# Combining SystemC, IP-XACT and UML/MARTE in model-based SoC design

Jean-François Le Tallec, Julien Deantoni, Robert de Simone, Benoît Ferrero, Frédéric Mallet, Laurent Maillet-Contoz

# Combining SystemC, IP-XACT and UML/MARTE in model-based SoC design

Jean-François Le Tallec*†, Julien DeAntoni*†, Robert de Simone†, Benoît Ferrero†, Frédéric Mallet*† and Laurent Maillet-Contoz‡
*UNS, Nice Sophia-Antipolis, France †AOSTE Team, INRIA Rocquencourt / INRIA Sophia Antipolis / Laboratoire I3S, France ‡STMicroelectronics, Crolles, France

***Abstract*—Modern SoC design may rely on models, or on high-level description languages. Although very close, the benefits obtained from either sides can be substantially different (and mismatch may occur). The IP-Xact formalism, now a standard (IEEE 1685), was introduced to help assemble component IP from distinct sources into an integrated design. Components could be expressed in high-level HDLs such as SystemC, so should be the full design after translation. Experience shows that in fact this is hardly the case, specially in publicly available methods and tools. The present contribution goes one step into linking SystemC designs to their IP-Xact structural representation by translation. It then exports the resulting IP-Xact model into the UML/MARTE profile modeling framework, to allow to annotating existing models with additional information (again in a publicly available fashion, as opposed to vendor extensions). Even if our approach is still far from being complete, it bridges a number of gaps induce by the combined uses of SystemC and IP-Xact.**

## I. Introduction

Design of digital circuits was always involved with many representation formalisms. Some of them are formal or engineering *models* of such circuits, some are *programming* languages initially aimed at simulating such models. Combinatorial and sequential netlists, Mealy and Moore FSMs, process networks fall into the first range; Verilog, VHDL and SystemC language in the second. In particular, SystemC is becoming a de-facto standard for more abstract representation of Systems-on-Chip at higher level.

While modeling and programming formalisms do complement one another, they do *not* always coincide as well as one should hope. This is certainly so because they follow distinct goals. Analysis models aim at faithfully representing physical objects, while allowing abstractions from details, or considering additional relevant views (consumption, timing closure). Programming languages promote execution efficiency to allow simulation of very large circuits, sometimes at the cost of modeling accuracy.

The interplay of models and programming languages has thus always been a big issue for correct design (as for instance, in synthesizability requirements). The matter has been renewed with the advent of model-driven engineering techniques (based on UML or similar formalisms), which should provide a middle point between the demands of formal models and mathematical properties on the one hand, programing and design efficiency on the other one. Currently the IP-Xact standard initiative (initiated in the Spirit consortium and now handled at Accelera) aims at providing a dedicated ADL (Architecture Description Language) to model the hierarchical structure of SoCs and interconnects, and support representation of additional feature aspects in a normalized way. Ties between IP-Xact and SystemC should be obvious, but currently they remain rather obscure and implicit, hidden in proprietary tools and implementations.

Our present work consists of two parts. First, we capitalize of previous efforts by others to extract a complete structural model in IP-Xact from a SystemC program during its so-called elaboration phase; it provides a way to compose and assemble SystemC programs more easily and modularly. Second, we consider the representation of these IP-Xact structural descriptions into the MARTE profile of UML. MARTE is the UML extension meant to deal with Real-Time Embedded aspects of systems, which suits closely digital circuits. MARTE allows also to represent various non-functional aspects (timing, consumption,...), in a standardized fashion. This comes as an extension to IP-Xact feature for introducing extra information, usually restricted to proprietary *Vendor Extensions*. In MARTE, additional features will become integral public parts of the model. We choose to take test cases in external library to test the robustness of our tool confronted to different coding styles. For that purpose, we stress our tools on the SoCLib library and generic SystemC examples.

The benefits of our results are two-fold. To the best of our knowledge no explicit translation from SystemC to IP-Xact that would preserve and even identify and promote the hierarchical structure was available to this date. Also, the potential ability of a dedicated format such as IP-Xact to support annotations for representation of additional feature aspects was again not something available at this stage. Our contributions should help in the future experiment with various such additions, with the main impact of using the same basic skeleton model to add or extract different feature aspects that can efficiently be dealt with for mathematical analysis by a large spectrum of existing tools. In this paper, we will present in section II different existing technologies on which our approach/tool is relying on. Then, we will explain more precisely in section III-A how we built it and what possibilities it permits. We will conclude in section IV by enumerating earthly perspectives for the future works.

This paper has been partially supported by the French ANR project HELP (ANR-09-SEGI-006).

## II. Overview

### A. SystemC

SystemC[1] is a Hardware Description Language, meant to represent circuits and SoCs at various levels of abstraction (in particular at RTL and TLM levels) [5]. Conceived as library extension of C++, it benefits from data types and compiler environments from this host language. It adds provisions for parallel threads, signal wires, clocks, timing features, low and high level communication mechanisms. These extensions are semantically dealt with by a specific non-preemptive scheduling discipline, for simulation of designs. Simulation goes through two successive phases. First, the *elaboration* phase instantiates the parallel processes and the static network of components (*i.e.,* the sc_modules and their interconnects), as requested by the program in an initial object creation part. Then the actual behavioral simulation itself takes place, combining the individual component bodies according to the scheduler. Simulation itself can be untimed (causal), cycle-accurate, approximate or even loosely timed. SystemC designs are usually strongly influenced by formal modeling with so-called Models of Computation and Communication (MoCC). It can be traced back to the pioneering work of Daniel Gajski and fellow co-authors on SpecC [4]. Also the seminal SystemC reference [5] mentions Kahn Process Networks. Modeling Kahn Process Networks, CSP and Synchronous Reactive systems in SystemC is also explored in [6]. Heterogeneous modeling in SystemC is addressed in [12], [17]. However, the connection remains implicit, or even unclear, when looking at actual programs. The elaboration phase can be seen as the construction of the structural parts of models (process networks and interconnect topology), while component behaviors can usually be seen as some form of interacting FSMs (possibly with timing). One of our goal is to extract such modeling information in an explicit way (in IP-Xact for structure and interfaces, in later work in UML MARTE for component behaviors).

### B. IP-XACT

IP-Xact[2] is an XML format dedicated to the design, integration and reuse of IP (Intellectual Property) components from various vendors into larger designs by enabling automatic configuration and integration through industrial tools. IP-Xact was created by the Spirit Consortium and is now handled by Accelera, recently becoming the IEEE 1685 standard. IP-Xact provides a means to describe information relative to the structural part of a design. A *component definition* describes the interface of a component and contains a reference to a file that describes the implementation of its functional behavior in a specific HDL. A *design* is a set of component instances and links that interconnect these instances.

Because correctness is an essential concern for integration and reuse, functional and extra-functional properties can be embedded in a component definition or a design by using so called *vendor's extensions*. The format and the kind of information contained in a vendor extension is not part of the standard, and thus can not be shared across different vendors tools. This prevents integration of IP developed individually in different tools. Another limitation is the lack of mechanism to easily handle parameterizable structures. These are the main reasons why we believe that more generic definition formalisms such as those found in UML could be useful here.

### C. UML/MARTE

MARTE[3] [11] is the profile extension to the UML modeling framework aimed to deal with Modeling and Analysis of Real-Time Embedded systems. It inherits from UML several diagrammatic styles (such as components for structural design, hierarchical FSMs and data-flow activity blocks for behaviors, amongst several others). It also provides standard annotation features (there called "stereotypes") to represent functional and extra-functional properties, as well as a way for the user to introduce others. As a result, further specialization of MARTE could allow encoding IP-Xact notions, as well as easy extensions like the IP-Xact vendor's extensions, but this time publicly exposed. The emphasis put in model-driven engineering (as around UML) on model transformation should then allow the relevant information to be directed to whichever proprietary tool may understand how to deal with them. Furthermore, it embeds a logical time model [2] to allow description of constraints to formally link different views of a model such as consumption, timing or safety and possibly a TLM to RTL refinement way [7]. Repetitive structure modeling (RSM) package defined in the standard UML MARTE profile also simplifies the representation of complex parameterizable structure.

### D. Our approach

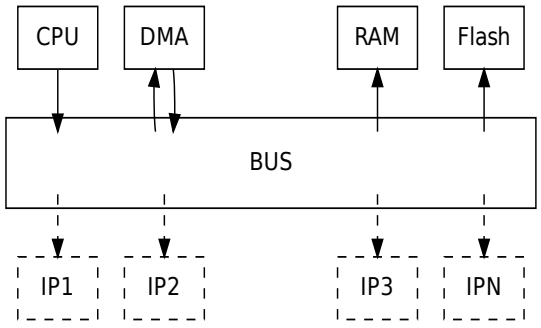


Figure 1. Common SoC Design

Classically, SoC platforms are depicted in a block diagram fashion as in figure 1. But such an informal picture is not a true model, as it lacks usually important information. As the purpose of IP-Xact is to promote platform assembly using IP blocks, it requires further information of the port interface and interconnect features. Furthermore, MARTE profile could provide graphical editing and formal support for annotation in

[1]**The Open SystemC Initiative** www.systemc.org

[2]**The Spirit Consortium** www.spiritconsortium.org

[3]**UML Profile for MARTE** www.omgmarte.org

distinct non-functional domains (e.g., power domain). It is our main goal to model such a platform as in figure 1, but with possibly all the details needed to understand it fully (such as the protocol description).

Thus, our global vision is that composite SoC design descriptions integrating IPs from different sources should be provided in the IP-Xact fashion, certainly extended with useful annotation features in a way less obscure than "vendor extensions". Generic model-driven engineering, in our case embodied in the UML/ MARTE profiles, could help prototyping such extensions. In turn, SystemC could be used as an efficient target simulation language, with programs benefiting from "correct-by-construction" features obtained by early model analysis. Individual component behaviors could also be provided as SystemC descriptions (or with formal MoCCs).

Still, there are currently many more design descriptions directly provided in SystemC, and lifting them back up into model-level descriptions in IP-Xact could greatly help populate the design methodology. This is why a model extraction scheme from SystemC programs is so desirable. It is the main topic of this paper, pointing out the potential difficulties as well as openings of the approach.

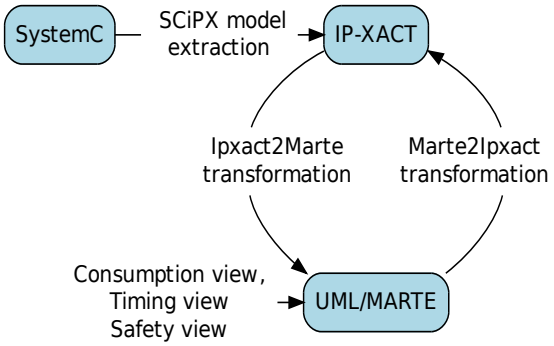


Figure 2. Global Tool Flow

So, we consider the automated bottom-up translation from SystemC to UML/ MARTE, through IP-Xact, as shown in figure 2. To translate from SystemC to IP-Xact, we extract information from the original program using two different methods. The first method consists in running a simulation along its *elaboration* phase, and recover from that a model for the fully instantiated network of components; this part is directly indebted to former work on PinaVM (see below). The second method uses static analysis introspection of the SystemC code to recover structural syntactic information that were not preserved during the compilation process; it relies heavily on the Doxygen C$^{++}$ static analyzer (again described below). After these two were conducted in parallel, we merge their results into a complete IP-Xact view that conforms to the IEEE 1685 standard description. We defined and realized a second translation, this time from IP-Xact to a corresponding subset of UML/MARTE. It uses advanced model transformation techniques. The main point here is that resulting models could then easily be extended to allow supplementary annotation features (such as timing or low-power), in a public representation format (as opposed to current vendor extensions in IP-Xact). While this part is still underdeveloped, we can envision potential further work, for instance in describing early abstract designs that are "bus protocol agnostic", and can then be refined to accommodate AMBA or OCP-IP requirements. Corresponding vendor extensions could then automatically be synthesized in a way compliant to IP-Xact. We have already implemented a reverse translation from UML/MARTE to IP-Xact, currently bare regarding such extended annotations, but which could easily support them once defined.

### *E. Inspirational works*

As mentioned above, the translation from SystemC to IP-Xact requires two distinct methods to provide results, one based on run-time simulation of the elaboration phase, one based on static analysis of the source code. We now describe in turn the two already implemented methods, due to others, upon which we built our process by proper modifications and enhancements. One main highlight of our approach in SCiPX is to combine both techniques to get the final combined models.

*a) Elaboration run-time: PinaVM:* PinaVM [9], [10] is a SystemC front-End based on the LLVM compiler. It provides an abstract representation of a SystemC programs after the elaboration phase (*i.e.,* it provides an abstract view of the network of components). It also provides an extendable project structure to plug specific backend, allowing the manipulation of this representation. The abstract representation provided by PinaVM is well suitable to verification / validation of the behavioral part but, due to the use of precompiled information, lacks of static information. For instance, as for every C$^{++}$ programs, the precompilation remove information on the attribute names and mangle information on attribute types. Moreover, PinaVM does not focus on architectural concern and does not directly allow the translation of the representation into a model view (either IP-Xact or UML-based). We chose to take advantage of the already existing project structure and to develop a back-end that have additional information from static SystemC code analysis.

*b) Static analysis: Doxygen:* Doxygen[4] is originally a documentation generator but can also be used as a static code source analyzer. It permits us to retrieve information about the component definition. The goal is to retrieve information that lacks in PinaVM like the name of the attributes, etc. Of course, Doxygen can not give any relevant information about the elaboration phase or the component network. It appears that PinaVM and Doxygen can fit together and provide enough information for the construction of both IP-Xact component definition and IP-Xact design. From these IP-Xact description, it is then possible to make an import into UML/ MARTE. These steps are detailed in the next sections.

### *F. Related Works*

The connection of SystemC to more engineering models and meta-modeling frameworks (such as UML) is also not entirely new [15], [3]. Early connections between IP-Xact and

[4]**Doxygen** www.doxygen.org

UML models were presented in [14], [16], [8]. The present work is original in that it provides a general translation of the structural aspects of SystemC programs into engineering models, **and** allows this translation to be later completed with various annotation aspects that capture more views of the design (like behavioral aspects for instance). While we focus on the code to model transformation at this point, to benefit from existing collections of SystemC programs, we are eager to reverse transformations so as to "synthesize" SystemC programs from high-level models (something which would sound obvious from many claims in the literature, but is not yet achieved by any means to the best of our knowledge).

## III. Detailed tool description

### A. SCiPX description

We now explain why our tool requires combination of dynamic execution (elaboration phase in PinaVM) together with static analysis (Doxygen). We illustrate this on the simple design depicted in Figure 3. *c1,c2* are two components of the design, respectively instances of *M1* and *M2*. *p1,p2,p1',p2'* are ports of the same type *T1*. *c1* contains *p1* and *p2*; *c2* contains *p1'* and *p2'*. *p1* is connected to *p1'* through *channel1* *p2* is connected to *p2'* through *channel2*. While this information alone allows to create a design view, more information about the admissible number of port *M1* and *M2* instances can accept should also be provided (not always "2" on all instances). The dynamic runtime analysis recovers the involved instances and their port instances as well as the link between the ports. However, they loose information like port naming, exact type of the definition (two ports or an array of size two makes no differences), and other attributes of the component. In the previous example, all ports have the same type and are, after pre compilation identified by their memory address. It is then impossible to deduce which address corresponds to which name. More generally, the name loss makes impossible to differentiate component ports of a same type. The static approach on the other hand, can fill up attribute name information. But, in case of dynamic (and / or conditional) instantiation, is not able to retrieve the component network. So we have to combine static and dynamic approaches in the proper way.

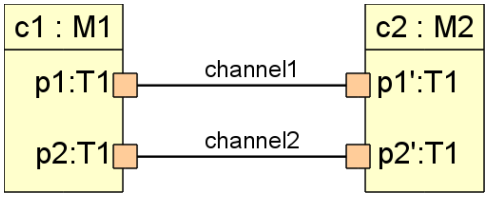


Figure 3. Simple example of two communicating SystemC sc_modules

First, retrieve static knowledge about the SystemC component definition; second, retrieve information about component network resulting from the elaboration phase (*i.e.,* need to retrieve run time information); finally merge all these pieces of information in order to have enough knowledge to create an IP-Xact system from the SystemC code.

This process is depicted in figure 4. First, we run Doxygen over a SystemC source code where macro has been replaced in order to get native C++ source code *(link n°1)*. It results in an xml file that contains all class definitions together with inheritance and containment.

This xml file can then be analysed to extract the various component definition (*i.e.,* component, interface, attributes, etc) *(link n°2)*. From the same xml file, a new main program is produced. It does not modify the existing SystemC *sc_main* code. The goal of this generated *sc_main* is to collect offset of each attribute of each components identified by Doxygen, using the standard ANSI C macro *offsetof()*. Technically, the offset represents the difference between the attribute address and the address of its class. A new xml file result of this. It represents the mapping table between an attribute name and its offset *(link n°3)*. It is important to notice that this mapping depends on compilers/compiling options. Consequently, this code must be compiled with exactly the same options and compiler than the one used by PinaVM and on the same execution platform.

In the next step, the original SystemC program is executed until the *end of elaboration* by PinaVM. When this point is reached, PinaVM returns to the back-end plug-in an access to its component network internal representation. Now, using the available attribute offset mapping table built before, information about attribute names (and furthermore port names) can be asserted by comparing the address of a port, the address of the owner component and the offset*(link n°4)*.

In order to retrieve the attribute types in a human readable form, we used RTTI (Run-time type information), a way to keep information about an object's data type in memory at runtime. This has to be done at runtime to make sure types are retrieved even when using template components.It becomes then possible to build a complete IP-Xact design linked to the appropriate component definitions *(link n°5)*.

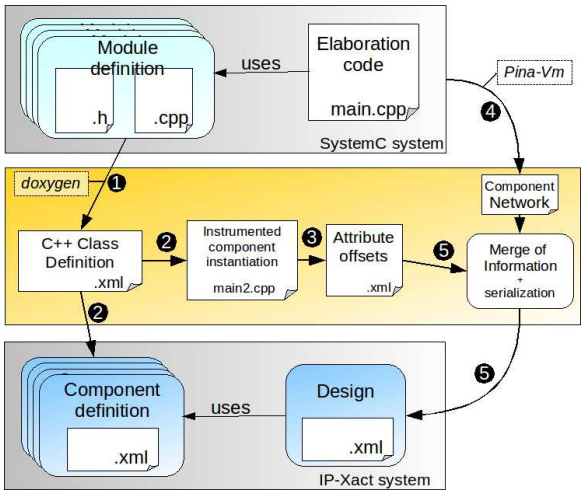


Figure 4. SystemC to IP-XACT details

The following source code illustrates a number of declarations that our SCiPX[5] tool is able to handle. The first

[5] **AOSTE** www-sop.inria.fr/aoste

piece of code is an implementation of a *Source* component. Assume that there is the mirror implementation for the *Target* component.

As highlighted on line 2,3 and 4 we deal with dynamic declaration of ports as well as with dynamic port array. As SystemC do not allow instantiation when the elaboration phases is done, any dynamic ports must have been created. Knowing the address of these port objects and the reference of port pointers in a component, a simple comparison between port address and port pointer value permits us to map dynamic ports. Dynamic port array is very similar to dynamic port. The whole array is declared at the same time. The first port of the array is detected as a simple dynamic port. A simple comparison between unmapped port addresses and the size of the port permits us to link unmapped ports to the first port of it array.

```
1  SC_MODULE(Source){
2    sc_out<statType> statOUT;
3    sc_out<dynType> * dynOUT;
4    sc_out<arrayType> * arrayOUT;
5    SC_HAS_PROCESS(Source);
6    Source(sc_module_name name, unsigned arraySize){
7      dynOUT = new sc_out<dynType>;
8      arrayOUT = new sc_out<arrayType>[arraySize];
9    }
10 };
```

The last piece of code is the instantiation of both *Source* and *Target* into an *sc_main*. Such a declaration is valid as it is simple C++. But, it could not be synthesized by classical static analysis (*i.e.,* without symbolic execution) because it contains loops and conditional statements. Combining the runtime and static approach permits us to overcome this limitation.

```
1  int sc_main(int argc, char **argv){
2    unsigned arraySize=2;
3    bool invertArray=true;
4    Source sourceInst("sourceInst", arraySize);
5    Target targetInst("targetInst", arraySize);
6    sc_signal<statType> statSIG;
7    sc_signal<dynType> dynSIG;
8    sc_signal<arrayType> * arraySIG;
9    arraySIG = new sc_signal<arrayType>[arraySize];
10   sourceInst.statOUT.bind(statSIG);
11   sourceInst.dynOUT->bind(dynSIG);
12   for(unsigned i=0;i<arraySize;i++)
13     sourceInst.arrayOUT[i].bind(arraySIG[i]);
14   targetInst.statIN.bind(statSIG);
15   targetInst.dynIN->bind(dynSIG);
16   if(!invertArray) for(unsigned i=0;i<arraySize;i++)
17     targetInst.arrayIN[i].bind(arraySIG[i]);
18   else for(unsigned i=0;i<arraySize;i++)
19     targetInst.arrayIN[i].bind(arraySIG[arraySize-i-1]);
20   sc_start();
21   return 0;
22 }
```

We ran SCiPX on a number of examples found in the standard SystemC library (ver 2.2), and were able to correctly generate IP-Xact representation for most. Examples that did not pass the processing all contained the same feature: to specify connections, these examples do not use *sc_port_base* which is the information handled in SystemC internals. Thus, asking SystemC internals do not reveal these kind of connections as it is not saved during the *elaboration* phase. One can argue that the problem comes from a poor SystemC coding style on these examples, but we are considering a work-around this.

### B. Ipxact2Marte and Marte2Ipxact

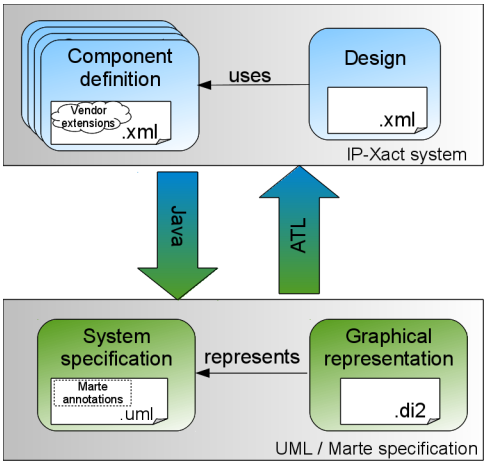


Figure 5. There and back between IP-XACT and UML / MARTE

IP-Xact allows the specification of additional, NFP (Non-Functional Property) annotations only through so-called vendor extentions, which remain private and not standardized. We used the UML standard profile MARTE as a substitute, where such NFP annotations can be defined and charaterized in an open public way and then possibly translated to IP-Xact in any vendir format that is made available.

Another advantage of MARTE is the possibility to use already existing tools to realize / visualize / edit an IP-Xact design. For visualization purpose, in addition of the IP-Xact to MARTE transformation, we developed an automatic $di2$ generator which permits us to display the extracted design in Papyrus [6], a free and open source UML/ MARTE environment. Another side effect is that behavioral and NFPs can be analysed by different available tools directly at the model level. For instance, works around data flow network allow the computation of static periodical schedule and results can be injected back in the MARTE model.

The transformation from IP-Xact to MARTE is a reverse implementation of the mapping described in [1]. Together with this previous work, we provide a mean to travel between IP-Xact and MARTE. It is then possible to import component definitions and (possibly) a design, to modify (or create) the design in a UML editor, to annotate the design, to analyze it and then to re-generate IP-Xact representation of our extended design keeping retro compatibility with IP-Xact tools. For now, we only prototype a specific vendor extension transformation

[6]**Papyrus** www.papyrusuml.com

from and to IP-Xact. Because vendor extension is a very general extension mechanism, there is no way to provide a generic transformation for that purpose. However, doing it on various vendor extensions promotes them to a first order concern, publicly exposed while keeping the advantage of the algorithm / tool in charge of the extension.

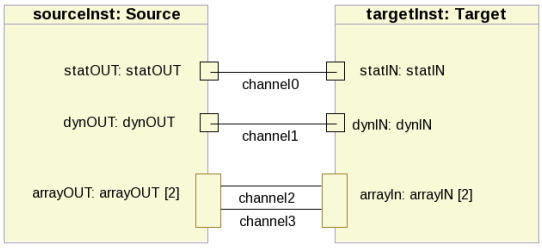


Figure 6. MARTE diagram of the example of section III-A

After processing the source code of the example of the section III-A, we have the MARTE model and diagram whose a screen-shot is given in figure 6. To stress our implementation, we ran our tool on the SoCLib SystemC library. We succeeded in generating IP-Xact component definitions and transforming then into UML/ MARTE components as well. Then, we were able to graphically create various designs in Papyrus MARTE with these extracted components. Then, we were able to generate IP-Xact design file from MARTE. As SCiPX, tools to achieve these transformations are also avaliable on AOSTE webpage.

## IV. Conclusion and Future Work

An ideal SoC design flow would allow to assemble well understood components models, through well defined structural interfaces. Early analysis at model level could help guarantee correctness properties, in the functional and non-functional domains. IP-Xact goes in this direction by providing clear interfacing and configuration mechanisms, but extra information is currently provided as vendor extensions outside the range of the standard. Also, component models are defered elsewhere, and in fact provided as HDL code. SystemC ambitions to allow the representation of MoCCs at several levels, but as it relies on code there is no guarantee that programs match this need (even though the language itself is expressive enough for that of course). Since there are currently many more SystemC designs than IP-Xact models openly available, we considered extraction of IP-Xact structural representation from composite SystemC designs, our first contribution. To allow further model annotations beyond vendor extensions we also studied further transformations to UML MARTE profile. Next one should provide useful examples of such annotations.

Currently we have not extended the model extraction from SystemC to component behaviors themselves (as PinaVM does to some extent). As IP-XACT does not provide features for behavior modeling, this would have to target directly MARTE behavioral models (which then are amply defined, with hierarchical FSMs for control and data-flow activity diagrams for netlists. While it could be interesting to check whether extracted models there reflect the designer's intuition, we strongly believe that proper design flow should produce efficient code representation in SystemC from MoCCs, not the other way around. Principles of abstraction/refinement (between RTL and various TLM sublevels), as well as timing level accuracy, would certainly find a more natural formulation at model level than code level.

## References


[1] C. André, F. Mallet, A.M. Khan, R. De Simone, and I.S.A. Méditerranée. Modeling SPIRIT IP-XACT with UML MARTE. In *Conf. on Design, Automation and Test in Europe (DATE), MARTE Workshop.(March 2008)*, volume 35, page 40, 2008.

[2] Charles André, Julien DeAntoni, Frédéric Mallet, and Robert de Simone. The time model of logical clocks available in the OMG MARTE profile. In Sandeep K. Shukla and Jean-Pierre Talpin, editors, *Synthesis of Embedded Software: Frameworks and Methodologies for Correctness by Construction*, chapter 7, pages 201–227. Springer Science+Business Media, LLC 2010, July 2010.

[3] Rabie Ben Atitallah, Eric Piel, Julien Taillard, Smail Niar, and Jean Luc Dekeyser. From high level mpsoc description to systemc code generation. In *International ModEasy'07 Workshop of the Forum on specification and Design Languages (FDL'07*, 2007.

[4] D. D. Gajski, J. Zhu, R. Doemer, A. Gerstlauer, and S. Zhao. *SpecC: Specification Language and Methodology*. Kluwer Academic Publishers, 2000.

[5] Th. Grötker, S. Liao, G. Martin, and S. Swan. *System Design with SystemC*. Kluwer Academic Publishers, 2002.

[6] Fernando Herrera, Pablo Sánchez, and Eugenio Villar. Modeling of CSP, KPN and SR systems with systemC. 2004.

[7] Jean-François Le Tallec and Julien DeAntoni. Toward a TLM to RTL refinement: a formal approach. In *Proc. of the 3rd Junior Researcher W. on Real-Time Computing, JRWRTC'09, in conjunction with the 17th Int. Conf. on Real-Time and Network Systems, RTNS'09*, Paris, France, October 2009.

[8] F. Mallet, Ch. André, and R. de Simone. Ip-xact components with abstract time characterization. In *Advances in Design Methods from Modeling Languages for Embedded Systems and SoCs*, volume 63 of *Lecture Notes in Electrical Engineering*. Springer Sciences+Business, 2010.

[9] Kevin Marquet and Matthieu Moy. PinaVM: a SystemC front-end based on an executable intermediate representation. Technical Report TR-2010-8, Verimag, 2010.

[10] Kevin Marquet, Matthieu Moy, and Bageshri Karkare. A theoretical and experimental review of SystemC front-ends. Technical Report TR-2010-4, Verimag, 2010.

[11] OMG. *UML Profile for MARTE, v1.0*. Object Management Group, Nov. 2009. OMG document number: formal/09-11-02.

[12] Hiren D. Patel and Sandeep K. Shukla. Towards a heterogeneous simulation kernel for system level models: A SystemC kernel for synchronous data flow models. *VLSI, IEEE Computer Society Annual Symposium on*, 2004.

[13] M.-A. Peraldi-Frati and Y. Sorel. From high-level modelling of time in MARTE to real-time scheduling analysis. In *MoDELS'08 W. on Model Based Architecting and Construction of Embedded Systems on ACES-MB*, pages 129–143, Toulouse, France, September 2008.

[14] S. Revol, S. Taha, F. Terrier, A. Clouard, S. Gerard, A. Radermacher, and JL. Dekeyser. Unifying hw analysis and soc design flows by bridging two key standards: Uml and ip-xact. In *Distributed Embedded Systems: Design, Middleware and Resources*, volume 271 of *IFIP International Federation for Information Processing*. Springer, 2008.

[15] E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio. A uml 2.0 profile for systemc: toward high-level soc design. In *EMSOFT '05: Proceedings of the 5th ACM international conference on Embedded software*. ACM, 2005.

[16] T. Schattkowsky, T. Xie, and W. Mueller. A uml frontend for ip-xact-based ip management. In *Conference in Design Automation and Test in Europe (DATE'09)*, 2009.

[17] J. Zhu, I. Sander, and A. Jantsch. HetMoC: Heterogeneous modeling in SystemC. In *Proceedings of the Forum on Design Languages (FDL'2010)*, 2010.