Building a Dynamically Reconfigurable System Through a High Development Flow

David de la Fuente, Jesús Barba,
Xerach Peña and Juan Carlos López
University of Castilla la Mancha
Email: David.Fuente@uclm.es

Pablo Peñil and Pablo Pedro Sánchez
University of Cantabria
Email: pablop@teisa.unican

***Abstract*—Partial Reconfiguration is one of the most attractive features of FPGAs. This feature provides new computing possibilities, such as the reduction of the total area required in a FPGA by means of functioning overlapping, or the modification of the design after its deployment, where a complete configuration is not needed. However, the design of partially reconfigurable systems is still a complex task. This work focuses on facilitating the design process and proposes a new development framework for dynamically configurable systems from high level UML/MARTE models which, starting from dynamically reconfigurable systems high level UML/MARTE models. Simulation and VHDL code are generated from those models, according to the specification requirements of the reconfigurable hardware captured in the specifications.**

To demonstrate this approach, a edge detection-based use case has been implemented with the developed framework.

***Keywords*—UML/MARTE, SystemC, VHDL, reconfigurability, automatic code generation**

I. Introduction

For nearly two decades, advances on reconfigurable devices have made FPGAs more and more attractive for embedded system design. Besides the improvement shown in design area, performance, efficiency, and dedicated resources (i.e. DSPs or BRAMs), most important FPGAs vendors have also included a special characteristic in some of their products: the partial reconfiguration facility.

This feature increases FPGA flexibility because its use is not limited to systems which are resolved at compilation time. A user can modify part of the device configuration at run time, allowing the modification of the initial design after its deployment without a complete configuration process. However, partial reconfiguration is far from being mainstream mainly because of the complexity of the development process [1].

One of the advantages of partial reconfiguration is the reduction of the cost of a system due to area reuse. One example of this strategy is software defined radio applications where it is not necessary to deploy all possible radio waveforms at the same time. Another advantage is the flexibility provided by the fact that new components can be developed and instantiated after the initial deployment of the design [2].

However, there are some disadvantages when considering partial dynamic reconfiguration. Only a few device families implementing run-time reconfiguration are available and, once a manufacturer is chosen, the designer must remain tied to the tools and the workflows imposed by the vendor. Partial reconfiguration forces developers to deal with traditional co-design techniques and high-level design problems [3]. In addition, vendor design flows, such as ISE or Vivado design tools from Xilinx, work at a very low-level of abstraction [4].

The work described in this paper reduces the complexity of building partially reconfigurable systems by means of a design flow developed to ease the design of this type of systems. During the design flow, multiple design alternatives can be considered and evaluated in relation to the number and type of functional components deployed in the FPGA fabric. The objective is to analyze the possible implications of each one of these variations in the entire system behaviour. Therefore, different properties such as power consumption, performance, critical path, etc., should be considered. In order to obtain the best design configuration, new design approaches performing early design space exploration (DSE) processes have been developed. This enables the search for the best FPGA element configuration according to the specific characteristics that determine the correctness of the system behaviour.

The development of electronic system-level (ESL) design methodologies [5] provides a strategy for designing complex systems, in which the initial key activity is specification. SystemC [6] is the most widely adopted language by the ESL community so as to write executable models of a system. Model-Driven Architecture (MDA) [7] is a design approach that enables expressing systems by means of models at different abstraction levels. As to MDA, the most widely accepted and used language the Unified Modelling Language (UML). Nevertheless, UML lacks of specific semantics required to support all the three embedded system specification, modelling and design steps. In order to overcome this limitations, several application-specific UML profiles have been proposed [8]. In this context, the standard MARTE profile [9] has been developed so that the modelling and analysis of real-time embedded systems is now possible. MARTE provides the concepts needed for capturing real-time features which are the corpus of the semantics in this kind of systems at different abstraction levels. By using this UML profile, designers are able to specify the system functionality composed of interconnected application components and the HW/SW platform where the application is executed. In addition to that, The UML Testing Profile [10] enables the definition of models that capture scenarios for system testing.

Following this combined approach, this paper presents an

infrastructure which generates, in an automatic way and using UML/MARTE model as the sole input, the code for both the simulation and implementation of dynamic reconfigurable systems. This infrastructure allows obtaining executable SytemC specifications for simulation and validation purposes according to the system requirements. Finally, our UML/MARTE methodology captures enough information about the target platform so that automatic VHDL code generation of the part to be allocated in the FPGA fabric is also possible.

The paper is organized as follows; in section 2, a study of the state-of-the-art is presented. In section 3, the complete design framework is described. Section 4 include a case study that will be used to explained all the aspects of the UML/-MARTE methodology. The experimental results are given in section 5 and finally, some conclusions are presented in section 6.

II. Related works

Nowadays, Xilinx development flow for building partial reconfigurable systems is very complex and difficult. This fact may be one the reason which prevents this technology from a higher degree of adoption in the industry.

In this case, the work presented in this paper proposes a solution close to Xilinx development flow, solving some restriction of the standard method and improving it to reduce the overall effort of the designer and the total number of errors in the early stages of the design flow.

The characterization of a system specification using a model helps the developer to obtain a global vision of the system to be built, since the model is developed using an approach based on the objectives of the system.

Targeting SystemC [11] enables building executable, platform agnostic validation environments. SystemC is a language which has been widely used for system-level and reusable test bench development, and moreover, the development of advance features intended to support verification and debugging.

Modelling a system brings about many benefits, among which the following can be highlighted: it helps to capture and organize the knowledge of the system, it allows the early exploration of alternatives, it facilitates the decomposition and modularization of the system, it reduces the total number of errors, it facilitates the reuse and maintenance of the system, increasing the productivity of the development team and, finally, it simplifies the documentation process, etc.

Several proposals for bridging the gap between UML specifications and SystemC executable models have been published. The first area where UML and SystemC where combined was the use of UML stereotypes for SystemC constructors. This combination focused on a system-on-Chip (SoC) design methodology, such as [13] or [14]. With the work of Riccobene et al. a SoC design flow was proposed based on a SystemC profile [15] aimed at the production of executable models from UML specification.

Most of the efforts spent on the integration of UML into the design process of embedded systems have targeted the synthesis of the hardware and software parts, indistinctly. Several research works on application code synthesis from UML models are characterized by the creation of state machine models or variations of them [16]. In [17], a formal design for reconfigurable, modular digital controller logic synthesis is presented. By means of UML state machines concurrent superstates are modeled, enabling the direct, automatic mapping on structured array of cells in FPGAs.

Nevertheless the major limitation of these approaches comes from the fact that UML, as a completely generic language, usually lacks of the semantics required to adequately model all the characteristics of embedded systems. In order to confront the challenge and cover the whole design flow of real-time embedded systems, the MARTE profile was created. Taking MARTE-based models as input, several synthesis approaches have also been proposed.

MoPCoM [18] is a methodology for the design of real-time embedded systems which supports UML and the MARTE profile for system modelling. Specifically, MoPCoM uses the NFP MARTE profile for the description of real-time properties, the HRM MARTE profile for platform description and the Alloc MARTE profile for architectural mapping.

Gaspard2 [19] is a design environment for data-intensive applications which enables MARTE description of both, the application and the hardware platform, including MPSoC and regular structures. More specifically, [20] presents a generic control semantics for the specification of system adaptability and dynamic reconfigurability in SoCs. The dynamic reconfigurability is implemented by generating the code for a reconfigurable region according to a high level application model, translating it into a hardware component, and generating the source code for a reconfiguration controller. The controller is in charge of the management of the different implementations related to the hardware resource.

In addition to that, [21] enables the design of FPGA-based embedded system, supporting automatic generation of VHDL descriptions from UML/-MARTE models, establishing a mapping rules to translate high-level elements into VHDL constructs, allowing the generation of fully synthesizable descriptions, including the embedded system structure and behaviour.

III. Design framework

The task of building a reconfigurable project is long and complex, mainly because of the available tools working in the background, at the lowest levels. The proposed design framework aims to rise the level of abstraction of the tasks which must be performed by the developer. As a consequence, the effort needed to turn a static design into a dynamic one is minimized. Our methodology starts from the creation of an UML/MARTE high level model of the dynamically reconfigurable system, based on a component-based methodology. Then, the flow provides two alternatives: the first one simulates the behavior of the system through a SystemC executable specification; and the second one links with the available vendor tools in order to automatically synthesize the bitstreams. Figure 1 illustrates the main steps of this design flow.

The graphical tool used to create the UML/MARTE model is Papyrus [23] which generates XMI files. A XML code

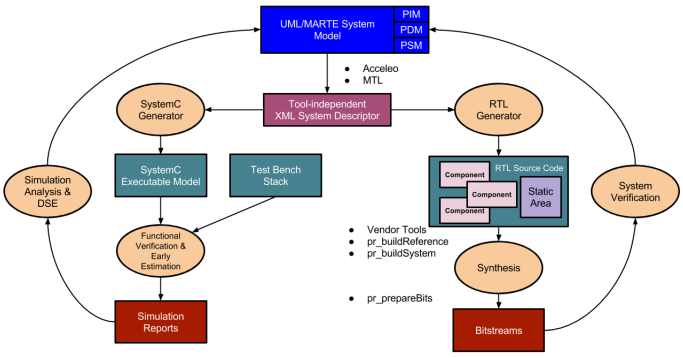

Figure 1: Design flow for dynamically reconfigurable systems design.

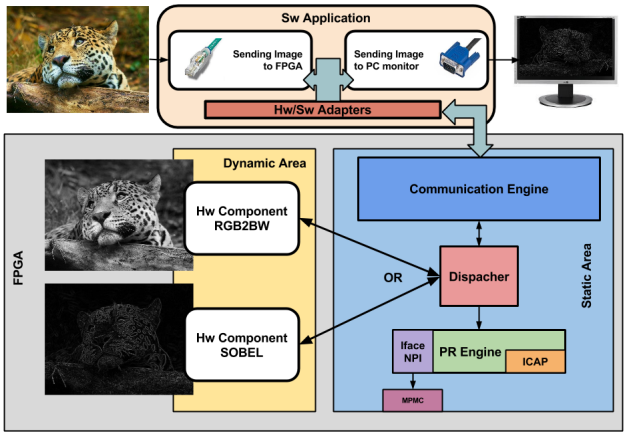

Figure 2: Case study application.

generator has been developed, using a set of generation templates written in the standard MTL language [24]. The development has been done through Acceleo [25], a code generation framework fully integrated in Eclipse. This XMI-XML transformation has been done so that other external tools can bind to the flow in this step. Based on Xilinx's reconfigurable technology, this flow adds three new tools which form a higher level front-end: pr_buildSystem, pr_buildReference and pr_prepareBits. In the following section, the flow will be explained by means of a image processing case study.

IV. Case Study

In this section we apply the presented flow to generate the source code for an image processing application. The main goal of the application is to perform edge detection on a image sequence applying the SOBEL filter. The filter requires grayscale pictures, so a previous "RGB to Black and White" transformation is needed. Hence, in the system there will be two components that will be exchanged over the same dynamic area at runtime. Finally, the obtained image will be shown in a computer monitor. All computation tasks will be carried out in an FPGA, in particular a Virtex-5 XC5VFX110T model of Xilinx. In order to receive the images to be processed, the FGPA is connected to a PC via Ethernet. This application works with images whose size is 640x480 pixels.

The application (illustrated in Figure 2) will be made up of four components: (a) the first takes a sequence of images from the camera and sends them to the FPGA via Ethernet, (b) the second one converts the image to grayscale, (c) the SOBEL filter will be applied by the third component and finally, and the last component (d) transfers the image to the video memory so it is displayed in a monitor.

Since it is considered a scenario where the available resources are not enough to place both the RGB2BW and SOBEL filters, it is necessary to use a dynamic area. On top of this, thanks to this separation, a single image device is provided which is able to reconfigure itself in order to get different behaviors.

A. UML/MARTE specification model

In this proposal, the goal is to enable the automatic generation of executable models and FPGA programming files from component-oriented models [26] by means of the application of the Model-Driven Architecture [27] principles and techniques to heterogeneous embedded systems.

However, UML, as a generic modeling language, usually lacks of all the semantics required to adequately model all the characteristics of embedded systems. To solve that, the OMG has proposed the MARTE profile [9] as an alternative to confront the challenge to cover the complete design flow of real-time embedded systems. Thus, the methodology has relied on the UML/MARTE standard to capture all the system details.

System specification starts with the definition of the *the Platform-Independent Model* (PIM) where the application components are modeled and the application structure is described as instances of these application components who exchanges data using a message exchanging system. After that, the model is complemented by a description of the HW/SW platform which conforms the *Platform Description Model* (PDM). In the PDM, the HW component resources are identified together with the required features for each one, depending on the specific application under consideration.

Finally, the PIM and PDM models converge into the *Platform-Specific model* (PSM) where a mapping between functional components and available resources in the HW/SW platform is established.

The graphical orientation of UML helps the designers to handle large systems in an easy way. However, the UML/-MARTE model must contain all the relevant, essential information, of the system in order to allow the execution of the simulation and synthesis process. Thus, it is required to define a UML/MARTE methodology combining the benefits of a visual language with large amounts of information. To solve this issue, the information contained in a UML/MARTE model is separated in specific concerns, depending on their application area. Each concern is captured in a model view, which is represented using the UML diagrams that most fit the concern.

1) Application components: Application components identify pieces of functionality that represent a certain behavior with a relevant role in the hierarchy of the system. The entire system is composed of application components which are, in

turn, interconnected. This interconnection is established by services, grouped in interfaces. In this methodology, the application components are modeled by the MARTE stereotype <<RtUnit>>. The functionality of each application component is enclosed in two elements: interfaces and files. The interfaces group the services provided/required needed for establishing the data transmission between the application components. The interfaces are modeled by means of UML interfaces specified by the MARTE <<ClientServerSpecification>> stereotype.

The case study defines three components but two of them will be implemented in SW in the main application (Figure 3). Each application component has an associated set of code files that define the component functionality. These files are modelled as UML artifacts using the UML standard stereotype <<File>>.

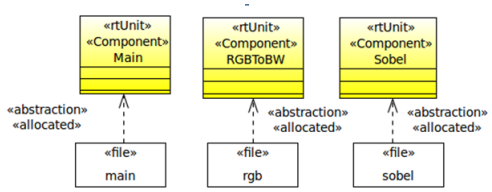

Figure 3: Application Components.

The communication is established using two elements: ports and channels. The ports enclose the interfaces that are provided or required by the component. The ports are specified by the MARTE stereotype <<ClientServerPort>>. Each port can support only one interface, and it can be specified with the values provided or required from the attribute kind of the ClientServerPort stereotype depending on how this interface is used in the component. The system is composed of interconnected instances of application components types (Figure 4). The application instances are connected by links between clients and servers who owns ports implementing a compatible interface. These instances are connected through UML connectors, linking ports (Figure 4).

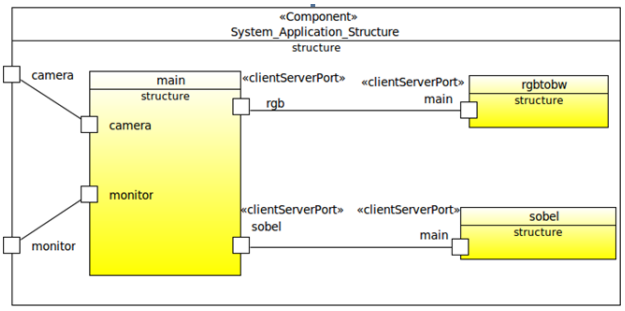

Figure 4: Application System structure.

After that, functional components are then mapped to memory spaces (Figure 5), which will be assigned to HW resources in the target platform model. Thus, components associated with a single memory space are always handled together. Specifically, all the application instances that are realized in HW should be grouped in the same memory space (Figure 5, "rgb2bw" and "sobel"). This association with memory spaces enables the mapping to non symmetric systems, since the model ensures that there are no memory access problems.

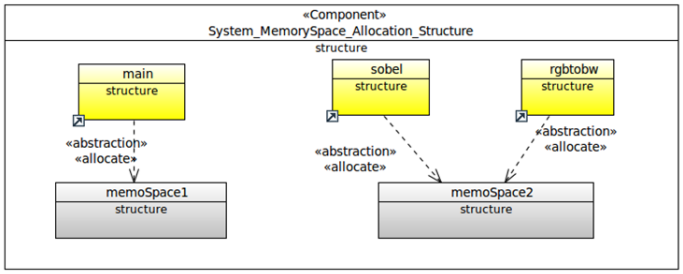

Figure 5: Memory spaces structure and application mapping.

2) HW/SW platform: The platform where the application is executed includes the HW resources and the SW infrastructure. The hardware platform is captured and supported through MARTE stereotypes of the Hardware Resource Modeling (HRM) subprofile (Figure 6). Processors are modelled with the stereotype <<HwProcessor>>. The bus communication element is also supported through the <<HwBus>> and as well as different types of memories as instruction cache and data cache stereotype <<HwCache>> and program memory stereotype <<HwRAM>>. Each stereotype enables the possibility to set the values of different attributes of the platform.

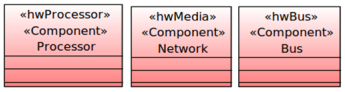

Figure 6: Hw component resources.

The other processing element is the FPGA. The FPGA is modeled by the MARTE stereotype <<HwPLD>>. In order to enable the synthesis process, this component defines compulsory properties for its characterization. These properties are $device, $package, $speedGrade, $architecture and $period (Figure 7).

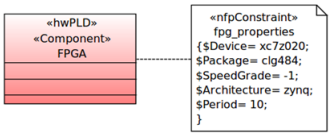

Figure 7: FPGA component.

Additional features have to be specified in the FPGA. Specifically, these features are related to the structure of the FPGAs that defines the different areas available to configuration. In a composite structure diagram associated to the HwPLD component, the different sections of the FPGA are modeled as parts typed as <<HwComputingResource>> MARTE stereotype. Then, these parts are specified by the MARTE stereotype <<HwComponent>>. In the attribute position, the localization of each section can be specified. Another set of different components considered in the methodology are the resources for communicating the processing elements. Networks and network interfaces can be included in the HW

model, with the <<HwMedia>> and <<HwEndPoint>> MARTE stereotypes, respectively (Figure 6). According to the additional properties of the HwEndPoint, different types of commutations are captured. When the network interfaces includes the IPAddress property the communication is IP connection, involving a different synthesis process (Figure 8). When the network interfaces includes the serialPort attribute (Figure 8) the communication is by serial port.

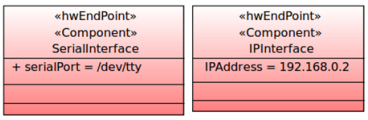

Figure 8: Communication interfaces components.

Regarding the SW infrastructure, the operating system is modeled by an UML component specified by the stereotype <<OS>>[22]. The attributes associated with this stereotype are the type of OS (Ubuntu, Fedora, Angstrom, etc.) and the policy which defines the process scheduling algorithm (Fixed Priority Preemptive, Round Robin, FIFO, EDF, etc.). Finally, when HW and SW components are defined, the target platform is created by using parts typed by the previous HW/SW components interconnected (Figure 9). In addition to that, the allocation of the memory spaces previously defined is captured (Figure 9).

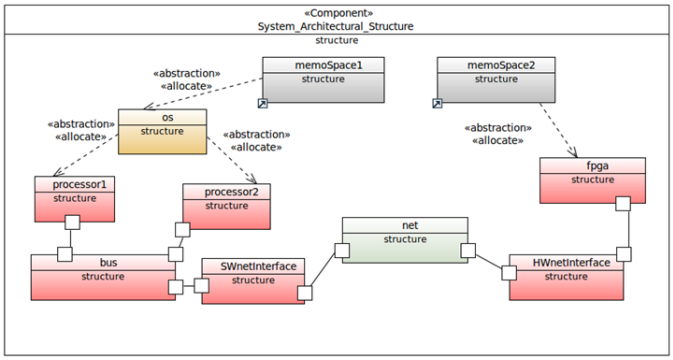

Figure 9: HW/SW platform and memory spaces allocation.

3) Environment: In order to support code generation for evaluation, test benches have to be integrated. To include them, a set of UML components is specified by means of stereotypes included in the standard profile UML UTP. The components which represent environment elements are specified by the UTP stereotype <<TestComponent>>. These application components have the corresponding files associated.

Another component is specified by the UTP stereotype <<TestContext>>. This component defines the structure of the environment and the interconnection of environment components with the system under test. The environment structure is modeled in a UML composite structure diagram associated with this TestContext component (Figure 10). This composite structure diagram contains instances of TestComponents previously defined connected to a UML property typed by the component where the application is captured (components defined in Figure 6, Figure 7 and Figure 8). This property is specified by the UTP stereotype SUT (System Under Test) as can be seen in Figure 10.

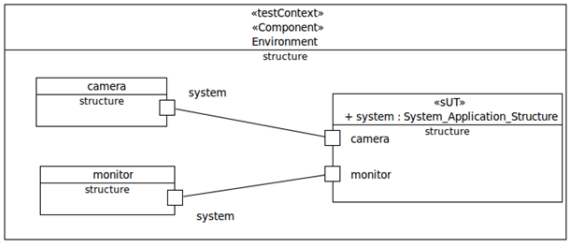

Figure 10: Environment structure.

B. Simulation Branch

The main goal of this step is to model and simulate the final behavior of the system at an early stage of the design process. As a result, the designer is provided with accurate measures on the performance of the system. It is the responsibility of the system architect to interpret and propose the necessary changes in order to meet the requirements. As a distinctive feature of this work, the reconfigurable process is also considered for emulation which gives an enhanced picture of the actual behavior of the final system.

All this can be achieved keeping the degree of manual intervention in a minimum level. Therefore, a reconfigurable system architect is only requested to write the behavior and specify some temporal parameters of the hardware components which will be deployed in the reconfigurable fabric.

A Reconfigurable Unit (RU) entity is defined as a container for the core function of an application component. Both elements (RU+core function) results in what is called SystemC application component. In order to emulate reconfigurability, each SystemC component is compiled as a dynamic library. This way, the programming of the FPGA which results in a change of the behavior of a reconfigurable area is simulated by means of loading/unloading the appropriate software library.

The second level of integration has to do with the management of the communication with other components. A Communication Adapter (CA) is responsible of the adaptation of the low level FIFO interface of the SystemC application component to TLM semantics. The new logical layer adds additional information to the messages to be exchanged; identification key of the message, the logical target address or the logical source address, for example. Figure 11 graphically represents the composition of the different abstraction layers from the core function to a fully functional system component.

As a result, the simulation step allows connecting not only a set of components described in the UML/MARTE, but also other type of systems, as long as the specific CA has been appended. In this way, not only flexibility in the communication topology is higher but also it achieves the capability to communicate with external elements connecting to a TLM bus and which do not belong to the simulation environment itself.

Once the components have been designed, it is necessary to provide a Reconfiguration Manager (rc_M) to model all

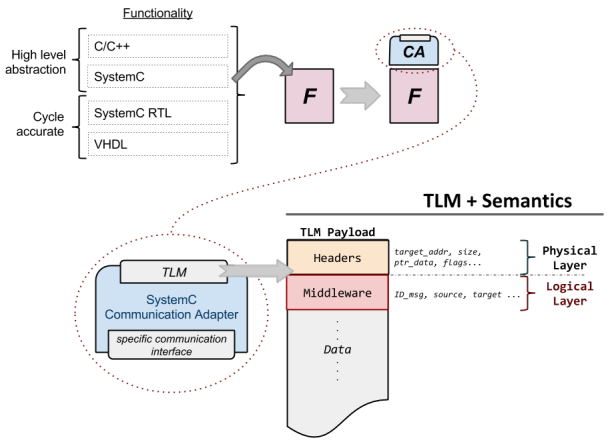

Figure 11: Communication Adapter (CA) Scheme.

aspects as to the reconfiguration such as the interchange of functionality (loading or unloading the dynamic libraries) and management of the execution state of the components (start and stop the process). In addition, the rc_M provides a communication mechanism which allows carrying out the reconfiguration of each component (Figure 12).

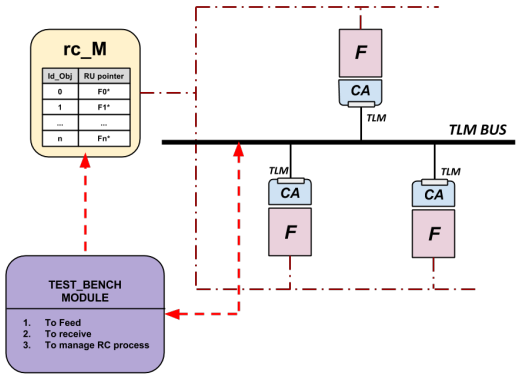

Figure 12: Test bench module for system simulation.

After modeling the reconfigurable system using SystemC, a behavior analysis is needed. Taking into account the temporal parameters of the components, reconfiguration latency or dynamic area distribution, the output of this phase provides an advanced knowledge on system behavior. With this, we can ensure that the system meets the user requirements. To get it, a test bench module (TBM) has been included in the SystemC infrastructure previously described (Figure 12).

TBM offers the ability to inject/receive data to/from the system and manage the reconfiguration process. The actions allow to emulate all possible tasks that may occur in the system, to evaluate its delays and to simulate different scenarios in order to compare different configurations of the system (Design Space Exploration). If the entire system behaves as expected, the following step is to implement the system, starting with the construction of the static area.

C. RTL Synthesis Branch

Dynamic reconfiguration management is typically solved using a software approach, where an embedded processor is in charge of transferring the reconfigurable bitstream to the configuration memory of the device in order to change the content of the dynamic area. The approach presented in this work, however, relies on a specialized hardware component: the Reconfiguration Engine (referred as prEngine). This has two main advantages: first, it obtains two orders of magnitude improvement in speed compared with the software approach [28] and; second, no processor is required for reconfiguration management freeing CPU clock cycles for other more critical tasks.

The infrastructure allocated in the static area of the FPGA is composed of the prEngine that offers a set of dynamic reconfiguration related services through a simplified interface. From the architectural point of view, the prEngine has three elements: the Factory, the Reconfiguration Controller and some kind of storage (i.e. external memory), although it is not really part of the engine, and can be shared with the rest of the resources in the system (Figure 13). The Factory component deals with the issues related to the transference and manipulation of partial bitstreams and the Reconfiguration Controller (rController) is responsible for high-level management of the reconfiguration process. It receives the reconfiguration commands and issues the corresponding operation requests to the reconfiguration areas or to the Factory.

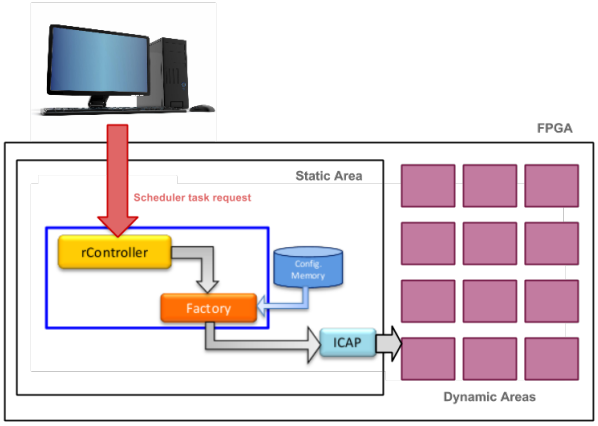

Figure 13: Block diagram of prEngine component.

The main purpose of the prEngine is to provide a common abstraction layer compatible with all the technologies, by means of encapsulating the mechanism as a transparent service to the upper abstraction layers. Therefore, the prEngine is automatically generated when a dynamic areas is defined in the UML/MARTE specification model. The prEngine structure does not change in any device, however there are some parts which depend on the FPGA technology. Therefore a specific tool from Xilinx called Coregen is used as the backend. A .cgp file is generated from the specification model which is processed by Coregen to build hardware components for a specific device or family. In our proposal, the partial bitstreams are stored in DDR memory. Thus, the read and write operations are supervised by the memory controller component (MPMC - Multi Port Memory Controller]).

Once the static part of the design has been identified, the layout of the dynamic reconfigurable areas must be defined. For each reconfigurable area at least a geometric location, its resources and a name are specified in the UML/MARTE model. The geometric location delimits a rectangular shape using the bottom-left and top-right slice identifiers.

In the solution proposed, one reconfigurable area is, actually, composed by a grid of smaller sub-areas (see Figure 14(a)). These sub-areas are interconnected by generic data and control signals which enable the communication between them. Also, a reconfigurable area may have one or more interconnection channels from/to the static area of the FPGA. The designer must enable or disable these interconnections channels and define the type of use they are going to support which can be a bus or point-to-point protocol, for example. Figures 14(b) and 14(c) have channel B disable whereas Figure 14(d) has both channels enable with a different configuration.

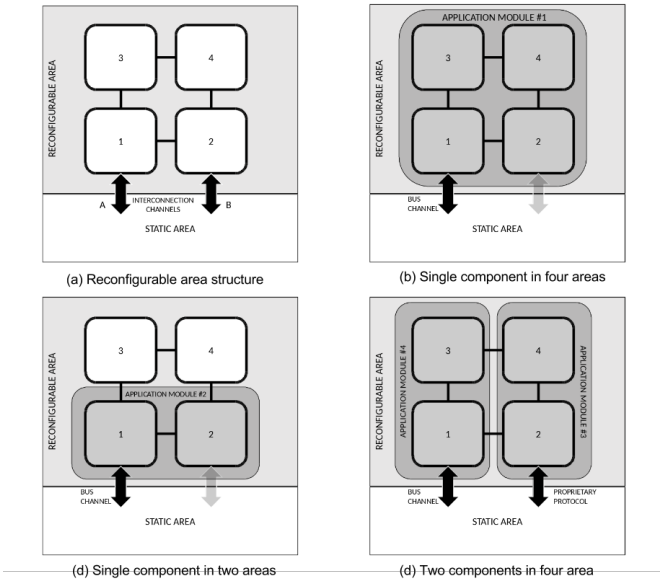

Figure 14: Hierarchical dynamic areas.

A top file described in VHDL is generated according to the dynamic areas defined in the model. This top file is composed by the static and the dynamic areas as defined in the model. The dynamic areas are treated as a black box, since the deployment of any component is possible. The assignment of functionality to areas is known as a configuration definition (described in a *.pxml file). The pr_buildReference uses this configuration to

The next step consists in the construction of the file system and template files of the reconfigurable project. This is performed by the pr_buildSystem tool, that requires as input both the top.ucf file and the static part of the project that can be obtained from the UML/MARTE model. The tool builds the wrappers and necessary templates to develop dynamic components and different combinations of reconfigurable modules.

Figure 15 shows an example of a reconfigurable system is depicted. This tool generates two main directories in the file system: regions and generated.

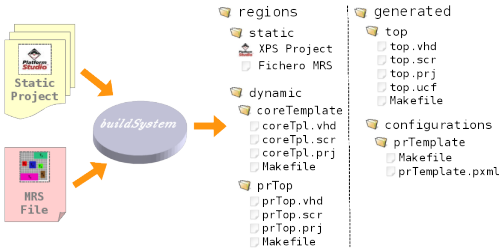

Figure 15: pr_buildSystem tool.

After the whole project infrastructure has been created, the developer must assign the components already tested of the static area in the corresponding directories. Each directory will generate a configuration compatible with a certain area. If a new component, not previously designed, is built, it can make use of the generated template. The next step is to build an initial configuration (reference configuration) by means of pr_buildReference tool. This configuration is used as both the initial design to be downloaded, and as a reference for the extraction of the partially reconfigurable areas.

The following step consists in the synthesis of the reconfigurable Project to get system bitstreams. Previous tools (pr_buildSystem and pr_buildReference), generate several scripts, which help designers to automate this process.

Once all the different bitstreams have been obtained, the last step consists in modifying them for being used by the factory component, which is responsible for the partial reconfiguration of FPGA. The prepareBits tool removes original bitstream headers and trashes the data after completing the command. The new bitstream is divided into three parts: a reduced header, which includes configuration and synchronization commands, the bitstream data, and a completion command. All these commands are aligned into 32 bits words. The tool is also able to join several partial bitstreams. The result is a bitstream, which has a header at beginning of file, then the data of each original bitstream and, finally, the completion command.

Finally, a system verification is needed to guarantee the correct working of the project, including all process: reconfiguration, application deployed into a dynamic area, etc., so the user has to define integration tests to verify the entire system. Due to some tools of Xilinx are beyond of our control, the system can be synthesized in a not desired way, In that case it might be necessary to make modifications to the reference UML/MARTE model.

V. Results

From the UML/MARTE model, the SystemC simulation process builds the entire simulation infrastructure allowing the behaviour/temporal system evaluation. In this case, all components have associated the time estimation required in the simulation phase. After analyzing the results of this stage, the simulation log file indicates the following:

- If only RGB2BW filter is used, the estimated performance is 63.19 frames per second.

- Using dynamic reconfiguration and the SOBEL filter, the performance is reduced around 35.27
- The reconfiguration rate is 180.3 MB/s.

To get an idea of the benefits of using the proposed design flow, the table I shows the lines of code that have been generated automatically instead of handwritten.

Elements (VHDL)	Lines of code	Elements (SystemC)	Lines of code
Static Area	4568	SystemC Platform	950
RGB2BW Wrapper	1483	RGB2BW RU	69
SOBEL Wrapper	1771	SOBEL RU	69
TOTAL	7822	TOTAL	1088

Table I: Source code generated.

For example, as can be seen in the table, about 8.000 lines of VHDL source code were generated, so that the developers only have to focus their efforts in the functionality code. In this scenario, both Hw filters (RGB2BW and SOBEL) has been generated from C/C++ using a tool of Xilinx called Vivado HLS. In the first one, the code that has been written by the developer only has 15 lines and the second one 16. With these few lines, the Vivado HLS tool generates 3254 lines of functional code automatically.

VI. Conclusion

This work provides a full Eclipse integrated design flow that allows tocreate dynamically reconfigurable applications from a high abstraction level (UML/MARTE model), to simulate different scenarios based-on design space exploration and to implement the final application over a FPGA device.

A hierarchical approach to the reconfiguration problem is taken, where top-level reconfigurable areas can be divided into a set of smaller independent regions, that can also be freely composed at run time. The tools rely on the use of a pr_Engine which is provided as an independent IP core. The core provides a common abstraction to the reconfiguration process which is device independent. New FPGA families can be supported with little effort.

VII. ACKNOWLEDGMENT

This work has been partly funded by the Spanish Ministry of Economy and Competitiveness under project REBECCA (TEC2014-58036-C4-1-R) and by the Regional Government of Castilla-La Mancha under project SAND (PEII_2014_046_P).

References

[1] Christophe Bobda, *Introduction to Reconfigurable Computing: Architectures, Algorithms, and Applications*, isbn 1402060882, 9781402060885, Springer Publishing Company, Incorporated. 2007

[2] D. Dye, *Partial Reconfiguration of Xilinx FPGAs using ISE Design Suite"* (UG702), Xilinx, 2011.

[3] W. Wolf, *High-Performance Embedded Computing. Architectures, applications, and methodologies*, Princeton Universit, 2006.

[4] Xilinx, "Partial Reconfiguration User Guide", Xilinx, 2011.

[5] N. Steiner, A. Wood, H. Shojaei, J. Couch, P. Athanas and M. French *Torc: Towards an Open-Source Tool Flow*, Nineteenth ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2011

[6] T. Cervero, S. Lopez, R. Sarmiento, T. Frangieh and P. Athanas *Scalable Models for Autonomous Self-Assembled Reconfigurable Systems*, International Conference on ReConFigurable Computing and FPGA, 2011.

[7] G. Martin, B.Bailey, A. Piziali. *ESL Design and Verification: A Prescription for Electronic System Level Methodology (Systems on Silicon)*. March 9, 2007. ISBN-10: 0123735513.

[8] Y. Vanderperren, W. Mueller, W. Dehaene: *UML for Electronic Systems Design: A comprehensive overview*, Design Automation for Embedded Systems, V.12, N.4. December 2008.

[9] OMG: "UML Profile for MARTE", www.omgmarte.org, 2013.

[10] OMG: "UML Testing Profile (UTP) 1.1", website http://utp.omg.org/ . Nov 2014.

[11] IEEE Std. 1666-2011. IEEE Standard for Standard SystemC®Language Reference Manual. Jan., 2012. Available at http://standards.ieee.org/getieee/1666/download/1666-2011.pdf.

[12] Y. Vanderperren, W. Mueller, W. Dehaene. UML for Electronic Systems Design: a comprehensive overview. Journal on Design Automation for Embedded Systems, Springer Verlag, August 2008.

[13] F.Bruschi, E. Di Nitto, D. Sciuto. SystemC Code Generation from UML Models. Forum on Specification and Design Languages 02.

[14] W.Muller et al. The SATURN approach to sysML-based HW/SW codesign. IEEE Annual Symposium on VLSI, ISVLSI 2010.

[15] S. Bocchio, E. Riccobene, A. Rosti,P. Scandurra. A SoC design flow based on UML 2.0 and SystemC. In DAC, Workshop UML-Sock'05.

[16] D. Harel, H. Kugler, and A. Pnueli, "Synthesis revisited: Generating statechart models from scenario-based requirements," Formal Methods in Software and System Modeling, 2005.

[17] M. Adamski. Design of reconfigurable logic controllers from hierarchical UML state machines. 2009 4th IEEE Conference on Industrial Electronics and Applications, ICIEA 2009.

[18] J. Vidal, F. de Lamotte, G. Gogniat, P. Soulard, J.P. Diguet. "A Code-Design Approach for Embedded System Modelling and Code Generation with UML and MARTE". In Proceedings of DATE'09. Dresden. March, 2009.

[19] E. Piel, R. Atitallah, P. Marquet, S. Meftali, S. Niar, A. Etien, J.-L. Dekeyser, P. Boulet: "Gaspard2: from MARTE to SystemC Simulation", proc. of the DATE'08 workshop on Modeling and Analysis of Real-Time and Embedded Systems with the MARTE UML profile, 2008.

[20] I.R. Quadri, Yu Huafeng, A. Gamatie, E. Rutten, S. Meftali, J.-L. Dekeyser. Targeting reconfigurable FPGA based SoCs using the UML MARTE profile: from high abstraction levels to code generation. International Journal of Embedded Systems, v 4, n 3-4, p 204-24, 2010.

[21] M. Leite, C.D. Vasconcellos, M.A. Wehrmeister. "Enhancing automatic generation of VHDL descriptions from UML/MARTE models". 12th IEEE International Conference on Industrial Informatics (INDIN), 2014.

[22] H.Posadas, P.Peñil, A. Nicolás, E.Villar, "Automatic synthesis of embedded SW for evaluating physical implementation alternatives from UML/MARTE models supporting memory space separation". Microelectronics Journal. Volume 45, Issue 10, October 2014, Pages 1281–1291.

[23] Papyrus webpage http://www.papyrusuml.org

[24] MOF Model To Text Transformation Language webpage http://www.omg.org/spec/MOFM2T/1.0

[25] Acceleo webpage http://www.acceleo.org

[26] C. Szyperski, Component Software: Beyond Object-Oriented Programming. Addison-Wesley Professional, 2002.

[27] D. C. Schmidt, "Model-driven Engineering" IEEE Computer, vol. 39 no. 2, pp. 25-31, 2006.

[28] J.D. Dondo, J. Barba, F. Rincón, F. Moya, J.C. López. "Dynamic Objects: Supporting Fast and Easy Run-Time Reconfiguration in FPGAs".JSA - Journal of systems Architecture - 01/01/2013