Generation of Abstract IP/XACT Platform Descriptions from UML/MARTE for System-Level Performance Estimation

Towards a MARTE to IP/XACT Generation Framework of HW platform descriptions for a DSE Multi-level Performance Estimation Framework

F. Herrera and E. Villar
Microelectronics Engineering Group, TEISA Dpt.
University of Cantabria
Santander, Spain
{fherrera, evillar}@teisa.unican.es

Abstract—UML/MARTE is enabling the development of methodologies for the specification of a whole real-time system, and of holistic MDA methodologies where the UML/MARTE description is taken as the source for different design activities, such as system-level performance estimation and implementation refinement. A crucial issue to make these methodologies working and efficient is the development of tools, such as code generators, able to handle and produce from the UML/MARTE model the level of information required by each specific design activity. In this line, this paper proposes a framework for the automatic generation of abstract IP/XACT descriptions of the HW platform of a system, synthetic and suitable for system-level performance estimation. This paper is a first step towards a generation framework, able to produce IP/XACT descriptions fitted to the information needs of the different design activities. While this work focuses on high-level performance estimation, the framework is aimed to enable the production of more detailed IP/XACT descriptions able to feed more detailed performance estimation frameworks and implementation tools.

I. INTRODUCTION

Embedded systems are becoming more complex everyday. Their design is becoming more difficult since it requires dealing with a growing amount of information. Reaching a competitive and efficient implementation requires enabling an early description of the whole system, which facilitates its understanding and an early and fast, but enough accurate, assessment of the involvements of the different implementation possibilities. In this line, Model Driven Architecture (MDA) [1] enables the description of a system through different views. For instance, a use case view of the system can provide information for the generation of the necessary test benches for a full system validation. Such use case view does not need to contain information from other views, such as the inner functionality, or the implementation architecture of the system. A proper handling of the abstraction level is also required. For instance, the generation of functional test benches should require handling less information than the generation of HDL test benches. In any case, handling more information than necessary can be cumbersome and counterproductive. In this line, MARTE [2] is a standard UML profile which is enabling the development of different methodologies [3][4][5] for building specifications which capture all the necessary information for the analysis and design of a real-time system. Regarding such analysis, the great variety of platform possibilities and mappings is leveraging the development of fast performance estimation technologies, such as SCoPE [6].

In this context, COMPLEX [7] is a holistic design space exploration (DSE) framework which supports UML/MARTE for the system description. A set of generators extract from the UML/MARTE model the information required to feed performance estimation tools. These tools are, in turn, driven by an exploration tool which decides the next point to be explored in the design space.

In order to make the COMPLEX methodology efficient, easy to use, and scalable, it is necessary that the generators are able to extract from the UML/MARTE specification a synthetic description, only with the information required by the system-level performance estimation tools, and under a format where such information can be grown with the additional information which will be required by further design activities, such as a detailed performance estimation and implementation.

This work has been funded by the COMPLEX (FP7-247999) project.

Figure 1. Generation of the Abstract IP/XACT description in COMPLEX.
Specifically, this paper tackles the extraction (thick arrow in Figure 1) from a COMPLEX UML/MARTE description of the HW platform information required by system-level performance estimation tools. This extraction has several distinctive points. First, the platform information from the UML/MARTE description will be extracted as an IP/XACT description. IP/XACT [8] is a well accepted standard for the exchange of IP components at electronic system and RTL level [9], and already adopted by commercial tools [10]. Second, this paper specifically focuses on the generation of an abstract IP/XACT description. It is abstract of generic, e.g., it uses a bus of reduce set of parameters, instead of an OPB or an AMBA model with a detailed set of characteristic parameters. It is also abstract in the sense of synthetic, because it only contains the information necessary, but sufficient, to serve as input to the COMPLEX system-level performance estimation chain, specifically to SCoPE [6]. Third, the generator and the produced IP/XACT description are scalable. That is, an IP/XACT description with more information suitable for lower-level performance estimation tools (slower but more accurate), or even for implementation, can be generated by adding further information (extracted from the UML/MARTE model) to the abstract IP/XACT description. A separated dealing of this information enables the development of a MARTE to IP/XACT generator suited to a multi-level performance estimation framework.

The rest of the paper is structured as follows. Section II overviews the previous work. It includes an introduction to the type and format of IP/XACT format generated. In section III, the UML/MARTE methodology is introduced, focusing on the description of the HW platform. Then, the generator is explained in section IV, where the more important translations rules implemented and the main features of a generator prototype are explained. Finally, section V provides the main conclusions and future lines of this work.

II. PREVIOUS WORK

A. COMPLEX and SCoPE

In the MARTE COMPLEX flow, there are three main phases/tools. First, the specification methodologies/tools for capturing in UML/MARTE the system description; second, the generation of the SystemC executables, whose execution provides performance figures; and, third, the DSE tool, able to select the next points of the design space to be explored and thus drive the exploration. In COMPLEX, the generation of the SystemC executable is actually split into two phases. In a former step, a set of generators is in charge of producing all the input files which are required for the generation of the SystemC executable. These files do not contain SystemC code, but the information which, in turn, is used for the generation of the SystemC executables.

Figure 2 sketches the three main formats automatically generated in the COMPLEX generation flow. A COMPLEX specific format (CFAM) serves for the description of the system application, including its structural information, its functionality, and the inherent concurrency. The CFAM will also include the real-time features included in the MARTE description. From the CFAM description, a SystemC platform independent model (PIM) will be generated, which enables a functional validation of the system, and moreover, feeds further stages of the COMPLEX flow.

The automatic generation from MARTE in COMPLEX will also produce XML files which contain information relative to the description of the platform, and to the allocation of application components into platform components. This XML information, which can be split into different files, will also reflect all the configurable parameters which are susceptible for exploration, and which can be tuned by the DSE tool. This highly configurable XML is a suitable format to enable the generation of configurable platform specific models in SystemC.

Specifically, SCoPE [15][16][17] is one of the performance estimation frameworks involved in COMPLEX. In COMPLEX, SCoPE will be in charge of system-level performance estimation. This means that SCoPE will be able to enable the faster iterations in the DSE cycle. For it, SCoPE relies on native simulation [16][17], and requires different information of the system configured, which regards to the application, to the platform and to the way the application components are allocated to the platform components. With abstract information and an abstract model of the system, SCoPE is able to provide different metrics of the system, such as power consumption, time latencies, usages of CPUs, etc.

From its origin, SCoPE has been able to support as input a concurrent application relying on a RTOS API (currently POSIX, WIN32 or uC/OS-II). In COMPLEX, SCoPE is being extended to SCoPE+, in order to support the higher level CFAM API. As SCoPE, SCoPE+ will be able to consider the performance effects of custom hardware, encrusted as TLM SystemC models, directly plugged to the system bus. Moreover, SCoPE already supports as an XML input the description of the platform, of the allocation of SW components to HW components, and the configurability of the platform. Finally, in a previous work [19], SCoPE was enabled to admit the description of the HW platform as an IP/XACT description, thus separated from the rest non-standard XML information.

In COMPLEX, the generation of an IP/XACT description of the HW platform is foreseen. Such an IP/XACT description reflects a fixed architecture (the optimum one, found by the DSE phase), and contains all the
In COMPLEX, the generation of the configurable a PSM SystemC executable requires from SCoPE+ to be able to read the application description as CFAM code, and the platform description, allocation information, etc, as XML code. Beyond the COMPLEX objectives, provided that SCoPE+ is able to admit an abstract IP/XACT as input for the description of the HW platform, this paper proposes enabling the direct generation of such synthetic IP/XACT descriptions already in the phase where generator a re enabling the direct generation of such synthetic IP/XACT descriptions. Different files are used for this specific XML representation of the hardware platform, and is identified by means of its vendor-library-name-version identifier, that is, its VLNV identifier [8].

Hanging from the root design element, there are two main sections. The entry \(<\text{spirit:componentInstances}\>\) delimits a section which contains component instances, recognized by the \(<\text{spirit:componentInstance}\>\) IP/XACT entry. A second section, defined by the entry \(<\text{spirit:interconnections}\>\) is in charge of containing the set of interconnections among the aforementioned component instances.

In the concise IP/XACT representation required by SCoPE, the component instance only requires a basic information defined by three attributes, which includes the component instance name (\(<\text{spirit:instanceName}\>\) entry) and the reference to the component (\(<\text{spirit:componentRef}\>\) entry). This component reference is done through the VLNV identifier of the component. The third entry \(<\text{spirit:vendorExtensions}\>\) is used to facilitate the automatic generation of high-level performance estimation and simulation models.

```xml
<spirit:design
xmlns:spirit="http://www.spiritconsortium.org/XMLSchema/SPIRIT/1.4"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.spiritconsortium.org/XMLSchema/SPIRIT/1.4 ">
    <spirit:vendor>teisa</spirit:vendor>
    <spirit:library>scope</spirit:library>
    <spirit:name>i_platform</spirit:name>
    <spirit:version>1.0</spirit:version>

    <spirit:componentInstances>
    ...
    </spirit:componentInstances>

    <spirit:interconnections>
    ...
    </spirit:interconnections>
</spirit:design>
```

This root element is equivalent to the top schematic representation of the hardware platform, and is identified by the component instance name as identified by the VLNV identifier of the component. The VLNV identifier serves to give a link to an IP/XACT component description which, in turn, gives again a structural description. However it does not provided a specific functional semantics to the component. In order to cover this lack, the IP/XACT format read by SCoPE uses context labels formally defined by the SPRINT project in the SCIPIV document [20]. This way, SCIPIV tags are used to identify the component type, and thus facilitate the generation of generic abstract models. SCIPIV tags can be also in component instances. This is the case of the example shown, where the \(i\) bus component instance has a context label \(<\text{spirit:isNetworkComponent}\>\).

SCIPIV tags are used for the integration of generic components in the SCoPE simulation model. SCoPE provides a default support or portfolio of generic components (processor, bus, memory, DMA, bridge, etc). If the Spirit design file contains component instances whose VLNV identifier does not fit with any component description file, they will be considered as generic if possible. Context clauses are used in this case for using the generic SCoPE components. This enables a fast and easy building of a platform specific executable model.
As well as the portfolio of components offered by SCoPE, the SCoPE user can provide and attach to the platform new components such as custom IPs and I/O devices. Moreover, the user can even override the use of generic platform components and use their own performance models. For it, a SystemC model with a TLM interface which can include behaviour and performance annotations has to be provided. In SCoPE, this model can be directly encurated at a SystemC level. Moreover, the IP/XACT support for the integration of these new components enables making an IP/XACT component declaration (by means of the entry <spirit:component>) separated in a file different from the one which contains the top design (<spirit:design> entry) and the architectural description defined by the component instances and interconnections sections. This enables a modular and cleanly extensible description of the hardware platform. The component description is identified again by a VLNV identifier, which can be referenced by a component instance. The component declaration also includes additional information which enables deconnection to the SCoPE platform structure, namely, a section for the bus interface description (<spirit:BusInterfaces>), a section for describing the mappings of the device to the memory map (<spirit:memoryMaps>), and, finally, a vendors extension section with the a SCPIPV tags to specify the type of component.

If a component instance references a component description which does not reference a behavioral model, and it is possible to infer a generic component, then the hardware platform model cannot be generated. 

The simple interconnection scheme required by SCoPE for building a fast executable performance model is reflected in requiring a simple description for the interconnections section. Specifically, SCoPE only requires the description of point-to-point interconnections among HW platform component instances. This is done by using the <spirit:interconnection> element for each interconnection. 

The interconnection is described by its name (<spirit:name>) and by the two connection extremes (using the <spirit:activeInterface> element). The tied component instances are defined by assigning the component instance name to the spirit:componentRef attribute of each spirit:activeInterface. In the following example, the component instances “i_mem” and “i_bus” are interconnected. Additionally, the IP/XACT description reflects the master or slave character of the component attached (e.g., the “i_mem” has a slave role). Connections of the type master to bus, and slave to bus (as the one in the example) can be described. This serves to reflect the simple computational scheme of a high-level simulation tool such as SCoPE, where a key for fast simulation is the typical master-slave configuration, where slaves are reduced to functional calls, and the number of masters is kept as reduced as possible.

```
<spirit:interconnection>
  <spirit:name>c2</spirit:name>
  <spirit:activeInterface
    spirit:busRef="Slave"
    spirit:componentRef="i_mem" />
</spirit:interconnection>
```

Another IP/XACT capability supported by SCoPE is the definition of configurable platforms. Through the entry <spirit:configurableElementValue>, a value of a component attribute can be fixed or even left as a parameter (_MEM_ADDR_) of the generated executable specification. Then, this parameter needs to be passed to the generated executable specification before the execution of executable specification.

### III. UML/MARTE SPECIFICATION METHODOLOGY

This work is in the context of the COMPLEX code generation framework, which takes as input an UML/MARTE system description under the UML/MARTE methodology. A detailed description of this methodology is out of the scope of this paper. Here a brief introduction of the overall methodology is given, to later focus on all the information which regards to the description of the HW platform.

The COMPLEX UML/MARTE specification methodology is a component-based approach which enables the description of the whole system, including the application, the platform, and the allocation of application components into hardware components. Moreover, a COMPLEX UML/MARTE specification enables the capture of a set of real-time features associated to the application components, and the definition of an environment, clearly separated from the system description which defines the different use cases and their related input stimuli.

A relevant feature of the COMPLEX UML/MARTE specification methodology is the separation of concerns. Specifically, this means that the different information related to the system description is separated into different views: namely, (1) a data view; (2) a functional view; (3) a concurrency and communication (C&C) view; a platform view; (4); and architectural view (5); and (6) a verification view. The former five views are devoted to the system description. Among them, the platform view and the architectural view are the ones which contain information for the extraction of the HW platform architecture, and thus for its IP/XACT description. Each view is captured as a UML package with a COMPLEX specific stereotype. Specifically, the <<PlatformView>>, <<ArchitecturalView>> stereotypes.

The platform view declares both, the SW components (RTOS, drivers) and the HW components (processors, memories, DMAs, buses, etc) which will build up the platform. It is done by including in the <<PlatformView>> package UML components with different MARTE stereotypes. Specifically, for HW components, stereotypes from the MARTE HRM subprofile <<HwBus>>, <<HwBridge>>, etc are used. The main software components are RTOS, which are captured as UML components stereotyped with the MARTE GRM <<Scheduler>> stereotype.
The architectural view contains a single component (a UML component stereotyped with the <<system>> COMPLEX stereotype) which reflects the internal architecture of the system by means of a UML composite diagram. This diagram contains:

- the application component instances (captured as parts typed as any of the components of the C&C view (stereotyped as <<RtUnit>>) and their interactions. (connectors between part ports).
- the platform (software and hardware) component instances (captured as parts typed as any of the components of the platform view) and the interconnections of HW component instances (captured as port connectors).
- the allocation of application component instances to platform component instances, such as RTOS instances and instances of computation resources, such as processors, or custom hardware. It is captured as associations with the <<allocate>> stereotype.
- information which distinguishes the methodology, such as DSE parameters and rules. They are specified as comments stereotyped with COMPLEX specific stereotypes such as <<dseScalarParameter>> and <<dseRule>>, which enable to define as tunable certain attributes of component instances.

Further information in the platform and architectural views corresponds to the values of the attributes associated to the different instances. The attribute values in the platform components are understood as default values inherited by any instance. An association to a <<dseScalarParameter>> comment can change the value of an attribute for a given instance. Among the different tunable attributes, resMult is a is used to specify in a compact way the number of instances associated to a single part in the UML composite diagram of the system architecture. Other attributes state non-functional properties, e.g., the frequency property for a HwProcessor instance, of NFP_Frequency UML/MARTE type.

The COMPLEX UML/MARTE specification methodology has been put into practice over a toolset based on MDT Papyrus [11], which is integrated in Eclipse Helios [12]. Despite this toolset does not provide specific features for the separation of views, the COMPLEX UML/MARTE methodology enables by means of the mentioned stereotyped packages. By keeping the architectural information (instances and connectors) and configuration information in a single view (architectural view), the need of any coherence check among views is eliminated.

IV. GENERATOR

A. Transformation Rules

Although the platform and architectural views contain all the information necessary for the generation of the IP/XACT HW platform description required by SCoPE, not all that information is required for the generation.

Figure 3 provides a basic view of the MARTE to IP/XACT mapping implemented by the generator.

The generator will produce a single file with the spirit:design entry. The generator does not produce files with spirit:component entries, since these are assumed to be available. However, the components are declared within the MARTE specification. Therefore, the generator basically produces the architectural description of the HW platform.

![Figure 3. Basic sketch of the MARTE to IP/XACT mapping.](image-url)

For each part typed as a component of the platform view with a stereotype corresponding to the hardware platform (HwProcessor, HwBus, etc), the generator will produce as many spirit:componentInstance entries as defined by the resMult attribute (1 by default). Therefore, parts typed as SW components are ignored. An inference of at least a single HW component instance generates the container spirit:componentInstances section. The generator directly uses the name of the name attribute of the part in the architectural view for the corresponding spirit:instanceName value. If resMult is bigger than 1, a different number is appended as suffix to distinguish instance names. By default, the generator produces a VLNV identifier and a context label for a generic SCoPE component. For instance, for a part typed with a <<HwBus>> component, it produces a pair VLNV={teisa,scope,bus,1.0} and context label <context:isNetworkComponent> which involves the usage of the generic bus of SCoPE. This immediate correspondence is kept in several cases. For instance, from a system part in the architectural view typed as <<HwProcessor>>, an IP/XACT component instance with VLNVs={teisa,scope,bus,1.0} and <context:isProcessorComponent> is produced.

The generator searches for all the connectors which relate HW component instances, discarding the remaining ones (e.g., those relating application components). For each connector, the generator produces a spirit:interconnection entry, whose spirit:name adopts the value of the name attribute of the connector. The generator checks that the connector relates two HW parts, and then produces the two spirit:activeInterface entries, one for each connector end. For each connector end, the generator gets the name of the part owning the port the connector end is tied to, and uses such name as value of the spirit:componentRef of the active...
interface. In order to produce the spirit:busRef entries, the generator needs a deeper navigation to determine the type of components tied and if it is master or a slave connection.

Finally, the generator infers from the dseScalarParameters comments, the spirit:configurableElementValue entries.

B. Generator Prototype

A prototype of the MARTE to IP/XACT generator has been implemented. The generator has been written as a set of generator templates under the standard Model to Text Language (MTL) [13]. This makes the generator open, easily extensible and portable to different code generation engines supporting MTL (e.g. Acceleo, Xpand, Jet). Specifically, the generator has been developed with Acceleo MTL [14], a code generation framework fully integrated in Eclipse Helios [12].

The generator is defined as a MTL module, the UML2 metamodel as input:

```
[module marte2ipxact('http://www.eclipse.org/uml2/3.0.0/UML')/]
```

This environment has facilitated the modular programming of the generator, through a hierarchical set of templates and navigation queries using OCL language as a complement to MTL. Notice that the COMPLEX MARTE2IPXACT generator requires a recurrent navigation of the specification from the architectural view to the platform view, and along the different elements which gets more complex in certain cases. For instance, to recognize if a certain UML connector relates HW components, the navigation goes from the end point of the connector through a port, its owner part, and then its component type. Then it checks that such component type belongs to a set of HW components, determined through a query.

The recognition of the application of MARTE stereotypes in the model is easily done through the following MTL code structure:

```
isStereotypeApplied(getApplicableStereotype('MARTE::...'));
```

The generator implements a set of basic checks, while the specification is navigated. For instance, the generation checks that there exists the two required views (platform and architecture), that the architectural view is not empty and that contains a single system component, that there is a minimum set of HW component instances in the architectural view, etc. Finally, the generator also includes a header, indicating the automatic generation, and XML comments to let track the UML/MARTE elements detected in the specification which have led to the associated IP/XACT code.

V. CONCLUSIONS

In this paper the generation of generic and concise IP/XACT descriptions from UML/MARTE system descriptions suitable for DSE of real-time complex embedded systems is addressed. Specifically, this generation reads and traverses the platform and architectural views of a COMPLEX UML/MARTE specification and produces a file with the architecture of the hardware platform in IP/XACT format. This IP/XACT code generated contains a minimum set of elements plus some functional information (SCIPIV tags) which synthetically describes the hardware platform and is directly readable by SCoPE for the automatic generation of an executable for high-level functional and performance estimation.

Future work will cover the generation of more detailed IP/XACT descriptions, enabling the generation of more detailed, and thus accurate, performance models, and moreover the assembly of platform components for actual implementation. Such work will take as reference the IP/XACT descriptions admitted by a Magillem generation tool. This tool encourages generators able to produce the aforementioned detailed SystemC virtual platforms.

REFERENCES

[17] www.teisa.unican.es/scope