Facilitating IP deployment in a MARTE-based MDE methodology using IP-XACT: a XILINX EDK case study

Gilberto Ochoa-Ruiz, Ouassila Labbani, El-Bay Bourennane, Sana Cherif, Samy Meftali, Jean-Luc Dekeyser

FACILITATING IP DEPLOYMENT IN A MARTE-BASED MDE METHODOLOGY USING IP-XACT: A XILINX EDK CASE STUDY

G. Ochoa-Ruiz, O. Labbani-Narsis, E. Bourennane
LE2I Laboratory, Université de Bourgogne, France
C. Author: gilberto_ochoa-ruiz@etu.u-bourgogne.fr

S. Cherif, S. Meftali, J.-L. Dekeyser
INRIA Lille Nord Europe, Villeneuve d'Ascq, France
(FirstName.SecondName)@inria.fr

Abstract – **In this paper we present framework for the deployment of hardware IPs at high-levels of abstraction. It is based in a model-driven approach that aims at the automatic generation of Dynamic Partial Reconfiguration designs created in Xilinx Platform Studio (XPS). Contrary to previous approaches, we make use of the IP-XACT standard to facilitate the deployment of hardware IPs, their parameterization and subsequent integration. We propose an extension to the MARTE profile for IP deployment, and we introduce the necessary model transformations to obtain a high-level representation from an IP-XACT component library. These models are then used to create a platform in MARTE that abstracts the technologic aspects of the chosen back-end. The so-obtained UML platform is transformed in an IP-XACT design, which is exploited to generate the files used by XPS for system implementation. In this way, we promote IP reuse and deployment while remaining back-end independent, by using specific vendor extensions. Finally, we analyze the advantages of the proposed methodology by a case study in system integration.**

Index Terms— UML MARTE, MDE, IP-XACT, EDA, Rapid System Prototyping.

1. INTRODUCTION

Model-Driven Engineering (MDE) [1], in tandem with UML has been used in SoC co-design methodologies recent years with relatively success [2]. Many of these approaches make use of the UML profile for "Modelling and Analysis of Real Time and Embedded Systems" (MARTE) [3]. UML 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 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. In the context of UML for SoC, this is basically an IP reuse problem; in this way the components can be configured and, through a system composition phase, a synthesizable implementation can be obtained.

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. The parameters and interface information of the IP have to be readily available in the high-level models. Therefore, the deployment phase must also provide a mechanism for retrieving the information of IP cores that are usually coded in languages such as VHDL. 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 do not promote IP reuse.

This issue has been addressed by the academia, with ongoing efforts aiming to integrate the IP-XACT standard [4] in MDE-based methodologies, as an intermediate representation (IR). The standard describes a set of XML schemas used to document IP meta-data for IP packaging and SoC integration. The goal of the specification is to provide an interchangeable description of HW components.

In this paper, we introduce a deployment approach used for deploying Xilinx Platform Studio (XPS) [5] IP Cores. We make use of IP-XACT as an IR between the MARTE models and the files used by XPS to implement the SoC platform. We transform the XPS files into IP-XACT components that are subsequently converted into MARTE models, used in the deployment phase of our approach.

Afterwards, the components in the MARTE platform are linked to those in the library, obtaining automatically a description that is converted to an IP-XACT design. This description contains the information of the deployed components, which is fed to a model transformations tool. In this way, an XPS platform which is used by the Xilinx tools to obtain synthesizable VHDL code is created.

The paper is organized as follows: in Section II, we describe similar approaches and its limitations in the IP deployment phase. Then, Section III describes the targeted back-end, and the needs in terms of models generation for HW specifications. In section IV, we present the proposed approach, the role of the proposed meta-models and the model transformations. In Section V we discuss how IP-XACT is embedded in the IP deployment and generation phases of the methodology. The next section introduces the MARTE extensions for IP deployment, and describes how the models are obtained from IP-XACT; then, we discuss IP-XACT design is obtained through model transformations. The transformations for obtaining the XPS models are described in Section VII. In section VIII we embark in a discussion of the reduction in design effort provided by the methodology, and Section IX concludes the article, providing some avenues of future work.

978-1-4673-2921-7/12/$31.00 ©2012 IEEE

2. RELATED WORKS

Several works have tackled the use of MARTE in SoC design, specifically at the deployment level, such as the MoPCoM [7] and GASPARD [8] frameworks. They are both based in a Y-schema co-design approach. The main disadvantage is that, as with many other MDE methodologies, both approaches make use of non-standardized deployment representations. This means that the deployment models obtained by these methodologies are not interchangeable, making them highly methodology-dependant. Another issue is that the parameters are retrieved from the implementation files in a non-automatic way (e.g. annotated as comments in UML).

The IP-XACT standard has been introduced as a means to overcome the aformentioned issues. Authors in [9] and [10] have created IP-XACT meta-models as extensions to MARTE in order to manage IP related information, such as parameters, ports and bus interfaces. However, having such levels of detail of the IP implementation betrays the use of MARTE, since the IP-XACT descriptions contain information that is not necessarily pertinent at high-levels of abstraction. Moreover, it forces the user of MARTE to deal with aspects that are too hardware-oriented, since the IP-XACT description contains very low-level meta-data.

We undertake a simpler approach: the IP-XACT components in our approach are created in such a way that allow us to extract information of the IP components (in another IR, used by XPS) by using a deployment package, which includes only the necessary information of the IP for parametrisation and interconnection. The components obtained in this manner are used to compose an architecture diagram, which is then parsed to obtain an IP-XACT design description. The IP-XACT design allow us to be back-end independant, but to generate the EDK files by using vendor extensions in the component descriptions.

3. THE XILINX PLATFORM STUDIO BACK-END

Xilinx XPS makes use of a series of files defined in the Platform Specification Format (XPSF) 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 top-level 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 Figure 1, 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 bundled 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.

```
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
```

Fig 1. Snapshot of an MPD file for IP description

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 Figure 2, Platgen (a Xilinx tool) reads a MHS as its primary design input. The tool 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

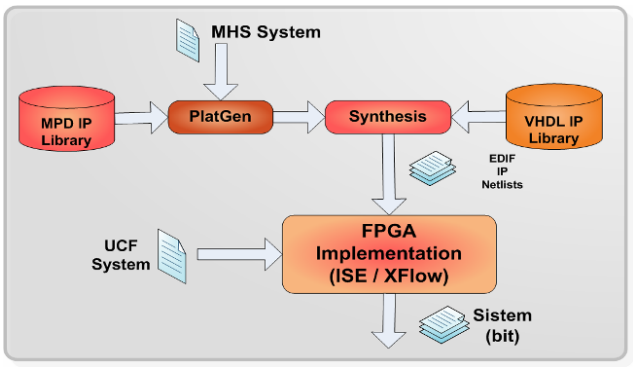

Fig 2. Xilinx EDK flow for processor-based design

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 deployment 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 XPSF descriptions would tie the approach to an specific back-end, making it difficult to adapt to updates or to adapt it to other vendors and flows. Therefore, we use IP-XACT, as it will be described in section 5, 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. We have created a series of EDK meta-models in order to perform these transformations.

4. PROPOSED MDE METHODOLOGY

In this section we explain in more detail the ideas introduced previously. Here, we embark in a thorough description of our approach and how it is embedded into the design flow of XPS embedded systems. The proposed MDE methodology, in terms of models and model transformations, is presented in Figure 3. We make use four abstraction levels, each making use of its corresponding component library. The entry point is a MARTE deployment model (a Composite Structure Diagram, CSD), which is created by choosing components from a MARTE Model Library (MML). The MARTE model is obtained in the deployed allocation phase, where sufficient information of 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 the MARTE profile for defining the features of the deployed IPs that allow us to link them to their IP-XACT counterparts and to obtain their properties automatically.

After having chosen an IP in the deployment model, the system designer can create a system description by using two views, the "Parameterization View" and a "Platform View"; these views contain a set of component instances to be parameterized 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 hierarchical connections.

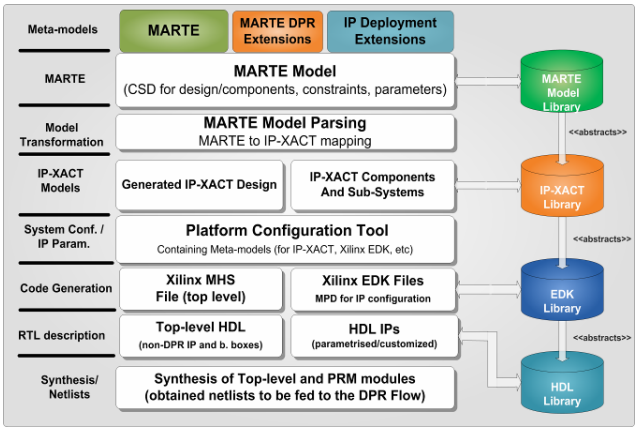

Fig 3. System integration approach in terms of model transformations

The IP-XACT design is imported to our chosen "design environment", Sodius MDWorkbench [12], in which the model transformations from IP-XACT to the XPSF models take place. The XML schemas have been processed by an improved XSD/Ecore meta-model importer in MDWorkbench, which leads to a Java/EMF implementation of the IP-XACT meta-model.

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 transformations from IP-XACT to this proprietary format, which is used by the Xilinx tools to obtain the top-HDL description, and the references to the HDL IPs, that are configured in this phase.

In this way, we aim to facilitate the design entry phase of the Dynamic Partial Reconfiguration Xilinx design flow, as described in [6]. In this paper, we discuss the meta-models and transformation rules in detail.

5. IP-XACT CONCEPTS USED IN OUR APPROACH

The IP-XACT standard defines four central object descriptions, which are *bus* and *abstract definitions*, *component*, and *design* descriptions. These four elements are sufficient for structurally describing a system and the IP cores the compose it. The main goal of this paper is to present a framework for the parameterization and integration of the modules that comprise the DPR platform.

5.1. Component descriptions.

A component description packages the information related to an IP core, as depicted in Figure 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 *<bus interfaces>* 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 ports to support controlled inclusion into the generation phase. This is important for IP customization and not considered in the current IP-XACT specification.

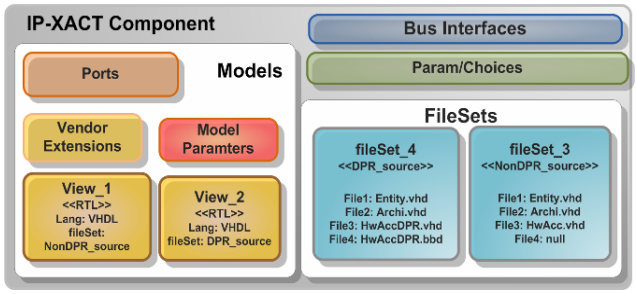

Fig 4. IP-XACT concepts for a component description

The <<views>> elements in the <<Model>> section describe implementation specific information, and specify the location of the implementation files.

5.2. 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 in IP-XACT: interconnections, ad-hoc and hierarchical connections.

We make use of design 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.

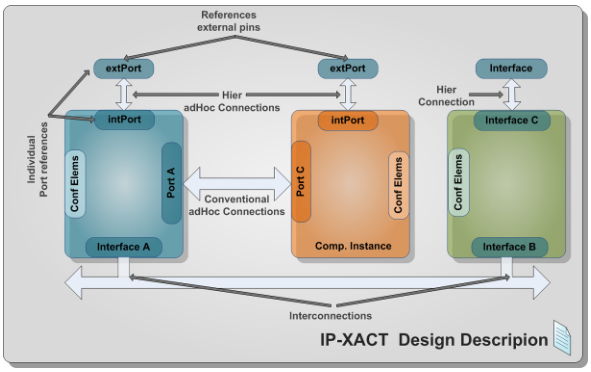

Fig 5. IP-XACT concepts for a design description

6. MARTE EXTENSIONS FOR IP DEPLOYMENT

6.1. IP Deployment Package

The deployment phase of any MDE methodology is instrumental, since enables the generation of a synthesizable SoC description from a high-level MARTE model. More precisely, sufficient information must be provided at this stage so that the code integration and parameterization on the IPs can be performed.

In order to promote IP reuse in our approach, we have introduced a deployment package as an extension to the MARTE Profile. It contains an IP stereotype, depicted in Figure 6, which uses resources from the *Generic Resource Modeling (GRM)* package in MARTE. Each elementary component is deployed and corresponds to an IP implementation in the IP-XACT library. The IP stereotype contains a set of attributes used to describe basic information of the IP at high-levels of abstraction. 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>> stereotypes* to provide a means to deploy the IP, specifically Identifier and ModelParameters.

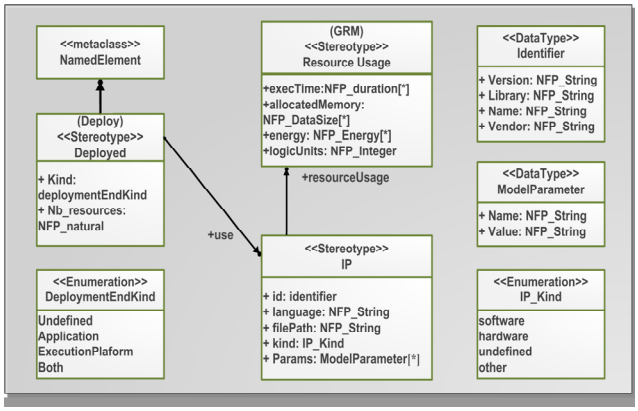

Fig 6 Stereotype IP in the Deployment package

In Figure 6 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 our approach, 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).

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.

6.2. MARTE models at the deployment level

We perform model transformations from the Xilinx MPD files to obtain our IP-XACT library; the components in the library, as described in Section 4.1**,** 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, via vendor extensions.

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 Figure 7a). 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.

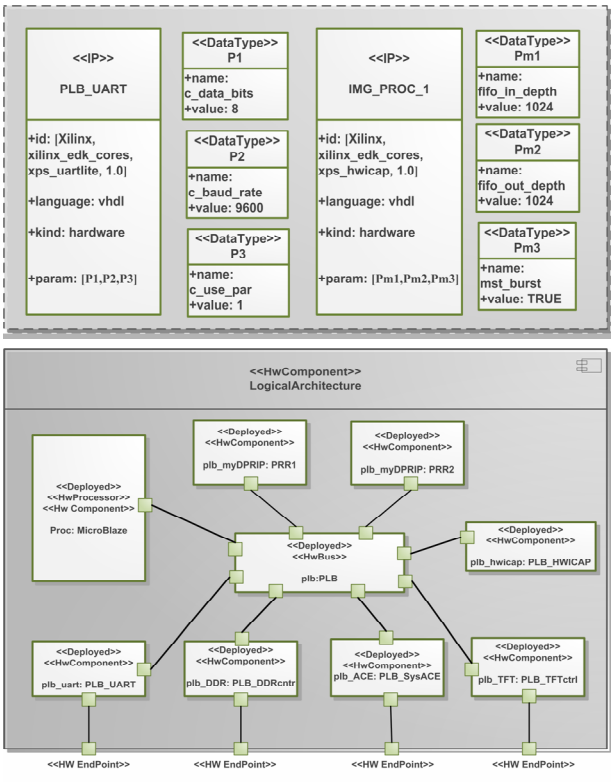

Fig 7. a) A detailed component instance snapshot

b) MARTE logical architecture view

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>* element of the component description. 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 parameters are converted into MARTE component by an IP-XACT to MARTE transformation.

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 element; 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 are labelled as *<<HWResource>>* or *<<HWComponent>>* are linked to the *<<IP>>* stereotypes and stored in the MML MARTE Library as a template. These modules are subsequently used as the building blocks of the *"Platform View"*, as shown in Figure 7b). As mentioned before, both views are parsed and used to create an IP-XACT design description exploited for system generation.

5.3. Model Transformation for IP deployment.

As described in Section 4 (Figure 3), the creation of a platform in MARTE starts by importing a set of components from the MML 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 liking the high-level components to their IP-XACT counterparts. Similarly, the meta-model is also used for creating the "Parameterization View" with parameters that control the inclusion of ports, bus interfaces and other parameters in the final MHS model (and 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.

Table 1. Models transformations for IP deployment, IP-XACT ⇔ MARTE

MARTE	IP - XACT
Class Instance	**Component**
id: identifier *(VLNV tuple)*	spirit: version: library: name: vendor
Port = name (kind: busIF)	spirit:busInterfaces:busInterface:name
Port = name (kind: adHoc)	spirit:Model:ports:port:name
Parameter = name	(if <immediate> and <configGroups> = user) spirit:ModelParameters:Parameters

Once the designer has parameterized and composed its platform, the models ("Parameterization" and "Platform" views) are parsed to produce an IP-XACT design description. The obtained XML file contains a *<spirit:design>* entry, which identifies it as the top level element in a SoC design. The MARTE CSD diagram contains a set of parts, 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 2. This description in imported to Sodius MDWorkbench for the back-end models generation.

Table 2. Transformations from a deployed model to an IP-XACT design.

MARTE	IP - XACT
CSD Diagram	**Design**
Part	spirit:ComponentInstance
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

For each component in a MARTE CSD an *<spirit:instanceName>* entry is created in the *<spirit:componentInstances>* section of the IP-XACT design file. The configurable elements for the components are inferred from the *<<Parametrisation 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:interconnections>*); the external pin information are obtained from the a combination of the enclosing IP-XACT component description and the *<spirit:adHocConnections>* with external ports references.

7. XPS BACK-END META-MODELS AND MODEL TRANSFORMATIONS

In this section, we will discuss the proposed back-end meta-models and outline how the model transformations are carried out. For questions of space, we will concentrate in the MHS file for obtaining the top-level implementation description from the IP-XACT description.

Xilinx PSF files are structured a textual format, which can easily be understandable by machines by defining a parser, but the first mandatory step if the meta-model definition. The 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. These meta-models reside in the Sodius tool, where the models transformations take place.

This file is used by the Xilinx tool to create the top-level description of the hardware platform The MHS description contains the same elements of the MPD file, with one difference: only the parameters, bus interfaces and ports that have been used by the user for the top-level description of the IP are displayed, as shown in Figure 8.

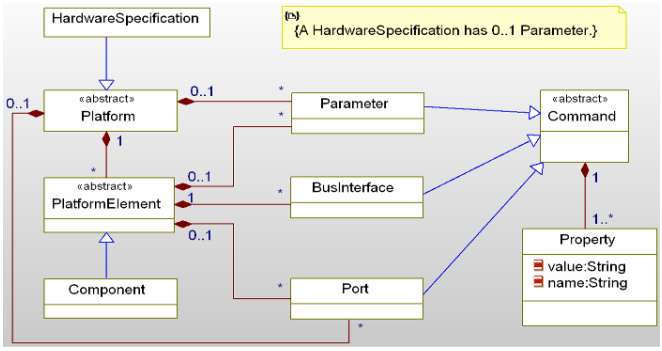

Fig 8. UML Mode for l Microprocessor Hardware Specification meta-model.

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).

Table 3. Rules for passing from an IP-XACT design to an MHS model

MHS Command	IP-XACT Design Counterpart
PARAMETERS	**CONFIGURABLE ELEMENTS**
INSTANCE	spirit:InstanceName
HW_VER	spirit:Version
PARAM_i = VALUE_i	spirit:configurableElementValue spirit:referenceId = value
BUS INTERFACES	**INTERCONNECTIONS**
BUS INTERFACE i INTERFACE = bus_std	if (componentRef = "instanceName") activeInterface:busRef = interconnection:name
PORTS	**PORT REFERENCES**
internal ports PORT_i = VALUE_i	if (componentRef = "instanceName") internalPortRef:portRefadHocConnection:name
external ports PORT_i = VALUE_i	if (componentRef = "instanceName") externalPortRef:portReadHocConnection:name

The components in EDK are associated to the implementation files using the instance name and hardware version values. IP-XACT provides a mechanism, the VLNV value to link the components from the MARTE models to their IP-XACT and 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 3 shows the list of mapping between IP-XACT and MHS.

8. CASE STUDY AND DESIGN EFFORT FOR THE CREATION OF A EDK PLATFORM

In this section, we discuss how the proposed methodology helps in simplifying the conception of SoC in Xilinx XPS, using the example depicted in Figure 7 (in Section 5.2). The system implementation has been described in detail in [6], it consists in a DPR image processing architecture implemented in a Virtex 5 FPGA (LX50) board; however, since the aim of the paper is on facilitating system composition and generation for EDK, the discussion that follows can be extended to similar systems, without impacting the validity of the results in general.

8.1. Implementation Results.

We have modeled the system in Figure 7 using Papyrus. The platform model is converted into an XMI file, which is parsed in MDWorkbench and then used to perform the mode transformation described in Section 5.3, in order to obtain an IP-XACT design description. Then, by using the XPSF meta-models and models transformations, we obtain the MHS file, which is fed to XPS, to obtain the VHDL description which, after synthesis, can be used as an input for the Partial Reconfiguration Xilinx design flow.

Table 4. Number of lines per model and executions time for synthesis

File Type	No Lines	Exec Time	Description
XMI	654	50 sec	MARTE model in XMI for transformation purposes into IP-XACT in MDE Workbench
IP-XACT	998	30 sec	Design object for platform description, imported to Sodius
MHS File	455	20 sec	XPS top level description file
VHDL	7986	30 sec	HDL description of the system. Obtained feeding Xilinx Platgen
Netlist	N/A	12 min	Time used for logical synthesis

Table 4 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 which relies completely on the Xilinx tools. Each of the steps described in Table 4 has been automated, facilitating the task of the creator working in high-levels of abstraction, one of the objectives of using MARTE in our approach.

8.2. Discussion on design effort.

In this section we elaborate on the design effort required to implement the system detailed in Figure 7, especially if we compare the proposed approach 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 instantiations, parameterization and signals declarations for interconnection. It is evident that creating such a design (composed of several components, and multiple sub-components) would take not only hours, but maybe days, in a process very prone to errors, as shown in Table 5.

Table 5. Design efforts using VHDL, XPS and the proposed methodology

Type of design capture	Time	Description
Pure VHDL Approach		This method is the less reliable, long and prone to error. Good for small systems Not support for DPR management
Manually integrating the platform	Days	
Manually modifying DPR IPs	Hours	
Using Xilinx EDK		EDK is justifiable for systems containing at least one processor (DPR manager) IP blocks need to be processed separately
Platform Integration in XPS	1h30 min	
Manually modifying DPR IP+ MPD	25 min/IP	
Proposed Approach		The time required for a platform creation is reduced, and the maintainability is improved
Platform Integration	40 mins	
Modifying DPR IP+ MPD	10min/IP	

Xilinx EDK, using the Platform Specification Format (PSF, 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 require a great deal of design effort and expertise of the tool and this is precisely one of the advantages of using the proposed 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.

Another advantage of using UML and MARTE is the maintainability and improved updatability of the models; this means that, contrarily to purely VHDL 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 5, 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 6, we provide a summary of the several languages, models and tools. For models transformations we have used the Model Query Language (MQL) and Java. The high-level models have been created using Papyrus, and the XPSF meta-models have been implemented using Rhapsody, and then imported to MDWorkbench. We have used IP-XACT Editor for creating IP-XACT golden rule models manually; these models are compared with those generated by MDWorkbench.

Table 6. Used languages, models and tools

Used Languages	Purpose
MQL	Ruleset definition
Java	Transformations
IP-XACT	Intermediate Model
VHDL	IP Implementations
Used Tools	**Purpose**
Papyrus	Modelling
Rhapsody	EDK Meta-models creation
MDWorkbench	Model Transformations
IP-XACT Editor	Golden Rule Models
Xilinx EDK	IP implementations

9. CONCLUSIONS

In this paper we have presented a design methodology that enables the deployment, parameterization and integration of hardware IPs into SoC platform at multiple levels of abstraction. For this, we have introduced IP deployment capabilities in MARTE, which aim at facilitating the import of selected low-level features into the high-level models, their modification, and the creation of an IP-XACT design description that is used to parameterize and integrate the underlying IP 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. 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 a generic IP deployment meta-model, which does not make particular assumptions of the nature of the low-level implementation details.

10. ACKNOWLEDGMENTS

This work has been supported by the ANR FAMOUS Project (ANR-09-SEGI-003) by the Agence Nationale de la Recherche.

The authors also wish to thanks Sodius for their support in this project and for providing access to their design tools.

11. REFERENCES

[1] J-L. Dekeyser, P. Boulet, P. Marquet, and S. Meftali, "Model driven engineering for SoC co-design," IEEE-NEWCAS Conference, 2005. The 3rd International, vol., no., pp. 21- 25, 19-22 June 2005 doi: 10.1109/NEWCAS.2005.1496724.

[2] S.Taha, A. Radermacher, S.Gérard, J-L. Dekeyser: MARTE: UML-based Hardware Design from Modelling to Simulation. FDL 2007.

[3] OMG, "Modeling and Analysis of Real-time and Embedded systems MARTE), 2009.

[4] "IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tools Flows," IEEE Std 1685-2009, Feb. 18 2010.

[5] Xilinx, "Platform Specification Format Reference Manual", UG642

[6]. G. Ochoa, E. Bourennane, O. Labbani, K. Messaoudi: IP-XACT and marte based approach for partially reconfigurable systems-on-chip. FDL 2011: 1-8

[7] J. Vidal, F. de Lamotte, G. Gogniat, J.- P. Diguet, and P. Soulard, "IP reuse in an MDA MPSoPC co-design approach," Microelectronics (ICM), 2009 International Conference on , vol., no., pp.256-259, 19-22 Dec. 2009.

[8] I. R. Quadri, S. Meftali, and J.-L. Dekeyser, "Designing dynamically reconfigurable SoCs: From UML MARTE models to automatic code generation," Design and Architectures for Signal and Image Processing (DASIP), 2010 Conference on, vol., no., pp.68-75, 26-28 Oct. 2010.

[9] C. André, F. Mallet, A. M. Khan, and R. de Simone, "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, 2008.

[10] F, Herrera, E. Villar, "A framework for the generation from UML/MARTE models of IPXACT HW platform descriptions for multi-level performance estimation," Specification and Design Languages (FDL), 2011 Forum on , vol., no., pp.1-8, 13-15

[11]. Xilinx, Embedded System Tools Reference Guide, Xilinx UG111.

[12]. Sodius, MDWorkbench, http://www.mdworkbench.com/, 2011.