Dynamic applications on reconfigurable systems: from UML model design to FPGAs implementation

Jorgiano Vidal*, Florent de Lamotte*, Guy Gogniat*, Jean-Philippe Diguet*, Sebastien Guillet*
* Lab-STICC – European University of Brittany - UBS – CNRS, UMR 3192, Lorient, France

***Abstract*—In this paper we propose a design methodology to explore dynamic and partial reconfiguration (DPR) of modern FPGAs. We define a set of rules in order to model DPR by means of UML and design patterns. Our approach targets MPSoPC (Multiprocessor System on Programmable Chip) which allows: a) area optimization through partial reconfiguration without performance penalty and b) increased system flexibility through dynamic behavior modeling and implementation. In our case, area reduction is achieved by reconfiguring co-processors connected to embedded processors, and flexibility is achieved by permitting new behavior to be easily added to the system. Most of the system is automatically generated by means of MDE techniques. Our modeling approach allows designers to target dynamic reconfiguration without being experts of modern FPGAs. Such a methodology allows design time speed-up and a significant reduction of the gap between hardware and software modeling.**

I. Introduction

Modern embedded systems require dynamic properties, i.e. the capability to change at run-time the execution of the application to fit the environment evolution. Such needs depend on the system being developed. As an example when dealing with multimedia systems where different CODECs are required, dynamic properties can lead to reduce power consumption by specializing some parts of the system or to reduce system chip area by sharing the same resources for different tasks that run at different times. Capturing such execution scenarios during the design phases becomes mandatory to guarantee the definition of an efficient system. During the development process flexibility, performance and chip area are the key concerns for the designer.

Existing FPGA technology (e.g. Xilinx Virtex devices) offers dynamic and partial reconfiguration (DPR) features that can be advantageously considered to address these points. DPR allows HW components to share the same resource if their execution is exclusive. This solution is very interesting as it reduces the total system area while still meeting performance constraints. For instance, supporting DPR allows multiprocessor systems to replace co-processors or accelerators at run-time which significantly increases platform flexibility. Unfortunately there is a lack of tools addressing the design of reconfigurable MPSoPCs (Multiprocessor System on Programmable Chip) at the higher abstraction levels.

In this paper, we propose to address this issue and to reduce the gap between the design of dynamically reconfigurable MPSoPCs and the targeted technology (FPGA). To achieve this goal, we propose to model a DPR system at a high level which allows a technology-independent modeling. Moreover we use the Unified Modeling Language (UML) [1] in order to build an interchangeable and standardized model format. We add dynamic system modeling on top of an existing UML-based embedded system design methodology [2].

978-3-9810801-7-9/DATE11/©2011 EDAA

The paper is organized as follows: in Section II, we discuss existing efforts related to our work. In Section III, we present a dynamic and partial reconfigurable platform. In Section IV, we show our modeling methodology for dynamic behavior. In Section V, we show some results. Finally, in Section VI, we conclude and present some future work.

II. Related work

The use of model based approaches for co-design has been discussed in [3], which pointed out some advantages: cost decrease, silicon complexity handling, productivity increase, etc. Several works have also shown the benefit of using UML for embedded system modeling. Dynamic and partial reconfiguration modeling using UML allows existing methodologies to take advantage of dynamic reconfiguration capabilities of modern FPGAs. Although there is a lot of work on embedded system modeling using UML, only few explore dynamic and partial reconfiguration capabilities [4], [5].

In [4], authors use UML sequence diagram with specific stereotypes to model dynamic reconfiguration. Their approach is very simple and efficient, but it lacks platform modeling. In their work the system platform is fixed: a processor with a reconfigurable device as an auxiliary computing unit. Also, it does not support dynamic and partial reconfiguration.

In [5], authors detail a dynamic reconfigurable system by extending UML/MARTE with specific stereotypes. Their approach is developed in a design environment called GASPARD [6], where VHDL code is generated. This approach is very target-dependent and requires a strong level of expertise as all elements of the Xilinx partial reconfiguration design process need to be modeled.

Compared to previous efforts, our approach only uses standard UML/MARTE elements and well known modeling solutions (for instance design patterns). Specific properties are required by the code generation tool that is target dependent. In order to allow a large adoption of dynamic and partial reconfiguration we hide from system designers many technology details. Reconfiguration services and resources are automatically added to the system during the code generation step. Furthermore we are able to capture both application and architecture dynamic properties within a single design methodology.

III. Dynamic and partial reconfiguration

To define a dynamically reconfigurable MPSoPC, we consider a platform architecture as described in Figure 1 embedded in a single FPGA. We also define a dynamic application, where some parts of the application behavior can change at run-time. Such a dynamic behavior is possible by using FPGA dynamic and partial reconfiguration capabilities. Our approach proposes to dynamically replace a co-processor.

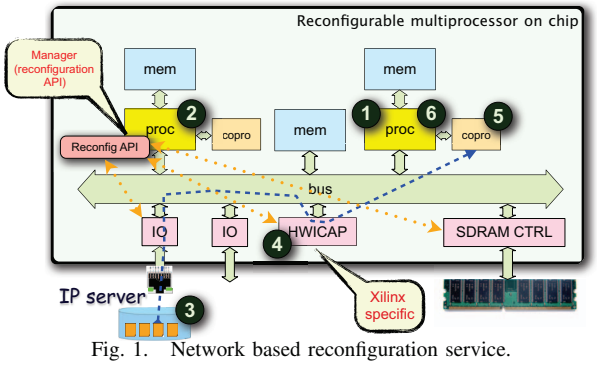

Fig. 1. Network based reconfiguration service.

FPGA configuration is performed by loading a bitstream into the device. The bitstream contains the system to be implemented into the FPGA. When dynamic and partial reconfiguration is done, a partial bitstream is required as only a part of the system is modified (in our case co-processors).

To perform dynamic reconfiguration, we need some target specific features. In this paper, we call *reconfiguration service* all the elements used to perform dynamic and partial reconfiguration, as defined in [7]. This service is composed of hardware and software elements (e.g. ethernet controller, Xilinx ICAP (Internal Reconfiguration Access Port), reconfiguration API) required to perform bitstream loading, as showed in Figure 1.

The service is offered to the system as a software API (*Application programming interface*), residing in one processor, identified as the *manager processor*. The reconfiguration is performed by target-dependent code that relies on specific platform components. Beyond the manager processor, a network Ethernet adapter, a SDRAM controller and a HWICAP (which is target specific) components must be embedded in the platform.

The reconfiguration request is done by the processor that wants to reconfigure one of its co-processor (1). The manager processor checks if the requested bitstream is in the SDRAM cache (2). If the bitstream is not present in the local cache, the processor sends a request to the IPserver (3) and after sends a reconfiguration command to the HWICAP (4). The co-processor is reconfigured by the HWICAP (5) through a direct access to the FPGA LUTs and at the end the manager processor sends a `END-OF-CONFIG` confirmation to the requesting processor (6). The reconfiguration service is accessible by the reconfig(id) function, where id is the bitstream identification to be loaded.

IV. Modeling

We use a co-design UML based approach to model the complete system. The modeling approach used is defined in [2], where three models are built to define a complete system: application, platform and allocation. The main rules for system modeling were first introduced in [8] and further enhanced in [9] and [10]. We first describe how to model dynamic behavior in section IV-A and how to generate code in section IV-B.

A. Dynamic system modeling

In order to facilitate understanding, speed-up design time and to allow seamless code generation, we define a set of design rules that allows DPR exploring.

1) Platform modeling: We consider the platform model as defined in [9] and we add reconfiguration specific information, as specified in [10]. The first point is to identify the reconfiguration manager processor. This is done by adding the ≪Manager≫ stereotype in the processor. Each reconfigurable zone is identified using the ≪HwPLD≫ stereotype from MARTE HRM profile, in the component.

Once the system model contains a reconfiguration manager processor and a set of reconfigurable zones (HwPLD components), the code generation tool automatically includes the reconfiguration library in the system. Also, in order to simplify the designer task, Xilinx HWICAP IP is automatically added in the system platform.

2) Application modeling: Modeling rules must be simple in order not to penalize productivity. Considering our application modeling methodology, we add dynamic behavior into an application component, i.e. dynamic property is defined at the task level.

Design patterns [11] have been used in object oriented modeling in order to build a common problem-solution catalog, speeding-up design time, creating a set of known terms (the pattern) shared by the community and allowing reuse of tested and approved solutions. In order to augment productivity and simplify the rules, we use well known design patterns to model a dynamic component: the *Strategy* and the *state* patterns, both are described in [11].

a) Strategy pattern: The strategy design pattern is defined as a software design pattern, whereby algorithms can be selected at run-time when it is necessary to dynamically swap algorithms in an application.

In Figure 3, we can observe the class diagram for the pattern. The Context active class captures the internal pattern behavior, where several objects inherit the Strategy abstract class. Once a strategy is chosen, the corresponding concrete object is pointed from the context. Details about our methodology modeling rules are explained in section IV-A4.

It is important to remark that, as we define a family of algorithms, the operations and associated parameters are the same for whatever algorithm is chosen.

b) State pattern: The state pattern is a behavioral software design pattern. This pattern is used in computer programming to represent the state of an object, where the behavior of an object depends on its state.

The state pattern class diagram is similar to the strategy one, as we can see in Figure 4. Each concrete object represents a context state with its associated behavior. Once the state changes the context pointer to the object changes.

3) Component modeling: To model a component dynamic behavior in our methodology we must consider two components: the client and the dynamic component.

We can observe the application dynamic part modeled in Figure 2. The first point that must be done is to inform that such a dynamic component can be able to reconfigure itself. This is done by adding the ≪adaptive≫ stereotype, indicating that this component changes its behavior dynamically.

The designer must also inform how that change is performed: by changing state (state design pattern) or by an explicit operation that tells the component to reconfigure (strategy design pattern). This is done by the adaptive stereotype tag method, which can be state or strategy. Modeling details change from the chosen method, as explained in the next sections.

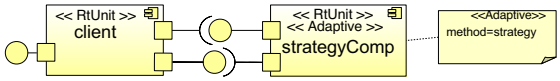

Fig. 2. The dynamic component is stereotyped with the adaptive stereotype and the method tag indicates the reconfiguration method.

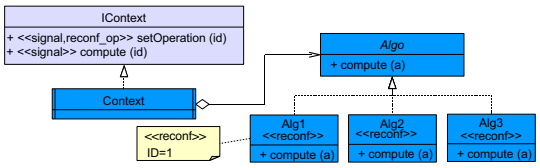

Fig. 3. Strategy: Reconfiguration operation and configurations IDs are indicated in the class diagram.

4) Strategy based reconfiguration modeling: When choosing the strategy method to allow dynamic components, the designer must model the component as follows: the reconfiguration operation must be indicated. This is done by the ≪reconf_op≫ stereotype in the chosen operation **and** the chosen operation must contain only one integer argument, which indicates what is the configuration to setup. Once the operation is defined, the designer also must indicate the ID tag of the ≪reconf≫ stereotype in the concrete classes. This information allows to download the correct bitstream.

Figure 3 shows the class diagram of the strategyComp component, where the setOperation operation is defined to be the reconfiguration trigger. In Section IV-B, we detail an example of the strategy pattern example code generation.

5) State based reconfiguration modeling: The state based method uses an internal variable to define which configuration to be used. This is done by the ≪reconf_state≫ stereotype in the active class state attribute of the component, as we can observe in the Figure 4, **and** the attribute type must be integer. The model is completed with the state indication in the concrete classes, like the strategy pattern, with the ID tag.

In both methods we need to define exactly **one** object in the dynamic component. This object corresponds to an instance of the active class in the class diagram. The set of concrete passive classes defines the number of configurations of the component.

Concrete classes are reachable from the abstract class through inheritance. Figure 5 shows a strategyComp and its associated state machine (actually, the active class state machine, which controls the component behavior).

6) Allocation modeling: Allocation modeling rules are the same for both methods: the client is allocated into a ≪HwProcessor≫ and the dynamic component is allocated into a ≪HwPLD≫ connected through dedicated buses to the processor containing the client, as show in Figure 6 for the strategy pattern. As we target Xilinx Virtex FPGAs in our tests, the allocation must be done into a co-processor connected to

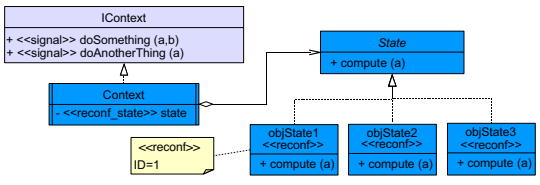

Fig. 4. Reconfiguration state and configurations IDs are indicated in the class diagram.

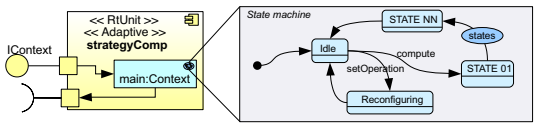

Fig. 5. Dynamic component internals: there is only one object, all other objects are derived from the modeling rules

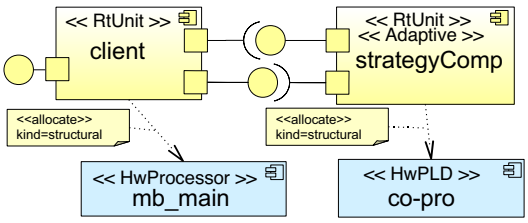

Fig. 6. Allocation of the dynamic component into the platform reconfigurable zone: the co-processor connected to the Microblaze processor.

a *Microblaze* processor through FSL dedicated buses. This limitation is required by the code generation tool, thus we can change target by adapting the tool or developing a new one, the model remains the same. In the next section, we show how to generate code from our dynamic application modeling.

B. Code generation

The code generation presented here enhances a set of existing rules introduced in [10]. Code generation algorithm depends upon chosen reconfiguration method. In both cases the generated code considers the reconfiguration service, as explained in section III, and the reconfiguration request source is the processor code, i.e. the reconfiguration client component.

1) Strategy method code generation: The code generation for the strategy method considers the reconf_op in order to generate appropriate reconfiguration call. The code generation tool replaces the method call with the reconfiguration command in the client component that runs on the processor. It is important to note that there is no inter-configuration variables, i.e. all values are lost from one configuration to another. Also, the code in the operation is not considered by the code generation tool.

As our code generation substitutes the call to the reconfiguration operation, the new configuration is in the initial state. This is an important point in the semantic of the tool, as any behavior implemented in the reconfiguration operation is not considered in the code generation.

The dynamic component is implemented as an hardware component. As we target Xilinx FPGAs series, we generate an ISE project for each concrete object in the class diagram that inherits from the abstract class pointed by the context object. Each ISE project contains VHDL code that is used to build the bitstream used to configure the corresponding co-processor in the system.

2) State pattern code generation: The state method code generation is more complex than the previous one, as the state is an attribute in the client object and potentially any operation can change its value. The code generator must find any assignment to the state variable and replaces it by a reconfiguration call by the client component.

In order to permit code generation in the state pattern, we constrain assignments to the state attribute as follows: first the method where the assignment operation is used should not

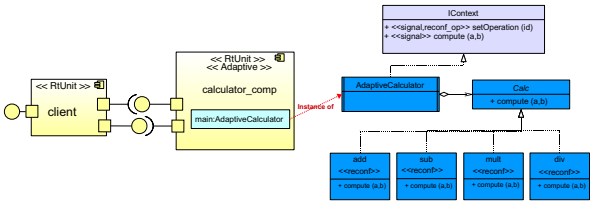

Fig. 7. Calculator application components and dynamic component class diagram.

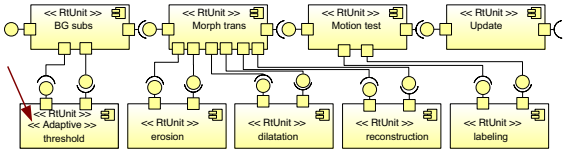

Fig. 8. The object tracking application **threshold** component can be implemented of several different ways.

change any data in the dynamic component. This is due to the fact that the reconfiguration cleans up the dynamic component implementation. The assignments operations are removed from the code and a `reconfig` command is inserted in the client object in order to trigger the reconfiguration. One ISE project must be generated for each state object in the diagram.

V. Results

We tested the strategy design pattern modeling and code generation. First, a calculator that computes any 2-number operations using the same chip area is considered. Figure 7 shows the components and the class diagram that models the dynamic component.

The client is allocated into a processor and the calculator_comp is allocated into an associated co-processor. It is not important how many Calc classes we define, only one of them is in the co-processor at any time.

The code generation tool replaces all calls to the setOperation method from the client component by the reconfig operation. From the calculator model, and following the design flow, we generated 6 bitstreams, 1 for the full initial configuration and 5 partial bitstreams to fit the reconfigurable zone (an empty one and one for each operation).

After validating the approach on this example, we also applied it to a more complex system, an object tracking application. Figure 8 shows the components of the application. The application consists of a set of image processing operations in order to mark moving objects. We consider 4 operations to be performed in the image: background substitution, morphological transformation, motion test and image update. Each of these operations is modeled as a component and allocated into a processor. Intensive computing parts of these operations are modeled as separated components (threshold, dilatation, erosion, reconstruction and labeling) and allocated into co-processors. The threshold component is required to be reconfigurable, as several threshold algorithms can be used, not at the same time.

The implemented system contains 60 platform IPs, where the threshold one is described by our dynamic component modeling technique. The chip area used by the threshold was about 2% of the total FPGA area (by LUT counting), a Xilinx Virtex 2 Pro FPGA packaged in the Digilent XUP board [12]. We implemented two different threshold algorithms.

VI. Conclusion and Future Work

In this paper, we have shown how to enhance a co-design UML based methodology in order to explore dynamic and partial reconfiguration of modern FPGAs. We used a reconfiguration service in order to abstract implementation details to the designer and to not overcharge design activity. Considering a fixed and well-defined reconfiguration service with its associated support on the target platform, we build a set of modeling methods.

We have tested our method within a co-design methodology in order to validate the concepts. In the examples, we tested the strategy design pattern modeling and code generation, and the state design pattern will be validated soon. The same idea can be re-used in any component-based modeling methodology in order to achieve a dynamic system modeling. Our approach speeds-up design time and facilitates model construction and understanding.

Next step on this work concerns the modeling and implementation of an internal reconfiguration manager. The main responsibility of the reconfiguration manager is to monitor the system internal behavior in order to take the decision to automatically reconfigure, e.g. change some algorithms due to a network quality of service changing.

References

[1] OMG, "Unified Modeling Language Specification," Object Management Group, Technical Report/2002-05-08, Specification formal/07-02-03, february 2002. [Online]. Available: http://www.omg.org

[2] A. Koudri, D. Vojtsiek, P. Soulard, C. Moy, J. Champeau, J. Vidal, and J.-c. Le lann, "Using marte in the mopcom soc/sopc methodology," in *workshop MARTE*, 03 2008.

[3] J. Dekeyser, P. Boulet, P. Marquet, and S. Meftali, "Model driven engineering for soc co-design," *IEEE-NEWCAS Conference, 2005. The 3rd International*, pp. 21–25, 2005.

[4] C.-H. Tseng and P.-A. Hsiung, "UML-Based Design Flow and Partitioning Methodology for Dynamically Reconfigurable Computing Systems." in *EUC*, 2005, pp. 479–488.

[5] Imran Rafiq Quadri, Samy Meftali, and Jean-Luc Dekeyser, "High level modeling of dynamic reconfigurable FPGAs," *International Journal of Reconfigurable Computing*, vol. 2009, p. 15, 2009.

[6] F. WEST Team LIFL, Lille, "Graphical array specification for parallel and distributed computing (gaspard-2)," 2005. [Online]. Available: http://www2.lifl.fr/west/gaspard/

[7] P. Bomel, J. Crenne, L. Ye, J.-P. Diguet, and G. Gogniat, "Ultra-fast downloading of partial bitstreams through ethernet," in *ARCS*, ser. Lecture Notes in Computer Science, M. Berekovic, C. Müller-Schloer, C. Hochberger, and S. Wong, Eds., vol. 5455. Springer, 2009, pp. 72–83.

[8] J. Vidal, F. de Lamotte, G. Gogniat, P. Soulard, and J.-P. Diguet, "A co-design approach for embedded system modeling and code generation with uml and marte," in *DATE '09: Proceedings of the conference on Design, automation and test in Europe*, 2009.

[9] J. Vidal, F. de Lamotte, G. Gogniat, J.-P. Diguet, and P. Soulard, "Ip reuse in an mda mpsopc co-design approach," in *ICM'09: Proceedings of the International Conference on Microeletronics*, 2009.

[10] ——, "Uml design for dynamically reconfigurable multiprocessor embedded systems," in *DATE*. IEEE, 2010, pp. 1195–1200.

[11] E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1995.

[12] X. INC, "Xup development board." [Online]. Available: http://www.xilinx.com/products/devkits/XUPV2P.htm