Model-Driven design flow for distributed control in reconfigurable FPGA systems

Chiraz Trabelsi*, Samy Meftali†, Rabie ben Atitallah * and Jean-Luc Dekeyser†
*LAMIH, University of Valenciennes, Valenciennes, France
Email:{chiraz.trabelsi, rabie.benatitallah}@univ-valenciennes.fr
†INRIA Lille Nord Europe - LIFL - Universite Lille1, Lille, France
Email:{Samy.meftali,jean-luc.dekeyser}@inria.fr

***Abstract*—One of the most challenging and time-consuming design tasks for dynamically reconfigurable FPGA (Field Programmable Gate Array) systems is the design of runtime-adaptation control. This aspect covers various points such as runtime-monitoring, reconfiguration decision-making and reconfiguration realization. In this paper, we propose a control design flow aiming at facilitating the designers work and enhancing their productivity through high design reuse and automation. The proposed flow combines control distribution and Model-Driven-Engineering (MDE). The distributed control structure proposed in this flow aims at facilitating the reuse of the control design by using separate controllers for local and global control problems. The flow enables designers to move from high-level control models, using an extended version of the MARTE (Modeling and Analysis of Real-Time and Embedded Systems) UML standard profile, to an automatic generation of the corresponding VHDL code. The flow was validated through a video processing case study from modeling to implementation in FPGA.**

***Index Terms*—UML MARTE, Model-Driven-Engineering, distributed control, Partial Dynamic Reconfiguration, FPGA**

I. INTRODUCTION

The progress in FPGA technologies has enabled them to embed a growing number of computing resources and to target increasingly sophisticated applications. This has led to a growing design complexity of such systems. This complexity increases even more with the dynamic reconfiguration feature of some systems, especially in runtime-adaptation control design. This aspect covers various points such as runtime monitoring, reconfiguration decisions and reconfiguration execution. In this context, a design methodology that facilitates design reuse and automation is required in order to reduce design time and to enhance designers productivity. This design methodology has also to consider an efficient implementation of the control in terms of performance in order to guarantee its fast reaction to runtime changes. In order to reach these objectives, this paper proposes a control design flow that combines a distributed control approach with Model-Driven Engineering (MDE) for an automatic hardware code generation.

The distributed control approach used in this paper targets systems supporting Partial Dynamic Reconfiguration (PDR) [1]. This mechanism offers the possibility of reconfiguring only some regions of the FPGA without disturbing the operation of the rest of the system. The content of such regions is modified at runtime through reconfiguration, by loading different configuration data (bitstreams). The proposed control structure is based on allocating a controller to each reconfigurable component of the system. Each controller contains three modules that handle three tasks: monitoring, reconfiguration decision-making and reconfiguration realization. In order to guarantee that the local reconfiguration decisions made by the distributed controllers respect the global system constraints/objectives, a coordinator is used. Compared to centralized control, the proposed control structure decreases design complexity by separating local and global control problems, which facilitates also the reuse of the controllers and the coordinator. Distributing the control allows to parallelize control tasks (monitoring, decision-making and reconfiguration), increasing thus control performance. The proposed control structure was presented in details in a previous work [2].

MDE is a very interesting approach for design automation. It uses high-level modeling languages such as UML (Unified Modeling Language) [3] for system description. Using MDE tools, high-level models can be automatically transformed into code (executable code, simulation code, etc). In order to use the MDE for a high level description of a system in a specific domain such as embedded systems, UML profiles are used. A UML profile is a set of stereotypes that add specific information to a UML model in order to describe a system related to a specific domain. Several UML profiles target embedded systems design such as the UML profile for SoC [4], the System Modeling Language (SysML) [5], and the Modeling and Analysis of Real-Time and Embedded systems (MARTE) [6] profile.

The contribution of this paper is the MDE-based design flow and tool applied to the previously proposed distributed control structure. The aim of this flow is to simplify the designers work through high-level modeling and to enhance their productivity through high design reuse and automation. High-level modeling in this flow uses an extended version of the MARTE profile. After the modeling step, the control models are transformed using a model transformation chain in order to get closer to the targeted RTL implementation. The output model of this chain is then translated into VHDL code.

The rest of this paper is organized as follows. Section 2 gives a summary of the related works. Section 3 gives a brief description of the distributed control mechanism. Section 4 gives an overview MDE-based control design flow. Section 5 illustrates the high-level modeling of the control through a

video processing application case study. Section 6 describes the model transformation and code generation steps of the design flow. In Section 7, the generated control code is validated from simulation to physical implementation. This section evaluates also the flow in terms of design reuse and automation.

II. Related works

High-level modeling and automatic code generation for FPGA systems were studied in several works. In this context, many High-Level Synthesis (HLS) tools were proposed [7] [8]. These tools enables designers to move from a high-level language, most commonly C, to RTL description. These approaches offer a high design automation. However, writing the design in the C language requires to specify a certain amount of implementation details in order to be translated into RTL description, which is a time-consuming task. In order to ensure a more abstract system specification and thus less design efforts, using the UML language for system modeling was proposed in many works. However, few works such as [9] [10] [11] [12] studied the UML design of the runtime-adaptation control (monitoring, decision-making, reconfiguration control) for dynamically reconfigurable FPGA systems. In the scheduling-based reconfiguration category, we cite the MOPCOM [9] and OverSoC [10] design frameworks, which propose a high-level modeling approach for the OS-based runtime-adaptation control. In the monitoring-based reconfiguration category, we cite the the GASPARD2 [11] and FAMOUS [12] design frameworks. In the GASPARD2 framework, the control modeling for FPGA reconfigurable systems was limited to a single UML state-machine inspired from the mode-automata formalism [13]. This state-machine (mode-automaton) describes the different modes of the systems and the transitions between them. For each mode, a different configuration of the system is modeled. The main limit of this approach is its scalability for large systems resulting in a very complex state-machine and data redundancy between the system configurations. In the FAMOUS framework, the control modeling is based on a set of mode-automata using the mode-automata formalism. These automata are coordinated through of set of contracts. The used MDE-approach allows a formal verification of the control system through a Discrete Control Synthesis, which generates a global controller in C. The main limit of this approach is that obtaining a synthesizable controller is not guaranteed, especially when a great number of automata and controllable variables are used. Another limit of the control design approaches of the GASPARD2 and FAMOUS frameworks is that their control modeling is limited to the decision-making aspects. The monitoring and reconfiguration aspects are handled manually outside the MDE design flow, which limits the automation process. Moreover, these approaches are limited to a centralized software control implementation, which may have non negligible impact on the control performance due to their low-level parallelism degree.

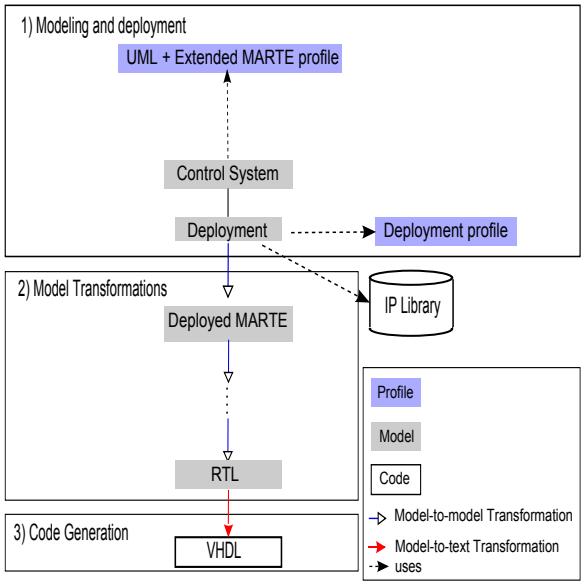

Fig. 1: An overview of the proposed control design flow

III. Distributed control for FPGA reconfigurable systems

In our control design approach, each distributed controller is composed of three modules handling monitoring, reconfiguration decision-making and reconfiguration realization for the controlled reconfigurable component. The monitoring module can collect information from the behavior of the controlled component and other external events such as those sent by sensors, for example. The monitoring module processes the collected data and transforms them if necessary into aggregates. Monitoring data are then sent to the decision module, which makes local decisions about whether or not a reconfiguration of the controlled component is required.

Due to the local vision of each controller, it cannot launch a reconfiguration without checking whether or not it can coexist with the current configurations of the other components. Therefore, before launching a reconfiguration, the decision module has to send a reconfiguration request to the coordinator. If the coordinator estimates that the requested reconfiguration implies also the reconfiguration of other components in order to respect the global system constraints, it sends reconfiguration suggestions to the concerned controllers. These controllers can accept or refuse those suggestions. After treating the controller responses, the coordinator gives its decision, which can be either the authorization or the refusal of the requested reconfiguration. If the coordinator authorizes the requested reconfiguration, each concerned decision module notifies the related reconfiguration module in order to launch the required reconfiguration of the controlled component.

IV. MDE-based control design flow

The MDE-based design methodology applied to this control structure aims at automating code generation from high-level models and thus reducing design time. The proposed flow is

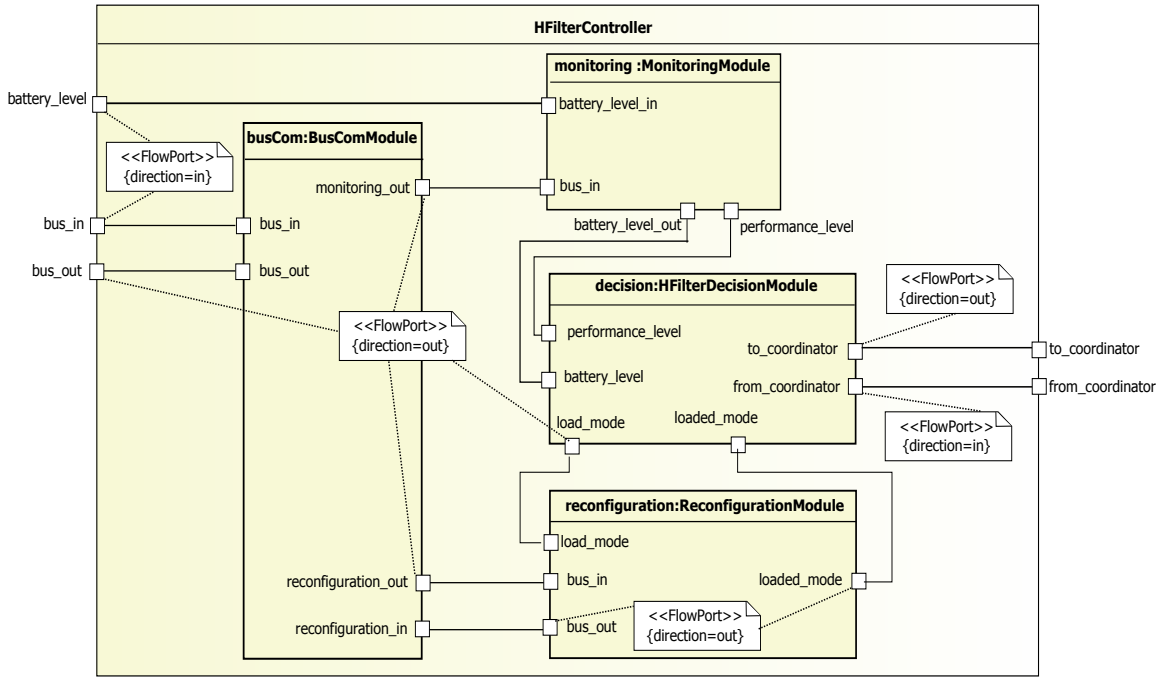

Fig. 2: The horizontal filter controller

implemented using the GASPARD2 [14] framework, which is an MDE-based SoC co-design framework. The high-level modeling in GASPARD2 is based on the MARTE UML standard profile, which is a very rich profile covering different views of an embedded system, such as application and architecture. This framework uses model transformations to target different languages such as Fortran, SystemC, OpenCL, C, VHDL, etc.

The design flow in GASPARD2 follows three steps: 1) system modeling (the application, architecture and association between them) and deployment, 2) model transformations, and 3) code generation. These steps concern the design of a whole embedded system including the application and the architecture parts. Our control design flow follows the same steps to generate control systems only as shown in Fig. 1.

V. High-level control modeling: a video downscaling case study

A. Video downscaling case study

The application of the case study is a video down-scaling application, which is very used for streaming in small form factor devices, such as mobile phones. It consists of a classical downscaler, which transforms a video signal, expressed in Common Intermediate Format (CIF:352x288 pixels), into a smaller size video (132x128 pixels). This application is composed of two main tasks: a horizontal filter that transforms the input video frame into a 132x288 format, followed by a vertical filter that generates a video frame in the final format. Each filter is composed of a repetition of an elementary filter task. In order to guarantee a high performance of the application, both filters are implemented by hardware accelerators. In our case study, the objective of the control is to adapt the downscaler application to changes in performance and power requirements.

Each accelerator related to one of the filters is implemented in a reconfigurable region using three different configurations (modes), varying the number of elementary tasks performed simultaneously. These modes are named $HFilter\_mode_1$, $HFilter\_mode_2$ and $HFilter\_mode_3$ for a reconfigurable region implementing the horizontal filter, and $VFilter\_mode_1$, $VFilter\_mode_2$ and $VFilter\_mode_3$ for a reconfigurable region implementing the vertical filter. The first mode for both filters ($HFilter\_mode_1$/$VFilter\_mode_1$) corresponds to the highest performance but at the same time the highest power consumption. Each reconfigurable region has a controller that controls its behavior by switching different modes depending on requirements in terms of performance and power consumption. The performance requirements are given by the user by selecting at runtime one of the three available performance levels: levels 1, 2 and 3, where level 1 is the highest performance level. The inputs of the controllers monitoring modules are thus the battery and performance levels.

B. Controller structure

There are two types of controllers in this case study: the *HFilterController* and the *VFilterController*, which are the controllers related to reconfigurable regions implementing the horizontal and vertical filters respectively. Fig. 2 represents the model describing the structure the *HFilterController*. This controller is modeled by a UML component having ports in order to communicate with the rest of the system. It is composed of 4 components representing monitoring, decision,

reconfiguration and bus communication modules. This last module is only used because the controller communicates with the processor in this case study (the performance level chosen by the user is sent to the controllers through the processor, and the processor is also used to communicate with the ICAP (Internal Configuration Access Port) in order to download bitstreams after retrieving their identifiers from the reconfiguration modules). The UML ports are stereotyped <<flowPort>> using the MARTE profile. This stereotype indicates the port direction (in, out or inout) as shown in Fig. 2. The controller inputs are coming from the battery (battery level), the bus (allowing to communicate with the processor), and the coordinator. The controller output ports allow it to communicate with the bus and the coordinator.

The controller related to the vertical filter (*VFilterController*) has the same structure. Note that the decision module of the *HFilterController* in Fig. 2 is named *HFilterDecisionModule*, whereas the other modules do not have the *HFilterController* in their names. This is because the monitoring, reconfiguration and bus communication modules are the same for both controllers. The difference between them is the decision module. As will be explained in the next sub-section, these decision modules do not have the same decision rules.

C. Decision module behavior

The decision-making system is composed of the decision modules of the controllers and the coordinator. In order to abstract and formalize the decision-making process and to facilitate its translation to different implementations and platforms, we use a modeling approach inspired from the mode-automata formalism [13]. Each decision module can be implemented by a mode-automaton, where each mode represents a given configuration of the controlled component. This automaton, has as inputs, monitoring data sent by the monitoring module and coordination data sent by the coordinator.

In order to enable the modeling of these aspects, we propose extensions to the MARTE profile. One of these extensions is the <<coordinatedTransition>> stereotype, which inherits from the <<modeTransition>> MARTE stereotype. To this new stereotype, we associate the <<coordinationConstraint>> stereotype in order to specify the reconfiguration request, acceptance, refusal conditions through the *actionKind* property as shown in Fig. 3.

In this case study the request, acceptance and refusal conditions depend on the required performance level and the battery level. For instance, as shown in Fig. 3, the controller decides that a reconfiguration to a less consuming mode is required only if the user requires a lower performance level, or the battery level is under a given threshold. Note that the considered battery thresholds are not the same for all controllers. Each controller determines its thresholds using a set of formulas that depends on the consumption of its controlled component modes.

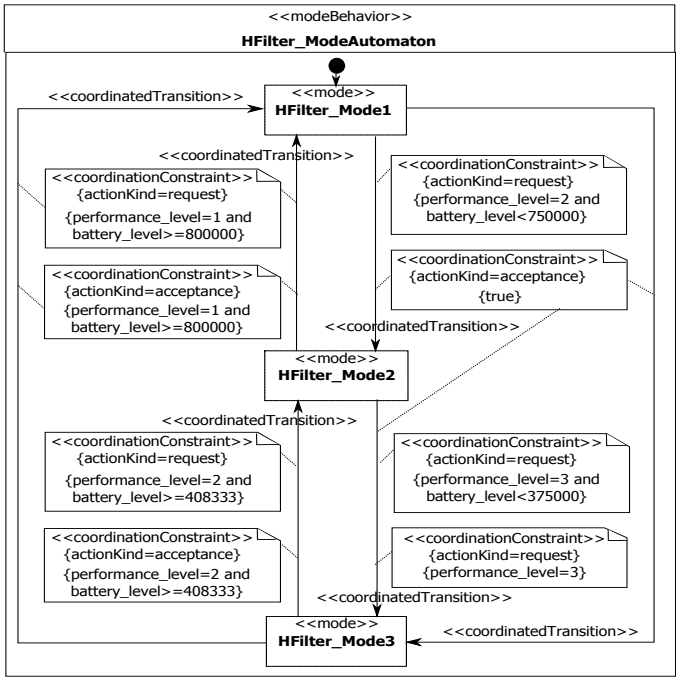

Fig. 3: The decision module mode-automaton model

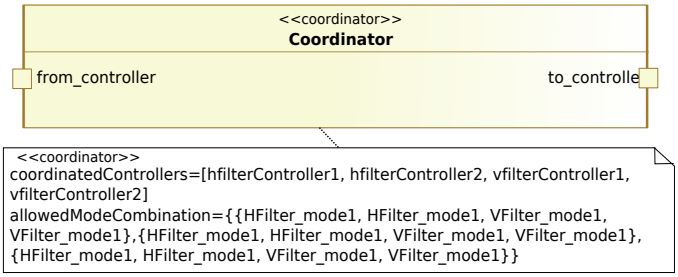

Fig. 4: The coordinator model

D. Coordinator

The embedded system considered in this case study contains 2 regions implementing the horizontal filter and 2 other regions implementing the vertical filter. Therefore, the related control system will be composed of 2 instances of the *HFilterController* and 2 instances of the *VFilterController*.

In order to enable the high-level modeling of the coordinator, we propose the <<coordinator>> stereotype, which inherits from the <<rtUnit>> MARTE stereotype. As specified in MARTE [6], an <<rtUnit>> represents a real-time unit, which is an execution resource that owns one or several schedulable resources and that is able to handle different messages at the same time. In this context, the coordinator can be seen as an execution resource handling a set of schedulable resources (the partial configurations) and different messages (requests and responses) coming from controllers. Fig. 4 illustrates the coordinator modeled for this case study. The *coordinatedControllers* property of the <<coordinator>> stereotype specifies the coordinated instances. The coordinated controllers are the 4 controller in-

stances of the system. The coordination table is represented by the *allowedModeCombination* property, which is a mode matrix having the number of allowed combinations as number of rows. Each row models a mode combination for the controlled regions. The coordinator table indicates the allowed mode combinations. In this case study, we suppose that the system constraint is that all the regions have to implement the same mode number as shown in Fig. 4.

E. Control deployment

After modeling the entire control system, we can deploy it by linking its elementary components to existing IPs in order to add details to get closer to the targeted platform. At this level, we used the GASPARD2 deployment profile. We associate VHDL code files (UML artifacts attached to the deployed UML components) to the monitoring, reconfiguration and bus communication modules since we use existing IPs for them. This is because their implementations depend on the targeted system (how the monitoring data are extracted, how they communicate with the processor if necessary, etc.). The controllers mode-automata and the coordinator are not concerned by the deployment phase since they will be generated automatically.

VI. Model transformations and code generation

The control transformation chain shown in Fig. 1 starts by an extended version of the UML2MARTE transformation in order to take the MARTE extensions that we propose into account. A set of transformations is then carried out in order to get closer to the targeted platform. The last transformation of this chain is the RTL transformation, which results in a model that translates the control system according the RTL concepts (RTL components, entities, architectures, etc).

The code generation tool takes as input the output model of the *RTL* transformation. It generates the decision modules of the controllers, the coordinator, and the communication between the modules of a controller and between the controllers and the coordinator. Mode-automata are implemented using VHDL case statements for both the decision modules and the coordinator. For the implementation of communication between the controllers and the coordinator, the generation tool uses the OCP (Open Core Protocol) standard protocol [15]. This protocol can be used to implement several communications types (peer-to-peer, bus, NoC, etc.) [16]. Furthermore, it offers a high design flexibility with a wide range of signals that can be activated if needed and whose sizes are configurable. In order to guarantee the control performance, the generation tool implements a peer-to-peer communication between the coordinator and the controllers.

The next section details the code generation results. The proposed flow is evaluated in terms of design automation and reuse. The generated code is validated from simulation to physical implementation on FPGA. The proposed control implementation is evaluated in terms of performance and resource overhead.

VII. Experimental results

A. Design automation and reuse

The modeling phase of the case-study control system took approximately 20 min. The model transformation and code generation phases took approximately 10 s. More than 1500 lines of code were generated automatically. The used host machine is equipped with Intel(R) Core(TM)2 Duo processor and a 4 GB RAM memory. This is a significant result compared to a manual writing of the same control system, which can take hours and requires from the designer to master low-level details such as the communication protocol between the controllers and the coordinator.

There are many design reuse possibilities offered by the proposed flow. As we explained previously the UML components representing the monitoring, reconfiguration, and bus communication modules are reused for both controllers. The whole control model of the case study can also be reused to target systems implementing the same downscaling application with more hardware accelerators by increasing the data-parallelism degree of the application. In this case, the designer only needs to increase the number of the used controllers and modify the characteristics of the coordinator, which are the coordinated controller instances and the coordination table. The new control system can then be generated automatically, using the same model transformation chain and the same code generation tool. This approach was validated by reusing the control model of the case study to target systems containing 2, 6, 8 and 10 controllers. This process requires approximately 10 s to add a controller instance and approximately 10 s to add/modify a line in the coordinator table.

The generated code by the proposed flow can also be reused at the deployment phase. This approach was also validated by adding the generated controllers from the case study control model to the IP library of the GASPARD2 framework. The control systems were then modeled by using an elementary UML component for each controller. The design reuse was validated by starting from a model containing only two controllers and a coordinator. The modeling of a controller and its linked code takes approximately 2 min. The control model was then reused by adding more controller instances (to obtain 4, 6, 8 and 10 instances) and by modifying the characteristics of the coordinator. The generation process was validated for all these control systems.

B. Hardware implementation

The generated control systems (with 2, 4, 6, 8 and 10 controllers) were validated during simulation, by testing all the controllers reconfiguration requests, and reconfiguration suggestion acceptances and refusals, using different simulation scenarios. Detailing the simulation scenarios used for this case study is out of the scope of this paper. Simulations showed that the coordination process duration in clock cycles follows this equation:

$$coordination\_duration = 4 + nb\_coordination\_rounds * 4 \quad (1)$$

Resources	Number of regions				
	2	4	6	8	10
Slice registers	375	744	1112	1479	1847
Slice LUTs	474	907	1388	2010	2499

TABLE I: Resource overheads of the generated control systems

where *nb_coordination_rounds* is the number of coordination rounds before the coordinator decides to accept or refuse reconfigurations. Here, if the coordinator authorizes reconfigurations after the first round, the coordination process lasts only 8 clock cycles, which is a small duration for a system containing a big number of controllers. This small duration is obtained thanks to the hardware implementation of the control system. However, for systems where decision-making is under hard real-time constraints, this solution may not be acceptable if the number of rounds is big. This can be solved by changing the communication mechanism between the controllers and the coordinator. This can be done, for example, by making all the controllers concerned in a coordination process send to the coordinator the reconfigurations that they might accept in a single data transfer. This reduces significantly the coordination process time at the cost of an increased hardware resources overhead of the controllers and the coordinator.

Table I describes the resource overhead of the generated control systems in terms of slice registers and LUTs for different numbers of controlled regions. This overhead is up to 1847 slice registers and 2499 slice LUTs, which corresponds to 0.61% of slice registers and 1.66% of slice LUTs of the Virtex6-xc6vlx240t FPGA. This overhead can be considered as acceptable for such a large FPGA (301440 slice registers and 150720 slice LUTs). It is even more acceptable for larger FPGAs such as Virtex-7 2000T [17], which is a 2 million logic cell device. As shows Table I, the resource overhead is linear with the number of controlled regions. This is due to the high parallelism degree offered by the proposed implementation for the coordinator (parallel suggestion sending, parallel response handling, etc.). This means that for very large control systems (100 controlled regions for example), the resource overhead of the proposed implementation may be not negligible. In this case, this implementation has to be adapted in order to reduce resource overheads (by decreasing the parallelism degree of the coordinator, using a heterogeneous HW/SW implementation of the control system, etc.) at the cost of performance degradation.

The proposed control flow was also validated by physical implementation for the 4-controller system of the case study. Three partial bitstreams were generated for each reconfigurable region, as well as a full bitstream for the initial system configuration. The control system was validated at runtime using the same scenarios as in the simulation.

VIII. Conclusion

This paper presents a Model-Driven Engineering (MDE)-based design flow for the runtime-adaptation control of dynamically reconfigurable FPGA systems. This flow aims at reducing design time and enhancing designers productivity through design reuse and automation. It is based on a distributed control approach, where a control system is composed of a set of coordinated modular controllers, which promotes design reuse. The MDE-based design of the proposed flow allows to move from high-level modeling of the control systems, using the MARTE UML standard profile to an automatic generation of their VHDL code reducing thus design time. This paper details the different steps of the proposed flow through a video processing case study. The generated control code was validated from simulation to physical implementation. As a future work, we plan to extend the proposed flow to cover different implementations of the control (HW, SW and both) in order to adapt to different performance and resource constraints. Our work could be also extended in order to integrate other modeling possibilities for the constraints and objectives handled by the coordinator instead of listing all the acceptable mode combinations.

References

[1] P. Lysaght, B. Blodget, J. Mason, J. Young, and B. Bridgford, "Invited paper: Enhanced architectures, design methodologies and cad tools for dynamic reconfiguration of xilinx fpgas," in *FPL*, 2006, pp. 1–6.

[2] hidden reference for blind review.

[3] OMG, *UML Superstructure*, 2009.

[4] ——, "Uml profile for system on a chip (soc) available specification, version 1.0.1," 2006, http://www.omg.org/spec/SoCP/1.0.1/.

[5] ——, "Omg systems modeling language (omg sysml), version 1.3," 2012, http://www.omg.org/spec/SysML/1.3.

[6] ——, *UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems*, 2011.

[7] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-level synthesis for fpgas: From prototyping to deployment," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 30, no. 4, pp. 473–491, 2011.

[8] D. Navarro, O. Lucia Gil, L. Barragan, I. Urriza, and O. Jimenez, "High-level synthesis for accelerating the fpga implementation of computationally-demanding control algorithms for power converters," 2013.

[9] J. Vidal, F. de Lamotte, G. Gogniat, J.-P. Diguet, and P. Soulard, "Uml design for dynamically reconfigurable multiprocessor embedded systems," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '10, 2010.

[10] S. Pillement and D. Chillet, "High-level model of dynamically reconfigurable architectures," in *Conference on Design and Architectures for Signal and Image Processing*, ser. DASIP '09, 2009.

[11] 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*, ser. DASIP '09, 2009.

[12] S. Guillet, F. de Lamotte, E. Rutten, G. Gogniat, and J.-P. Diguet, "Modeling and formal control of partial dynamic reconfiguration," *Reconfigurable Computing and FPGAs, International Conference on*, vol. 0, pp. 31–36, 2010.

[13] F. Maraninchi and Y. Remond, "Mode-automata: a new domain-specific construct for the development of safe critical systems," *Science of Computer Programming*, vol. 46, 2003.

[14] A. Gamatie, S. L. Beux, E. Piel, R. B. Atitallah, A. Etien, P. Marquet, and J.-L. Dekeyser, "A model driven design framework for massively parallel embedded systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 10, 2011.

[15] O. I. Partnership, *OCP-IP*, http://www.ocpip.org.

[16] ——, "Open core protocol specification," Tech. Rep. Release 2.0, 2008.

[17] Xilinx, "7 series fpgas overview, advance product specification," Tech. Rep. DS180 (v1.8), 2011.