Marte Timing Requirement and Spirit IP-XACT

Aamir Mehmood Khan, Frédéric Mallet, Charles André, Robert de Simone

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Marte Timing Requirement and Spirit IP-XACT

Aamir Mehmood Khan — Frédéric Mallet — Charles André — Robert de Simone

N° 6647 — version 2

initial version Septembre 2008 — revised version Juin 2009

ISSN 0249-6399 ISRN INRIA/RR--6647--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
SOPHIA ANTIPOLIS - MÉDITERRANÉE

Marte Timing Requirement and Spirit IP-XACT

Aamir Mehmood Khan , Frédéric Mallet* , Charles André* ,
Robert de Simone

Thème : Systèmes embarqués et temps réel
Équipe-Projet Aoste

Rapport de recherche n° 6647 — version 2 — initial version Septembre 2008
— revised version Juin 2009 — 26 pages

Abstract: Large System-on-Chips are built by assembly of existing components modeled at different representation levels (TLM, RTL). The IP-XACT standard was developed to ease interoperability of IPs from different vendors. Currently, it focuses on structural, typing and memory-related information and does not fully face behavioral and timing representation issues. UML MARTE profile explicitly focuses on the rich expression of time (physical or logical).

Combining both specifications allows for introducing a higher timed representation level and for extending IP-XACT with timing characteristics. Such timing characteristics are used to validate IP-XACT models by composing component behaviors and compare existing TLM and RTL implementations.

Key-words: MARTE, IP-XACT, interoperability, IP, timing requirement

* Université de Nice Sophia Antipolis

Centre de recherche INRIA Sophia Antipolis – Méditerranée
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex
Téléphone : +33 4 92 38 77 77 — Télécopie : +33 4 92 38 77 65

Marte pour les exigences temporelles de Spirit IP-XACT

Résumé : Les Systèmes sur puces (SoCs) sont contruits par assemblage de composants disponibles à des niveaux de modélisation différents (TLM, RTL). Le standard IP-XACT a été développé pour faciliter l'interopérabilité des composants (IPs) assemblés et provenants par différents vendeurs. Jusqu'à présent, ce standard ne s'intéresse qu'aux aspects structurels des composants, son interface, ses ports et leur type, les informations liées à la mémoire et néglige le comportement et les caractéristiques temporelles. Le modèle de temps du profil UML MARTE, pour sa part, se concentre sur l'expression de propriétés temporelles.

En combinant les deux spécifications, on introduit dans IP-XACT un niveau plus abstrait temporisé. Cela permet également d'ajouter à IP-XACT des aspects concernants les exigences temporelles. Ces exigences servent à valider des modèles IP-XACT par composition des comportements individuels. Elles servent également, à établir des propriétés communes entre les implémentations de différents niveaux.

Mots-clés : MARTE, IP-XACT, interoperabilité, IP, exigences temporelles

1 Introduction

Reuse and integration of heterogeneous Intellectual Properties (IPs) from multiple vendors is a major issue of System-on-Chip (SoC) design. Existing tools attempt to validate assembled designs by global co-simulation at the implementation level. This fails more and more due to the increasing complexity and size of actual SoCs. Thus, there is a clear demand for a multi-level description of SoC, with verification, analysis and optimization possibly conducted at the various modeling levels. In particular, analysis of general platform partitioning based on a coarse abstraction of IP components is highly looked after. This requires interoperability of IP components described at the corresponding stages, and the use of traceability to switch between different abstraction layers. Although this is partially promoted by emerging standards, it is still insufficiently supported by current methodologies. Such standards include SystemC [1], IP-XACT [2], OpenAccess API [3], and also recent Unified Modeling Language (UML [4]) based standards like the UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE [5]) that specifically targets real-time and embedded systems.

System Modeling requires representation of both structural/architectural/platform-based aspects at different levels of abstraction as well as behavioral/functional aspects possibly considering time-related viewpoints such as untimed/asynchronous-/causal, logical synchronous/cycle-accurate or physical/timed models. Semantics provides behavioral meaning to the whole systems from the combination of the behavior of its components.

For *system structure* representation, UML uses class, component, and composite structure diagrams, while SysML [6] uses block diagrams. Tools like Esterel Studio, and virtual platforms like CoWare, Synopsys CoreAssembler and ARM RealView, introduce their own architecture diagrams. IP-XACT provides some ADL (Architecture Description Language) features for externally visible common interfaces and recognized state encodings, together with administrative information (ownership, tool chains, versioning ...). IP-XACT has its own XML syntax, for specification of IP meta-data and tool interfaces.

For *component behavior* representation, SystemC provides programming libraries to represent IP component behavior at different abstraction levels, from Transaction Level Modeling (TLM) to RTL but it requires additional support for architecture modeling. Indeed, it does not provide yet support for the whole range of abstraction levels usually considered [7]. In that way, the commonly used SystemC levels, such as TLM/PV, TLM/CC, or RTL, are complementary to UML state-machine, sequence and activity diagrams that can be thought as closer to TLM/CP (Communicating Processes) level models. MARTE could be seen as introducing the relevant *timed* version at this level (like PVT does for PV - Programmer View), through logical time and abstract user-defined clock threads.

However, MARTE is general-purpose and lacks some specific features of IP-XACT. Therefore, we propose to extend MARTE with IP-XACT-specific stereotypes. Relying on a profiling approach allows easy creation and extension of model editors for IP-XACT based on existing UML graphical editors (*e.g.,* Eclipse UML, MagicDraw by NoMagic, Rational Software Architect by IBM, Artisan, Papyrus, ...). Selected UML structural models are extended with IP-XACT capabilities and UML behavior models complement the current IP-XACT-SystemC

specifications. MARTE time model adds the necessary abilities to specify time requirements. This combined approach makes IP-XACT part of a more abstract level modeling environment. Transformation engine must be built to import existing IP-XACT models into this environment but also to export models to IP-XACT-dedicated tools.

We chose to do this by specializing the MARTE profile, which already provides a number of modeling features for extra-functional aspects (such as logical timing elements). We only define new stereotypes when nothing equivalent exist either in standard UML or in MARTE. One immediate bonus is to benefit from MARTE time model to attach time/behavioral information to IP-XACT models. As an example, we include the time information extracted from the IP datasheets and show how this information can be used to generate test-benches tailored for the different abstraction levels.

Related work: There have been several propositions to use UML in SoC Design [8, 9] including usage of profiling mechanisms (UML for SoC [10, 11], Omega-RT [12] and UML for SystemC [13]). There are also some combined UML/SysML-based environments to support analysis and produce SystemC outputs [14]. However, our work specifically focuses on the interoperability among IP-XACT models and makes an extensive use of the MARTE profile and its time model.

Some preliminary works [15, 16, 17] have started to consider solutions to model IP-XACT designs in general purpose modeling languages like UML with or without the support of MARTE profile. These approaches mostly focus on structural aspects, whereas we also consider behavior and time information of IPs.

The recent UML profile for ESL [18, 19] supports bidirectional transformations between UML and IP-XACT as well as the generation of SystemC code skeletons based on the register map information provided by IP-XACT. This profile focuses on TLM models and abstracts away all the RTL-related information. It was designed to provide a good integration with ST Microelectronics TLM design flow. Our work is complementary. Revol focuses on the structural aspects while we also look at the relationship with the behavior.

Our contribution is threefold. First, we show that we can make an extensive use of the MARTE profile to generate a complete IP-XACT specification from a UML model. Second, we use the MARTE time model to add timing information to IP-XACT models. Last, we check the conformity of TLM and RTL implementations with the added time requirements.

To achieve the first goal, we have built a UML metamodel of IP-XACT 1.4. Section 2 uses an excerpt from this metamodel to introduce IP-XACT. Then, section 3 gives an overview of MARTE concepts used in our approach. Section 4 describes the structural mapping of UML-based models to IP-XACT. Section 5 proposes an extension of IP-XACT to integrate time information and gives an operational process to check the conformity of TLM/RTL candidate implementations through the use of observers and testbenches. As a running example, we use the IP-XACT specification of the Leon2 architecture, released as part of the IP-XACT 1.4 RC1 distribution package (`http://www.spiritconsortium.org`).

2 SPIRIT IP-XACT metamodel

The IP-XACT XML schema definition (XSD) is the core of the IP-XACT specification. It contains seven top-level schema definitions, each of which defines a different kind of object: Component, Bus definition, Abstraction definition, Design, Abstractor, Generator chain, and Configuration. This section introduces our *domain view* that considers only the first four kinds. A domain view is a technology-independent representation of domain concepts and allows interactions with domain experts, which are not necessarily familiar with the UML profiling mechanism. The domain view also serves as a reference model to ensure that all concepts have been implemented in the chosen technology, *i.e.,* a UML profile in our case. Building a domain view before building a profile is considered as best practice in the profiling community [20].

It is worth mentioning that the domain view presented here differs from the Ecore metamodel of Section 4.4, which is automatically generated from the IP-XACT XSD description. Using an automatically generated Ecore metamodel guarantees that IP-XACT files produced by our transformation are XMI-compatible with other IP-XACT tools.

2.1 Component

Component is the basic model element in the SPIRIT IP-XACT. Every IP is described as a component without distinction of type, whether it represents a computation core (processor, co-processor, DSP), a peripheral device (DMA controller, timer, UART), a storage element (memory and cache), or an interconnect (simple bus, multi-layer bus, cross-bar, network-on-chip).

Figure 1 shows the main features of components as considered in IP-XACT, their interface and their memory hierarchy. A component identifier (also known as VLNV) is unique, it gives the name of the component, the containing library, the vendor and the version number. A textual description can be added to precise the intended role of the component. A component also contains a precise model of the memory hierarchy: address spaces and memory mappings.

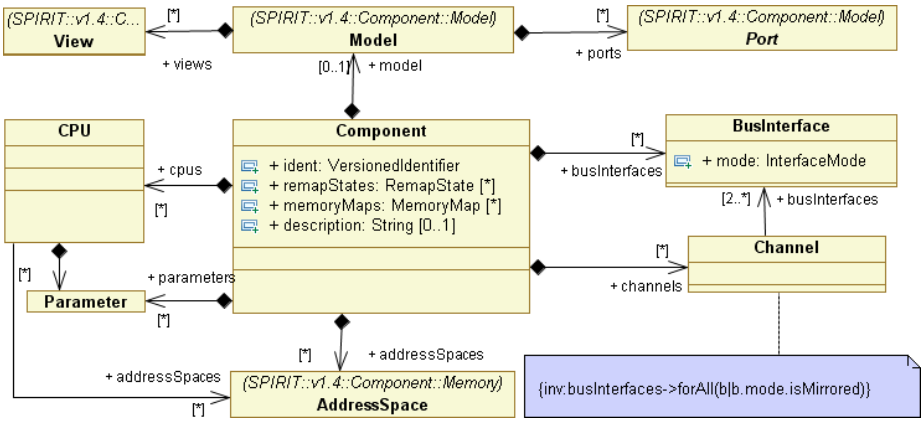

Figure 1: IP-XACT Component metamodel.

The component metamodel describes physical ports, the different views available (RTL, TLM, documentation) and a set of parameters.

The *view* mechanism is a provision for having several models of the same component at different levels of abstraction. The ports can be *wire ports* (for RTL and occasionally TLM) or *transactional ports* (for TLM only). Transactional ports allow only pure binary values or vectors of binary values.

BusInterface groups together ports that collaborate to a single protocol. Components communicate with each other through their bus interfaces tailored for a specific bus. The bus interfaces map the physical ports of the component to the logical ports of the abstraction definition (see next subsection). They also identify the interface mode (master, mirrored master, slave, mirrored slave). The mirroring mechanism guarantees that an output port of a given type is connected to an input port of a compatible type, and vice versa. Channels describe multi-point connections between components when the interfaces are not directly compatible and require some adaptation.

2.2 Abstraction and Bus Definition

A BusDefinition (see Figure 2) describes the high-level attributes of the interfaces connected to a bus. For instance, it defines a maximum number of masters and slaves, and whether a master interface can be directly connected to a slave interface or should rather go through mirrored master/slave interfaces. IP-XACT also provides a mechanism to extend bus definitions. Extending an existing bus definition allows the definition of compatibility rules with legacy buses. For instance the AHB (Advanced High-performance Bus) definition extends the AHBlite definition. An example of compatibility rule is that an extending bus definition must not declare more masters and slaves than the extended one.

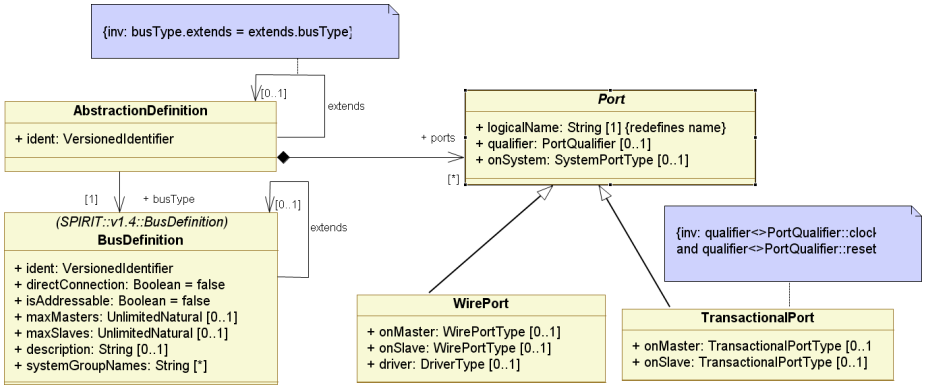

Figure 2: IP-XACT BusDefinition metamodel.

An AbstractionDefinition gives more specific attributes for a given BusDefinition. There can be several abstraction definitions for the same bus definition, like AHB_rtl and AHB_tlm. In the same way, an abstraction definition can extend another one with also some compatibility constraints to enforce. The abstraction definition specifies the ports, which have to be defined by the bus interfaces, and constrains them (type, direction ...).

2.3 Design

A Design (see Figure 3) represents a system or a sub-system. It defines a set of component instances and their interconnections. *Ad-hoc connections* connect two ports directly, wire ports but also transactional ports, without using a bus interface. *Interconnections* are point-to-point connections of bus interfaces from two sibling components, whereas *hierarchical connections* (HierConnection) connect components from different hierarchical levels (*e.g.*, a parent to one of its children).

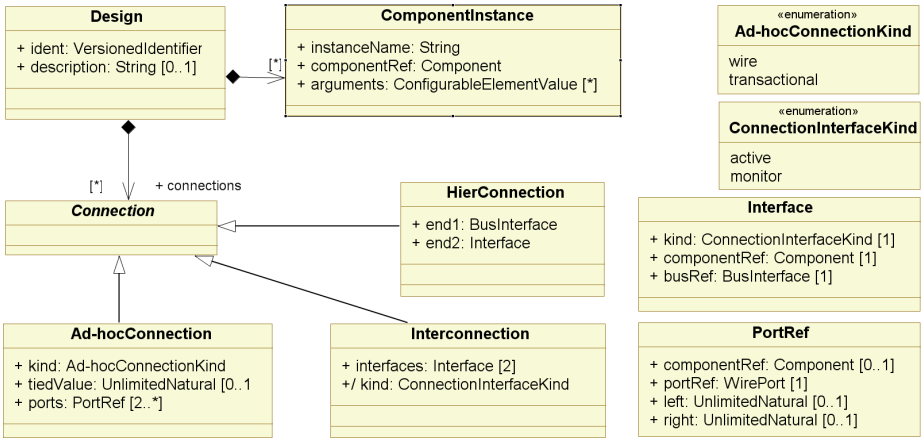

Figure 3: IP-XACT Design metamodel.

2.4 Address space

Address space specifies the addressable area as seen from bus interfaces in *master* mode. *Memory map* specifies the addressable area as seen from bus interfaces in *slave* mode. The addressable area can either be a single *address block* or a *bank* itself further decomposed into other banks or address blocks. Some address blocks can be reserved to be locations for memories or registers. The *registers* are further decomposed into *fields*, with which a value is associated.

Our metamodel reflects the actual IP-XACT specification. However, Revol has proposed an alternative meta-model [18], which is much more flexible and relies on the pattern Item/Descriptor. It distinguishes *register definitions* from *register instances*. The former consists in a generic definition of a register whereas the latter gives the specifics. For instance, all registers of the same size and type have the same definition. This is a valuable improvement to the IP-XACT specification, however, the purpose of our domain view is to provide a synthetic and faithful overview of IP-XACT concepts and to check the conformity of the proposed profile with this view. We do not intend here to improve this particular aspect of IP-XACT and therefore we adopt the metamodel shown in Figure 4.

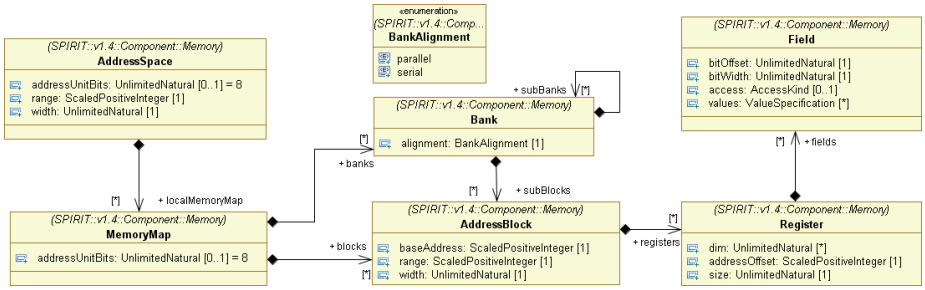

Figure 4: IP-XACT AddressSpace metamodel.

3 The UML Profile for MARTE

3.1 General overview

The new OMG UML profile for MARTE supersedes and extends the former UML profile for Schedulability, Performance and Time (SPT [21]). MARTE also addresses new requirements: specification of both software and hardware model aspects; separated abstract models of applications and execution platforms; modeling of allocation of the former onto the latter; modeling of large domains of Time and Non-Functional properties.

MARTE consists of three main packages. The first package defines the *foundational concepts* used in the real-time and embedded domain. These foundational concepts are refined in the two other packages to respectively support modeling and analysis concerns of real-time embedded systems. The second package addresses *model-based design*. It provides high-level model constructs to depict real-time embedded features of applications, but also detailed software and hardware execution platforms. The third package addresses *model-based analysis*. It provides a generic basis for quantitative analysis sub-domains. Our profile for IP-XACT reuses several model elements from the first and second packages. The following subsections briefly describe these borrowings.

3.2 Resources and Allocation

The central concept of *resource* is introduced in the Generic Resource Modeling (GRM) package of MARTE. A *resource* represents a physically or logically persistent entity that offers one or more *services*. A Resource is a classifier endowed with behavior (a BehavioredClassifier in UML terminology), while a ResourceService is a behavior. Resource and ResourceService are types of their respective *instance* models.

Several kinds of resources are proposed in MARTE like ComputingResource, StorageResource, CommunicationResource, TimingResource. Two special kinds of communication resource are defined: CommunicationMedia and CommunicationEndPoint. The communication endpoint acts as a terminal for connecting to a communication medium; typical associated services are data sending and receiving.

For structural modeling, MARTE enriches the concepts defined in the UML composite structures. StructuredComponent defines a self-contained entity of a system, which may encapsulate structured data and behavior. An *interaction*

port is an explicit interaction point through which components may be connected.

The MARTE *Allocation* associates functional application elements with the available resources (the execution platform). This comprises both spatial distribution and temporal scheduling aspects, in order to map various algorithmic operations onto available computing and communication resources and services. It also differentiates *Allocation* from *Refinement*. The former deals with models of a different nature: application/algorithm on the one side, to be allocated to an execution platform on the other side. The latter allows navigation through different abstraction levels of a single model: System-level, RTL and TLM views.

The Detailed Resource Modeling (DRM) package of MARTE specializes these concepts. It consists of two sub-packages: Software Resource Modeling (SRM) and Hardware Resource Modeling (HRM). Only the latter is considered in this paper.

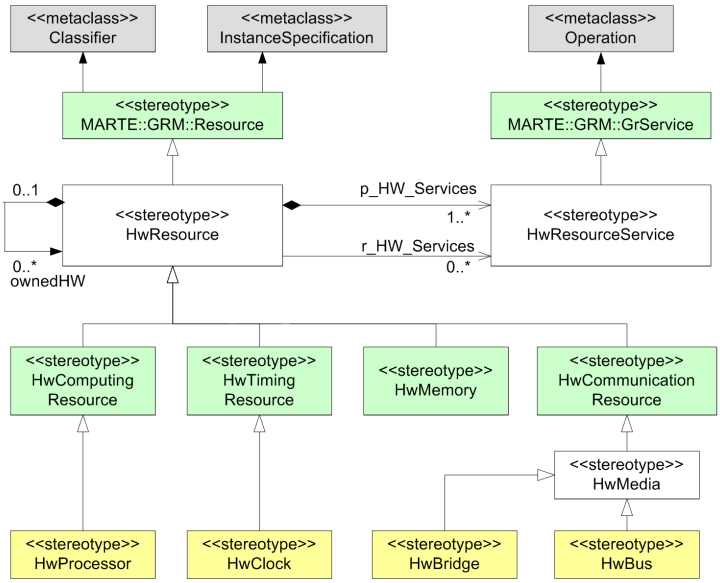

Figure 5: Excerpt from MARTE hardware resource profile.

As shown in Figure 5, HwResource (HwResourceService resp.) specializes Resource (ResourceService resp.) defined in the GRM package. A hardware resource *provides* (hence the prefix 'p_' in the role name) at least one resource service and may *require* ('r_' prefix) some services from other resources. Note that a HwResource can be hierarchical. The HRM package is further decomposed into two sub-packages: HW_Logical and HW_Physical. The former provides a functional classification of hardware entities; the latter defines a set of active processing resources used in execution platform modeling and close to several SPIRIT IP-XACT concepts. HwResource is specialized in the same way as the generic resource of the GRM package (lower part of Figure 5).

3.3 Time in MARTE

Both Resource and Allocation refer to the time model defined in the Time package of MARTE. While SPT considered only time models based on *physical time*, MARTE introduces two distinct models called *chronometric* and *logical* time. The former supersedes the SPT model and its time values are expressed in classical time units (second or one of its sub-multiples). The latter may "count" time in ticks, cycles, busCycles, or any other units. In fact, any event can define a logical clock that ticks at every occurrence of the event. Thus, logical time focuses on the ordering of instants, not on the physical duration between instants. Another noteworthy feature of the MARTE time model is the support of multiple time bases, required to address distributed embedded systems and modern electronic sytem designs.

In MARTE, the underlying model of time is a set of time bases. A time base is an ordered set of instants. Instants from different time bases can be bound by relationships (coincidence or precedence), so that time bases are not independent and instants are partially ordered. This partial ordering of instants characterizes the *time structure* of the application. This model of time is sufficient to check the logical correctness of the application. Quantitative information can be added to this structure when quantitative analyses become necessary.

A Clock is the model element that gives access to the instants of a time base; a ClockConstraint—a stereotype of UML Constraints—imposes dependency between instants of different time bases. Complex time structures and temporal properties can be specified by a combined usage of clocks and clock constraints. An example is given in Section 5.

MARTE also introduces the concept of *timed model element*. A TimedElement associates at least one clock with a model element. This association enriches the semantics of the element with temporal aspects. Thus, a TimedValueSpecification necessarily refers to clocks. A TimedEvent is an event whose occurrences are explicitly bound to a clock. A TimedProcessing represents an activity that has known start and finish times, or a known duration, and whose instants and durations are explicitly bound to clocks. The stereotype TimedProcessing may be applied to UML Action, Behavior, and even Message.

Details about the MARTE Time and Allocation models are presented in a previous paper [22].

4 Modeling IP in UML

This section describes our profile to extend the UML with IP-XACT concepts and the transformation rules to generate IP-XACT descriptions from UML models. Following B. Selic [20], we have tried to define *stereotypes* with parsimony and to create new ones when no equivalent concepts were available in UML or in MARTE. In addition to stereotypes, we have also defined a *model library* to provide a set of data types equivalent to IP-XACT primitive types. The new stereotypes and the model library are gathered within a new profile named UML *profile for* IP-XACT .

In the following, we go again through the main IP-XACT concepts and for each of them we explain our mapping rules and justify the creation of new stereotypes when required.

4.1 Component

Similarly to other approaches mentioned before, we use UML component diagrams to model IP-XACT components. We apply MARTE stereotypes from the Hardware Resource Modeling (HRM) package to identify components that must be transformed into IP-XACT components. More specifically, we apply the stereotype «hwResource» and some of its sub-stereotypes. Components stereotyped by «hwProcessor», «hwMemory» and «hwBus» are all transformed into IP-XACT components. Memories can be further specialized into specific memories using stereotypes like «hwRAM», however, IP-XACT makes no differences between memories and consider all of them as components. UML components stereotyped by «hwBridge» and «hwTimer» are the respective equivalent of IP-XACT bridge and timer components.

Figure 6 shows some UML components extracted from the Leon2 architecture example. All these components are transformed into IP-XACT components. Note that we have followed the naming convention imposed by the authors of the Leon2 architecture even though it is generally admitted that class names should start with a capital letter. This is absolutely required to allow a fully automatic transformation. This remark concerns all UML diagrams of this section.

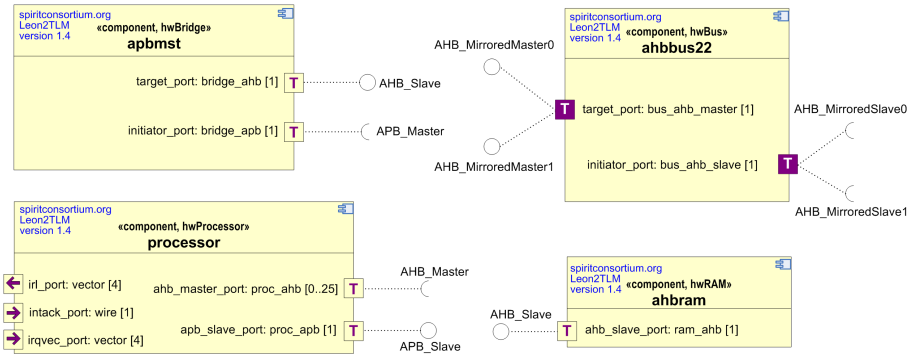

Figure 6: Component definition in UML.

Components interact with communication media through ports that may require or provide a specific bus interface. Contrary to IP-XACT, UML does not differentiate *initiator* ports from *target* ports. Instead of introducing a new stereotype we choose to define two classes: pv_target_port and pv_initiator_port. These classes or rather some of their user-defined subclasses should be used to type UML ports equivalent to IP-XACT ports. Since classes can implement interfaces, using classes instead of stereotypes mimics the relation between ports and bus interfaces directly with a built-in UML mechanism. Moreover, the information introduced via these classes (direction, width, type) is maintained even when the profile is unapplied. This is even better, since this information is relevant even outside our specific context of code generation.

A single IP-XACT component can have different views (*e.g.*, RTL or TLM). Depending on the view we use different port representations. At RTL level, all ports are declared as wire ports (using «portWireType») whereas at TLM level,

the models can have both wire (for ad-hoc connections) as well as transactional ports (identified by «portTransactionalType»). These stereotypes have properties specific to the port kind.

In Figure 6 transactional ports are denoted by a T and mirrored transactional ports by a T. Ports are typed by user-defined classes (*e.g.,* proc_ahb, proc_apb), which are subclasses of either pv_target_port or pv_initiator_port. The multiplicity (*e.g.,* [0..25]) on ports denotes IP-XACT properties minConnections and maxConnections associated with stereotype «portTransactionType». Wire ports are marked with an arrow that denotes their direction (*i.e.,* in or out). Their multiplicity denotes the size of the port. For instance, size 4 for vector ports, size 1 for single ports.

An expanded view of bus interfaces associated with ports is also given. The classes typing the ports either *use* or *realize* a UML interface. This interface should be stereotyped by «busInterface» . The relationship use serves to model required interfaces and the relationship realize models provided interfaces. Here again, UML natively provides a mechanism to model these IP-XACT features. Consequently, we need not define specific stereotypes for that. In the end, all the interfaces are defined in a separate class diagram to avoid diagram cluttering.

Stereotype «busInterface» extends metaclass Interface. It adds properties that exist neither in UML nor in MARTE. These properties identify the related bus/abstraction definitions and the interface types (Master, Slave, mirroredMaster, mirroredSlave ...). A mandatory attribute portMaps contains a reference to the logical and physical port names for the port connected to the bus interface. Interfaces associated with master ports are represented as *sockets* (a usage dependency in UML), whereas Interfaces associated with slave ports are represented as *lollipops* (a realization dependency in UML), since slave ports *provide* services to other components. A «busInterface» refers to a «busDefinition» and an «abstractionDefinition» with its properties busType and abstractionType. Boolean property isMirrored identifies whether the interface is *direct* or *mirrored*. Property mode of type InterfaceModeKind is for the interface mode (master, slave or system). Enumeration InterfaceModeKind, defined in our model library, contains valid modes.

4.2 Abstraction and Bus Definition

IP-XACT terminology may cause confusions. Abstraction and Bus Definition represents communication protocols. Nevertheless, physical buses that have some behavior of their own (like arbitration, address decoding ...) must be modeled as components. For instance, in the Leon2 example, component ahbbus models an AHB bus (AMBA High-Performance Bus). In the final design (see Section 4.3), an instance of this component interacts with instances of components processor and ahbram through connectors, *i.e.,* IP-XACT buses: AHB, APB. Contrary to other mentioned approaches, which do not explicitly model the abstraction and bus definitions, we use class diagrams to model them. Two stereotypes («abstractionDefinition» and «busDefinition») have purposely been defined in our profile, as shown in Fig 7. A new compartment was added to standard classes to display the information related to these stereotypes.

Properties of these stereotypes are directly derived from IP-XACT concepts. They include the IP identity typed as Ident, the maximum number of masters

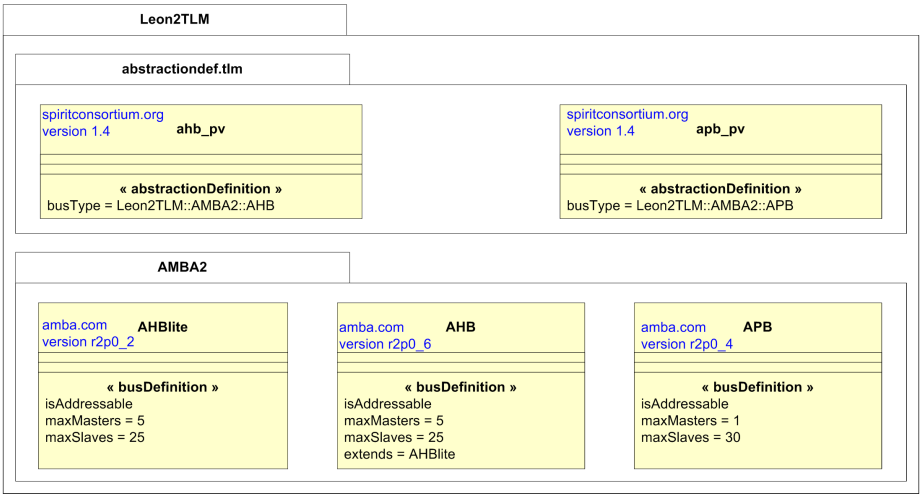

Figure 7: Bus and Abstraction Definition classes in UML Model.

(property maxMasters) and slaves (property maxSlaves). Our model library introduces a custom data type called VersionedIdentifier to uniquely represent all IP-XACT objects.

As explained in Section 2, IP-XACT introduces a mechanism to extend bus and abstraction definitions. In our approach, this mechanism is modeled using MARTE Refinement concept, which enhances UML concept of Refinement by making explicit the constraints implied by the refinement relationship. This enhancement comes handy to specify IP-XACT bus-extension compatibility constraints (like the restriction on number of masters or slaves previously described).

4.3 Design

When all components and buses have been defined, their instances must be combined and interconnected to build the targeted design. Other than the approaches that use component diagrams for both the definition and the integration, we use UML composite structure diagrams for the integration phase. This approach eases the reuse of already defined components and allows for having several parts (component instances) of the same classifier without corrupting the classifier itself. This also results in very simple composite structure diagrams while maintaining all the detailed information in the model on the classes themselves. Several instances of the same component can be used in the same design. These instances are stereotyped by «configurableElementValues» to provide values for component configurable parameters. The type of parts identify the related hardware resource, *i.e.,* an IP-XACT component stereotyped by «hwResource» or one of its substereotypes.

Figure 8 shows a partial composite structure diagram of the Leon2 design. Processor Leon2 uproc is connected to memory uahbram through component ahbbus.

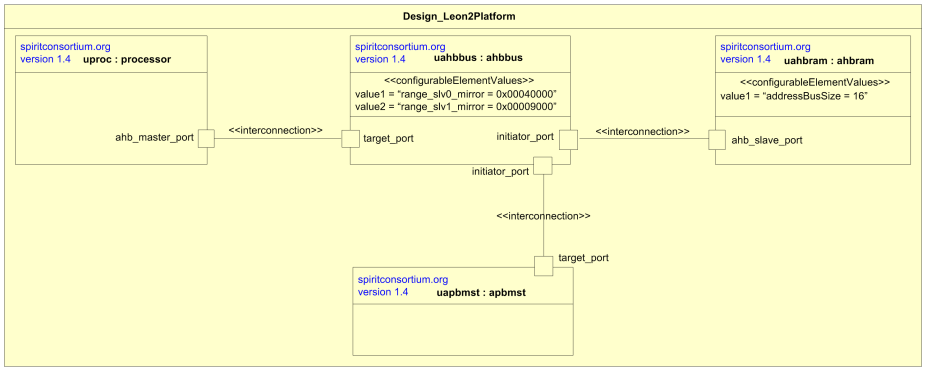

Figure 8: The Leon2 design in UML.

The parts are connected together by connectors linked to their ports. Even though, in component diagrams, ports are used to show the relation to bus interfaces, they are used here to show the interconnections. Both standard interconnections and ad-hoc connections are modeled as connectors. Hierarchical interconnections are represented with a delegate dependency between the port of the parent and a port of its children. Stereotypes like «interconnection», «adhocConnection» and «hierConnection» have been defined to model different IP-XACT interconnection kinds. The stereotype «interconnection», which extends the UML metaclass Connector, contains reference to the bus interfaces on both sides of the connector. Indeed, standard UML connectors connect connectable elements like ports or parts, which stand for component instances. However, IP-XACT connects the bus interfaces associated with the ports. As ports can have more than one bus interface, we have to identify each connection with the bus interface uniquely. So, the stereotype «interconnection» has a property called busIntfEnds to identify uniquely the bus interfaces attached to the connection.

4.4 Implementation

Both our new profile and the examples shown above have been implemented within Papyrus (`http://www.papyrusuml.org`), an open-source UML graphical editor. Table 1 summarizes our mapping rules from IP-XACT concepts to UML concepts and stereotypes.

We have implemented these rules within an ATL transformation model that allows the automatic generation of IP-XACT models from the UML models. ATL (ATLAS Transformation Language) [23] is a model transformation language and toolkit. An ATL transformation model is composed of rules that define how source model elements are matched to create and initialize the elements of the target models. It ensures that the generated target models conform to the target Ecore metamodel. In our case, we rely on an IP-XACT Ecore metamodel directly generated from the Spirit Consortium XSD specification. This ensures that the generated IP-XACT models conform with the IP-XACT standard and can be used with other IP-XACT tools (*e.g.,* Magillem, IP-XACT editor plug-in for Eclipse ...). For the demonstration purpose, we applied this ATL transforma-

SPIRIT IP-XACT	UML	MARTE	Profile for IP-XACT
Processor/CPU	Component	«hwProcessor»	
RAM	Component	«hwRAM»	
AHB/APB Bus	Component	«hwBus»	
VLNV	Model Library		VersionedIdentifier
Port	Port		«wirePort», «transactionalPort»
Bridge	Component	«hwBridge»	
Timer	Component	«hwTimer»	
Timing Constraints	Model Library		
Bus Definition	Class		«busDefinition»
Abstraction Definition	Class		«abstractionDefinition»
Bus Interface	Interface		«busInterface»
Bus Interface Mode	Model Library		InterfaceModeKind
Design	StructuredClassifier		
Component Instance	Property (as parts)		
Connection	Connector		«interconnection»

Table 1: Mapping IP-XACT concepts to UML

tion to make a full generation of the IP-XACT model of the Leon2 architecture, provided by the SPIRIT consortium. One rule generates IP-XACT code for the UML design composite structure diagram. Mainly it contains the component instance types to represent UML parts and interconnection types for connectors between the various parts. Two other rules extract the data present in the bus and abstraction definition class diagrams to produce the relevant IP-XACT XMI descriptions.

Our transformation model implements all the rules presented in Table 1. Table 2 contains additional rules borrowed from the UML Profile for Electronic System Level (ESL) [18].

SPIRIT IP-XACT	UML	Profile for ESL
MemoryMap	Class	«registerMapDef»
	Property	«registerMap»
Register	Class	«registerDef»
	Property	«register»
Field	Class	«fieldDef»
	Property	«field»

Table 2: Mapping IP-XACT concepts to UML profile for ESL

Figure 9 summarizes the full model transformation process. A model (leon2tlm.uml) taken from the Papyrus tools and that conforms to the UML2 Ecore metamodel, is transformed into several models (Output.xmi) that conform to the IP-XACT Ecore metamodel (ipxact.ecore). The transformation is defined by the transformation model (uml2ipxact.atl) which itself conforms to a model transformation metamodel (ATL). This last metamodel, along with the source and target metamodels, have to conform to a meta-metamodel (such as MOF or Ecore). Hence,

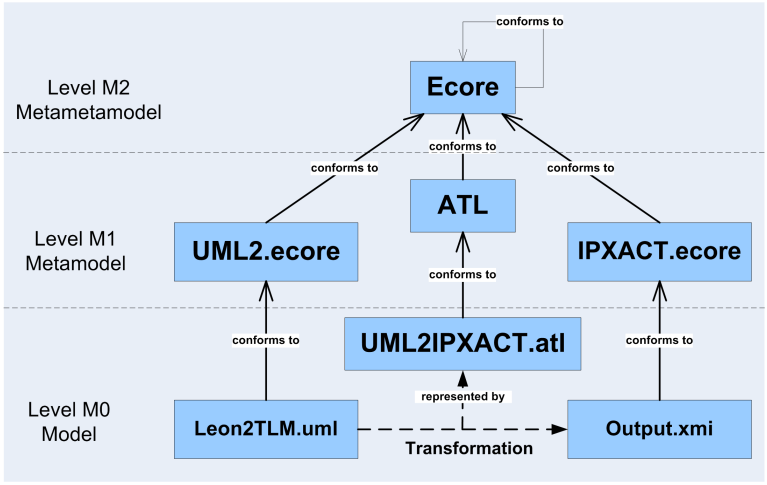

Figure 9: Model transformation and ATL

the ATL transformation model relies on two metamodels, UML and IP-XACT. The UML metamodel is provided by the Eclipse environment which is used by ATL, whereas the IP-XACT metamodel is generated from the XSD schema description given by the SPIRIT consortium. Instead of creating our own metamodel, this generated metamodel has the advantage that it produces the output adhering to the IP-XACT standard and will therefore allow quick adaptation whenever there are some minor changes in the standard. It means that for a newer version of IP-XACT coming in the future, we will have to generate IP-XACT Ecore metamodel from the new XSD schema description to make the tool compatible. The XSD schema files are converted to Ecore metamodel using dedicated transformation models provided by the Eclipse tool. However, major modifications of the standard would probably require a thorough rewriting of the transformation model.

4.5 Discussion

Our transformation model has been tested by building a UML description of the Leon2 and by automatically producing the IP-XACT specification. The conformity with the standard has been established by running an XML validation tool and by comparing the result with the implementation delivered by the Spirit Consortium. The model has been released to industrial partners of a French research project, the goal of which is to establish connections between UML and other standards like IP-XACT, SysML, AADL ....

The main expected advantages of this model-driven approach is to integrate IP-XACT in a more abstract specification flow where UML is used as a pivot. The idea is to benefit from lots of rapidly improving graphical editors and maintain in a single repository all models related to a single project. This is complementary to other approaches that generate behavioral codes to reduce the design time. In our case, the actual gain is difficult to quantify since our approach opens new possibilities and offers verification facilities but does not automatically produce any behavioral code. Additionally, we have focused on being able to generate

a full IP-XACT specification from the UML model and to extend IP-XACT with time properties so we can check candidate implementations against the time specification. We elaborate on this latter aspect in the next section.

5 Time Characterization of IP

5.1 Overview of our proposal

IP-XACT and SystemC provide little or no support to model at an abstraction level higher than TLM PV (Programmer View). Moreover, IP-XACT mostly deals with the structural aspects of IPs. The way IPs interact, their time characteristics or in general their behavior, are not covered by IP-XACT 1.4 and only depend on the associated SystemC/HDL code. However, early validation of IP-XACT design requires an abstract description of component behavior and of their time characteristics. The SPRINT Project (`http://www.sprint-project.net`), which aims at providing an extension of IP-XACT, highlights these needs.

Being a general-purpose modeling language, UML offers various diagrams (like activity, state machine or sequence) to represent the system *untimed* behavior. The MARTE profile, which includes the non-normative Clock Constraint Specification Language (CCSL) [24], extends the UML with explicit time-related concepts. Combined with UML behavioral elements it provides support for building models of the whole IP behavior at a *timed* Communicating Process (CPT) level.

By extending UML with IP-XACT specific features, we get a larger notion of component that combines the structural features of IP-XACT with timed UML behavior. Figure 10 shows this unified notion of component: on the left side, the structural aspects covered by IP-XACT; on the right side, the behavioral aspects brought by the UML (shown with a hatched background). The UML functional models can be annotated with time information using the MARTE time model. This defines the timed behavior of the whole IP. These aspects are further discussed in subsection 5.2. The relationship between the structural aspects and the behavior, which is implicit in IP-XACT is made explicit here. Writing in fields of control registers may trigger the execution of some behavior, which in turn uses values of other registers as input parameters.

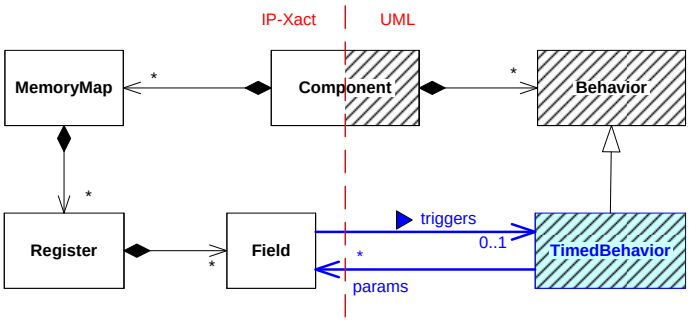

Figure 10: Unifying IP-Xact and UML components

In this unified environment we get structural, functional as well as time description of IPs. We propose to go one step further by extending IP-XACT

descriptions with time requirements of IPs. It is neither practical nor desirable to include the whole behavior since it would require adding all UML behavioral model elements to IP-XACT. It is also not practical in most cases since implementations of the same IP at different abstraction levels are usually made by different teams and may result in components that do not even have the same interfaces. For example, a simple read or write communication at TLM level boils down to more complex control signals at RTL level. The same RTL signals can also be shared by completely different transactions. Rather than addressing the whole IP behavior, we focus on their time requirements. These requirements, missing in IP-XACT, are usually described as waveforms in datasheets. We specify them as a CCSL specification (see subsection 5.3). Then, we rely on CCSL operational semantics to execute the specification and automatically produce waveforms. In a second phase, we use this specification to generate dedicated observers that are to be run in testbenches against candidate implementations both at the RTL and at the TLM levels (see subsection 5.4). Finally, we consider that timing requirements are fundamental for the specification of IPs and must be included in IP-XACT. Therefore, in a third phase, we augment IP-XACT to include such a specification (see subsection 5.5).

5.2 Behavior modeling including time

To illustrate our approach, we choose a generic memory. Our memory is an IP-XACT component that stores data and provides two services (reading and writing) for data access. We consider a timed abstract specification and intend to compare it with lower-level, timed and untimed, functional implementations written in SystemC, VHDL, ...

In hardware components, a time-related information is often a physical duration (expressed in seconds or one of its sub-multiples) or a clock frequency. In MARTE, this is relevant to the *chronometric* time model. The *logical* time, also supported by MARTE, allows a better gradation from the untimed to the (physical-)timed models, by introducing logical-time models. This is illustrated in the next subsection.

Since we model hardware, a flow-oriented description of the behavior has been chosen. Figure 11 contains a UML *activity diagram* that specifies the behavior of a memory. The two services read and write are represented by UML actions: MemRd and MemWr respectively. The memory interacts with its environment through two signals and four activity parameter nodes. Sel (Selection) is the event used to trigger one of the two services. Eoa (End of access) is the signal emitted at the completion of an access. Addr (Address), Din (Data input), RW (Read/Write), Dout (Data output) are *streamed* parameters, thus their values can change independently of the action executions. Such a concurrent modeling of the behavior gives rise to critical race conditions. To avoid such undesirable behaviors the model has to be further constrained, which is what we do with logical time and clock constraints expressed in CCSL.

Stereotypes from the profile MARTE are used to select model elements on which time requirements apply. These model elements are associated with what MARTE calls logical clocks that act as activation conditions. A synchronous memory is driven by a special signal we name *clk* (its clock). This clock is explicitly introduced in the activity diagram (Figure 11) by applying the stereotype «timedProcessing». Thus, action durations and occurrence dates of parameter

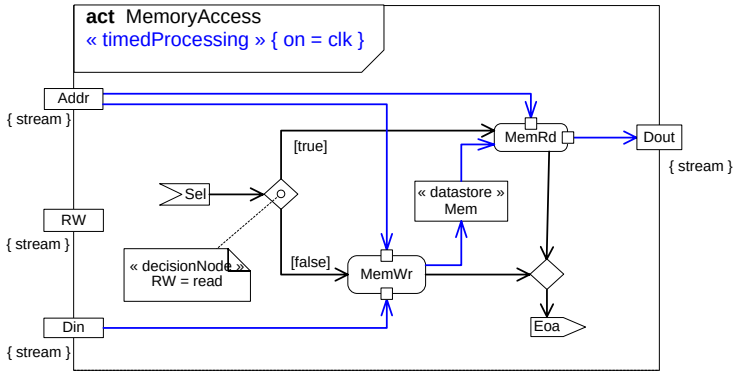

Figure 11: UML Activity for a memory with MARTE annotations.

changes can refer to *clk*. For instance, a memory write could be specified as lasting for 2 ticks of *clk*. (MemWr.duration = 2 on *clk*).

The MemoryAccess activity diagram is much more informative than a simple transactional model. It represents, through the "flow semantics" of the UML activity diagrams, many causal relationships. For instance, changing the value of Addr may cause a change in the stored value (Mem), and also in the output data (Dout). These causality chains are implied by the directed activity edges linking the corresponding activity nodes. Such a rich behavioral model is certainly quite much refined to be efficiently used in high-level modeling such as TLM or ESL, where components should be considered as *black-boxes*. The next section explains how CCSL can provide more abstract descriptions of the behavior of components, considering only interface signals, in a way similar to *Interface Automata* [25].

5.3 Time specification with CCSL

With CCSL, we have first to identify the clocks. As said in Section 3.3, a (logical) clock can be associated with any event. In the memory example, events Sel and Eoa are natural candidates. The changes in the parameter values are other events that may be associated with logical clocks. For convenience, we adopt the following notation: the clock associated with an event AnEvent is denoted *anEvent* (the same name but italicized and with a lower case initial letter). Dependency between events can then be expressed by clock constraints written in CCSL. In the MARTE profile, a clock constraint is a specialization of an NfpConstraint, which is a stereotype of the UML Constraint. As an NFP constraint, a clock constraint conveys an attribute that gives its kind. required and offered are the possible values of interest for our approach. required is used in specification when the clock constraint is *imposed* to a design. offered characterizes constraints that are *assumed* (constraints related to the environment) or that represent some capability of a component. Offered constraints are then used in performance evaluation. The clock constraints given hereafter are *required clock constraints*

written in CCSL.

$$sel \text{ strictly alternatesWith } eoa \quad (1)$$

$$eoa = sel \text{ delayedFor } 1 \text{ on } clk \quad (2)$$

$$addr \text{ strictly alternatesWith } sel \quad (3)$$

$$addr \text{ isSporadicOn } clk \text{ min } 2 \quad (4)$$

The first clock constraint (Cstr 1) says that the occurrences of Sel (*i.e.*, the ticks of *sel*) alternate with the occurrences of Eoa (*i.e.*, the ticks of *eoa*). This is a basic functional requirement demanding that each request is eventually served and that the accesses are not overlapping. This behavior is often implicitly assumed in transactional modeling.

The second clock constraint (Cstr 2) introduces a time quantitative information. The memory must complete an access on the second tick of *clk* following a tick of *sel*. This imposes an upperbound to the completion of an access, which was not the case with Cstr 1.

The third constraint links *addr* and *sel*. The intent is to impose that the value of Addr be set before a request for a new access (a tick of *sel*).

When we submit these 3 constraints to TIMESQUARE we get the possible execution trace in Figure 12. TIMESQUARE is an Eclipse plug-in that we have developed. It compiles CCSL constraints and produces simulation traces. TIMESQUARE is available at `http://www.inria.fr/sophia/aoste/time_square`. Instant relations can be displayed by TIMESQUARE: dashed arrows stand for precedence relations, whereas vertical edges stand for coincidence relations.

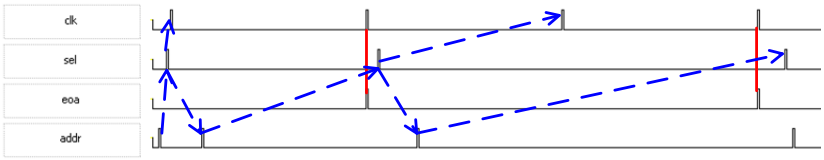

Figure 12: Simulation trace of Cstr1 to Cstr3.

The trace shows that the address value set on the second tick of *addr* is changed before its sampling by *clk*. This is obviously a faulty behavior that can be corrected by adding a stability constraint (Cstr 4). The new behavior is given in Figure 13.

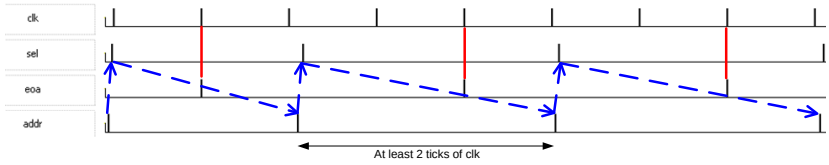

Figure 13: Simulation trace of Cstr1 to Cstr4.

Note that this memory specification is simplified (we do not constrain other input signals), and it is just a particular example. Other access protocols can be specified. CCSL could even express constraints such as found in memory datasheets.

5.4 Comparing RTL and TLM implementations

As discussed before, comparing implementations at different levels is very difficult. So instead, we have investigated the possibility to generate a skeleton from the abstract specification of the behavior. However, this is not practical because of huge libraries of legacy IPs and because it would impose one single methodology and design flow to all IP providers. It appears much more sensible to verify the equivalence of implementations by establishing common properties amongst them. Figure 14 illustrates this proposal.

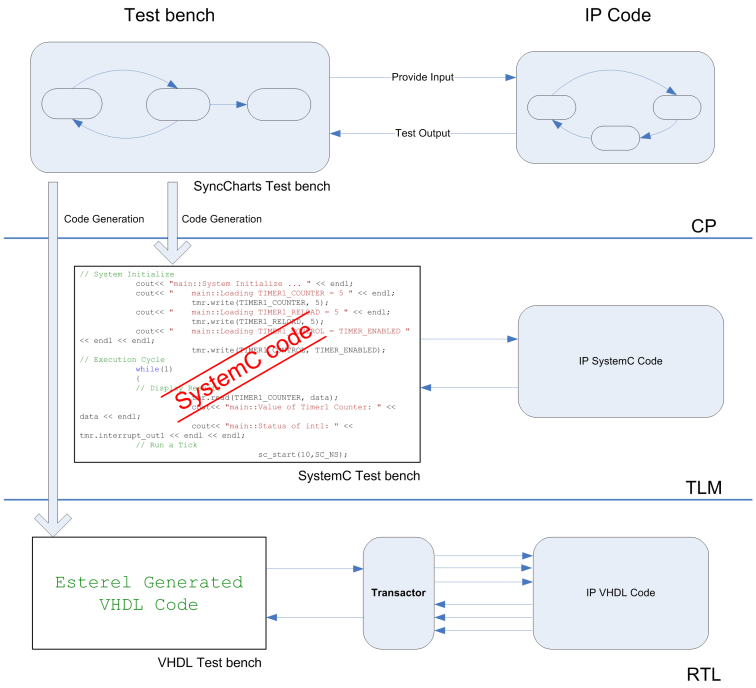

Figure 14: IP Test bench at various abstraction levels.

We start with a specification language (at CP level) that supports formal verification and code generation. At this level, we build an abstract model of the IP under design (right-hand side) and a testbench (left-hand side) that relies on *observers* to establish some properties on the IP.

In the first phase, we have used Esterel Studio to build a SyncCharts model of the memory and of its testbench. In our case, the testbench focuses on time properties. SyncCharts are graphically very closed to UML state machines but are semantically equivalent to the Esterel language. Using Esterel Studio verification tools, time properties are checked. Both the testbench model and the model of the memory are ajusted to exhibit the required time properties.

In the second phase, Esterel Studio generates SystemC code for the TLM level and VHDL code for the RTL level from the testbench model. Note that we do not generate code for the whole behavior, but only the code for the testbenches. The generated code for the testbenches is run against the respective IP implementations. Therefore, our process does not guarantee that the two implementations

are correct, but it does verify (in simulation) that the timing properties checked on the golden model still hold in the RTL and TLM implementations.

Since interfaces of the RTL and TLM models are not necessarily identical, the validation at the RTL (and possibly also at the TLM level) requires interface adaptors, also called *transactors*. For instance, read/write transactions at TLM level are mere reading/writing data to a port whereas at RTL level it involves many more signals, like bus strobe, chip select, bus arbitration signals. These transactors can often be generic but must nevertheless be validated separately. Of course, the generated testbenches are not complete and must be combined with other level-specific testbenches, but at least, we can identify common time properties.

To ease the building of testbenches at the CP-level, we have used a library of predefined *time patterns* that can be composed together to build complex testbenches. Let us recall, that one strength of synchronous languages is to provide a parallel composition operator as a primitive construct. This library of patterns reflects the CCSL specification. For each CCSL kernel operator, we have built an equivalent Esterel observer. We have also built some generic transactors to adapt CCSL clocks to Esterel signals. Therefore, our CCSL specification can be used in a systematic way to derive an Esterel testbench.

With the same idea, we have also built libraries of SystemC and VHDL observers for CCSL operators. So we can directly check candidate implementations without relying on Esterel Studio code generation tools. However, using Esterel Studio, the verification at the CP level is exhaustive so we can guarantee that the CP model will never exhibit a property that has been checked. Whereas, on the lower level, the simulation will not cover 100% of the functionality.

5.5 Augmenting IP-XACT with time requirement information

In the same way that IP documentations include waveforms to specify time properties, we think that IP-XACT descriptions should include equivalent information. Rather than including the waveform itself, it is more sensible and more productive to include a description of the time requirements. In this whole section, we showed that a CCSL specification can be used in several ways to improve the confidence we can have in IP implementations. We plead for adding a specification language with the same expresiveness within IP-XACT.

In any case, it is always possible to add such information as a vendor extension. For instance, by adding parameters named timeRequirement and whose value is a CCSL relation. This is illustrated in the code below. It is definitely better to have such a language standardized.

```
<spirit:component
   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
   http://www.spiritconsortium.org/XMLSchema/SPIRIT/1.4/index.xsd">
  <spirit:name>Memory</spirit:name>
...
  <spirit:vendorExtensions>
   <spirit:parameters>
   <spirit:parameter>
```

```
   <spirit:name>timeRequirement</spirit:name>
   <spirit:value>sel strictly alternatesWith eoa</spirit:value>
   </spirit:parameter><spirit:parameter>
   <spirit:name>timeRequirement</spirit:name>
   <spirit:value>eoa = sel delayedFor 1 on clk</spirit:value>
   </spirit:parameter>
   </spirit:parameters>
  </spirit:vendorExtensions>
</spirit:component>
```

As a vendor extension, proprietary tool can extract the information the way they like. With the example given above, an Eclipse plugin can extract the parameters named *timeRequirement* and produce a CCSL file that in turn is fed into the TimeSquare plugin to run simulations.

5.6 Results

We have presented a simple example so we can focus on the process itself and not on the understanding of the example. However, we have also considered the AHB Bus Arbiter and APB Bridge from the Leon2 Architecture. After building a CCSL description from the official AMBA specification, we have tested the code released by the Spirit Consortium to illustrate the use of the IP-XACT 1.4 specification.

For the AHB Arbiter example, we have not found any problem with the implementation. However, the implementation for APB bridge does not strictly conform to the ARM AMBA specification. The APB Bridge accepts requests from the masters and forwards them to the slaves. The ARM specification indicates that at least two buffer registers are required to hold the data while transfering to the slow speed APB bus. The implementation accepts only one input from an AHB Master and holds the signal ready low until it is flushed out on the APB bus. This kind of protocol violation can be tricky to detect manually and are more easily tackled by automatic verification techniques as the one we proposed.

6 Conclusions

Early validation of systems built by assembling components requires to associate abstract behavioral and timing models with IPs. The recent SPIRIT IP-XACT 1.4 only considers structural aspects of the IPs. After describing our UML profile for IP-XACT that builds on MARTE to model IP-XACT designs, we propose to use the MARTE Time model and its constraint language CCSL to extend IP-XACT. The CCSL description can act as a specification of timing requirements for IP-XACT components as waveforms and timing diagrams do in the paper datasheets. We also show how this specification can be used to generate testbenches of valid scenarios that must be satisfied by the component implementations whatever their level of abstraction and the language used is. Generating testbenches for models at various abstraction levels from the same formal specification is an important step towards establishing the equivalence, or at least the consistency of RTL and TLM implementations.

In addition to considering the time requirements of the IPs themselves there is also a demand to describe the time requirements of the environment in which

the IPs are executed. Such requirements act as pre-conditions to be enforced. We have not addressed that second aspect and shall consider it in future work.

References

[1] IEEE Standards Association: Open SystemC Language Reference Manual. Open SystemC Initiative. (2005) IEEE Std. 1666–2005.

[2] SPIRIT: IP-XACT v1.4: A specification for XML meta-data and tool interfaces. Spirit Consortium. (March 2008) http://www.spiritconsortium.org.

[3] Guiney, M., Leavitt, E.: An introduction to OpenAccess: an open source data model and API for IC design. In Hirose, F., ed.: ASP-DAC, IEEE (2006) 434–436

[4] OMG: Unified Modeling Language, Superstructure. Object Management Group. (November 2007) Version 2.1.2 formal/2007-11-02.

[5] OMG: UML Profile for MARTE, beta 2. Object Management Group. (June 2008) OMG document number: ptc/08-06-08.

[6] Weilkiens, T.: Systems Engineering with SysML/UML: Modeling, Analysis, Design. The MK/OMG Press, Burlington, MA, USA. (2008)

[7] Cai, L., Gajski, D.: Transaction level modeling: an overview. In Gupta, R., Nakamura, Y., Orailoglu, A., Chou, P.H., eds.: CODES+ISSS, ACM (2003) 19–24

[8] Chen, R., Sgroi, M., Lavagno, L., Martin, G., Sangiovanni-Vincentelli, A., Rabaey, J.: UML and platform-based design. In: UML for real: design of embedded real-time systems, Norwell, MA, USA, Kluwer Academic Publishers (2003) 107–126

[9] Schattkowsky, T.: UML 2.0 - overview and perspectives in SoC design. In: DATE, IEEE Computer Society (2005) 832–833

[10] Martin, G., Mueller, W.: UML for SoC Design. Springer (2005)

[11] OMG: UML profile for System on a Chip v1.0.1. Object Management Group. (2006) OMG document number: formal/06-08-01.

[12] Graf, S., Ober, I., Ober, I.: A real-time profile for UML. STTT **8**(2) (2006) 113–127

[13] Riccobene, E., Scandurra, P., Rosti, A., Bocchio, S.: A Soc design methodology involving a UML 2.0 profile for SystemC. In: Conf. on Design, Automation and Test in Europe (DATE), IEEE Computer Society (March 2005)

[14] Viehl, A., Schönwald, T., Bringmann, O., Rosenstiel, W.: Formal performance analysis and simulation of UML/SysML models for ESL design. In Gielen, G.G.E., ed.: DATE, European Design and Automation Association, Leuven, Belgium (2006) 242–247

[15] Zimmermann, J., Bringmann, O., Gerlach, J., Schaefer, F., Nageldinger, U.: Holistic system modeling and refinement of interconnected microelectronics systems. In: Conf. on Design, Automation and Test in Europe (DATE), MARTE Workshop. (March 2008)

[16] André, C., Mallet, F., Khan, A.M., de Simone, R.: Modeling Spirit IP-XACT in UML Marte. In: Conf. on Design, Automation and Test in Europe (DATE), MARTE Workshop. (March 2008) 35–40

[17] Schattkowsky, T., Xie, T.: UML and IP-XACT for Integrated SPRINT IP Management. In: Design, Automation Conference (DAC), UML for SoC workshop. (June 2008)

[18] Revol, S., Taha, S., Terrier, F., Clouard, A., Gérard, S., Radermacher, A., Dekeyser, J.L.: Unifying HW analysis and SoC design flows by bridging two key standards: UML and IP-XACT. In Kleinjohann, B., Kleinjohann, L., Wolf, W., eds.: Distributed Embedded Systems: Design, Middleware and Resources. Volume 271 of IFIP. Springer Verlag (2008) 69–78

[19] Revol, S.: Profil UML pour TLM: contribution à la formalisation et à l'automatisation du flot de conception et vérification des systèmes sur puce. PhD thesis, INPG, Grenoble, France (June 2008)

[20] Selic, B.: A systematic approach to domain-specific language design using UML. In: ISORC, Los Alamitos, CA, USA, IEEE Computer Society (2007) 2–9

[21] OMG: UML Profile for Schedulability, Performance, and Time Specification. Object Management Group, Object Management Group, Inc., 492 Old Connecticut Path, Framing-ham, MA 01701. (January 2005) OMG document number: formal/05-01-02 (v1.1).

[22] André, C., Mallet, F., de Simone, R.: Modeling time(s). In Engels, G., Opdyke, B., Schmidt, D.C., Weil, F., eds.: MoDELS. Volume 4735 of Lecture Notes in Computer Science., Springer (2007) 559–573

[23] Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: ATL: a model transformation tool. Sci. Comput. Program. **72**(1-2) (2008) 31–39

[24] Mallet, F., André, C.: Clock Constraint Specification Language: Specifying clock constraints with UML/MARTE. ISSE **4**(3) (2008) to appear

[25] Alfaro, L., Henzinger, T.A.: Interface automata. In: Proceedings of the Ninth Annual Symposium on Foundations of Software Engineering (FSE), ACM, ACM Press (2001) 109–120

Contents

Centre de recherche INRIA Sophia Antipolis – Méditerranée
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399