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

Septembre 2008

Thème COM

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

Marte Timing Requirement and Spirit IP-XACT

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

Thème COM — Systèmes communicants
Projet Aoste

Rapport de recherche n° 6647 — Septembre 2008 — 24 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

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex (France)
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 Property (IP) 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 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 [4] (UML)-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 aspects at different levels of abstraction and behavioral/functional aspects possibly considering timing viewpoints such as untimed/asynchronous/causal, logical synchronous/cycle-accurate or physical/timing models. Semantics provides behavioral meaning to full systems from the combination of the behavior of its components. For *system structure* representation, UML uses 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 ...). It uses 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, sequence and activity diagrams that can be thought as closest to TLM/CP (Communicating Processes) 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.

One good feature of UML is its extendability, with the profiling mechanism. On the other hand the set of considered features should be as much as possible standardized and agreed upon (if to allow interoperability), which is the main goal of the SPIRIT consortium. Therefore a UML profiling approach for IP-XACT based on relevant metamodels would allow easy creation and extension of model editors for IP-XACT as the standard evolves, as well as experimentations with prototypal versions. Models can then be translated into IP-XACT

syntax for tool integration. 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).

So, our goal is to use UML as a pivot or the focal point reusing the existing UML graphical editors (*e.g.,* Eclipse UML, MagicDraw by NoMagic, Rational Software Architect by IBM, Artisan, Papyrus, ...), thus integrating IP-XACT into a higher abstract level modeling environment. Hence, we will be able to export UML models to IP-XACT descriptions. One immediate bonus from using MARTE is to benefit from its time model to attach timing/behavioral information to IP-XACT models. As an example, we include the timing information extracted from the IP datasheets and show how this information can be used to generate testbenches tailored for the different abstraction levels.

Related work: There has been several propositions to use UML in SoC Design [8, 9] including using profiling mechanismes (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 interoperabiblity with IP-XACT models and makes an extensive use of the MARTE profile and of its time model.

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

To achieve the first, we have built a UML metamodel of IP-XACT 1.4. Section 2 uses an extract of this metamodel to introduce IP-XACT. Then, section 3 gives an overview of MARTE concepts used in our approach. Section **??** describes the structural mapping of UML-based models to IP-XACT. Section 4 proposes an extension of IP-XACT to integrate time information. As a running example, we use the IP-XACT specification of the Leon2 processor, released as part of the IP-XACT 1.4 RC1 distribution package (*http://www.spiritconsortium.org*).

2 Spirit IP-Xact metamodel

The IP-XACT schema is the core of the IP-XACT specification. It contains seven top-level schema definitions, each of which defining a different kind of object: Component, Bus definition, Abstraction definition, Design, Abstractor, Generator chain, Configuration. We consider only the first four.

2.1 Component

Component is the basic model element in SPIRIT. Every IP is described as a component without distinction of type, whether they represent computation cores (processors, co-processors, DSPs), peripheral devices (DMA controllers, timers, UART), storages (memories and caches), or interconnects (simple buses, multi-layer buses, cross bars, networks on chip).

Figure 1 shows the main features of components as considered in IP-XACT, their interface and their memory hierarchy. A component identifier is unique, it specifies its name, the containing library, the vendor and the version number. A textual description may comment its intended role. A component also contains a precise description of the memory hierarchy: address spaces and memory mappings.

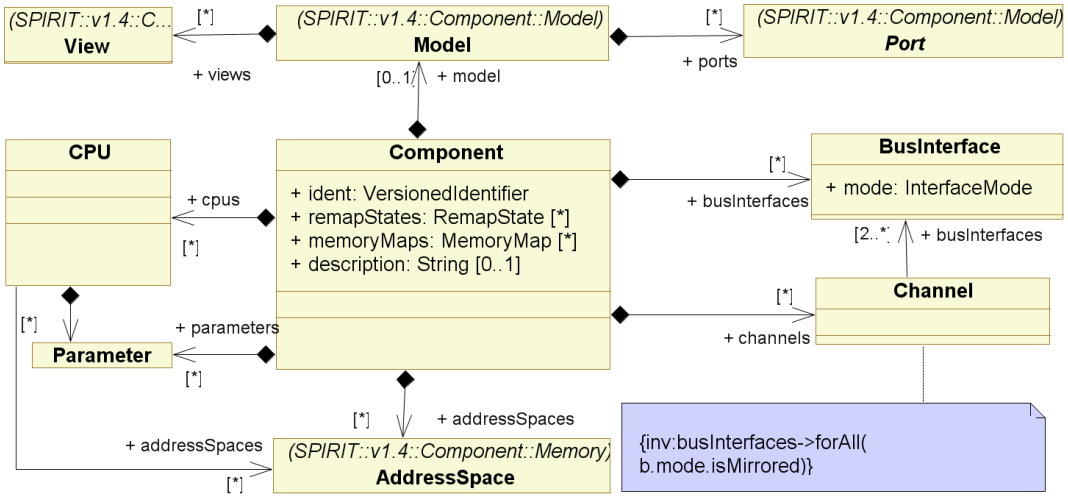

Figure 1: IP-XACT Component metamodel.

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 (only for TLM). Transactional ports allow only pure binary values or vectors of binary values.

BusInterface allows for grouping 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.

An AbstractionDefinition gives lower-level 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 defines the ports, which have to be defined by the bus interfaces, and constrains them (type, direction ...).

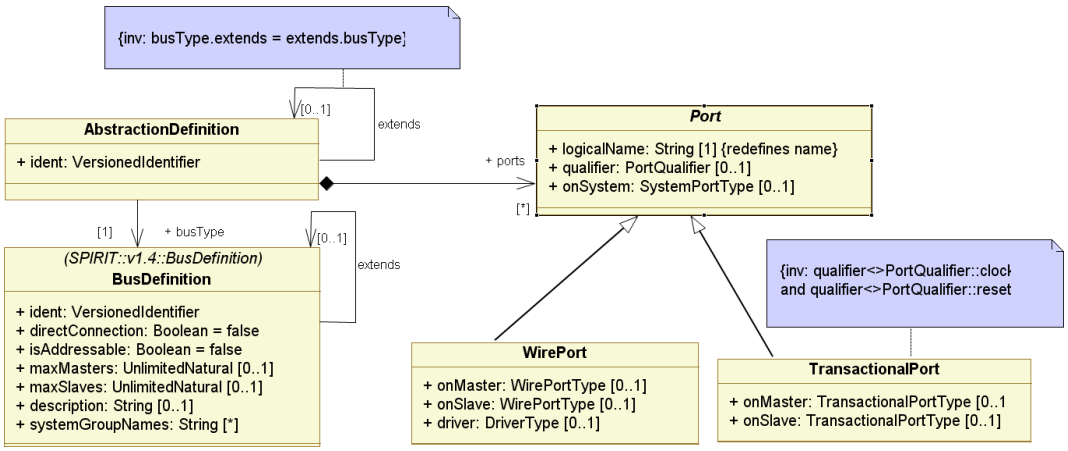

Figure 2: IP-XACT BusDefinition metamodel.

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 siblings components, whereas hierarchical connections (HierConnection) connect components at different hierarchical levels (*e.g.*, a parent to one of its children).

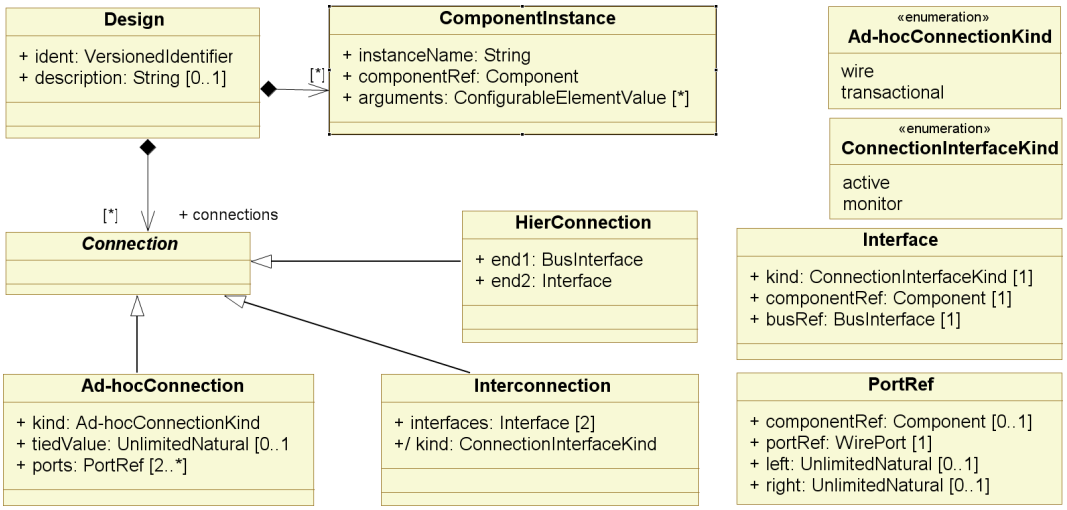

Figure 3: IP-XACT Design metamodel.

3 The UML Marte Profile

3.1 General overview

The new OMG UML profile for MARTE supersedes and extends the former UML profile for Schedulability, Performance and Time (SPT [20]). MARTE also adresses 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 media; 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/algorithms on 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: from the System-level view, to the RTL view, via the TLM view.

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.

As shown in Figure 4, 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 in 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 concepts. HwResource is specialized in the same way as the generic resources of the GRM package (lower part of Figure 4).

3.3 Time in MARTE

Both Resource and Allocation refer to the time model defined in the Time package of MARTE.

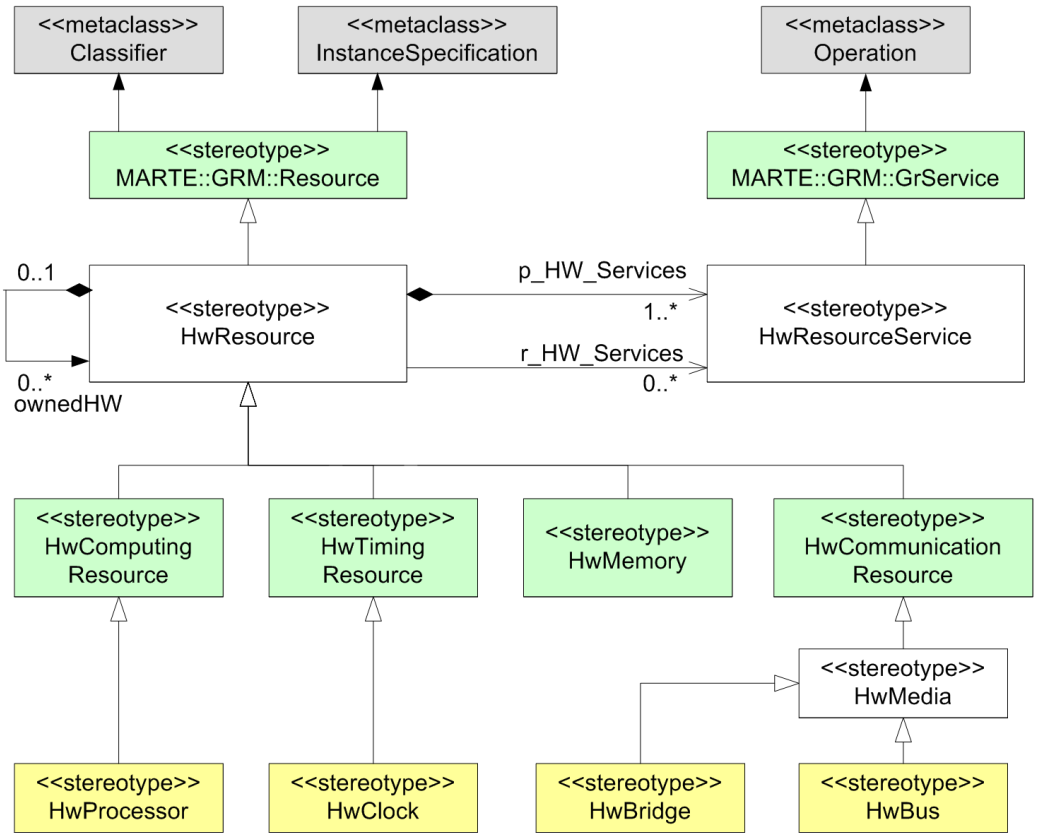

Figure 4: Excerpt from MARTE hardware resource profile.

While SPT considered only a time model based on *physical time*, MARTE introduces two models called *chronometric* and *logical* times. The former supersedes the SPT model and its time values are expressed in classical time units (second or one of its derivatives). The latter may "count" time in ticks, cycles, busCycles, or any other units. In fact, any event can define a logical clock logical 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 MARTE time model is to support multiple time bases, making it suited to distributed embedded system and modern electronic sytem design.

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 4.

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 [21].

This section describes our mapping rules from UML metaclasses to IP-XACT concepts. Following B. Selic [22], 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 that we call *UML profile for IP-XACT*.

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

3.4 Component

Similarily to other approaches mentioned before, we use 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 employ stereotype HwResource and some of its substereotypes. 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 bridges and timer components.

Figure 5 shows some UML components extracted from the Leon2 architecture example. All these components are transformed into IP-XACT components.

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* 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

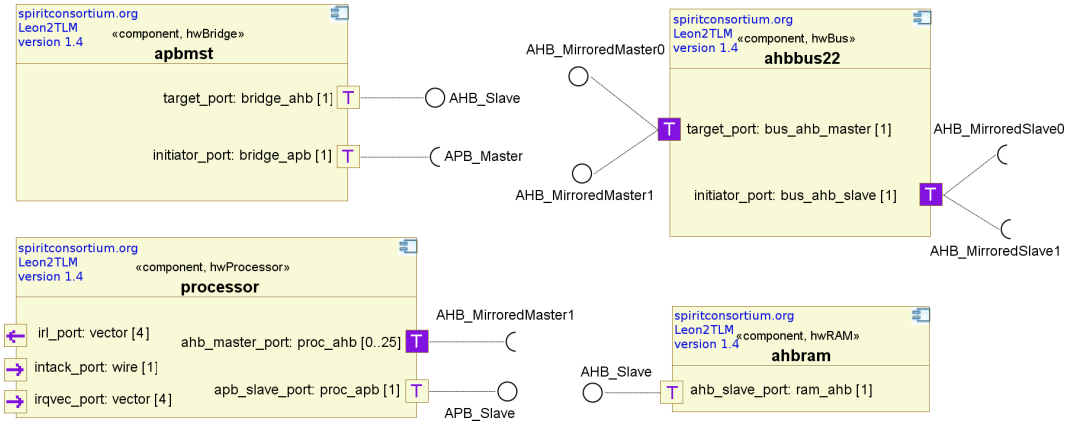

Figure 5: Components definition in UML.

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. Which 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. Like at RTL level, all ports are declared as wire ports (using «portWireType») whereas at TLM level, the models can have both wire (for adhoc connections) as well as transactional ports (identified by «portTransactionalType»). These stereotypes have properties specific to the port kind.

In Figure 5 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 vectors, size 1 for normal 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 the stereotype 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 IP-XACT mechanisms. Consequently, we need not define specific stereotypes for that. In the end, all the interfaces are defined in a separate class diagram. This separation avoids 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 type (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. Stereotype 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.

3.5 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 a bus AHB (AMBA High-Performance Bus). In the final design (see Section 3.6), 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 model explicitly abstraction and bus definitions, we use class diagrams to model them. Two stereotypes («abstractionDefinition» and «busDefinition») have purposedly been defined in our profile, as shown in Fig 6. 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 (property maxMasters) and slaves (property maxSlaves). All IP-XACT objects are uniquely identified by their identifier VLNV (Version, Library, Name, Vendor). Our model library introduces a custom data type called VersionedIdentifier to represent these identifiers.

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).

3.6 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

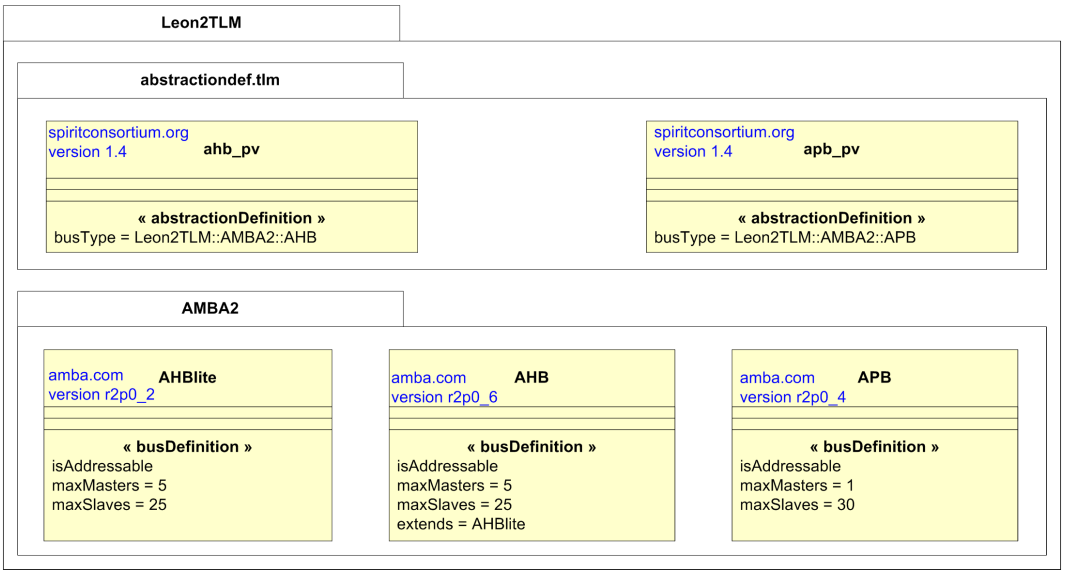

Figure 6: Bus and Abstraction Definition classes in UML Model.

diagrams for the integration phase. Our 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 can be stereotyped with «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 substereotype.

Figure 7 shows a partial composite structure diagram of the Leon2 design. Processor Leon2 (uproc) is connected to memory uahbram through component ahbbus.

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 IP-XACT standard and ad-hoc interconnections are modeled as connectors. Hierarchical interconnections are represented with a delegate dependency between the port of the parent and one 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 of 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

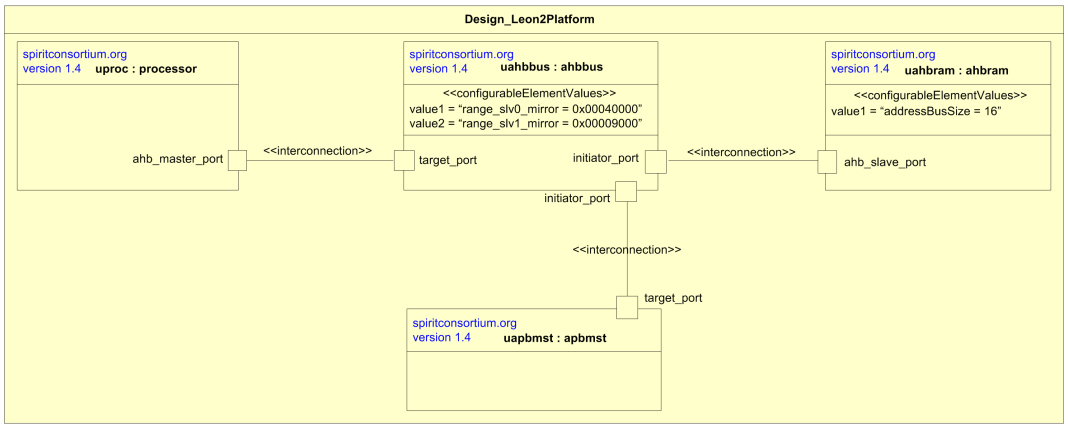

Figure 7: The Leon2 design in UML.

with the ports. As ports can have more than one bus interfaces, 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.

3.7 Implementation

Both our new profile and the examples shown above have been implemented within Papyrus (*http://www.papyrusuml.org*), an open-source graphical editor. In Table 1 below, we summarize our mapping rules from IP-XACT concepts to UML concepts. Following these rules, we have implemented an ATL transformation model that allows the automatic generation of IP-XACT models from UML models. For demonstration purpose, we applied it to make a full generation of the IP-XACT model of the Leon2 processor, provided by the SPIRIT consortium.

One difficulty came from the need to generate several IP-XACT files from one single UML model. The ATL code has been split into several rules and the UML model is parsed several times to produce the outputs. 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 files. Another ATL rule generates one file for each component.

Figure 8 summarizes the full model transformation process. A model (leon2tlm.uml), conforming to a metamodel (UML2), is here transformed into several models (Output.xmi) that conform to the 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 metametamodel (such as MOF or Ecore). Hence, for the model transformation using ATL, we

SPIRIT IP-XACT	UML
Processor/CPU	«hwProcessor»
RAM	«hwRAM»
AHB/APB Bus	«hwBus»
VLNV	VersionedIdentifier
Port	«wirePort», «transactionalPort»
Bridge	«hwBridge»
Timer	«hwTimer»
Timing Constraints	Model Library
Bus Definition	«busDefinition»
Abstraction Definition	«abstractionDefinition»
Bus Interface	«busInterface»
Bus Interface Mode	InterfaceModeKind
Design	Composite Structure
Component Instance	Part
Connection	Connector

Table 1: Mapping IP-XACT concepts to UML

provide two metamodel files, UML and IP-XACT. 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 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 the newer version of IP-XACT coming in future, we will have to generate the new XSD schema description and make little changes in the transformation code to make the tool compatible. The XSD schema files are converted to Ecore metamodel using Eclipse tool too. However, major modifications of the standard would probably require a thorough rewriting of the transformation model.

Figure 8 shows that, other than the metamodels, we have input and output files. As input files, we use the UML file obtained from Papyrus, in which MARTE was already available and our profile for IP-XACT has been implemented. Output files are in XMI format conforming to the IP-XACT description standard.

4 Behavioral description

Both 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 the IPs. The way IPs will interact, their timing characteristics or in general their behavior, is not covered with IP-XACT 1.4 and only depends on the associated SystemC code. However, early validations of IP-XACT design require an abstract description of

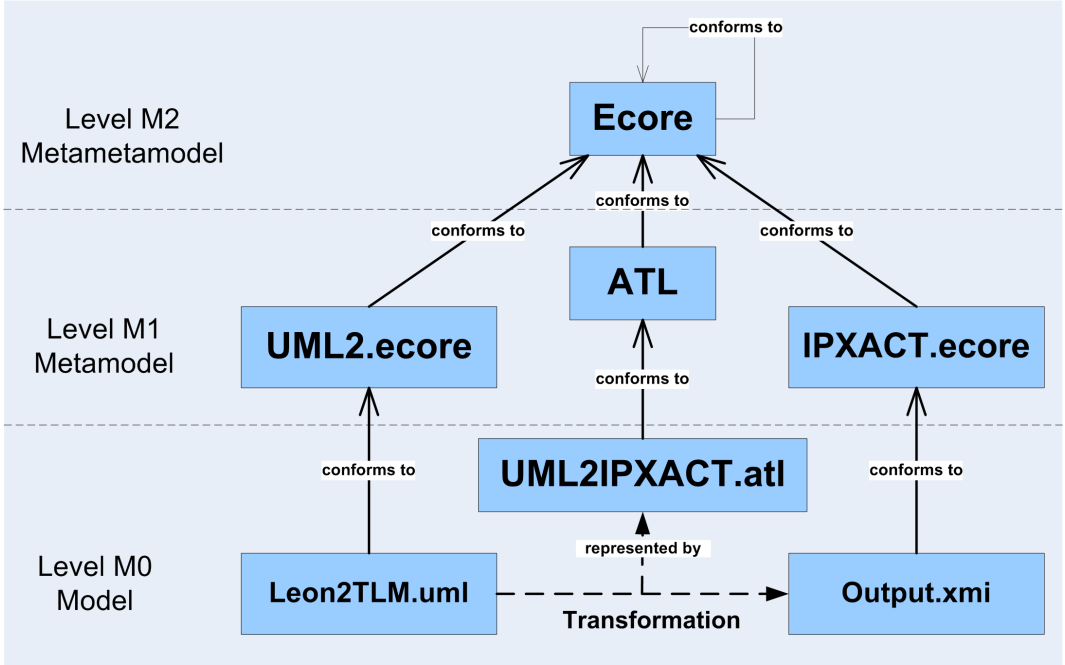

Figure 8: Model transformation and ATL

component behavior and of their characteristics. Early descriptions of the SPRINT Project (http://www.sprint-project.net/), which aims at providing an extension of IP-XACT, highlighted these needs.

Being a general-purpose language, UML offers various diagrams (like activity, state machine or interaction) to represent the system *untimed* behavior. MARTE profile, which includes a non normative Clock Constraint Specification Language (CCSL) [23], extends UML with time-related concepts. Combined with UML behavioral elements it provides a support for building models at a *timed* Communicating Process (CPT).

Figure 9 shows a timed activity that describes a memory at a higher level than the equivalent PV description, not to mention the RTL description. This memory supports two abstract services (MemRd and MemWrite represented as UML actions). Since we model hardware, a flow-oriented description has been chosen. The behavior is therefore specified by a UML activity. Sel (Selection) is an event used to trigger one of the two aboved-mentioned services. Eoa (end of access) is a 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. A MARTE clock (Clk) is added to control

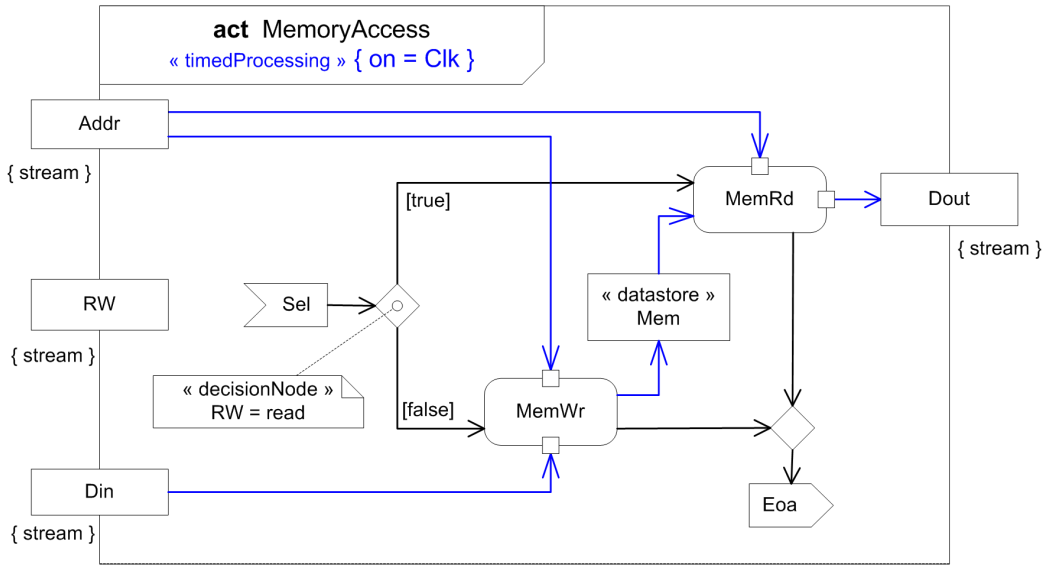

Figure 9: UML Activity for a memory with MARTE annotations.

flows and avoid memory dysfunctions. This is done by applying stereotype TimedProcessing to the activity. Combining IP-XACT models with such timed activities has many advantages.

Firstly, it provides a base for comparing RTL and TLM views of the same component. This is not an easy task since most of the time the two views are written by different people and do not even have the same interfaces (set of ports). Section 4.1 proposes a new approach to compare the two implementation views by generating observers directly from a timed activity.

Secondly, we go a bit further by augmenting IP-XACT specification with a new property called timingRequirement that characterizes the component timing properties. Details about this property and the way it is integrated in our observer-based validation process are given in Section 4.2.

4.1 Comparing RTL and TLM implementations

MARTE CCSL has been devised to combine synchronous, asynchronous and mixed operators to offer a wide range of modeling capabilities. If we restrict ourself to purely synchronous operators, we can represent the behavior with the Esterel language [24] along with its UML-like graphical syntax (Safe State Machines also known as SyncCharts). Using environments like Esterel Studio, we gain access to a large set of tools from the synchronous community to formally validate the time properties. Once it is validated, the question remains on how to validate the actual final implementations.

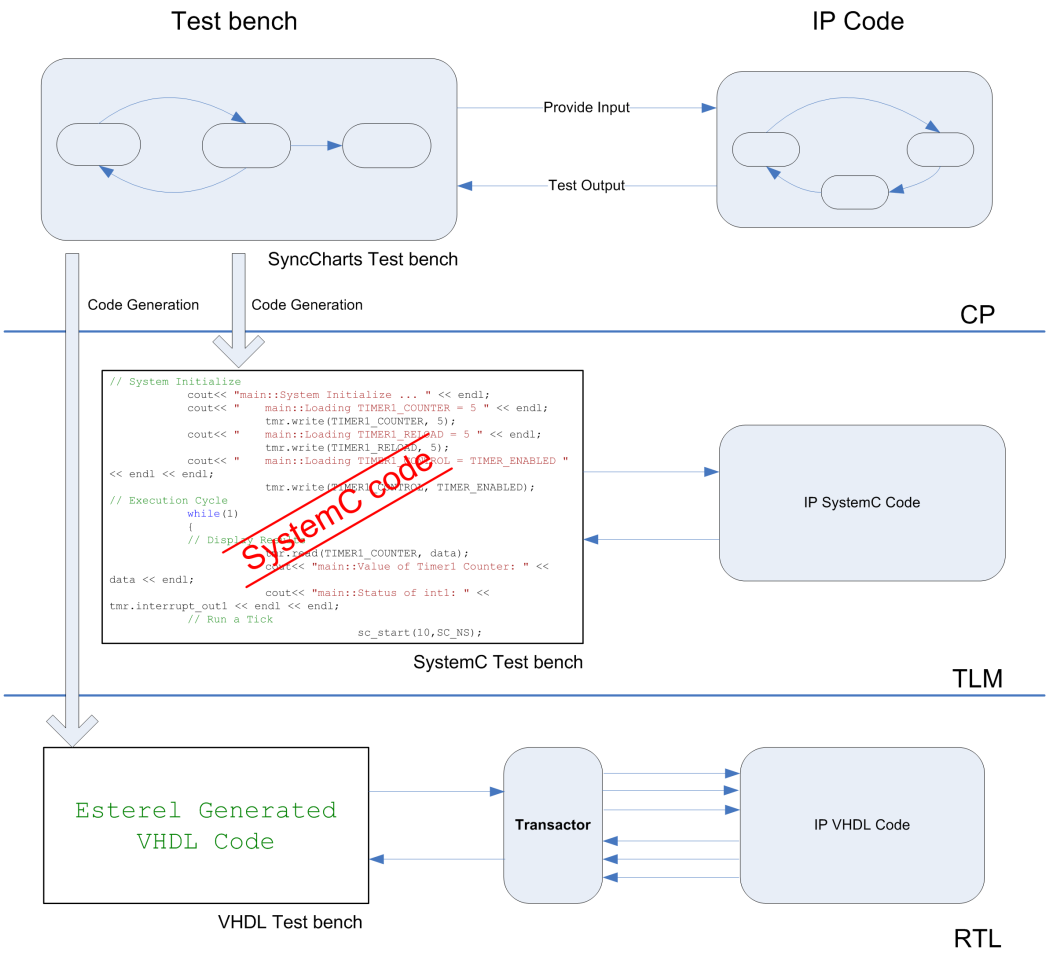

Figure 10: IP Test bench at various abstraction levels.

Figure 10 illustrates our proposed solution. We use Esterel Studio code generation facilities to produce VHDL or SystemC code. However, from an abstract CP-level model it is impossible to generate complete full-fledge TLM or RTL implementations. It would only be possible to generate skeletons of the behavior. That was our first attempt. Unfortunately, it appeared that models at different levels are often built by different teams. This results in implementations that do not have the same structure at all and, very often, do not even have the same interface. It may be very difficult to impose a common structure to all teams. Instead, we chose a more practical approach. We want to validate properties on *existing*

implementations and show that implementations at different levels exhibit the same timing characteristics than the synchronous model, used as a *golden model*. Here is how we proceeded for Leon2 plateform architecture.

We have built testbench models in Esterel. These testbenches are composed with the CP-level models of components from the Leon2 case study (see the upper part in Figure 10). Using Esterel Studio verification tools, time properties are checked. Both the testbench model and the golden model are ajusted to exhibit the required time properties. For the memory, we have proceeded in the same way, a synchronous program equivalent to Figure 9 is used as the golden model. Then, we generate SystemC code for the TLM level and VHDL code for the RTL level. 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 various 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 RTL and TLM implementations.

Obviously, since interfaces of the RTL and TLM models are not necessarily identical, the validation at the RTL (and possibly also at the TLM level) may require some 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 timing properties.

To ease the building of testbenches at the CP-level, we have used a library of predefined *timing 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. To give examples, we have identified the pattern cause/effect, when a signal is set long enough, then another signal must be set within a given time range and must remain valid long enough. The pattern watchdog states that if a signal is not set during n clock cycles, then a timeout signal must be set for m clock cycles. Many more patterns can be identified.

4.2 Augmenting IP-XACT with timing requirement information

If we do not restrict to synchronous evolutions, we can specify IP timing by clock constraints expressed in CCSL. We propose to add to IP-XACT component model element a new property called timingRequirement, in which CCSL expressions can be added.

In CCSL, we first need to identify clocks. As said in Section 3.3, a (logical) clock can be associated with any event. For the memory example, the first obvious clock is Clk given when applying the stereotype TimedProcessing. We also consider clocks ^Sel and ^Eoa associated with events Sel and Eoa. Other clocks are ^Addr, ^RW, ^Din, ^Dout for eponym parameters, where events are changes in parameter values.

These clocks are now to be constrained to express causal relationships. The first constraint imposes alternation of occurrences of ^Sel and ^Eoa. This is typically an asynchronous

clock constraint involving the precedence relation. The second constraint says that ^Eoa is a subclock of Clk. This synchronous constraint states that each instant in ^Eoa is coincident with an instant in Clk. Signal Eoa is emitted at the end of the clock cycle following the occurrence of Sel. This is expressed by the third constraint. Other constraints are stability requirements: there is no new tick of a clock for a specified duration or time interval. This is a clock constraint pattern. It recourses to a locally defined clock ^SelA (constraint 4). The stability itself is given by the fifth constraint. Similar constraints have been applied to ^Din and ^RW, they are omitted for the sake of simplicity.

$$\hat{}Sel \quad \text{alternatesWith } \hat{}Eoa \tag{1}$$
$$\hat{}Eoa \quad \text{isSubClockOf } Clk \tag{2}$$
$$\hat{}Eoa \quad = \quad (\hat{}Sel \text{ sampledOn } Clk) \text{ delayedFor } 1 \text{ on } Clk \tag{3}$$
$$\hat{}SelA \quad \text{is} \quad \hat{}Addr \text{ sampledOn } \hat{}Sel \tag{4}$$
$$\hat{}SelA \quad \text{alternatesWith } (\hat{}SelA \text{ delayedFor } 2 \text{ on } Clk) \tag{5}$$

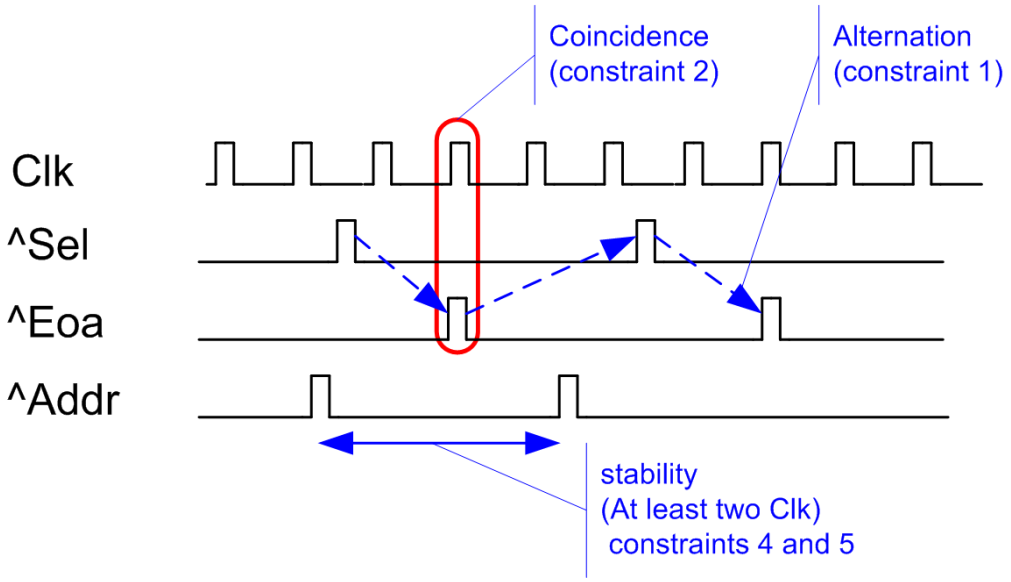

Figure 11: CCSL simulation waveform.

We have built a tool, called TimeSquare, that compiles CCSL constraints and produces simulation traces (see Figure 11). Traces have been adorned with comments to refer to some of the above constraints. If the system is overspecified or if there are inconsistencies, the compilation fails with an appropriate message. If the system is underspecified, the

simulation is not deterministic and several simulation policies are offered to the user. One can add constraints, make random choices or interactively select a solution. Of course, since this is simulation, we generate some valid traces, not all valid traces. These traces can play the exact same role as of waveforms or timing diagrams in paper datasheets.

Currently these traces are generated in the Value Change Dump (VCD) format, which is the dump format defined as part of Verilog HDL IEEE Standard. However, we could have chosen to generate some SystemC or VHDL testbenches, instead.

5 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 IPs. After describing our UML profile for IP-XACT that elaborates on MARTE to model IP-XACT designs, we propose to use 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 for paper datasheets. We also show how this specification can be used to generate testbenches of valid scenarios that must be satisfied by component implementations whatever their level of abstraction and the language used. 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.

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] 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).

[21] 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

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

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

[24] Benveniste, A., Caspi, P., Edwards, S.A., Halbwachs, N., Guernic, P.L., de Simone, R.: The synchronous languages 12 years later. Proceedings of the IEEE **91**(1) (2003) 64–83

Contents

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

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

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