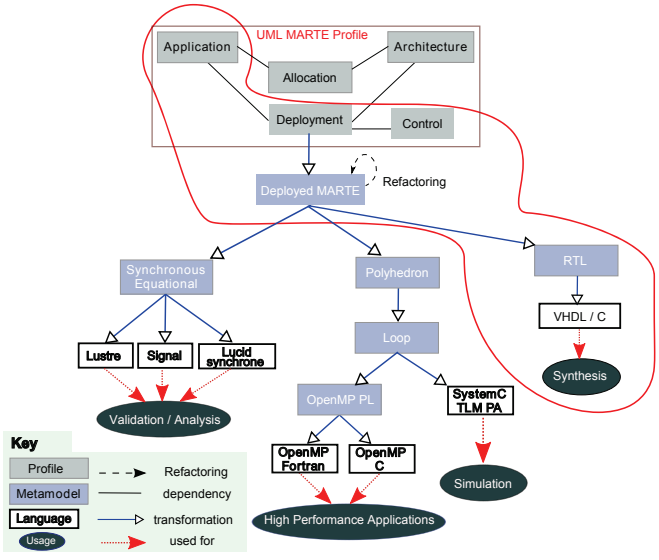

Figure.1: Overall view of Gaspard2 framework with emphasis on our specific contributions

Gaspard2 adapts QVTO [21], an implementation of MOF QVT standard [22] for model query and transformations, 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].

IV. Integrating Dynamic Reconfiguration in Gaspard2

A. Global overview of our design flow

We first give a brief description of our design flow. Figure 2 shows the global overview of the model transformation chain related to implementing partial dynamic reconfiguration in our design flow, as discussed earlier in this paper. Initially a complex data intensive parallel computation Gaspard2 application is modeled and deployed, along with the associated control semantics; in the Gaspard2 environment with Papyrus modeling tool[6] conforming to an extended version of the UML MARTE profile. This modeling is independent from any implementation details until the deployment phase.

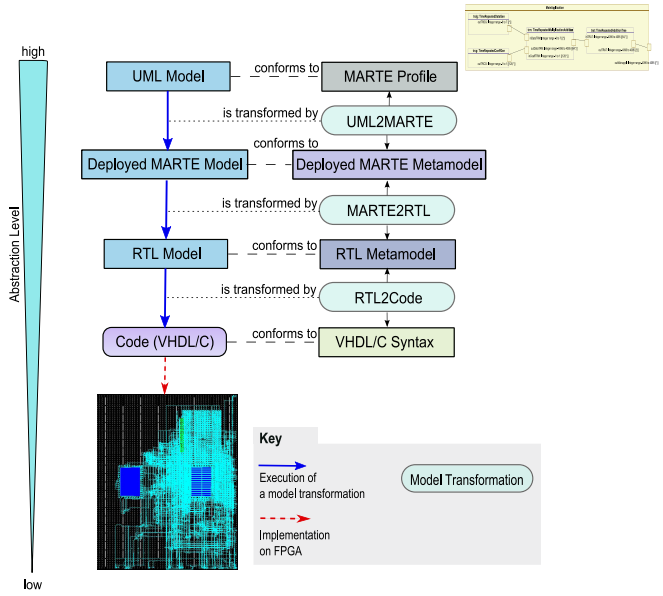

Figure.2: An abstract overview of the model transformation chain. Several intermediate metamodels help to bridge the gap between high level modeled UML diagrams and the final RTL code generation

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 electronic register transfer level. This model is considered as a low abstraction level and provides details related to creation of a dynamic hardware accelerator 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

[4]http://www.acceleo.org/pages/home/en
[5]http://www.omg.org/cgi-bin/doc?ad/2004-4-7
[6]http://www.papyrusuml.org/

partial code for the reconfiguration controller. Once the source code for the application and the reconfigurable controller is obtained, the reconfigurable FPGA based SoC can be created by using usual Xilinx design tools related to partial dynamic reconfiguration, such as Xilinx ISE [23] and EDK [24]. These aspects are detailed later on in the paper.

We now present the different contributions that enable us to create RTL code from high level UML MARTE models.

B. Introduction of generic control semantics

For dynamic reconfiguration in modern SoCs, an embedded controller is essential for managing a dynamically reconfigurable region. This key component is usually associated with some control semantics such as state machines, Petri nets etc; which in turn must be generic enough to be applied to both software/hardware design aspects. Similarly for a framework incorporating high abstraction levels and specially MDE, the control model must respect all the related criteria. While several control models exist, mode automata [25] based control are promising as they incorporate aspects of modularity present in component based approaches, for describing SoC in an incremental fashion to build these complex systems.

The controller normally has two functionalities: one responsible for communicating with the FPGA *Internal Configuration Access Port* hardware reconfigurable core or ICAP [26] 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, and is treated as a macro. Regarding the second functionality, being state based in nature, it can be easily modeled at high abstraction levels such as using UML state machine diagrams. Then the model transformations can be used for code generation. Afterwards the code is generated, it can be merged with the hand written macro for execution in the controller.

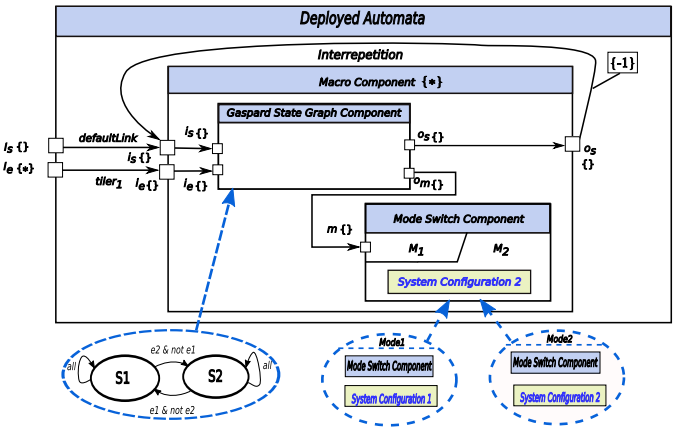

Figure.3: Overview of control concepts

Regarding the control semantics, we have provided a detailed description in [18] and here only a brief overview is given. Several key concepts related to our control semantics are illustrated in the abstract Figure 3, which is very close to actual UML modeling. A `Gaspard State Graph Component` acts as a controlling component and provides some input values to a `Mode Switch Component` acting as a controlled component. This component in turn contains several system `modes` all having the same external interface. Here the system modes correspond to different *configurations* of the modeled application. These configurations are modeled and illustrated later on in the paper.

The behavior of the controlling component is determined by a `State Graph` containing number of states equal to the modeled application configurations. Upon receiving some particular events, the transition between states takes place, producing a mode value by the `Gaspard State Graph Component` that is received by the `Mode Switch Component`, whose internal behavior is determined by some associated `Collaborations`. For example, a mode value `Mode1` received by the controlled component determines the behavior by means of the associated collaboration. We place a condition that only one exclusive mode can be executed at a particular time instant in a `Mode Switch Component`.

These two components are then placed in a `Macro Component` that expresses a complete control structure. However, as compared to semantics of mode automata having continuous transitions, this composite structure represents only a single transition. We thus utilize some MARTE concepts present in the RSM package, namely: `Tiler`, `defaultLink` and `Interrepetition` dependency, to resolve this issue. The details regarding the utilization of these concepts is out of the scope of this paper, however they help to connect the various repetitions of the `Macro Component` to the `Deployed Automata` component that represents a classic mode automata.

Once the control semantics were specified, an interesting challenge was to select an appropriate SoC design level for their integration. Several solutions were explored, such as integrating the control at application, architecture or allocation level. The advantages and disadvantages of control integration at these levels have been highlighted in [27]. However, integration of control for reconfiguration aspects at these levels was restricted by two main problems. Firstly, integration at the above three levels either influenced other design levels. For example, changing system architecture also caused changed in the allocation of the application onto the architecture. Secondly, strict conditions were required so that the influence range of control does not affects other design levels.

Hence, we proposed control integration at the IP deployment level in Gaspard2, where its influence range is local in nature. Another advantage being that the application, architecture and allocation models can be reused. Also *Quality-of-Service* (QoS) criteria can be applied to IPs such as consumed FPGA configurable resources etc.

C. Extending the MARTE metamodel and profile

Our second contribution relates to extending the MARTE profile and related metamodel for integrating our control

semantics at high abstraction levels. While modeling tools supporting MARTE profile permit to specify behavioral concepts such as state machines, the current underlying MARTE metamodel is not detailed enough for specifying system behavior. In turn, the model transformations required for eventual code generation will not be able to interpret the high level models expressing our mode automata control aspects. Similarly, the notion of deployment and IPs for elementary components does not exist in MARTE. Hence these concepts need to be integrated in the MARTE profile and metamodel. For this, we make use of the merge mechanism [28] as illustrated in Figure 4, an implementation of which has been developed internally in our research team. This concepts allows to extend the MARTE metamodel and profile with our proposed control semantics and IP deployment, which are themselves in the form of metamodels and associated profiles.

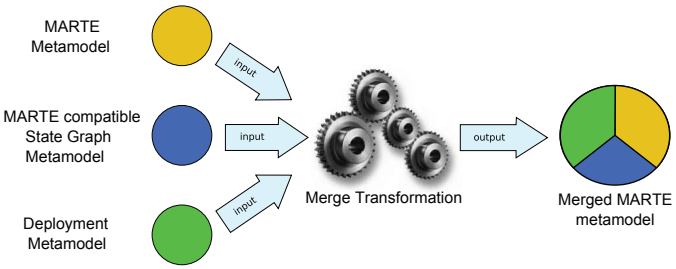

Figure.4: The merging mechanism in our design flow

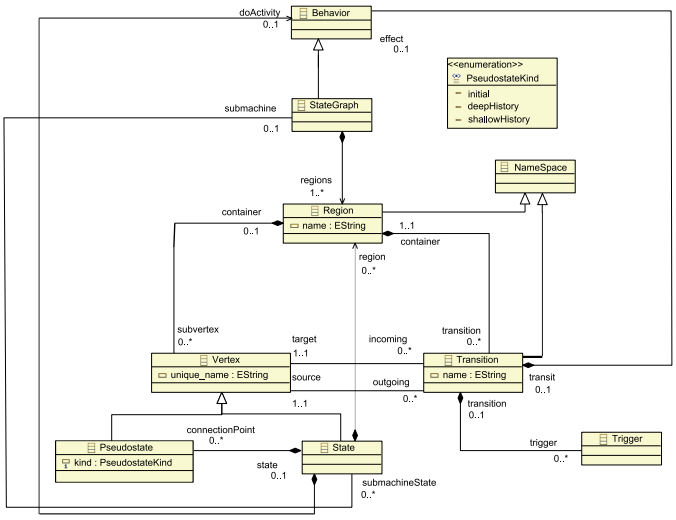

Figure.5: An extract of the Stategraph concept in MARTE

Figure 5 shows an extract of the State Graph metamodel related to our control semantics. A `StateGraph` is associated with MARTE `Behavior` concept and itself contains concepts such as `Region`, `State`, `Transition`, `Trigger` etc. This metamodel is itself inspired from the UML state machines concepts in UML specifications, and can be viewed as a specific subset.

D. Integrating configurations in Gaspard2: Extending the IP deployment modeling level

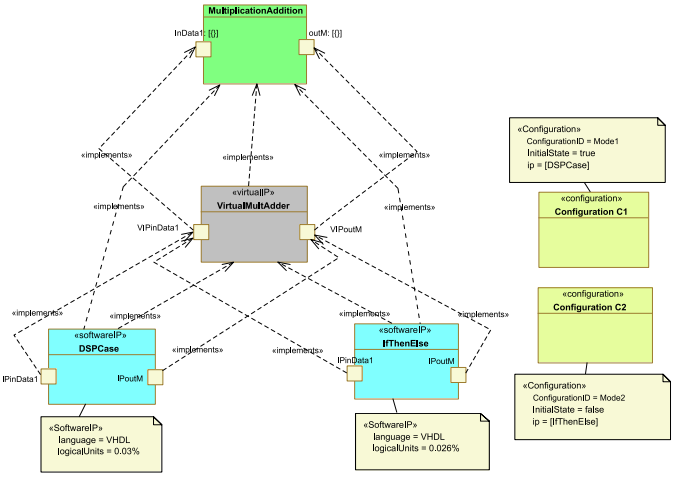

Figure.6: Deploying an elementary component of our case study application

Currently one of the main features of Gaspard2 is the ability to link the elementary components of the modeled application/ architecture, to the available user defined or third party intellectual properties. Along with the addition of control semantics, the current deployment level has also been extended to integrate the notion of *Configurations*, which are unique global implementations of a high level modeled application functionality, with each configuration comprised of different combinations of IPs related to the elementary components. These configurations usually depend upon designer requirements and environment specifications. Initially once an application is modeled, its elementary components can be deployed to respective IPs. In our extension, an elementary component can be linked to different IPs that represent different QoS characteristics: execution time, consumed FPGA resources etc; as illustrated in Figure 6. It is also possible that an elementary component can be associated with one unique IP in several user defined configurations.

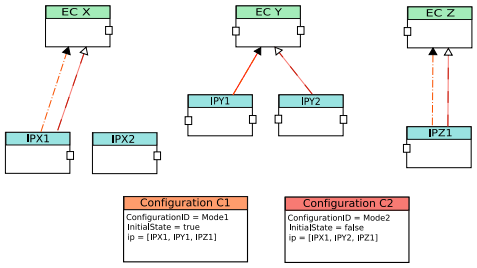

Figure.7: Abstract overview of configurations in deployment

Using a combination of the deployment level and the introduced control semantics, it is possible for a designer to specify several desired configurations as illustrated in Figure 7. Here a hypothetical application is illustrated having three elementary

components EC X, EC Y and EC Z; each with associated IP(s). Here two configurations C1 and C2 are created each having different combinations of the available IPs. Moreover, An IP can be partially or globally shared between configurations or not at all.

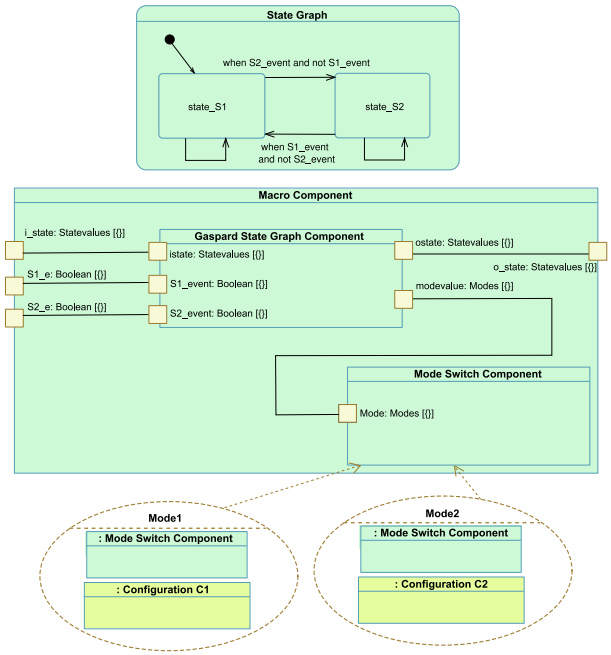

Figure.8: Modeling mode automata concepts at Gaspard2 IP deployment level

As illustrated in Figure 8, integrating control at deployment level permits a designer to change one global configuration by another with different results [18]. Here only a partial extract of the control modeling is illustrated, due to space limitations.

There are several advantages of using mode automata control semantics. A dynamically reconfigurable region in a reconfigurable SoC 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 a dynamically reconfigurable hardware accelerator equivalent to a modeled high level application specified using the MARTE profile. Using the control aspects in the Gaspard2 deployment level, it is possible to create different configurations of the modeled application, as illustrated earlier. 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.

E. Concepts for transforming high level models into Register Transfer Level equivalents:

This contribution introduces the *Register Transfer Level* (RTL) metamodel which comprises of two different aspects. Firstly, it incorporates the concepts for translating the application model into a hardware accelerator using a hardware execution model defined in [7]. This hardware accelerator is treated as a dynamically reconfigurable region in a targeted reconfigurable SoC, with the modeled application configurations serving as its different implementations. Among several available execution models, we have selected an execution model that corresponds to the requirements of dynamic reconfiguration. Secondly, the metamodel enriches the control semantics with details related to RTL. The two aspects although share certain common metaclasses, are quite different in nature. The RTL model, which is an instance of the RTL metamodel, is the end model from which eventual code can be generated. The part of the metamodel related to the hardware accelerators is independent from syntax related to any specific HDL, yet its low abstraction level enables code generation for a desired HDL. Similarly the enriched control concepts can either be interpreted for generation of HDL code in case of an HDL based hardware controller module; or a high level language such as C/C++ for implementation in a microprocessor based controller such as a PowerPC.

The RTL metamodel inspires from the MARTE metamodel itself. Concepts such as components, ports and connectors found in MARTE have also been translated into near equivalent metamodel elements such as illustrated in Figure 9. However the MARTE metamodel (or even our extended MARTE metamodel) does not provides detailed semantics for the generation of an integrated circuit at the RTL level. Description at RTL requires enriched details related to execution platforms which do not; and should not exist in the high level metamodels. Finally, we refer the reader to [7] for a detailed description of this metamodel.

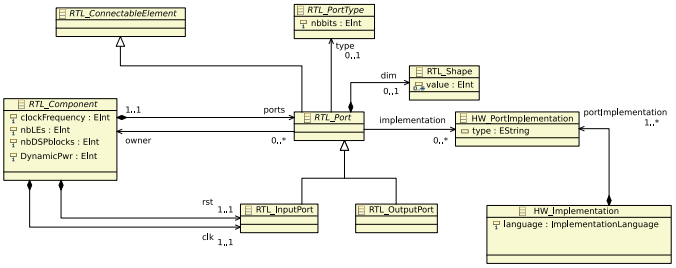

Figure.9: An extract of the RTL metamodel related to component ports and associated types

F. A complete transformation chain

Model transformations in our design flow enable creation of a complete model transformation chain for automatic code generation of high level MARTE compliant UML models. The *model-to-model* transformations in our flow permit to move from high level UML models to the RTL model, all the while enriching the intermediate models. Thereafter, a *model-to-text transformation* generates HDL code equivalent to the different implementations of the hardware accelerator. At the same time, C/C++ language code is generated for the switch mechanism related to the reconfiguration controller. Henceforth by using commercial synthesis tools, it is possible

to create a dynamically reconfigurable SoC. Finally, Figure 9.10 represents a screen shot of the Gaspard2 environment in the Eclipse environment[7] that illustrates the different models present in our design flow as well as the automatically generated RTL code.

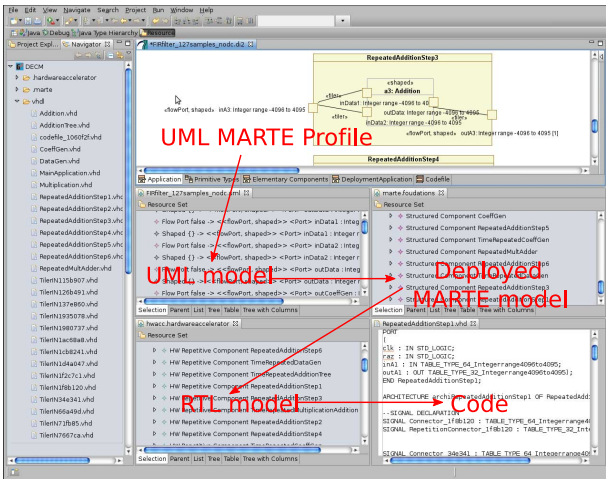

Figure.10: The transformation flow related to our design flow

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. Details related to our transformation rules can be found in [7].

V. Case Study

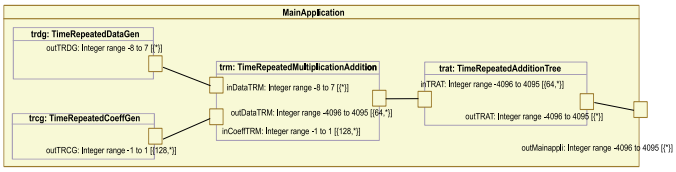

Figure.11: The top level view of the DECM functionality

We now present the case study which validates our design flow related to partial dynamic reconfiguration. The case study is related to a delay estimation correlation module (DECM) in a large anti-collision radar detection system [11], [29]. These systems when installed in vehicles permit collisions by detecting objects in the line of trajectory. The DECM module is modeled using the MARTE concepts. The radar system is responsible for emitting a wave with a code of reference, when this wave collides with an object such as an incoming car, it is reflected and received by the DECM. The DECM eliminates the noise presented in the received signal and executes a correlation algorithm, a peak observed in the correlation results indicates the presence of an object. For this, we have simulated a signal in MATLAB for some initial results.

[7]http://www.eclipse.org/

Initially, the DECM module is modeled as illustrated in Figure 11; and integrates aspects such as task and data parallelism. We specified two modeled configurations *DSP configuration* and *If-then-else* which correspond to two unique configurations of the modeled DECM, each having a unique combination of IPs and different QoS criteria. Afterwards, we developed the mode automata for the control aspects.

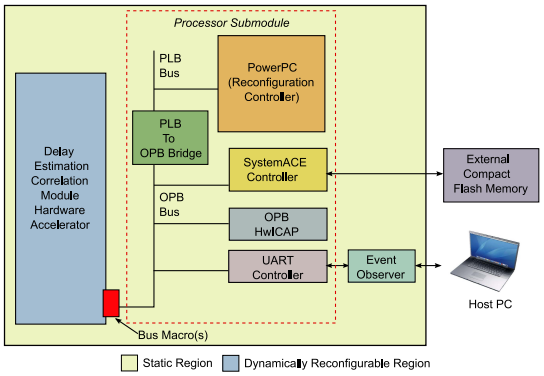

Figure.12: Block Diagram of the architecture of our reconfigurable system

Subsequently using the model transformation chain, we were able to generate the code related to the DECM, its related implementations as well as the source code for the reconfiguration controller. The generated code related to the different implementations (or configurations) of the DECM was verified by simulation and we obtained a near perfect match to the results obtained by the MATLAB simulation [11].

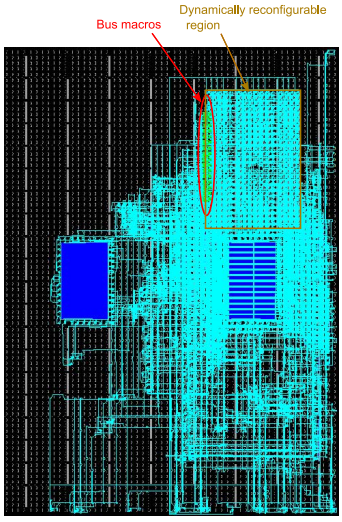

Figure.13: Full bitstream related to the reconfigurable SoC

Afterwards, we move onto implementing partial dynamic reconfiguration, using Xilinx *Early Access Partial Reconfiguration* flow [1]. In the initial design partition phase, we design the reconfigurable system using Xilinx ISE and EDK tools before passing onto the second phase using the Xilinx PlanAhead tool [1] for final bitstream generation. In the design partition

phase, we specify the global architecture for implementing dynamic reconfiguration as shown in Figure 12. The system consists of a dynamically reconfigurable region corresponding to the reconfigurable hardware accelerator and its associated implementations. This region is connected to the static region using bus macros that allow communication between the static/dynamic regions. The static region mainly consists of a processor submodule that contains the reconfiguration controller, a hardcore PowerPC running at 100MHz which is connected to several peripherals. The processor submodule is created in Xilinx EDK tool, while a hand tuned wrapper helps to integrated the dynamically reconfigurable accelerator in the overall architecture. These components are then instantiated in the top level of the reconfigurable system before making use of PlanAhead for the eventual bitstream generation as illustrated in Figure 13.

Using the mechanism of partial dynamic reconfiguration, we were able to switch between the two configurations at runtime, depending upon the user input. Table I shows the results related to the two configurations. 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, of about more than 400Kb. Using an external RAM memory can greatly decrease the reconfiguration times, similarly various other optimizations can be carried out, such as introducing a customized ICAP controller.

	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 I: Results related to the two configurations for the hardware accelerator

VI. Conclusion

This paper presents a novel high abstraction level design methodology based on the UML MARTE profile for targeting dynamically reconfigurable FPGA based SoCs. Our approach incorporates aspects of Model-Driven Engineering and allows a designer to specify key components of a dynamically reconfigurable SoC. Afterwards, using model transformations, automatic code generation is possible. Finally the produced code is taken as input in commercial tools for final implementation of partial dynamic reconfiguration.

References

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

[2] OMG, "Portal of the Model Driven Engineering Community," 2007, http://www.planetmde.org.

[3] S. Sendall and W. Kozaczynski, "Model Transformation: The Heart and Soul of Model-Driven Software Development," *IEEE Software*, vol. 20, no. 5, pp. 42–45, 2003.

[4] OMG, "Modeling and Analysis of Real-time and Embedded systems (MARTE), Beta 3," http://www.omgwiki.org/marte-ftf2/doku.php, 2009.

[5] DaRT team, "GASPARD SoC Framework," 2009, http://www.gaspard2.org/.

[6] A. Gamatié and S. L. et al, "A model driven design framework for massively parallel embedded systems," *ACM Transactions on Embedded Computing Systems (TECS)*, 2010, to appear.

[7] I. R. Quadri, "MARTE based model driven design methodology for targeting dynamically reconfigurable FPGA based SoCs," Ph.D. dissertation, USTL - Lille, France, 2010. [Online]. Available: http://tel.archives-ouvertes.fr/tel-00486483/en/

[8] A. Koudri et al, "Using MARTE in the MOPCOM SoC/SoPC Co-Methodology," in *MARTE Workshop at DATE'08*, 2008.

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

[10] I.-R. Quadri, S. Meftali, and J.-L. Dekeyser, "A model driven design flow for fpgas supporting partial reconfiguration," *International Journal of Reconfigurable Computing*, 2009, Hindawi Publishing Corporation.

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

[12] S. Pillement and D. Chillet, "High-level model of dynamically reconfigurable architectures," in *Conference on Design and Architectures for Signal and Image Processing, (DASIP'09)*, 2009, pp. 1–7.

[13] A. Brito, M. Kuhnle, M. Hubner, J. Becker, and E. Melcher, "Modelling and Simulation of Dynamic and Partially Reconfigurable Systems using SystemC," in *IEEE ISVLSI'07*. IEEE Computer Society, 2007, pp. 35–40.

[14] K. G. Nezami, P. W. Stephens, and S. D. Walker, "Handel-C Implementation of Early-Access Partial-Reconfiguration for Software Defined Radio," in *IEEE WCNC'08*, 2008, pp. 1103–1108.

[15] F. Berthelot and F. Nouvel and D. Houzet, "A Flexible system level design methodology targeting run-time reconfigurable FPGAs," *EURASIP Journal of Embedded Systems*, vol. 8, no. 3, pp. 1–18, 2008.

[16] S. Gupta, N. Dutt, R. Gupta, and A. Nicolau, "Spark : A high-lev l synthesis framework for applying parallelizing compiler transformations," *VLSI Design, International Conference on*, vol. 0, p. 461, 2003.

[17] P. Coussy, G. Corre, P. Bomel, E. Senn, and E. Martin, "High-level synthesis under i/o timing and memory constraints," in *IEEE ISCAS 2005*, 2005, pp. 680–683 Vol. 1.

[18] I. R. Quadri, S. Meftali, and J.-L. Dekeyser, "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)*, 2009.

[19] A. Gamatié, E. Rutten, H. Yu, P. Boulet, and J.-L. Dekeyser, "Synchronous modeling and analysis of data intensive applications," *EURASIP Journal on Embedded Systems*, 2008.

[20] H. Yu, "A MARTE based reactive model for data-parallel intensive processing: Transformation toward the synchronous model," Ph.D. dissertation, USTL, 2008.

[21] OMG, "M2M/Operational QVT Language," 2007, http://tiny.cc/tFGGx.

[22] ——, "MOF Query /Views/Transformations," 2005, http://www.omg.org/cgi-bin/doc?ptc/2005-11-01.

[23] Xilinx, "ISE Foundation Software," 2009, http://www.xilinx.com/ise/logic_design_prod/foundation.htm.

[24] ——, "Embedded Development Kit (EDK) Software," 2009, http://www.xilinx.com/ise/embedded_design_prod/platform_studio.htm.

[25] F. Maraninchi and Y. Rémond, "Mode-automata: a new domain-specific construct for the development of safe critical systems," *Sci. Comput. Program.*, vol. 46, no. 3, pp. 219–254, 2003.

[26] B. Blodget and S. McMillan and P. Lysaght, "A lightweight approach for embedded reconfiguration of FPGAs," in *Design, Automation & Test in Europe, DATE'03*, 2003.

[27] J.-L. Dekeyser, A. Gamatié, S. Meftali, and I. R. Quadri, *Heterogeneous Embedded Systems - Design Theory and Practice*. Springer, 2010, ch. High-level modeling of dynamically reconfigurable heterogeneous systems.

[28] Object Management Group Inc., "Uml 2 infrastructure (final adopted specifcation)." http://www.omg.org/spec/UML/2.1.2/Infrastructure/PDF/, Nov. 2007.

[29] I. R. Quadri, Y. Elhillali, S. Meftali, and J.-L. Dekeyser, "Model based design flow for implementing an Anti-Collision Radar system," in *9th International IEEE Conference on ITS Telecommunications (ITS-T 2009)*, 2009.