A High-level Methodology for Automatically Generating Dynamic Partially Reconfigurable Systems using IP-XACT and the UML MARTE Profile

Gilberto Ochoa-Ruiz, Ouassila Labbani, El-Bay Bourennane, Philippe Soulard, Sana Cherif

Des Autom Embed Syst
DOI 10.1007/s10617-012-9098-6

A high-level methodology for automatically generating dynamic partially reconfigurable systems using IP-XACT and the UML MARTE profile

Gilberto Ochoa-Ruiz · Ouassila Labbani · El-Bay Bourennane · Philippe Soulard · Sana Cherif

Received: 12 March 2012 / Accepted: 11 October 2012

Abstract Dynamic Partial Reconfiguration (DPR) has been introduced in recent years as a method to increase the flexibility of FPGA designs. However, using DPR for building complex systems remains a daunting task. Recently, approaches based on Model-Driven Engineering (MDE) and UML MARTE standard have emerged which aim to simplify the design of complex SoCs, and in some cases, DPR systems. Nevertheless, many of these approaches lacked a standard intermediate representation to pass from high-levels of descriptions to executable models. However, with the recent standardization of the IP-XACT specification, there is an increasing interest to use it in MDE methodologies to ease system integration and to enable design flow automation. In this paper we propose an MARTE/MDE approach which exploits the capabilities of IP-XACT to model and automatically generate DPR SoC designs. We present the MARTE modeling concepts and how these models are mapped to IP-XACT objects; the emphasis is given to the generation of IP cores that can be used in the Xilinx EDK (Embedded Design Kit) environment, since we aim to develop a complete flow around their Dynamic Partial Reconfiguration design flow. Finally, we present a case study integrating the presented concepts, showing the benefits in design efforts compared with a purely VHDL approach and using solely EDK. Experimental results show a reduction of

G. Ochoa-Ruiz (✉) · O. Labbani · E.-B. Bourennane
LE2I Laboratory, Burgundy University, Dijon, France
e-mail: gilberto-ochoa-ruiz@etu.u-bourgogne.fr

O. Labbani
e-mail: ouassila.labbani@u-bourgogne.fr

E.-B. Bourennane
e-mail: ebourenn@u-bourgogne.fr

P. Soulard
SODIUS, Nantes, France
e-mail: psoulard@sodius.com

S. Cherif
INRIA Lille Nord Europe, Villeneuve d'Ascq, France
e-mail: Sana.Cherif@inria.fr

Published online: 31 October 2012

the design efforts required to obtain the netlist required for the DPR design flow from hours required in VHDL and Xilinx EDK, to less the one hour and minutes for IP integration.

Keywords Dynamic Partial Reconfiguration · UML MARTE · MDE · IP-XACT · Rapid System Prototyping

1 Introduction

Run-time reconfiguration (RTR) has been introduced in recent years as a means of virtualizing hardware tasks in FPGA systems [1]. However, it wasn't until the introduction of Dynamic Partial Reconfiguration (DPR) technologies by Xilinx that these systems became a reality. In DPR systems, parts of the system can be reconfigured at run-time while the other functionalities in the FPGA remain operational [2]. This capability can provide many benefits to the systems designers, such as power and resources reduction, amongst others.

However, despite the efforts by Xilinx and many industrial and academic endeavours, using DPR in very complex systems remains a daunting task. This is due, in the first place, to the complexity of the design flow [3], which requires an in-depth knowledge of many low level aspects of the FPGA technology. Secondly, efforts in the academia to extend the capabilities of DPR design flow have further increased the complexity of DPR SoC designs. Furthermore, the creation of SoC DPR-based systems has very specific requirements, in particular, IP reuse capabilities in which the parameterisation and integration of IP cores (both DPR and non-DPR components) is performed in such a way that facilitates the design process.

Model-Driven Engineering (MDE) techniques, in tandem with UML, have been used in co-design methodologies in the last years with relatively success in embedded systems modeling [3]. Many of them make use of the UML profile for "Modelling and Analysis of Real Time and Embedded Systems" (MARTE) [4]. UML/MARTE models are used not only for communication purposes but, using model transformations, to produce concrete results such as a source code. For this purpose, MDE methodologies for SoC make use of a deployment phase in which the building blocks of the high-level models are linked to the low implementations that embody the related behaviour. This is basically an IP reuse problem, and in this way the components can be configured, and a synthesizable top-level implementation can be obtained. The main issue is that most MDE methodologies found in the literature make use of non-standardized representations. This means that a Deployed Allocation model is not easily interchangeable, and that IP reuse is highly methodology-dependant.

The SPIRIT consortium has developed the IP-XACT specification [5] that describes a standard way documenting IP meta-data for SoC integration. Several industrial case studies have demonstrated that the adoption of IP-XACT facilitates the configuration, integration, and verification in multi-vendor SoC design flows [6, 7]. Additionally to the IP packaging and integration, IP-XACT can provide an effective means for IP reuse by linking the low level implementation to their high-level counterparts in an MDE approach.

The contributions of this paper relate to presenting an MDE approach that uses the MARTE profile and that enables moving from high level models to HDL code generation for system description, including the DPR components. Since we aim for a component based approach, the seamless integration and interoperability of the used IP is a necessity. IP-XACT is used as an intermediate model, used to configure the deployed IPs in the DPR system and to automate the system integration and parameterisation. The rest of this paper

is organized as follows: In Sect. 2 we introduce the Xilinx DPR design flow and the motivation of the work presented in this paper. In Sect. 3, we examine the related works in the areas of hardware resource modelling using UML and the efforts done by the academia to integrate IP-XACT into MDE approaches. Section 4 summarizes the most prominent features of IP-XACT and how they are exploited in our MDE methodology. In Sect. 5 we present the proposed methodology in detail, and in Sect. 6 we elaborate on the used model transformations. Then, in Sect. 7, we embark in a case study for the integration of two implementations of an image processing IP into a SoC based DPR design. Finally, Sect. 8 we present the conclusions and perspectives for future work.

2 Related works

The use of model based approaches for co-design has been thoroughly discussed in [13]. UML/MDE has been adopted in co-design methodologies in the last years with relatively success. The extensions mechanisms introduced in UML have stimulated its use in embedded systems modeling [14, 15]. In particular, structural modeling has been the most prominent application of the UML in SoC design, for specification of requirements, behavioural and architectural modelling, and system integration. There are several approaches which use the UML profile and extensions to support embedded hardware resources modelling. Many of them made use of the UML profile for "Modelling and Analysis of Real Time and Embedded Systems" (MARTE) [4].

A typical MDE methodology for high-level system co-design is presented Fig. 1; many approaches are based in similar design flows, which are inspired by the Y chart. The Y schema is generally adapted to represent the SoC Co-Design approaches; its three axes represent functional behavior, hardware architecture and final implementation in specific technologies. The central point of these three axes denotes the association of the application onto the hardware resources. In parallel, elementary concepts in software and hardware can be deployed with user defined Intellectual Property (IPs) blocks.

The deployment phase of such a methodology is instrumental, since enables the generation of a complete and synthesizable SoC description from a high-level description in MARTE. Designers must be able to precisely associate abstract components with their corresponding IP low-level implementations, while remaining at a high abstraction level. More

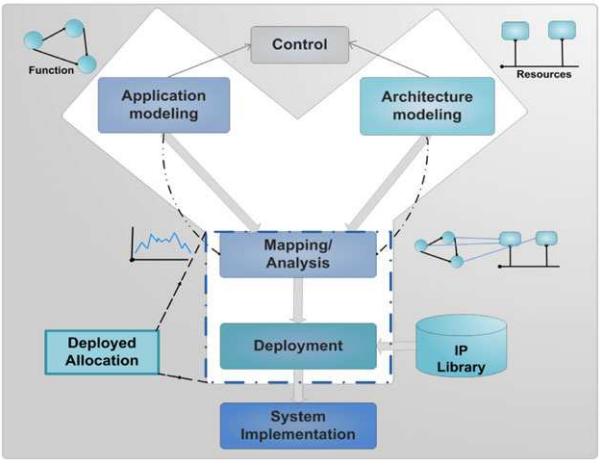


Fig. 1 Typical SoC Co-design methodology, the Deployed Allocation phase is the focus of this work

precisely, sufficient information must be provided at this stage so that the code integration and parameterization on the IPs can be carried out automatically at the time of the system generation.

The main disadvantage of recent efforts in applying MDE to SoC design has been precisely in the passage from the high-level models to the code generation. Some approaches have performed this mapping manually, whilst others have defined deployment meta-models to link both levels. However, these meta-models remain highly methodology dependant, and MDE for SoC approaches should aim at effectively promoting IP reuse, and the intermediate representation should be interchangeable. In this paper we propose an MDE approach that makes use of IP-XACT for promoting IP reuse; this is achieved by using a phase, the Deployed Allocation in which the allocation model in converted into a new model that contains technology specific information of the selected IPs. The proposed Deployed Allocation level offers a merge between simple allocation in MARTE and the deployment level of UML which defines the relationship between elementary components and their IPs.

In sections that follow, we discuss the related works in DPR SoC modeling using UML-MARTE, concentrating first in the efforts that do not employ IP-XACT.

2.1 Hardware resource modeling with UML profiles

Several works have tackled the use of UML/MARTE methodologies in SoC design, specifically at the deployment phase. An interesting approach has been carried out by the MoPCoM project [16] that aims to target modelling of FPGA based embedded systems using the MARTE profile [17]. In their approach, three models for the system are defined: functional, platform, and allocation. The allocation maps the behaviour onto the platform components, and then HW/SW partition is performed. The authors only present results in which they are capable of generating the microprocessor hardware specification file for input in Xilinx EDK tool. The authors further refine their approach in [18] to add IP-reuse capabilities, which was lacking in their original proposal. However, IP reuse information is annotated directly in the model, making the process difficult to automate. Furthermore, they don't take into account DPR concepts.

Several other works explore embedded system modelling using UML, but only a few explore dynamic and partial reconfiguration capabilities. In [20], authors detail a dynamic reconfigurable system by extending MARTE Profile with specific stereotypes. Their approach is developed in the GASPARD2 [19] design environment. The approach requires a strong level of expertise as all elements of the DPR design flow need to be modelled. Despite its complexity, in [21] the authors demonstrate how their methodology can be exploited to move from MARTE models to automatic code generation. They make use of the so-called Deployment Meta-model [22], which in fact provides several mechanisms to link low-level implementation (e.g. component interfaces, configuration parameters, hardware, and software implementation files) to the high-level models. However, authors do not detail the specificities to link the MARTE model to this level nor describe how the meta-model is exploited to move to lower levels of abstraction. Moreover, the proposed meta-model, as in other approaches, it is too methodology dependant, lacking sufficient generalization capabilities to be applied to a variety of design flows and to permit tool interoperability.

Despite the limitations of both approaches, we made use of certain ideas presented in these works regarding the modeling of the IPs. However, the IP blocks in our approach have an IP-XACT representation, simplifying their linking to the MARTE component models (and thus, their deployment); having this representation facilitates the parameterization and customization of the components in an automatic manner, which is not the case with previous approaches.

2.2 Hardware modeling using UML and IP-XACT

The interoperability issues discussed previously have been addressed by the academia in recent years, with ongoing efforts looking to integrate the IP-XACT specification in MDE-based methodologies. The standard describes a set of XML schemas used to document IP meta-data for SoC integration, in a structured manner. Several industrial cases studies have demonstrated that its adoption facilitates the configuration, integration, and verification in multi-vendor SoC design flows. Furthermore, IP-XACT also provides ways to automate the design flows where different tools are used. It is meant to be used by IP providers and system integrators and all major EDA vendors as way to standardize the access to IP information.

The goal of the IP-XACT standard is to provide a standard XML abstraction of hardware components, whatever the language. Hence, these files can be seamlessly interchanged between EDA tools to favour IP-Reuse. The IP-XACT standard has generated enormous interest in the industrial and scientific communities as a means to overcome the complexity of system integration. Initial attempts at bridging the gap between the MARTE profiles with IP-XACT have been presented in [23, 24]. The authors aim to create an ad-hoc UML profile for IP-XACT by introducing stereotypes and concepts to the MARTE profile to represent IP-XACT objects. However, their approach is only sketched, without presenting implementation results.

In [25], the authors have investigated the application of the UML for modelling IP-XACT compatible component and system descriptions by mapping several IP-XACT concepts to corresponding UML concepts. They present an application targeting a Core Connect based system, but it is oriented to generation of SystemC Transaction-Level Model (TLM) without reporting the integration of RTL components. A similar approach can be found in [26], which maps the TUT UML profile for embedded systems design to an IP-XACT model. The resulting IP-XACT design flow using UML is also presented which allows automatic RTL component integration based on the proposed transformation rules. Subsequently, the authors further demonstrated their approach in [27], adding modelling concepts, to implement a complex MP-SoC.

Despite the relatively large numbers of proposals in the modeling of SoCs using UML MARTE in the one hand, and a combination of UML and IP-XACT in the other, so far there are not approaches that use a Meta model-based approach for DPR systems. We believe that the combination of MARTE Profile and IP-XACT can improve the applicability of the model-driven approaches to the design entry phase of DPR systems.

In this paper we propose an approach for integrating Xilinx EDK cores supporting Core Connect interfaces. We make use of IP-XACT as an intermediate format between the MARTE models and the files used by the Xilinx Platform Studio Tools (XPS). We transform these files into IP-XACT component descriptions that are subsequently converted into MARTE components, used in the Deployed Allocation phase of our approach, as presented in the Y schema introduced in Fig. 1. These transformations are done automatically, facilitating the task of the high-level designer and making readily available the information for parameterization and interconnection. The designer composes a hardware platform in MARTE, by instantiating this high-level components into a MARTE Composite Structure Diagram, that is subsequently transformed in an IP-XACT design description, which configures the underlying IP Xilinx IPs by transforming it into a proprietary XPS model used for implementing the system. In this way, we promote IP reuse in our approach, while keeping the IP descriptions decoupled from the used front and back-ends. We make use of the transformations proposed in [24] and [25] for obtaining the IP-XACT description, but we

model our IP-XACT components descriptions in such a way that facilitates the transformations from IP-XACT to MARTE, avoiding the use of complex stereotypes and complicating the work of the designer.

3 Motivation of the proposed methodology

3.1 Traditional Dynamic Partial Reconfiguration Design Flow

A brief discussion of the DPR Design Flow is provided in this section. For a more detailed description, the reader is directed to the Xilinx User's guide [8]. Partial Reconfiguration parts from the principle that only a small area of the FPGA can be modified at run-time. For this, the designer must define explicitly the areas of the FPGA that will be dynamically reconfigured (known as PRRs, for Partial Reconfigurable Regions); then, a set of modules are assigned to these physical partitions (known as PRMs, for PR Modules). These modules are subsequently converted into partial bitstreams which can be downloaded at run-time to map the desired functionalities into the destined partitions. A use case of the DPR flow is depicted in Fig. 2, where processor-based DPR design is used. A typical system includes a reconfiguration manager (e.g. a processor such as the MicroBlaze), a DDR controller (to access the configuration data stored in external memory), an ICAP controller (responsible for the reconfiguration process) and some reconfigurable modules.

The technology, as it was originally proposed by Xilinx, has been used to swap tasks that are mutually exclusive, often without regard of real-time constraints. The DPR flow is comprised of four phases: first, the design entry (1), in which the structural information of the system (described in an HDL language) is performed. Afterwards, netlists for the top level and the reconfigurable modules are obtained (2) and imported to PlanAhead, where the third phase takes place; in this stage, the reconfigurable areas are floorplanned and the system implementation is performed (3). The final phase corresponds to the PR management running in a FPGA platform, using the obtained partial bitstreams for performing the configuration at run-time (4).

The DPR Design Flow is based on a bottom-up synthesis approach; the methodology requires that netlists for each partition are generated independently. In parallel, the top module

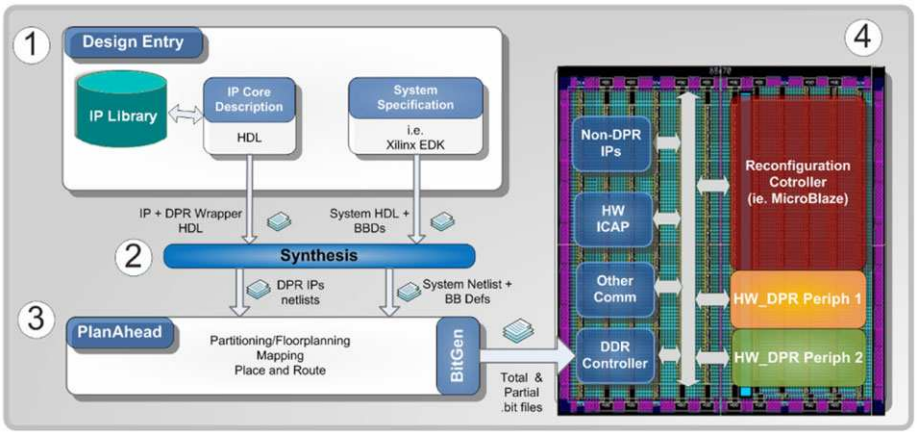


Fig. 2 Traditional design flow for the conception of dynamically reconfigurable systems

of the design is synthesized, containing black-boxes for the reconfigurable partitions. This implies that the IP blocks are obtained independently (i.e. library); these IP blocks may require being customized and parameterized before being synthesized, during the design entry phase. The same can be said about the top level integration, in which several parameters at the system level need to be set. Therefore, automating the IPs and system parameterization and their subsequent transformation into netlists can positively improve the DPR systems design cycle.

3.2 MDE and IP-XACT in the context of the proposed flow

We believe that the DPR design flow can be fully exploited by integrating it into an MDE approach. The presented work is part of the FAMOUS project which aims at integrating DPR concepts into the MARTE profile [28]. Several extensions are proposed, including features for control (for further reference the reader is directed to [29]), application modeling, and the definition of the DPR architecture, among others. The work presented in this paper deals with the generation of the DPR architecture from a high-level description in MARTE, as depicted in Fig. 3; in particular, we aim at facilitating the design entry phase of the DPR flow.

We concentrate our efforts in the so-called Deployed Allocation level, which provides IP reuse capabilities to our MARTE based approach by linking the abstract models to lower-level representations. The MARTE component models are obtained by model transformations from the IP-XACT component descriptions, but contain less information, which is enriched in the generation phase. The high-level model of the platform is parsed to generate an XML file that is used to gather information of the associated components from an intermediate component library (1). This library contains IP-XACT descriptions used for several purposes; the specification has become a de-facto standard for promoting IP reuse in the EDA industry, and we exploit its features and capabilities to generate different outputs used in the Xilinx design flow for SoC in EDK.

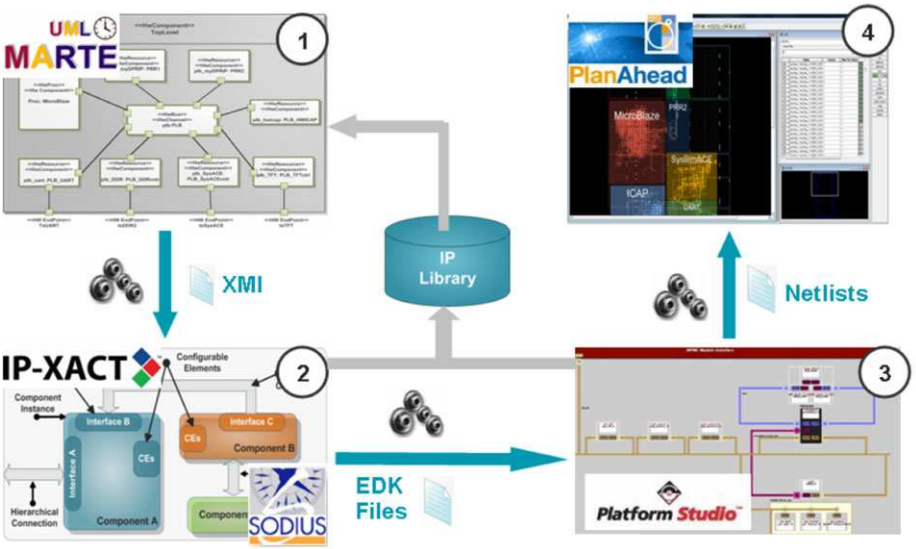


Fig. 3 Proposed design and tool flow

The IP-XACT standard requires the use the so-called Design Environment (DE) which is used for importing and to parameterise the IP blocks, and then to interconnect them to obtain a top-level system description. For this purpose, we make use of Sodius MDWorkbench [30], a tool that enables us to import the IP-XACT description of the top level design and of the IPs, for subsequent processing (2). The tool makes use of several meta-models and transformation chains in order to generate files used by the Xilinx EDK design flow for implementing the top-level SoC description of the system. We have chosen to target this flow since it facilitates the conception of soft processor-based systems and because we aim at modeling different aspects of the Xilinx DPR design chain.

The configured IP-XACT description is used to generate several files utilized by the Xilinx Embedded Development Kit [9, 10] (EDK) tool. Examples of these files are the Microprocessor Hardware Specification (MHS) and the Microprocessor Peripheral Description (MPD). The MHS and MPD files are employed by the Platgen tool [11] to generate the SoC platform. This tool generates the top-level HDL description, whilst the HDL files for the reconfigurable modules are gathered from an independent library. Finally, the top level design and each of the reconfigurable modules is synthesized separately (3). These netlists are used as inputs to the Xilinx PlanAhead tool [12], where the physical implementation of the DPR system is performed (4).

4 IP-XACT concepts used in our methodology

The IP-XACT standard defines an XML-based data for IP and system description. It defines four central object descriptions, which are bus definition, abstract definition, component, and design descriptions. These four elements are sufficient for structurally describing a system and the IP cores the compose it. As mentioned before, the main goal of the work presented in this paper is the generation of the inputs for the Xilinx DPR design flow, which comprises the structural information of the top level design and of the IPs to be used in the platform.

The goal of the section is to give the reader enough information to understand how IP-XACT is embedded in our MDE methodology and how determines the transformations from the deployed models, and to the Xilinx EDK back-end. It must be noted that the standard is intended for describing systems in a standardized way, but punctual vendor extensions and the way they are used in a design flow or tool flow, are methodology dependant.

4.1 Component description

A component description packages the information related to an IP core, as depicted in Fig. 4. We have chosen this block-like representation of the IP-XACT concepts instead of the schemas in the standard, since it facilitates their comprehension. Here, we have included the most widely used concepts for structural representation, logical implementation and parameterization. The component descriptions make use of ⟨*businterfaces*⟩ for interconnecting the parts to other elements in a design description; the bus interfaces make use of other two IP-XACT objects: the bus and abstract definitions, used to describe a bus protocol and how implements an interface logically-wise. In our methodology, we have extended the ⟨*busInterfaces*⟩ descriptions using vendor extensions in order to distinguish between different kinds of interfaces (not only bus based interfaces). ⟨*Parameters*⟩ are used for configuring the IP underlying implementations, but also for specifying flow dependant meta-data; the same can be said about ⟨*choices*⟩ that define parameters as enumerated lists of predefined values. We have introduced ⟨*vendor extensions*⟩ into chosen parameters, bus interfaces and

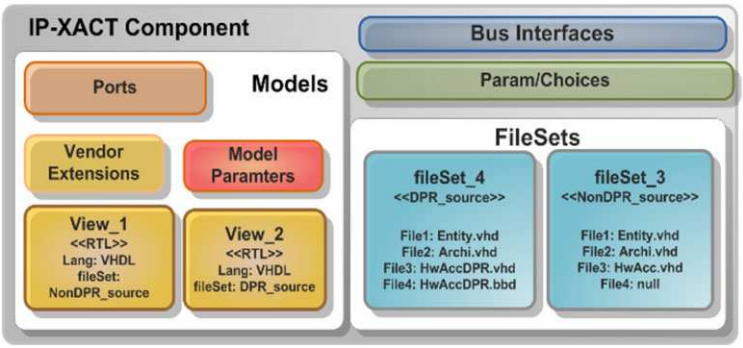


Fig. 4 IP-XACT concepts for a component description

ports to support their controlled inclusion into the generation phase. This is important for IP parameterization and customization and not considered in the current IP-XACT specification.

The ⟨*model*⟩ element describes the views, ports and model-related parameters of a component. An IP can contain different views such as RTL, TLM and software, to name a few. Views are used in tandem with ⟨*filesSets*⟩ and generators information to enable the automation of component related tasks (such as FPGA synthesis, driver/source code compilation). The fileSets and views elements of an IP-XACT component are very closely related, since a given implementation (⟨*view*⟩ in IP-XACT jargon) references to a specific ⟨*fileSet*⟩.

In our methodology, we exploit this capability of the view elements for describing components with different purposes but having the same interface. As mentioned in Sect. 3, the Xilinx DPR design flow requires as inputs, the netlists of the top-level design, but also of the individual reconfigurable modules. In the latter case, the reconfigurable modules functionality must be synthesized independently, while maintaining the same interface in the top level implementation. Thus, we implement the DPR IP in such a way that the DPR RTL ⟨*view*⟩ can be selected as a parameters in MARTE, impacting which parameters are used for configuration (hence the importance of controlling the inclusion of parameters, bus interfaces and ports depending on other parameters in the component description); likewise, the ⟨*view*⟩ element points to a ⟨*FileSet*⟩ that specifies which set of files has to be synthesized for the IP implementation.

4.2 Design descriptions

An IP-XACT design object describes an actual top level implementation as a set of component instances, which can be configured through configurable elements. The sub-elements in a design are connected between bus interfaces (that conform to predefined bus definitions). There are three kinds of connections, named interconnections in IP-XACT: interconnections, ad-hoc and hierarchical connections.

While an IP-XACT design, as depicted in Fig. 5, with referenced components and interconnections, describes most of the information for a design, some information is still missing, such as the exact port names used by a bus interface. To resolve this, a component description (referred to as a *hierarchical component*) is used. This *component description* contains a view with a reference to the design description. Together, the component and referenced design description form a complete single-level hierarchical description.

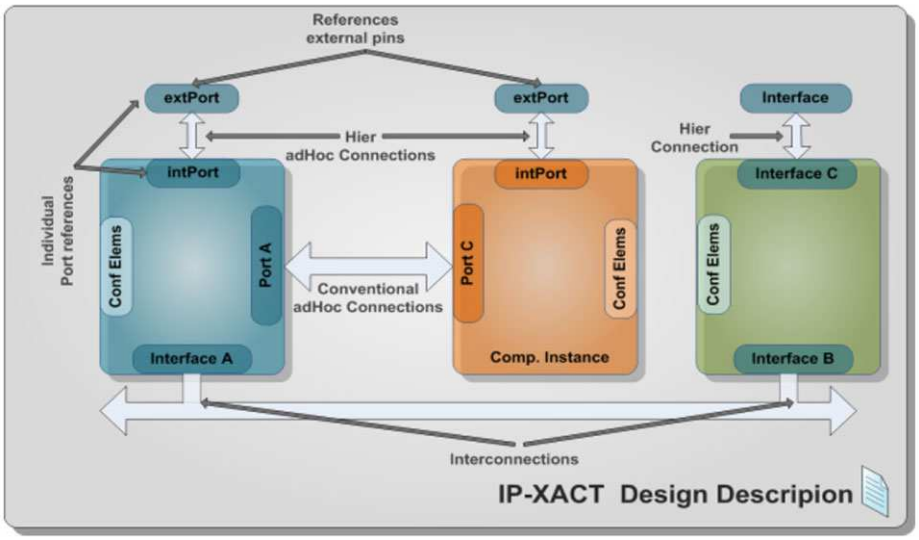


Fig. 5 IP-XACT concepts for a system description

We make use of design hierarchical descriptions as a means of describing the top level architecture in a flow agnostic way. The IP-XACT design description contains most of the information required to generate systems described in languages such as VHDL, Verilog or even SystemC. The descriptions are tailored by adding vendor extensions or flow dependant configurableElements; IP-XACT defines the concepts of generators and generator chains for accessing the meta-data contained in these descriptions. They are used configure the IPs in the component library, to generate drivers or customize components. This task is carried out by Sodius MDWorkbench, in which we import the IP-XACT descriptions of the top-level to generate the Xilinx MHS file, effectively decoupling the intermediate representation (IR) from the intended front and back-ends. Therefore, we can envision scenarios in which the departing model is not described in MARTE, but in AADL, to cite and example. The back end can also be customized by choosing a different view in the components description.

5 Proposed meta-model driven DPR Design Flow

This section explains in more detail the ideas discussed in Sect. 3. Here, we embark in a thorough description of our approach and how it is embedded into the design flow of DPR embedded systems. The proposed MDE methodology, in terms of models and model transformations, is presented in Fig. 6. We make use four levels of abstraction, each making use of its corresponding component library. The entry point is a MARTE Deployed Allocation model (a Composite Structure Diagram, CSD), which is created by choosing components from a MARTE Model Library (MML). The MARTE model is to be obtained after the association phase, where sufficient information about the components to use is available. At this phase, components are seen as simple IP blocks containing interfaces to be connected and parameters to be set by the designer. We have created an extension to MARTE for defining the characteristics of the deployed IPs that allow us to link them to their IP-XACT counterpart (i.e. a VLNV stereotype) and to obtain the information of the IPs automatically.

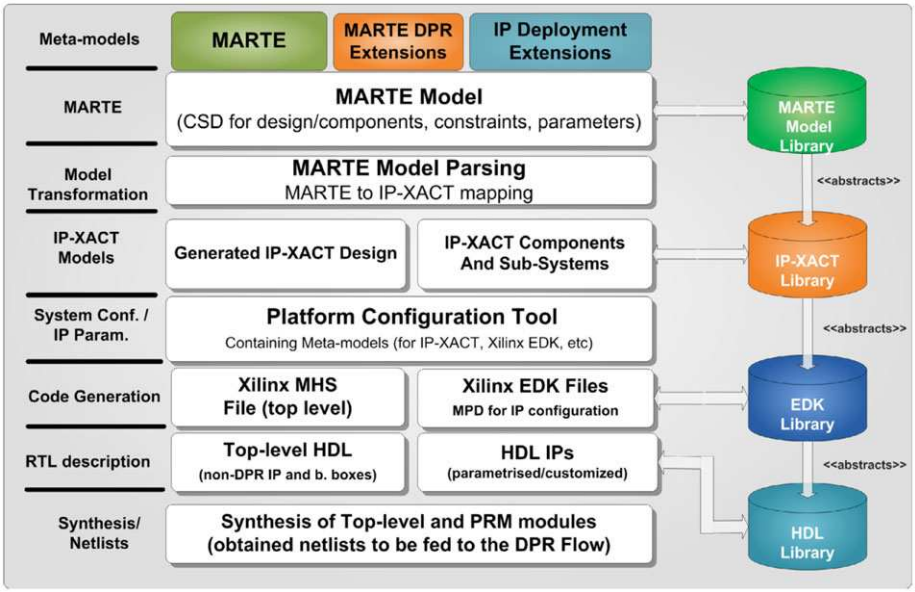


Fig. 6 DPR system integration in terms of model transformations

After having deployed the IPs to compose the platform, the system designer can create a system description by using two views, the "*Parameterisation View*" and a "*Platform View*"; these views contain a set of component instances to be parameterised and a CSD for interconnecting the different IPs in the platform, respectively. The parameters and interfaces for the components in these models are obtained from the IP-XACT library. Both views are parsed in the MARTE model parsing phase to obtain an IP-XACT system description, using of a simplified version of the model transformations proposed in previous works. The components in the IP-XACT library are parsed to gather the valid parameters and interfaces used to create the complete IP-XACT design description; this description contains the chosen component instances, bus interfaces, configurable elements, the connections between the component instances and the hierarchical connections to external signals.

The IP-XACT design is imported to our chosen "*design environment*", Sodius MDWorkbench, in which the model transformations from IP-XACT to the Xilinx EDK take place. The purpose of this tool is to generate several files used by EDK configure the hardware and software component of a SoC platform. The configuration of the components is performed through the creation of a Microprocessor Hardware Specification (MHS) file, which contains a set of component instances and their parameters. We have defined the transformations from IP-XACT to this proprietary format, which is used by the Xilinx tool to obtain the top-level HDL description, and the references to the enclosing HDL IPs, that are configured in this phase. The obtained HDL code is then synthesized to obtain the netlists used as input to the DPR design flow, as detailed in Sect. 3.1.

5.1 Targeted generation back-end: Xilinx platform studio

Xilinx EDK makes use of a series of files defined in the Platform Specification Format (PSF) document [11], which formalizes the description of different components in the Xilinx

design flow for processor-based systems. These files are used as an abstraction of the IPs implementations, and as a means to configure the IPs used in the platform via a design description. The VHDL description of an IP contains only information about the in/out ports, and in the best case, generics allowing the designer to parameterize and customize it. If the VHDL implementation had to be associated with a high-level description (typically containing parameters and bus interfaces), there will not be an easy and automatic way to determine which ports of the IP belong to a bus interface, and to use additional information important for the design flow.

Xilinx solves the aforementioned problems by providing an intermediate representation layer, the Microprocessor Peripheral Description (MPD) file, as depicted in Fig. 7(a), which shows a section of such a file. The MPD file contains basic information of underlying IP VHDL/Verilog implementation (generics, ports), adding flow dependent attributes, used for configuration. The ports can be bundle together using the concept of "bus interface", allowing the designer to customize the use of certain interfaces by setting attributes such as DataType, isValid, Permit, etc. Similarly, parameters and options can be made dependant on other parameters, and attached to specific groups (e.g. parameters that affect certain interfaces but not others, parameters that are only used when another feature have been chosen by the user). An important aspect of the MPD file is that allows adding information about the IP that is tool/technology specific, which facilitates the configuration of the IP in different scenarios, customizing their behaviour.

The IP implementations abstracted by the MPD files need to be parameterised at a higher level; this is done through the components instantiation in the Microprocessor Hardware Specification (MHS) file. As shown in Fig. 7(b), Platgen reads a Microprocessor Hardware Specification (MHS) file as its primary design input. Platgen also reads various hardware Microprocessor Peripheral Description (MPD) files from the EDK library and any user IP repository referenced in the MHS file. Platgen produces the top-level HDL design file for the embedded system that stitches together all the instances of parameterized IP blocks contained in the system. In the process, it resolves all the high-level bus connections in the MHS into the actual signals required to interconnect the processors, peripherals and on-chip memories. It also invokes the XST (Xilinx Synthesis Technology) compiler to synthesize each of the instantiated IPs.

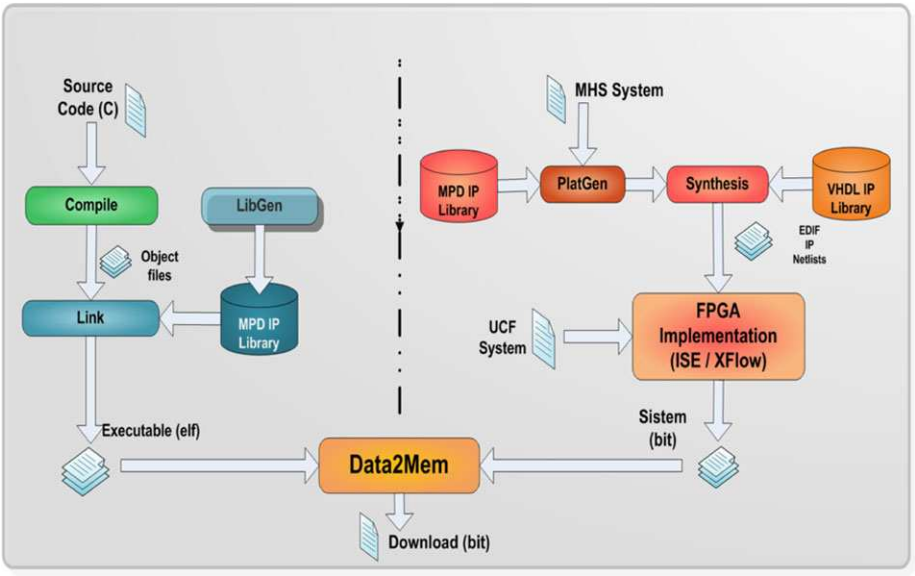
The EDK intermediate description, based in the MHS and MPD file (among others), represents an improvement over a purely VHDL description, since the textual representation has a formal semantic. Therefore, in our methodology there is an interest in being able to integrate these models for platform generation; for this, we have proposed several meta-model of the Xilinx PSF files. However, an MDE methodology should be platform independent before the Deployed Allocation model; it is at this phase where back-end and technology dependant information is added to the platform components. Linking the components directly to the Xilinx PSF descriptions would tie our methodology to the back-end, making it difficult to adapt to changes in the industry or to adapt it to other vendors and flows. Therefore, we use IP-XACT, as described in Sect. 4, with vendor extensions to support specific attributes to generate the EDK files, and to support features such as customization in parameters, bus interfaces and ports, which are not supported in the current IP-XACT specification.

5.2 Proposed meta-models for the Xilinx PSF files

Xilinx PSF files are structured a textual format, which can easily be understandable by machines by defining a parser, but the first mandatory step is the meta-model definition. The

```
62  ## Bus Interfaces
63  BUS_INTERFACE BUS = SPLB, BUS_TYPE = SLAVE, BUS_STD = PLBV46
64
65  ## Generics for VHDL or Parameters for Verilog
66  PARAMETER C_BASEADDR = 0xffffffff, DT = std_logic_vector, BUS = SPLB,
67  PARAMETER C_HIGHADDR = 0x00000000, DT = std_logic_vector, BUS = SPLB,
68  PARAMETER C_MEM_WIDTH = 16, DT = INTEGER, RANGE = (8,16), PERMIT = BAS
69  PARAMETER C_SPLB_AWIDTH = 32, DT = INTEGER, BUS = SPLB, ASSIGNMENT = C
70  PARAMETER C_SPLB_DWIDTH = 32, DT = INTEGER, BUS = SPLB
71  PARAMETER C_SPLB_P2P = 0, DT = INTEGER, BUS = SPLB
72  PARAMETER C_SPLB_MID_WIDTH = 3, DT = INTEGER, BUS = SPLB
73  PARAMETER C_SPLB_NUM_MASTERS = 8, DT = INTEGER, BUS = SPLB
74  PARAMETER C_SPLB_NATIVE_DWIDTH = 32, DT = INTEGER, BUS = SPLB, ASSIGNM
75  PARAMETER C_SPLB_SUPPORT_BURSTS = 0, DT = INTEGER, BUS = SPLB, ASSIGNM
76  PARAMETER C_FAMILY = virtex5, DT = STRING
77
78  ## Ports
79  PORT SPLB_Clk = "", DIR = I, SIGIS = Clk, BUS = SPLB
80  PORT SPLB_Rst = SPLB_Rst, DIR = I, SIGIS = Rst, BUS = SPLB
81  PORT PLB_ABus = PLB_ABus, DIR = I, VEC = [0:31], BUS = SPLB
82  PORT PLB_UABus = PLB_UABus, DIR = I, VEC = [0:31], BUS = SPLB
83  PORT PLB_PAValid = PLB_PAValid, DIR = I, BUS = SPLB
84  PORT PLB_SAValid = PLB_SAValid, DIR = I, BUS = SPLB
85  PORT PLB_rdPrim = PLB_rdPrim, DIR = I, BUS = SPLB
86  PORT PLB_wrPrim = PLB_wrPrim, DIR = I, BUS = SPLB
87  PORT PLB_masterID = PLB_masterID, DIR = I, VEC = [0:(C_SPLB_MID_WIDTH-
88  PORT PLB_abort = PLB_abort, DIR = I, BUS = SPLB
89  PORT PLB_busLock = PLB_busLock, DIR = I, BUS = SPLB
```

(a)



(b)

Fig. 7 (**a**) Snapshot of an MPD file for IP description. (**b**) Xilinx EDK flow for processor-based design

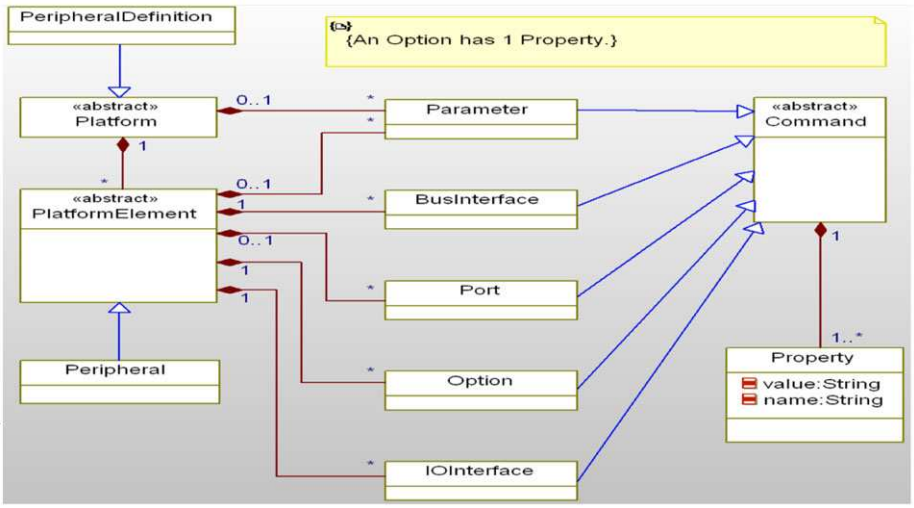


Fig. 8 UML Model of Xilinx PSF meta-model—microprocessor peripheral definition

Ecore formalization of these meta-models does not exist by nature, and has to be entered, in UML for instance. We have created meta-models for the different Xilinx files used in EDK, such as the MHS and the MPD, among others. For a more detailed explanation of the use of these files, the reader is directed to the PSF guide [11]. The import into MDWorkbench of the complete UML model of the Xilinx PSF meta-model translates it into an Ecore description, and then produces a Java/EMF implementation, which can then be used to perform the model transformations.

5.2.1 Microprocessor Peripheral Definition file

Figure 8 shows the UML model for the MPD file. As it can be observed, a peripheral (IP core in Xilinx jargon) is defined as a platform element, that contains a set of attributes. Most of the attributes in the MPD file can be mapped directly to concepts in the IP-XACT component description; specifically, the parameters and ports are concerted in ⟨*ModelParameters*⟩ and ⟨*Ports*⟩ in the ⟨*Model*⟩ element, which describes the implementation specific details of the component. The bus interfaces and options are mapped to the corresponding concepts in IP-XACT (⟨*choices*⟩ in the latter case). More details will be provided in Sect. 6 when discussing the model transformations from IP-XACT to the EDK formalism.

5.2.2 Microprocessor hardware specification file

This file is used by the Xilinx tool to create the top-level description of the hardware platform, as shown in Fig. 9. The MHS description contains the same elements of the MPD file, with one difference: only the parameters, bus interfaces and ports that are necessary for the top-level description of the IP are displayed. This is achieved by parsing the MPD files and checking for valid parameters (controlled by those in the MHS file).

The MHS file is created from an IP-XACT design description, which contains as well component instances, parameters associated with them (named configurableElements in IP-XACT jargon). The components in EDK are associated to the implementation files using the

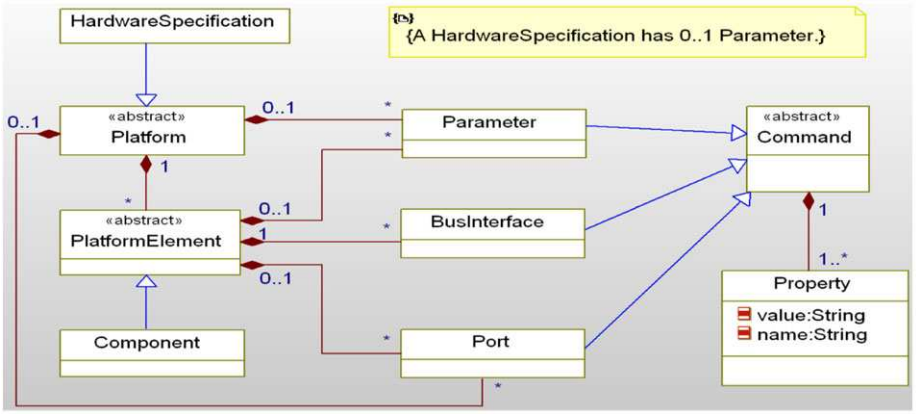


Fig. 9 UML Model of Xilinx PSF meta-model—hardware specification

instance name and hardware version values. IP-XACT provides a mechanism, the VLNV value, that contains this information, and that links the components from the MARTE description to the IP-XACT components and its EDK/VHDL counterparts. The bus interfaces are inferred from the ⟨busRef⟩ tags associated with the bus interconnections between the component instances. Regarding the individual, top-level ports (typically used for ad-hoc connections between components or to external FPGA pins) their tags are retrieved from the ah-hoc connections in the IP-XACT description.

5.3 MDE development tool and IP-XACT as intermediate meta-model

The MDWorkbench model-driven platform has been specifically designed to support the creation of meta-models from various formats, and includes several post-processing tools to improve the Ecore formalization of non-ecore meta-models. Along with meta-model definition (OMG's M2 level) for a given tool, language or standard, it is of primary importance to complete with the design of reader/writer connectors (OMG's M1 level), enabling to support data transfers from/to the platform. In some cases, this connector is based on a native XML/XMI file format. But, if the original format is a textual grammar, it may require the creation of a parser which can on-the-fly instantiate a syntactic model from any file compliant to a given grammar. This is the case of the meta-models for the Xilinx PSF presented in the previous sections, and that have been developed in our chosen MDE tool. Once the meta-models and connectors are operational for each source and target domains/tools/language, it can be used for further model transformation (m2m) or text generation from models (m2t). We make use of the Sodius MDWorkbench as a means of developing meta-models specifications for the different models in our design chain. Furthermore, the tool also provides means to describe transformation rules and to perform model transformations; this implies the use of the tool as a backbone for federating the heterogeneous data manipulated in our design flow.

As discussed in Sect. 2, we use IP-XACT as a means to share the same information between all the actors, using a common way to describe this information, and to automate the generation of multiple formats depending on the task needs and to perform checks between steps. We propose as well IP reuse by providing IP descriptions that remain interchangeable regardless of the added vendor extensions. The IP-XACT specifications provide a set

of XML schemas (.xsd) for representing different concepts in SoC design. This set of XML schemas has been processed by the improved XSD/Ecore meta-model importer in MDWorkbench, which leads to a Java/EMF implementation of the IP-XACT meta-model. Figure 10 shows a snapshot of the component meta-model in MDWorkbench. This meta-model is used to perform a series of model transformations from IP-XACT to different targets, as it will be detailed in Sect. 6, but also to be able to promote IP reuse by converting IP-XACT component descriptions into MARTE template models, as will be described in the next section.

5.4 MARTE extensions for IP deployment

In order to promote IP reuse in our approach, we have introduced the Deploy package as an extension to the MARTE Profile. At the Deployed Allocation level, we present two kinds of UML diagram. First, we have to identify what can be deployed the logical view—behavior or structure, and what can serve as a target of a deployment, the physical view—a resource or a service. The stereotype ⟨⟨deployed⟩⟩ is used for this matter in the "Deployed Component diagram". This stereotype belongs to the extended Deploy package in MARTE. Each element component corresponds to an IP implementation in the component library (i.e. IP-XACT component library). In order to enable this, we define a new stereotype labelled as ⟨⟨IP⟩⟩, which makes use of several resources from the *Generic Resource Modeling (GRM)* package in MARTE.

This stereotype is applied on classes in the Diagram Class of IPs, which is the second kind of UML diagram used at this level. Each elementary component is deployed and corresponds to an IP implementation in the component library (i.e. IP-XACT component library in our approach). The ⟨⟨IP⟩⟩ stereotype contains a set of attributes used to describe basic information of the IP at a high-level of abstraction, as depicted in Fig. 11. We have decided to keep these attributes to a minimum, since the designer at higher levels of abstraction does not need to know all the parameters of the IP. We have defined two ⟨⟨*DataType*⟩⟩ to provide a means to deploy the IP; these data types are defined in more details as follows.

We show mainly the elements necessary for the "*Parameterization View*". The IP_Kind enumeration is used to identify the type of implementation of the IP core. This provides a mechanism to identify which parameters should be used in the flow, since the kind of implementation determines their configuration. In this work, we assume that all the IPs are implemented as HW components (hence, the IP language attribute should be VHDL or Verilog); this information is obtained from the IP-XACT component description, particularly from the ⟨*View*⟩ element). However, this can be extended for software implementation functionalities.

The Id element types "identifier" which is a ⟨⟨*DataType*⟩⟩ in MARTE and contains a set of attributes to link the high-level descriptions to their IP-XACT counterparts. This type provides a means to unequivocally identify a component in the library by defining the *VLNV (Version, Library, Name, and Version)* tuple used in the IP-XACT standard to name the components descriptions. For Xilinx EDK IPs, the Name and Version values are obtained from the MPD file (Name and HW_Ver). We have decided to use a FilePath attribute for cases in which the designer wants to point the location of the implementations files; this information can be retrieved from ⟨*FileSet*⟩ element in the IP-XACT component description, under the ⟨*Dependency*⟩ element.

Finally, the most important aspect of our approach for parameterization (and eventually customization) of the IP composing the hardware platform, is being able to import the most relevant parameters of the IP. We perform model transformations from the Xilinx MPD files to obtain our IP-XACT library; the components in the library, as described in Sect. 2.1,

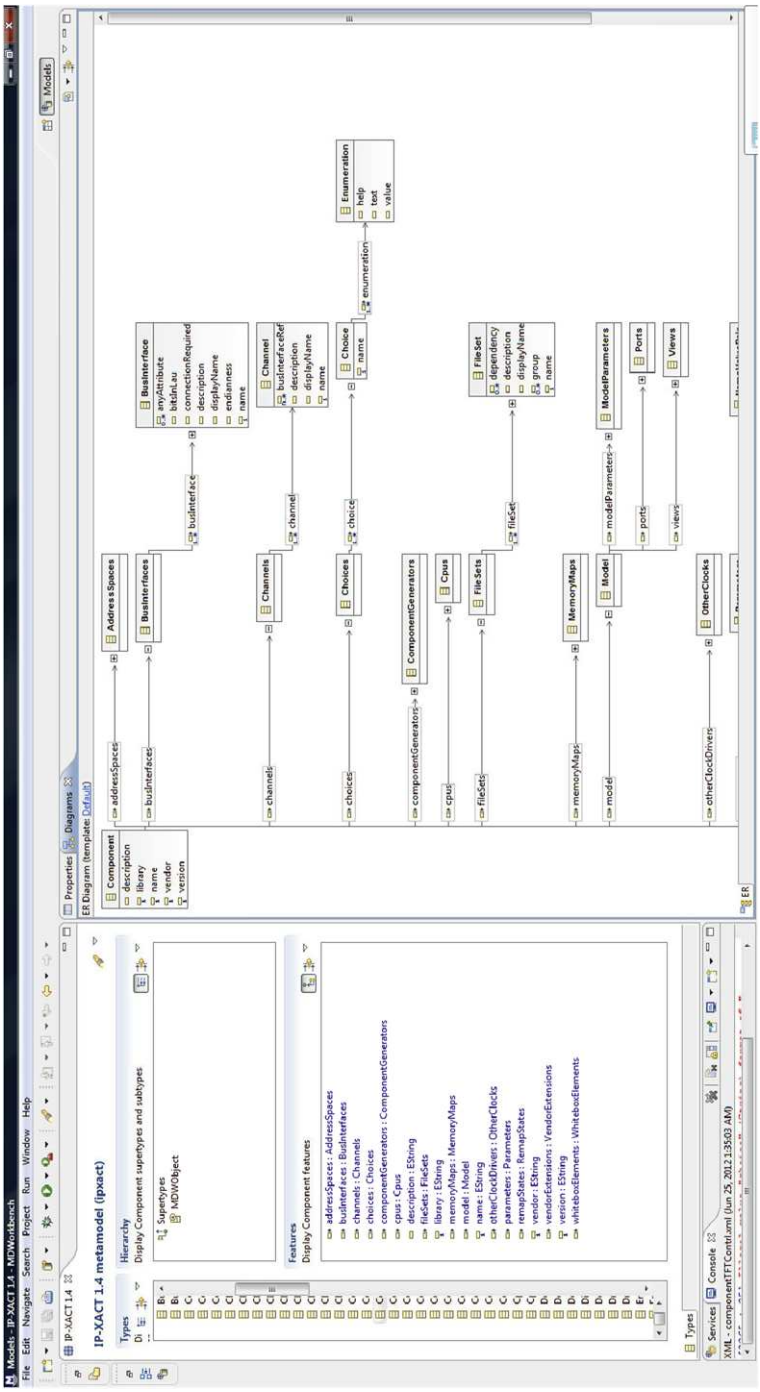


Fig. 10 Example of the IP-XACT meta-model in sodius MDWorkbench

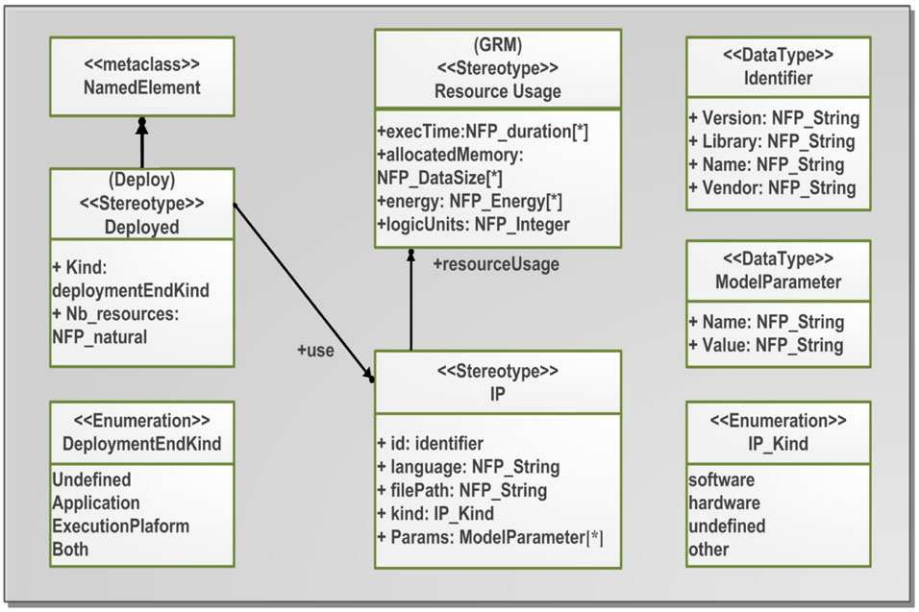


Fig. 11 Stereotype IP in the deployment package

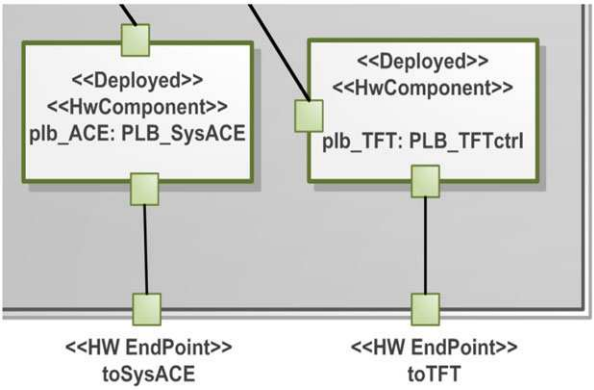
contain a set of elements described in the standard. The ⟨*Model*⟩ element contains, among other features, a description of the ⟨*ModelParameters*⟩ of the IP, typically implementation dependant information. We store the parameters information from the MPD file into the ⟨*ModelParameters*⟩ section of the component description; we have extended the definition of parameters with Xilinx specific attributes, through vendor extensions. These extensions will be described in more detail in Sect. 6 when discussing the model transformations from IP-XACT to the Xilinx generation models.

Figure 12(a) illustrates a snapshot of a deployed component view diagram in which we make use of stereotypes from MARTE HRM package (e.g. HwComponent) in order to describe the logical architecture. We also use ⟨⟨deployed⟩⟩ stereotype from Deploy package to match each component to its respective IP defined in a class diagram. This is the so-called "Platform View"; using a CSD, the designer is interested in describing the way the system is to be connected, not concerned to the low level aspects of the design.

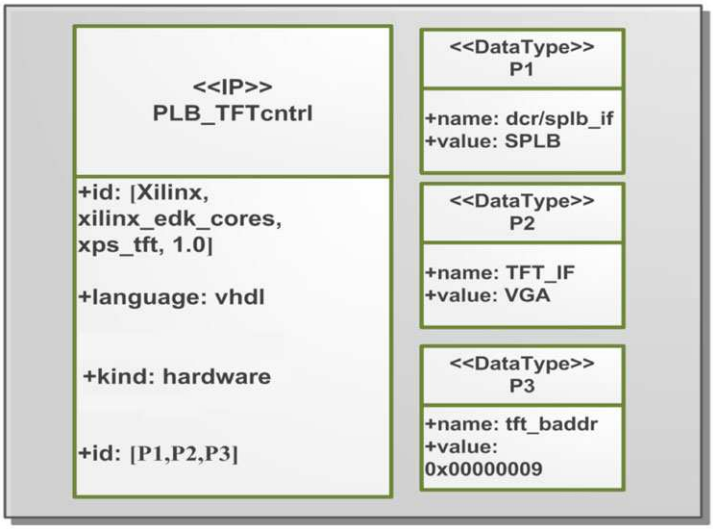
The MARTE extensions discussed in the previous section allow us to import the IP description to generate the views used for parameterisation and integration, as depicted in Fig. 12(b). We promote IP reuse in our approach by importing important parameters into a ⟨⟨*IP*⟩⟩ instance in MARTE; the creation of both views is done automatically by models transformations.

In order to accomplish this, we need first to define a transformation from MPD to an IP-XACT component description; the transformations are defined in the next section, but the basic principle is to categorize important features in the MPD model into meaningful groups in order to obtain only the required information for the high level models. For instance, parameters can be categorized as visible, visible when valid, optional and constant; these attributes are defined used the ⟨*configGroups*⟩ tag under ⟨*Views:Parameters*⟩ elements.

By separating the parameters into different groups, we can define which sets can be imported into the MARTE component description; only visible and visible when valid pa-



(a)



(b)

Fig. 12 (**a**) Snapshot of the MARTE architecture view. (**b**) Example of an IP instantiation with parameters

rameters are converted into MARTE component by an IP-XACT to MARTE transformation. The reason for this is that optional and constant parameters are typically part of the IP description, but not used in the parameterisation phase. Another aspect in the reuse of IPs is the customization of different components of the implementation; the VHDL components can be designed in such a way that code templates can be added or removed from in the synthesis phase by controlling parameters in the MPD description. Therefore, these parameters control the inclusion of other parameters, ports and bus interfaces into the final IP description; the visible when valid tag is thus applied to all this three groups when converting the MPD description into an IP-XACT description. When performing the conversion into MARTE models, the parser looks for the visible and visible when valid parameters, ports and bus interfaces and creates a MARTE model accordingly. However, dependencies on other parameters are not supported by the current specification of IP-XACT (v1.5.); the

standard specifies ways to control the values of certain parameters and choices dependant on equations involving other parameters, but not methods to control the inclusion or not of certain elements in the final generation phase. In order to support this, we have defined ⟨*vendor extensions*⟩ in the ⟨*parameter*⟩ elements, which are parsed to resolve this attribute.

The generation of the structural information of the components is more straightforward: bus interfaces and ports are converted to UML ports and named after the IP-XACT description; similarly as in the case of parameters, only those with the visible and visible when valid tags are used to generate the MARTE component. The components labelled as ⟨⟨*deployed*⟩⟩ in the deployed architecture diagram, are linked to the ⟨⟨*IP*⟩⟩ stereotypes in the class diagram of IPs. Each deployed component corresponds to an IP class and stored in the MML MARTE Library as a template that can be subsequently used as the building block of the "*Platform View*", as shown in Fig. 12(b). As mentioned before, both views are parsed and used to create an IP-XACT design description exploited for system generation.

5.5 MDE approach in terms of the used tools

In this section, we embark in a thorough description of our design methodology, concentrating more in the way the different tools are integrated, and the role of the models transformations. Figure 13 depicts the complete component based methodology for the generation of DPR platforms. The four libraries introduced at the beginning of the section are coupled with Sodius MDWorkbench; as explained before, the tool is used as a backbone for federating the heterogeneous data manipulated in our design flow. The first step is the creation of the IP-XACT component library from the EDK components descriptions, and subsequently the generation of the MARTE model library, by using the model transformations briefly introduced in the previous section, and that will be discussed in more detail in Sect. 6.

Once the IP libraries in different levels have been obtained, the designer of an application can create a top level description of the system in a modeling environment such as Papyrus (we are looking forward to support Rhapsody as well). The modeling environment contains the MARTE meta-model with different extensions to support partial reconfiguration modeling, but here we concentrate in the integration and parameterisation of static and DPR IPs. The deployed platform model created in Papyrus is exported as an XMI file that is parsed in Sodius MDWorkbench by using a model parser, which objective is to obtain an IP-XACT design description of the platform (along with a hierarchical component description for enclosing the design).

The IP-XACT design description contains already the component instances, their parameterised configurable elements and the interconnections between the components. This information is used, along the meta-models for the Xilinx PSF files (MHS and MPD) and the transformation rules, to generate a MHS file that is exported to Xilinx Platform Studio; the MHS file invokes the encompassing IPs in the MPD and HDL libraries for their parameterisation and integration. A top-level VHDL file is obtained by PlatGen, which is then synthesized.

Once the top-level and DPR IPs implementation have been synthesized, the obtained netlists are fed to the Xilinx PlanAhead tool [12] to generate the physical implementation of the DPR system, and subsequently the bitstreams necessary to configure the FPGA. It must be noted the some steps have not yet been completely automated; for instance, the methodology could be further improved by automating the synthesis process and by being able to run automatically generated scripts by IP-XACT generators.

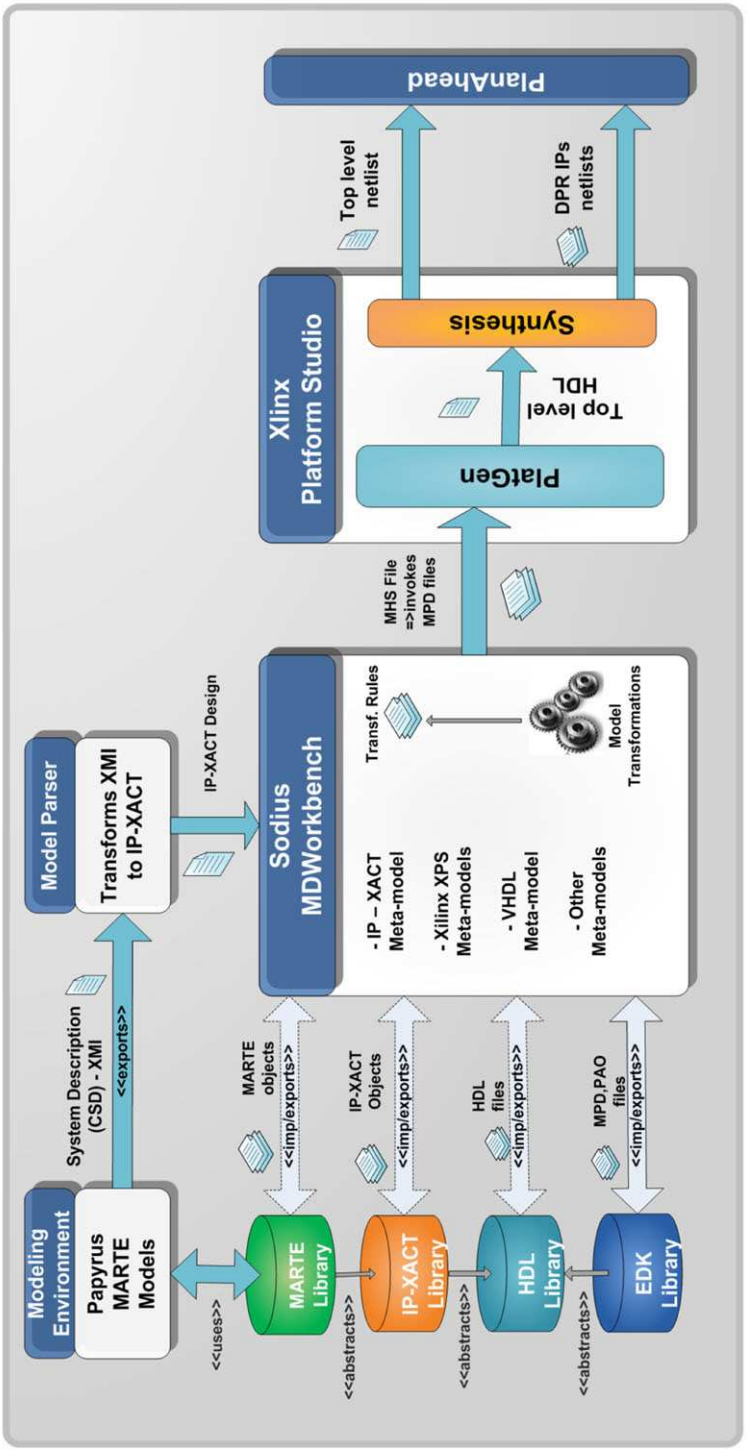


Fig. 13 Proposed MDE methodology and tool chain in-depth

6 Model transformations from the deployment level.

In this section we discuss in depth the transformations rules necessary to implement the complete MDE methodology proposed in Fig. 13. We discuss first the transformation from IP-XACT components to MARTE deployed component models; this is an important step, since we provide mechanisms for IP deployment absent in the MARTE profile. Then we discuss the transformations from the MARTE deployed platform description to an IP-XACT design description; the definition of the design and its eventual design build are the most important aspects in an IP-XACT based approach. Previous works have been proposed to carry out the transformation from MARTE to IP-XACT, but in comparison with those proposals, we make use a simplified component model in UML, while previous works contain a complete meta-model, an approach that we consider not suitable for most cases, since importing a complete IP-XACT component description in MARTE can render its manipulation extremely heavy. In this section we discuss in depth the transformations rules necessary to implement the complete MDE methodology. Afterwards, we discuss in detail the transformations from the IP-XACT description from and to the Xilinx EDK back-end. We show some of the proposed vendor extensions to achieve this task. Finally, we show the implementation details of the proposed transformation rules.

6.1 MARTE ⇔ IP-XACT transformation rules

As depicted in Fig. 13, the designer of a platform imports a set of components from the MARTE Library. This library is populated by transforming the IP-XACT component into MARTE deployment templates. We make use of the proposed a deployment IP meta-model that allow us to perform this mapping and to promote IP reuse in our methodology by linking the high-level components to their IP-XACT counterparts and creating a "Parameterization View" with predefined parameters that control the inclusion of ports, bus interfaces and other parameters in the final MHS model (and subsequently in the underlying VHDL IP implementations). Instead of having a complete IP-XACT meta-model in MARTE, our simplified IP meta-model allow us to have a reduced set of elements, which can be easily transformed in both directions. The mapping for this phase is detailed in Table 1(a).

Once the designer has parameterized and composed its platform, the design is parsed to produce an IP-XACT design description from the "Parameterization" and "Platform" views. The obtained .xml file contains a ⟨*spirit:design*⟩ entry, which identifies it as the top level element in a SoC design or hierarchical components. The MARTE CSD diagram contains a set of parts as in [24], including a custom data type, the VLNV id to link the high-level components to their IP-XACT counterparts. The transformation rules are shown in Table 1(b).

For each component in a MARTE CSD an ⟨*spirit:instanceName*⟩ value is created in the ⟨*spirit:componentInstances*⟩ section of the IP-XACT design file. The configurable elements for the components are inferred from the "*Parameterisation View*" and its values assigned from the description itself. The components in the MARTE platform diagram contain HW connectors, which are mapped to IP-XACT interconnections (⟨*spirit:interconnection*⟩). The external pin information is obtained from a combination of the enclosing IP-XACT component description and the ⟨*spirit:adHocConnections*⟩ which contains an external port reference.

6.2 MPD ⇔ IP-XACT transformation rules

In this sub-section will be further describe how the mapping between the Xilinx EDK files and their IP-XACT counterparts is performed. In Sect. 5 we described how the MPD files

Table 1 (**a**) Transformation rules for an IP-XACT component to a MARTE deployed component model. (**b**) Transformation rules from a MARTE CSD to an IP-XACT design

(a)

MARTE	IP-XACT
Class instance	Component
id: identifier (VLNV tuple)	spirit: version: library: name: vendor
Port = name port kind = busIF	spirit:busInterfaces:busInterface:name
Port = name port kind = ad-hoc	spirit:Model:ports:port:name
Parameter = name	(if ⟨immediate⟩ and ⟨confGroups⟩ = user) spirit:ModelParameters:Parameters
areaType = static or DPR	spirit:view: NODPR or DPR (choosing FileSet Implementation)

(b)

MARTE	IP-XACT
CSD diagram	Design
Part Name	spirit:ComponentInstance name
id: identifier	spitir:ComponentRef
Parameter = value	spirit:configurableElement:ReferenceID
Connector = name	spirit: interconnection portRef (busIF name)
Ednpoint = name	AdHoc Connection = name with intPortRef and extPortRef

are defined and the elements of the IP it describes; we also introduced the proposed metamodels. As mentioned previously, the MPD file contains all the parameters, ports and bus interfaces of an IP; however, the MHS file has precedence over the MPD file. This means that the values set at the top-level change which elements of the MPD/IP core will be implemented.

We start with the bus interfaces and IO interfaces of the IP. This concept exists in IP-XACT in the bus interfaces element; however, the nested parameters of the MPD file are not part of the standard. For this reason, we have decided to store these nested parameters as vendor extensions in each of the bus interfaces elements of a component description, as shown in Table 2(a). We have defined an attribute, INTERFACE_TYPE, to differentiate normal bus interfaces from IO interfaces in the IP-XACT description. Depending on the value of these parameters, the subsequent elements in the interfaces descriptions are used to create the MPD description, if this is the objective of the transformation phase. An important parameter for the bus interfaces description is the IS_VALID option, which controls if the interface is included or not in the design.

For the PORTS section of the MPD description, we have used the ports description in the Model element of the IP-XACT component description, as depicted in Table 2(b). As with the parameters, setting the configuration of the ports in this section, helps in keeping the component description independent of the intended flow. For VHDL modules, only the name, direction and size of the ports are required; this information is used for generating the entity to be connected in the top-level description. However, ports in a SoC flow are

Table 2 Transformation rules for IP-XACT to/from MPD for (**a**) bus interfaces, and (**b**) ports

(a)

MPD command	IP-XACT component counterpart
BUS_INTERFACE	spirit:busInterface if (parameter:interface_type) = bus_interface spirit:parameters:spirit:
INTERFACE_TYPE	parameter:Interface_Type
BUS	parameter:bus:value
BUS_STD	parameter:bus_std:value
BUS_TYPE	parameter:bus_type:value
GENERATE_BURSTS	parameter:generate_bursts:value
ISVALID	parameter: value = ((Depedency(param_id)) ? 0:1)
IO_INTERFACE	spirit:busInterface if (parameters:interface_type) = hier_interface spirit:parameters:spirit:
INTERFACE_TYPE	parameter:Interface_Type
IO_IF	parameter:IO_IF
IO_TYPE	parameter:IO_TYPE

(b)

MPD command	IP-XACT component counterpart
PORTS	model:Ports spirit:port:spirit
NAME	name
DIR	wire:spirit:direction
VEC	vector:left—vector:right
PORT PARAMS	spirit:port:spirit:vendorExtensions:spirit:Parameters
PORT_TYPE	parameter:portType (bus_signal, hier_signal, ah-hoc_signal) (bus_signal, hier_signal, ah-hoc_signal)
PORT_GROUP	parameter:portGroup
SIGNAL_TYPE	parameter:signalType (clk, interrupt, bus, io, three_st)
ISVALID	parameter:isValid if ((Depedency(param_id)) ? 0:1)
PERMIT	if (param:port_type = “hier_signal” and permit = user) PERMIT = base_user
BUS	if (portType = “bus_signal”) then parameter:bus_name
IO_IF	if (portType = “hier_signal” or “ad-hoc_signal”) then parameter:bundle_name

more complex. For instance, in Xilinx EDK, each of the ports in the MPD description has a set of nested parameters or attributes. These attributes depend on the type of port we are dealing with; we have defined three parameters to identify the type of port: PORT_TYPE (bus signal, external, internal), PORT_GROUP (to associate a port with a bus interface or IO interface), and SIGNAL_TYPE (clk, interrupt, bus, io, three_st). Depending on the values of these signals in the MPD file or in the design entry phase, the rest of the ⟨*parameters*⟩ in the ⟨*port*⟩ description will be created or parsed, depending on the scenario.

The last element of the MPD file to be mapped is the parameters, which are used for different purposes. IP-XACT provides the possibility to store parameters in different elements of the component description. Since we are targeting Xilinx EDK cores, we have decided

Table 3 Transformation rules for the component parameters

MPD command	IP-XACT component counterpart	Description
PARAMETERS	model:modelParameters	Describe parameters in EDK and its attributes
PARAM_i = VALUE_i ASSIGNMENT	modelParameter : value (constant, optional, require, update)	A parameter name with a tied value. The parameter can have attributes, as listed below Classifies parameters depending on configurability
DT	dataType	Determines data type of the parameter (e.g. string, int, std_logic)
ISVALID	vendorExtensions:parameters: isValid:((Depedency(param_id)) ? 0:1)	v_ext: if present in the description, the presence of the bus int in the MHS will depend on the value of param_id
PERMIT	isValid:((Depedency(param_id)) ? 0:1) and resolve = "user"	Determines is a parameter is configurable by the user
RANGE	value:minimum—value:maximum	If both values are present then RANGE = (MIN:MAX)
VALUES	if value:spirit:format = "choice" then VALUES = (enum1, . . . , enumn)	A list of enumerated values, to be retrieved from the referenced choice in the parameter description

to store these parameters in the ⟨*modelParameters*⟩ section; in this way, the rest of the IP description can be more generic and not tool dependant. Table 3 shows how the parameters have been mapped to IP-XACT. Table 3 shows these parameters and how certain among them depend on the values of the first three parameters.

6.3 MHS ⇔ IP-XACT transformation rules

The MHS file is created from an IP-XACT design description, which contains as well component instances, parameters associated with them (named ⟨*configurableElements*⟩ in IP-XACT jargon). The components in EDK are associated to the implementation files using the instance name and hardware version values. IP-XACT provided a mechanism, the VLNV value, that contains this information, and that links the components from the MARTE description to the IP-XACT component implementation and its EDK/VHDL counterparts. The bus interfaces are inferred from the ⟨*busRef*⟩ tags associated with the bus interconnection between the component instances. Regarding the individual, top-level ports (typically used for ad-hoc connections between components or to external FPGA pins), their values and tags are retrieved from the ah-hoc connections elements in the IP-XACT description. Table 4 shows the complete list of mapping between IP-XACT and MHS.

6.4 Summary and implementation details

In this sub-section we provide a summary of the model transformation presented before. The rules sets for all the transformation were specified using the Model Query Language (MQL). The transformations were implemented in Sodius Workbench using Java, Table 5

Table 4 IP-XACT to MHS mappings for the top-level hardware description

MHS command	IP-XACT design counterpart	Description
PARAMETERS	CONFIGURABLE ELEMENTS	Parameters in MHS are instance name, hw version,
INSTANCE	spirit:InstanceName	Name of the component instance
HW_VER	spirit:componentRef spirit:Version	Version of the IP, typically obtained from the MPD and IP-XACT descriptions for EDK cores
PARAM_i = VALUE_i	spirit:configurableElementValue spirit:referenceId = value	Different parameters of the IP. Only visible parameters appear in the MHS file
BUS INTERFACES	ACTIVE INTERFACES	Interfaces to bus (e.g. SPLB, MPLB) or other Xilinx
BUS INTERFACE i INTERFACE = bus_std	for each (interconnection) if (componentRef = "instanceName") activeInterface:busRef = interconnection:name	Each interconnection needs to be parsed to find all the IP's interfaces and their names
PORTS	INTERNAL PORT REFERENCES	
internal ports PORT_i = VALUE_i	for each (adHocConnection) if (componentRef = "instanceName") internalPortRef:portRef = adHocConnection:name	For each adhoc Conn, the parser must find the port reference and its name and update it in the MHS
external ports PORT_i = VALUE_i	for each (adHocConnection) if (componentRef = "instanceName") externalPortRef:portRef = adHocConnection:name	For ports with external port reference sin IP-XACT the information about direction and width is obtained from the hierarchical component

Table 5 Implemented model transformations from/to IP-XACT

Transformation	Nb of lines	Purpose
MPD to IP-XACT	113	Obtain IP-XACT library from Xilinx EDK
IP-XACT to MARTE	98	Obtain MARTE Library templates
IP-XACT to MHS	87	Generate top-level Xilinx EDK description
IP-XACT to MSS	47	Generate software drivers description of IPs

shows a summary of the implemented transformations, the number of code lines taken per each one and the purpose in the flow.

7 Case study

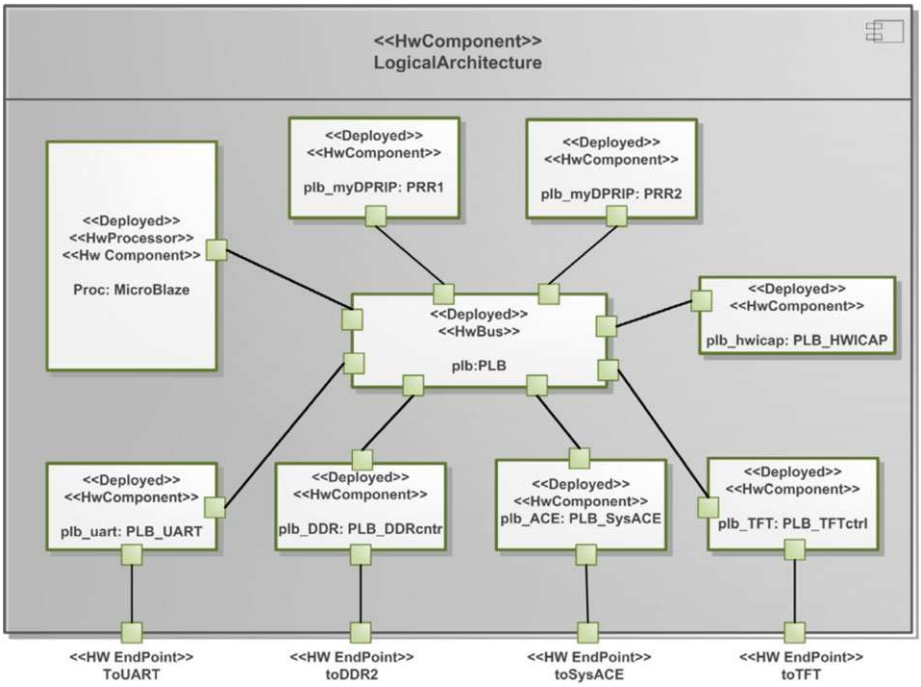
In this section, we present a case study in which we show how the methodology is used to implement a MicroBlaze-based SoC platform, integrating some DPR blocks. We start by describing MARTE related modeling concepts for the individual components and the platform. This description will make emphasis on how these MARTE concepts are related to their IP-XACT counterparts, which is one of the goals of this work. Similarly, we show how a DPR component can be parameterized and integrated from a MARTE description.

7.1 MARTE modeling of the top architecture.

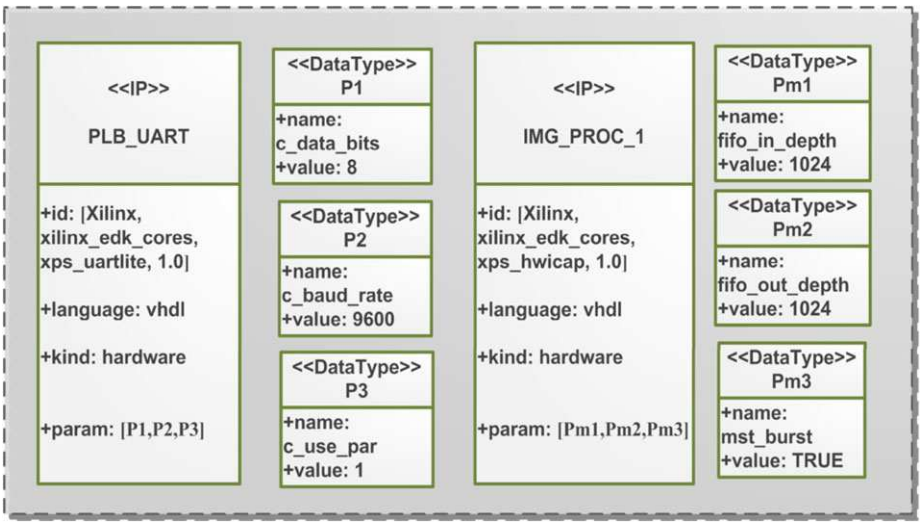
The Deployed Allocation level contains a set of views for executable models generation; in this paper we deal with the generation of the logical/architectural netlist of the top-level of a given design, and of the IP descriptions that compose it. As mentioned before, the objective is the generation of the structural information that is used as inputs for the DPR design flow. Figure 14(a) shows the modeling phase of a reconfigurable system, targeted to an embedded architecture to be exploited by the Xilinx's EDK environment. This diagram represents a deployed logical architecture where each component is labeled ⟨⟨HwComponent⟩⟩ and ⟨⟨deployed⟩⟩; the first one is used to define the component as hardware module from MARTE HRM package, while ⟨⟨deployed⟩⟩ stereotype belongs the Deploy package. This is the so-called "Platform View"; using a CSD, the designer is interested in describing the way the system is to be connected, not concerned to the low level aspects of the design. Every hardware component has two type definitions, one being functional (the type of modules) and the other physical. In the case of our case study, we make use of two dynamic reconfigurable regions (labeled as PRR in the diagram). In addition, we have made use of components such as the PLB_HWICAP (in charge of managing the partial reconfiguration data), the SystemACE controller (to store data and configuration bitstreams), the UART controller, and of course the PLB bus and the MicroBlaze processor.

The type of IP, static or reconfigurable, determines which ⟨*View*⟩ of component will be chosen for implementation in the flow; this aspect will be further detailed in the next subsection when discussing the modeling of the non-static IPs, which can contain multiple views referencing to different implementations of the IP, known as ⟨*fileSets*⟩.

The designer creates first what we call a "*Parameterization View*", as depicted in Fig. 14(b), which shows only a section of the view consisting of two components. Each component contains a specific MARTE stereotype; most of the components in this figure are ⟨⟨*HwComponent*⟩⟩, since they reference real IP cores. The components in the MML



(a)



(b)

Fig. 14 DPR system: (**a**) deployed architecture diagram. (**b**) Snapshot of the IP class diagram

library are related to their IP-XACT counterparts by their VLNV values (as mentioned before, using the id attribute). Each class in this diagram contains therefore a reference to the configurable elements that this component instance contains, which must be set by the

designer. This view has been separated, since it might become much cluttered to have all information in a single, architecture diagram. Therefore, as explained in the previous section, only those parameters in the component IP-XACT description whose resolve attributes are defined as ⟨*immediate*⟩ and that belong to the ⟨*configGroups:visible*⟩, are accessible in the "*Parameterization View*". Some parameters control which interfaces and ports are available in the "Platform View", as detailed previously, since components might have different technological configurations. Both descriptions are parsed for generating the IP-XACT for the top-level implementation.

7.2 Example of implementation and Integration of the DPR IPs

In order to demonstrate the feasibility of our methodology, we have implemented a series of IPs associated with the reconfigurable modules of the "Platform View". Since we are integrating these IP blocks into a Xilinx EDK based SoC, we need to provide a means to communicate with the other elements of the platform; this is done by using the CoreConnect PLB bus, as depicted in Fig. 15. The image processing IPs (or hardware accelerator, HWA) need to be wrapped by a Xilinx proprietary interface (IPIF) that. In most cases, the interface of the IP cannot be directly connected to the IPIF, and some sort or Protocol Adaptation Logic needs to be inserted between the two components. If each implementation of the HWA has a different interface, a different version of the PAL has to be used. Together, they comprise what we call the Dynamic Wrapper (DW, the component that declares a DPR partition in a static design, a necessary requirement of the Xilinx DPR flow). We facilitate the integration of DPR IPs into the proposed flow by, in the first place, providing MARTE front-end for their parameterisation and integration (using the associated VHDL and MPD files), as described in previous sections. The DPR IPs are stored in the library along fixed IPs, but the areaType attribute define them as dynamically reconfigurable; when defined as static, a default implementation is used instead.

The second feature of the DPR IP descriptions is that, by using the ⟨*views*⟩ and ⟨*fileSets*⟩ elements of the IP-XACT description we provide a means for pointing the location of the different implementations of the DW into the IP implementation directory. The ⟨*fileSets*⟩ element in the component description specifies all the files used to describe a component. In particular, a least one ⟨*fileSet*⟩ is destined for specifying the HDL sources using to implement the IP functionalities. A component can contain multiple implementations, each represented by a ⟨*View*⟩ referencing a ⟨*fileSet*⟩, as depicted in Fig. 15. The ⟨⟨DPR_Source⟩⟩ ⟨fileSet⟩ is parsed in the IP-XACT design generation phase, in order to retrieve the location of each implementation of the DPR IP. This information is subsequently used to synthesize each IP configuration separately, as required by the DPR design flow for the PRMs. Using the regular design flow, this task would required to be performed manually, with the additional burden of using a separate tool (Xilinx ISE). In Fig. 15 show a block representation of the implemented IPs; there are two of them in the platform of Fig. 14: each of them treats a half of a input image and sent to the TFT controller in the card for display, as shown in Fig. 16(b), and implemented several image processing tasks (binarization, inversion, edge detection, and greyscale reduction), as hardware accelerators (Image Proc 1 to 3).

7.3 Implementation results

In this section, we present the implementation results of the architecture presented in Fig. 14, using the IPs introduced in the previous section. We have created an script the reads the input MHS file in MDWorkbench, following the directives specified in [10] for automating

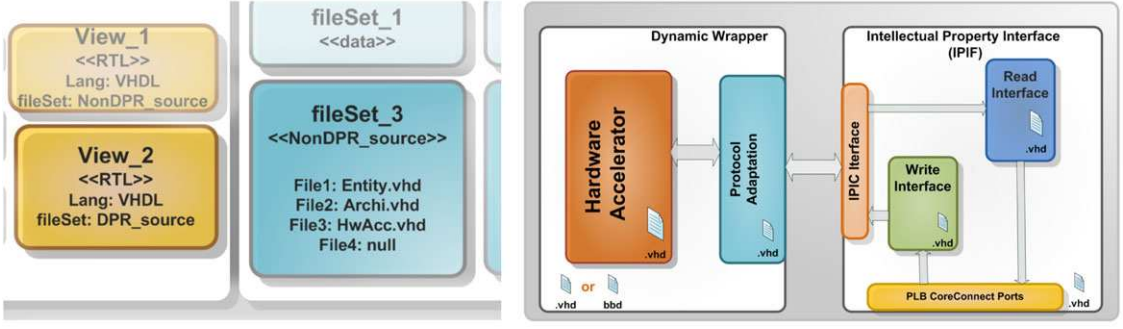
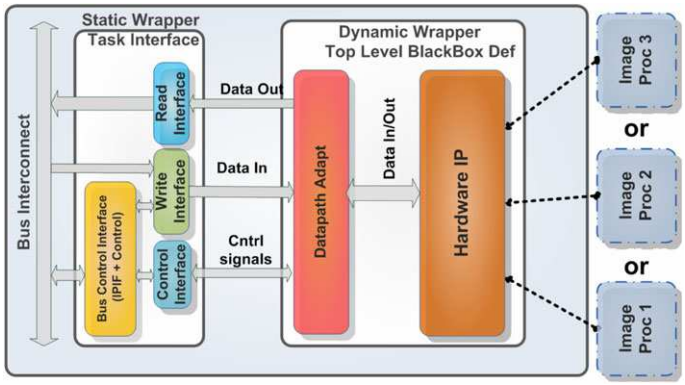
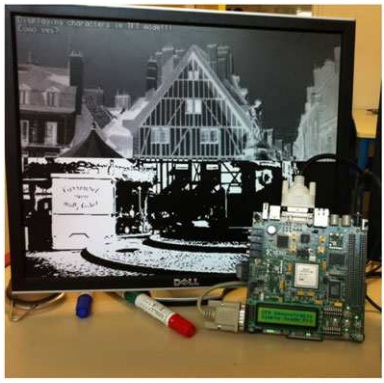


Fig. 15 IP-XACT abstract representation for the DPR IP



(a)



(b)

Fig. 16 (**a**) Architecture of DPR IPs used in the case study. (**b**) Implemented system running on board

the generation of the EDK platform; subsequently, the TCL script launches the synthesis process for the obtained VHDL file, obtained a system.ngc file, implementation the static component of the DPR design. Table 6(a) shows the synthesis results of the DPR architecture for a Xilinx Virtex 5 FPGA (LX50). The last row shows the total resources consumed by the system implementation; we have decided to show only the synthesis results of the most representative components. We can observe from that there are two components (IMG Proc IP1 and IP2) whose resource utilization is identical; they correspond to the static wrappers described in Fig. 16(b). As mentioned previously, we have defined two PRR regions in which we map different image processing algorithms.

In Table 6(b) we can observe the implementation results of two simple pixel-based operations. The resource utilization in the same; we provide as well the partial bitstream size (5 KB for each), which means that for using the throughput provided by the HWICAP we can attain a configuration time of 50 microseconds. In Table 7(c) we provide a more complex example, in which we have mapped a Discrete Cosine Transform (DCT) and the Discrete Wavelength Transform (DWT); as it can be observed, the increased resource utilization increases the partial bitstream sizes and accordingly, the configuration times. It must be noted that the configuration time for this FPGA would be of the order of seconds, to give some perspective of the gain provide by DPR.

Table 6 (**a**) Resources utilization for the presented platform. (**b**) and (**c**) Resources for the implemented DPR IPs, the memory footprints of the partial bitstreams and the associated reconfiguration time

(a) Resource utilization for different modules

Module	Resources utilization		
	LUT	FF	BRAM
Microblaze	1445	1518	0
BRAM Ctntrl	0	0	16
MB_PLB	182	854	0
RS-232	160	142	0
DDR2 Controller	3698	2680	5
SysACE Cntrl	217	103	0
TFT Cntrl	767	619	1
HWICAP Cntrl	520	635	0
IMG Proc IP 1	1008	847	0
IMG Proc IP 2	1008	847	0
System	9592	8688	22

(b) Reconfiguration metrics for Example 1

Example 1	Resources utilization					
DPR Module	LUT	FF	DSP	RAM	KB size	Conf time
IMG Proc 1	1008	847	0	0	5 KB	0.05 ms
IMG Proc 2	1008	847	0	0	5 KB	0.05 ms

(c) Reconfiguration metrics for Example 2

Example 2	Resources utilization					
DPR Module	LUT	FF	DSP	RAM	KB size	Conf time
DCT	1419	1636	8	8	47 KB	0.47 ms
DWT	940	389	0	4	44 KB	0.44 ms

Table 7 Lines and execution times per transformation

File type	Nb of lines	Exec. time	Description
XMI	654	50 s	MARTE model in XMI for transformation purposes into IP-XACT in MDE Workbench
IP-XACT Design XML	998	30 s	Intermediate model used in the transformation phase to obtain the MHS file
MHS File Top Level	455	20 s	This file is used by EDK to instantiating the IP blocks, parameterisation and interconnection
VHDL Top Level	7986	30 s	HDL description of the system. Obtained by feeding Xilinx Platgen with the MHS and MPD files
Netlist Top Level	N/A	12 min	The system VHDL top level file is used to obtain a NGC for the static part of the platform

Table 8 Use languages and tools

Used languages	Purpose
MQL	Ruleset definition
Java	Transformations
IP-XACT	Intermediate Model
VHDL	IP Implementations
Used tools	
Papyrus	Modelling
Rhapsody	EDK Meta-models
MDWorkbench	Model Transformations
IP-XACT Editor	Golden Rule Models
Xilinx EDK	DPR SoC
Xilinx ISE	IP development

Table 7 summarizes the required times to achieve each of the transformations, from the MARTE model conversion into a top level netlist, along the number of lines of each of the intermediate representations. It must be noted that the longest time corresponds to the synthesized top-level netlist that might change from system to system. Similarly, Table 8 summarizes the use languages and tools in our methodology.

7.4 Discussion on design effort and advantages of the proposed approach

In this section we elaborate on the design effort required to implement the system detailed in Fig. 14, especially if we compare it with a purely VHDL approach and, as in the case of the generation back end of this methodology, using Xilinx EDK. Let us consider for instance the obtained VHDL top level design, which is generated in around 30 seconds by PlatGen; the top-level VHDL description contains 7986 lines of code, and mainly contains components instantiation, parameterization and signals declarations for interconnection. It is evident that creating such a design (composed of 21 components, and multiple sub-components) would take not only hours, but maybe days, in a process very prone to errors, as depicted in Table 9.

Xilinx EDK, using the Platform Specification Format (PFS, notably MHS and MPD files), makes the design process more amenable: the designer can start creating a design through an easy to use Graphical User Interface (GUI), and then parameterize the design by choosing different options through IP specific TCL files and GUIs. These changes are automatically updated in the MHS files by parsing the corresponding MPD file and checking for any dependencies on parameters. However, the creation of the platform in Xilinx EDK is not completely automated, and a lot of steps still need to be performed manually; for instance, importing IPs into the platform, their interconnection and parameterization. All these steps required a great deal of design effort and expertise of the tool and this is precisely one of the advantages of used the proposed MDE methodology: by using a high-level description, the designer does not to know all the specificities of the used tools, which often are difficult to grasp by people who are not proficient into FPGA design and VHDL. The DPR aspects of the flow further complicate the proceedings, since more tools need to be used for generating the DPR design. For instance, the design has to be generated in EDK, with black boxes for the reconfigurable modules (RMs); the RMs need to be synthesized as independent projects, in Xilinx ISE, and then imported along the top level into PlanAhead.

Table 9 Design efforts using VHDL, EDK and our approach

Type of design capture	Time	Description
Pure VHDL Approach		This method is the less relaible, long and prone to error. Good for small systems
Manually integrating the platform	Days	
Manually modifying DPR IPs	Hours	No support for DPR management
Using Xilinx EDK		EDK is justitfiable for systems containing at least one processor (DPR manager)
Platform Integration in XPS	1 h 30 min	
Manually modifying DPR IP + MPD	25 min/IP	IP blocks need to be processed separately
Proposed Approach		The time required for a platform creation is reduced, and the maintainability is improved
Platform Integration	40 min	
Modifying DPR IP + MPD	10 min/IP	IP blocs creation (VHDL) takes the same time, but integration is improved

Table 10 Use languages and tools

File type	Number of files
Comp IP-XACT	21
MPD	21
VHDL	21 (top-level IP) 700 (sub-components)
Netlist	1 Top Level 3 DRP Modules

Further advantages of using UML and MARTE is the maintainability and improved updatability of the models; this means that, contrarily to purely VHLD or EDK flows, a change in the platform requires much less effort: since every step of the design flow is automated, the designer does not even need to make use Xilinx EDK or ISE. The IP-XACT descriptions also facilitate the updatability of the approach by changing the vendor extensions or the target meta-models, but not the implementation files. In Table 9, we show the design efforts for each of the aforementioned methods. It must be noted that we consider design capture times by non experts. In Table 9 we provide a summary of the number of files used for the implementation of the platform, note that an IP block can be composed by a few or even hundreds (260 VHDL files for the Multi-Port Memory Controller).

A comparison of the integration of DPR is also provided in Table 9. The typical approach would require manually inserting the black boxes in the IP VHDL descriptions, and in parallel, to synthesize each of the IPs to be mapped in these reconfigurable partitions; the same applies for an EDK based approach. If we consider the number of files to be integrated, as shown in Table 10, it can be observed how rapidly the design effort can explode. In our approach, these descriptions are available in the library, and their synthesis is automated through TCL scripts that access the IP-XACT components description, and stores the netlists in a new project folder. Along with the top-level netlist, they comprise the necessary inputs for the DPR floorplanning phase.

8 Conclusions

During the last decade, DPR has been widely studied as a research topic. Despite its intuitive appeal, the technique had eluded widespread adoption, particularly in industrial ap-

plications. This is due to the complexities of the provided design flow and the in-depth knowledge of many low level aspects of FPGA technologies used to implement DPR systems. However, with recent developments in FPGA technologies and with the automation of many of the burdensome steps in the design flow, this trend is expected to change, and some exciting new products have already been demonstrated.

In this paper, we have concentrated our efforts in the creation of the structural description of the system that is used as an input to the DPR design flow to facilitate the design entry phase of the DPR design flow. The presented approach is based on two widely used standards, UML MARTE and IP-XACT that until recent years had been developed in parallel. A great deal of research has been carried out to unify both standards, given the opportunities offered by the IP-XACT standard for interchanging IP descriptions among EDA tools. However, as it has been exposed in this paper, IP-XACT can also be exploited as a means for providing and intermediate system description that can be used to pass from UML MARTE models to HDL code generation. An IP-XACT compliant design environment facilitates the configuration and interconnection of complex systems, and provides mechanisms for EDA that can be used to control automate many of the burdensome tasks associated with SoC design flows.

We have showed how the IP-XACT can be used to generate the top level HDL description of the system, along with the necessary reconfigurable IPs that are gathered from a component library. We made this by separating the target back-end models from the high-level descriptions. The presented IP-XACT component descriptions contain vendor extensions that allow us to integrate our methodology in the Xilinx design ecosystem for DPR systems, but in such a way that allow us not to impact the interchangeability of the models. Therefore, the IP-XACT models can be extended for targeting different back-ends, allowing to easily evolve the methodology to changing requirements or to adapt it to other vendors.

Furthermore, we have demonstrated our methodology through a case study in which an image processing IP is integrated into MicroBlaze based SoC design. Using MARTE and IP-XACT makes the design or DPR more amenable, and at the same time, helps in decoupling the high-level models from the intended back-end. This his achieved through the use of a generic IP deployment meta-model, which does not make particular assumptions on the nature of the low-level implementation details.

Moreover, we have presented an deployment extension to the MARTE profile that enable us to import relevant information to the UML models that are subsequently used for system generation purposes, facilitating IP reuse in the process. Then, we introduced the model transformations necessary to move from UML MARTE to IP-XACT, and from the utilized Xilinx PSF models to generate the EDK platform, our targeted back-end.

Acknowledgements This work has been supported by the ANR FAMOUS Project (ANR-09-SEGI-003). The authors also wish to thanks Sodius for their support, and for providing access to their design tools.

References

1. Manet P (2008) An evaluation of dynamic partial reconfiguration for signal and image processing in professional electronics applications, EUSASIP J Embedded Syst
2. Compton K, Hack S (2008) Reconfiguration management. In: Reconfigurable computing: the theory and practice of FPGA-based computing, Chap 4. Morgan Kaufmann, San Mateo
3. Donlin A (2007) Applications, design tools and low power issues in FPGA reconfiguration. In: Designing embedded processors a low power perspective, Chap 22. Springer, Berlin, pp 513–541
4. OMG (2012) Modeling and analysis of real-time and embedded systems MARTE, Beta 3
5. IEEE Standard for IP-XACT (2010) Standard structure for packaging, integrating, and reusing IP within tools flows. IEEE Std 1685–2009, pp C1-360, Feb 18 2010

6. Kruijtzer W et al (2008) Industrial IP integration flows based on IP-XACT Standards. In: DATE'08, March 2008, pp 32–37
7. Lennard C (2006) Industrially proving the SPIRIT consortium specifications for design chain integration. In: DATE'06, March 2006, pp 1–6
8. Xilinx (2011) Partial reconfiguration user guide, Xilinx UG208
9. Xilinx (2011) EDK concepts, tools, and techniques a hands-on guide to effective embedded system design
10. Xilinx (2011) Embedded system tools reference guide, Xilinx UG111
11. Xilinx (2011) Platform specification format reference manual, Xilinx UG642
12. Xilinx (2009) PlanAhead user's guide, Xilinx UG632
13. Dekeyser J-L, Boulet P, Marquet P Meftali S (2005) Model driven engineering for SoC co-design. In: The 3rd international, IEEE-NEWCAS conference, 19–22 June 2005, pp 21–25. doi:10.1109/NEWCAS.2005.1496724
14. Taha S, Radermacher A, Gérard S, Dekeyser J-L (2007) MARTE: UML-based hardware design from modelling to simulation. FDL
15. Taha S, Gerard S, Dekeyser J-L (2007) An open framework for detailed hardware modeling. In: SIES 2007, pp 118–125
16. Koudri A et al (2008) Using MARTE in the MOPCOM SoC/SoPC CoMethodology. In: MARTE workshop at DATE'08
17. Vidal J, De Lamotte F, Gogniat G (2009) A co-design approach for embedded system modeling and code generation with UML and MARTE. In: Design, automation and test in Europe (DATE'09)
18. Vidal J, de Lamotte F, Gogniat G, Diguet J-P, Soulard, P (2009) IP reuse in an MDA MPSoPC co-design approach. In: International conference on microelectronics (ICM), 19–22 Dec 2009, pp 256–259
19. West team of LIFL laboratory (2012) Graphical array specification for parallel and distributed computing (Gaspard)
20. Quadri IR, Muller A, Meftali S, Dekeyser J-L (2009) MARTE based design flow for partially reconfigurable systems-on-chips. In: 17th IFIP/IEEE international conference on very large scale integration (VLSI-SoC 09)
21. Quadri IR, Meftali S, Dekeyser J-L (2010) Designing dynamically reconfigurable SoCs: from UML MARTE models to automatic code generation. In: Conference on design and architectures for signal and image processing (DASIP), 26–28 Oct 2010, pp 68–75
22. Quadri IR (2011) MARTE based model driven design methodology for targeting dynamically reconfigurable FPGA based SoCs. PhD Dissertation
23. Revol S, Taha S, Terrier F, Clouard A, Gérard S, Radermacher A, Dekeyser J-L (2008) Unifying HW analysis and SoC design flows by bridging two key standards: UML and IP-XACT. In: DIPES 2008, pp 69–78
24. André C, Mallet F, Khan AM, de Simone R (2008) Modeling SPIRIT IP-XACT with UML MARTE. In: Proc DATE workshop on modeling and analysis of real-time and embedded systems with the MARTE UML profile
25. Schattkowsky T, Tao X, Mueller W (2009) A UML frontend for IP-XACT-based IP management. In: Design, automation & test in Europe conference & exhibition, 2009, DATE'09, 20–24 April 2009, pp 238–243
26. Arpinen T et al (2008) Model-driven approach for automatic SPIRIT IP integration. In: UML-SOC'08, June 2008
27. Arpinen T, Koskinen T, Salminen E, Hamalainen TD, Hannikainen M (2009) Evaluating UML2 modeling of IP-XACT objects for automatic MP-SoC integration onto FPGA. In: Design, automation & test in Europe conference & exhibition, 2009, DATE'09, 20–24 April 2009, pp 244–249
28. Cherif S, Quadri IR, Meftali S, Dekeyser J-L (2010) Modeling reconfigurable systems-on-chips with UML MARTE profile: an exploratory analysis. In: DSD 2010, pp 706–713
29. Guillet S, de Lamotte F, Rutten E, Gogniat G, Diguet J-P (2010) Modeling and formal control of partial dynamic reconfiguration. In: ReConFig 2010, pp 31–36
30. Sodius Corporation (2011) MDWorkbench. http://www.mdworkbench.com/