On PSL Properties Re-use in SoC Design Flow Based on Transaction Level Modeling

Nicola Bombieri
STMicroelectronics
Agrate-Milano, Italy
nicola.bombieri@st.com

Andrea Fedeli
STMicroelectronics
Agrate-Milano, Italy
andrea.fedeli@st.com

Franco Fummi
University of Verona
Verona, Italy
franco.fummi@univr.it

Abstract

In this paper we present some key concepts concerning the Properties Specification Language (PSL) utilization in a system level verification flow for System on Chip (SoC) designs. As Transaction Level Modeling (TLM) is the de-facto reference model for SoC design flow, we evaluate PSL adoption in TLM context. How to save time and effort in the verification phase during system development steps and how to overcome global system verification limitations through a compositional approach are discussed. Two PSL-based techniques, "properties re-use" and "properties refinement", are described and compared in terms of refinement effort and simulation speed delay.

1. Introduction

As RTL design is a logic implementation methodology, it is simply not suitable for designing a multi-processor system with complex protocols, and its relatively low level of abstraction makes its use in IP integration and evaluation methodologies tedious, time consuming, and error prone [1]. Consequently, SoC design with an RTL integration platform will result in an increasing design failure rate. For these reasons, functional specification and performance analysis are becoming a challenge easily bringing to expensive re-spins [2].

Electronic System Level (ESL) design is a SoC design performed at a level of abstraction above RTL. ESL design enables functional verification and performance analysis of complex architectures and protocols very early during system development. At this high level of abstraction, a designer can run large numbers of test cases to identify and fix corner case problems that are often discovered only at system integration, too late to safely operate corrections in short times. With an ESL design methodology, HW/SW partitioning can be paired with up-front performance analysis that identifies bottlenecks long before implementation. The need for additional processing capacity and deployment of dedicated hardware accelerators is identified while there is still time available to explore different ways to meet requirements.

SystemC language [3] well supports ESL design as it satisfies all requirements of data timing, system concurrency and communication characteristics necessary to model and evaluate performance of real hardware. SystemC models can be equipped with parametric data types that enable analysis of system performance (data throughput, datapath latencies, and power consumption) as well as utilization of processing units, memory and system buses, for different values of the parametrized quantities.

Using SystemC, the designer can model the entire system, hardware and software, in terms of both functionality and timing in the early stages of design flow. The SoC architecture is designed with SystemC IP blocks connected via Application Program Interfaces (API's) to implementation-independent high-level bus models. These architectural models are simulated using system type transactions as function calls and packet transfers, and are thus called Transaction Level Models (TLM) [4].

Using TLM, the architecture is designed and verified in terms of functionality as characterized by high level block input/output events and inter-block data transfers. System IP components and buses may be modified or replaced in an easier way than at RTL and simulated more than 1,000 times faster. Therefore, designers can quickly optimize the design to achieve the best tradeoff. The transactional model of the resulting optimized architecture constitutes an executable specification that drives the entire subsequent RTL implementation. Thus, a TLM-based design flow provides the design flow with a series of steps that starting from an abstract system description evolves toward more detailed implementations till it gets to RTL. Every step of design flow that involves a change in the system design needs a verification check, in order to preserve the *golden model* functionalities ascertained at the preceding step.

The verification activity, therefore, involves three main phases: first, the design implemented at the higher abstraction level is validated considering the system functionalities; then, design changes following architecture exploration and performance analysis are validated taking into account the temporal behavior. Finally, whenever a step of the design flow implies a change in the system design, a further verification check is required in order to preserve

Proceedings of the Sixth International Workshop on Microprocessor Test and Verification (MTV'05)
0-7695-2627-6/05 $20.00 © 2005 IEEE

Level	OCP, CoWare, Synopsys	Frank Ghenassia STMicroelectronics Unicad	Caj-Gajski Univ. of California Irvine	Adam Donlin Xilinx Research Labs.
	Algorithmic			
TL3	Message		Specific Model	Communicating Processes (Timed/Untimed)
TL2	Transaction	TLM_Architecture	PE assembly model	Programmer's View (Timed/Untimed)
TL1	Transfer	TLM_Microarchitecture	Bus arbitration model	Cycle Accurate
	RTL			

Figure 1. TLM Levels matching.

the *golden model* functionalities ascertained at the preceding step.

Several papers regarding functional verification based on simulation for TLM design flows have been presented [5, 6, 7]. Assertion-Based Verification (ABV) using the PSL language [16] is currently gaining acceptance as an essential method for functional verification of SoC [7]. However, to the best of our knowledge, there are no works in the literature precisely related to exploiting PSL properties *re-use* for functional equivalence checking in TLM based SoC design flow. In the present paper we depict a verification methodology based on PSL in order to guarantee functional properties re-use and to overcome global system verification limitations.

The experimental results show that the price to pay adopting this approach, that is an additional overhead in simulation time, is acceptable in comparison with saved effort for properties refinement and (and avoided refinement mistakes risks that manual transformation comes along with). We give emphasis to the additional overhead simulation and saved translation effort trade-off more than to property coverage measurements or how many properties are needed to provide a good coverage to guarantee a reasonable certainty of system correctness.

The paper is organized as follow: Section 2 summarizes the most important concepts of TLM. Section 3 presents the verification methodology and the adopted compositional approach. Section 4 presents a case study. Concluding remarks and future plans are summarized in Section 5.

2. Transaction Level Modeling

TLM is a high-level approach to digital systems modeling where details of communication among modules are separated from details of implementation of functional units or details of communication architecture [8]. Transaction-level is a design and verification abstraction above RTL, that more emphasis on functionality of data transfer (what data are transferred to, to which location, and from what location) than on their actual implementations (e.g. on actual protocol used for data transfer).

In spite of its name, *transaction-level* does not denote a *single* level of description; rather, it refers to a group of abstraction levels, each varying in the degree of functional or temporal details used and expressed.

2.1. TLM Levels

So far, a common agreement on terminology for TLM levels is still missing. Different interpretations (and terminology for the same concepts) have been proposed by both industry and academia [4, 9, 10, 15]. Main contributions are matched in Figure 1. Factoring out common elements, key concepts are:

1. To implement a system at higher level means to implement the system in a more abstract way, that is to leave implementation details in order (mainly) to speed-up simulation for functional verification purposes.

2. To implement a system at lower level means to add implementation details to the system in order to simulate it in a more accurate way (for performance analysis purpose).

Main use and features of every TLM level are summarized in Figure 2, according to common lines of different proposals reported in Figure 1.

2.2. SystemC 2.1 for TLM

SystemC, as a broad-range level of abstraction modeling language, well addresses TLM. However, lack of established standards and methodologies means that each organization adopting TLM has to invent its own usage methodologies and API's. In addition to this redundant cost, these

Level	Use	Features
TL3	Executable specifications and first level of functional partitioning of data and control. System proof of concepts. Functional verification with fast simulation.	➢ Implementation architecture-abstract. ➢ Untimed functionality modeling. ➢ Event-driven simulation semantics. ➢ Point-to-point Initiator-Target connection. ➢ Abstract data types.
TL2	Architectural performance and detailed behavior analysis. HW/SW partitioning and co-development.	➢ Mapping ideal architecture into resource-constrained world. ➢ Memory/Register map accurate. ➢ Event driven simulation ➢ Bit-width and transfer-size constrained data types to allow mapping to bus bursts or fragments of bursts.
TL1	Detailed analysis and low level SW development. Modeling clock-accurate interfaces for abstract simulation models of IP blocks such as embedded processors. Clock-accurate performance analysis.	➢ Clock-accurate protocols mapped to the chosen HW interfaces and bus structure. ➢ Interface pin are hidden. ➢ Byte-accurate data Transactions have internal structure (protocols, data, clock). ➢ Transactions map directly to bus cycles. ➢ Parametizable to model different bus protocol and signal interfaces.

Figure 2. TLM Levels use and features.

methodologies easily differ, making IP exchange and reuse more difficult.

OSCI Working Group has recently released a TLM standard [11] to rigorously define implementation rules; besides implementation improvements, we want to exploit this standard for PSL-based functional verification. We will take that standard as a guideline to map different levels and behaviors of SoC components [11].

3. PSL-based Verification Methodology

A realistic TLM-based SoC design flow usually combines both top-down and bottom-up approaches: whilst going top-down, starting from a set of specifications implemented at algorithmic level (TLM level 3), implementation details are added going through TLM levels downward to RTL; whilst moving bottom-up, re-use of already designed and verified components is exploited to reduce design effort. As a consequence, the verification phase throughout SoC design flow performs both reference model validation and equivalence verification between reference model and refined(abstracted) model. That is, equivalence verification checks that different representations of the same system portion are kept consistent whilst moving down (refining) or moving up (abstracting) to get to the final design whilst performing architecture exploration, refinement, and blocks assembly.

3.1. PSL-properties in TLM context

The verification methodology exploits PSL [16] for properties specification in TLM context and it aims at three different goals:

1. To validate design functionalities at higher implementation level (i.e., TLM level 3).
2. To validate lower level implementation designs whilst carrying out architectural choices and performance analysis (architecture exploration).
3. To check that different representations of the same system functionalities are kept consistent whilst moving down (refining) or moving up (abstracting) to get to the final design.

The first goal can be reached expressing a number of PSL properties to provide a good enough coverage to guarantee design functionalities correctness. At this abstraction level, design implementation is untimed (see Figure 2). Functionalities are checked regardless time, delays and throughput. For this reason, PSL properties are exploited only as assertions, regardless PSL temporal layer. For example, a functional property checks if an IP slave implementing an integer square root produces correct results regardless how many time it spends and which communication protocol is shared with its master (Figure 3).

```
// TLM level 3 property:
// if root arg is p and its integer root is q, then
// p-2q-1<q*q<=p;
    vunit root_in_tolerance {
    ...
    assert always (p>0 -> p*p ≤ q && p*p > p-2*q-1)
    }
```

Figure 3. Functional property example.

The second goal can be reached checking a set of PSL properties to provide a good enough coverage to guarantee design correctness, taking into account temporal constraints, communication protocols, time delays and throughput. In this case, properties exploit all PSL power, involving complex temporal relations between signals. Figure 4 shows an example: considering delay in producing

integer square root at TLM level 1, the property checks if root IP is able to produce a result for any received input. Note that the property does not check square root numerical correctness.

```
// TLM level 1 property:
// an output value is eventually calculated (result_
// _is_ready) for any received input (number_is_ready)
    vunit root_causality_more_ins_than_outs {
    ...
    assert G{{number_is_ready(0);number_is_ready(0)[->]}
           && {!result_is_ready(0)[*]}}(false);
    }
```

Figure 4. Temporal property example.

The third goal can be achieved checking functionalities expressed for the first goal on the refined implementation. Two description will be then considered functionally equivalent if they satisfy at least the same properties set. Meaningfulness of this equivalence criterion relies in meaningfulness of property set, which we intend to measure by means of a *property coverage* metrics, as defined in [13].

Property sharing between two descriptions to be compared can be performed in two different ways: by direct reuse if the interface is kept unchanged during the refinement step, and by property refinement if interface has been reshaped. The first solution is obviously the best as let us save properties translation time avoiding translation mistakes.

Note that by using a common property specification language (PSL) from the highest TLM level throughout RTL, we can use a common coverage metric (the property coverage metric) to validate and to verify model as well as properties re-use.

PSL-based verification methodology relies on the *substitution principle*, a compositional approach that is explained in the next section. Main motivation for that are:

• *Functional properties re-use.* As explained before, we aim to save functional property refinement time and effort.

• *Easier verification.* Checking equivalence of two whole system descriptions is by far out of capacity of any technique applicable today. System level simulations are usually aimed at checking very specific working scenarios, as simulation of whole system is typically too expensive even taking into account a very rough representation of physical aspects.

• *Design and verification modularity necessity.* The modularity feature plays a fundamental role in SoC design flow as it allows design and verification teams independence. In many organizations, there are more distinct groups of engineers. One group understands the application domain very well but it is not particularly expert in C++ nor interested in details of TLM transport level or signal level communication protocol. Another group does not necessarily understand application domain but it does know the underlying protocol and it is expert in C++ techniques needed to model them. Because of this division in skills and focuses, it is useful to define a protocol specific boundary between these groups.

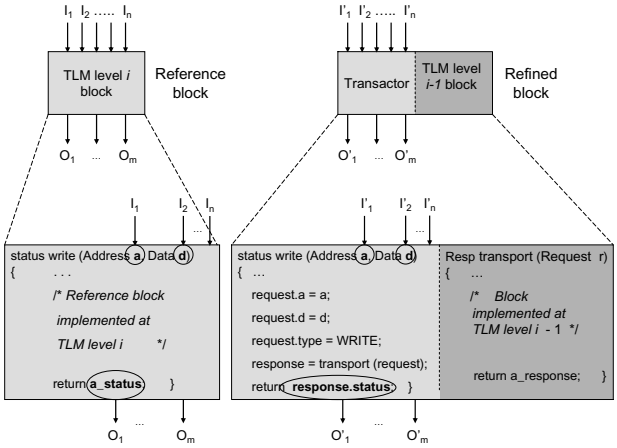

Figure 5. Transactor example

3.2. The substitution principle

In order to allow PSL properties re-use, easier verification, and modularity necessity, we present an essentially compositional approach: by keeping the interface between one component and the rest of the unaltered system , design and verification are eased by large when moving from a level of abstraction downward to the immediately less abstract level of description.

TLM API's are unchanged while moving through design refinement, exploiting the *transactor* concept. A transactor works as a translator from a function call to a sequence of statements implemented at a lower abstraction level (Figure 5). For example, between a TLM level 1 block and a RTL block a transactor gets an abstract request from the channel (TLM side) and waits for an opportunity to send this out over RT level bus by means of a RTL signals sequence. It then waits until it receives a response from RTL bus, at which point it puts the abstract response back into the channel.

Then, we check equivalence between two bordering levels of a SoC design implementation in more easier steps.

A SoC design implemented at TLM level *i* is composed by a set of communicating and concurrent blocks. An example is showed in Figure 6, where block *A* is Master for Slave block *B* and *B* is Master for Slave block *C* as well. System blocks communicate by TLM level *i* communication interfaces (TLM API's) in row 1 of Figure 6. As a consequence of refinement, block (*C*) and the corresponding component implemented at lower level undergoes equivalence verification. As explained before, we can check blocks functional equivalence with two different techniques. Next Subsection presents and compares these techniques in a more detailed way.

After a positive response, the original block can be substituted by the "equivalent" block whilst the remaining system does not change (second row of Figure 6). Note that

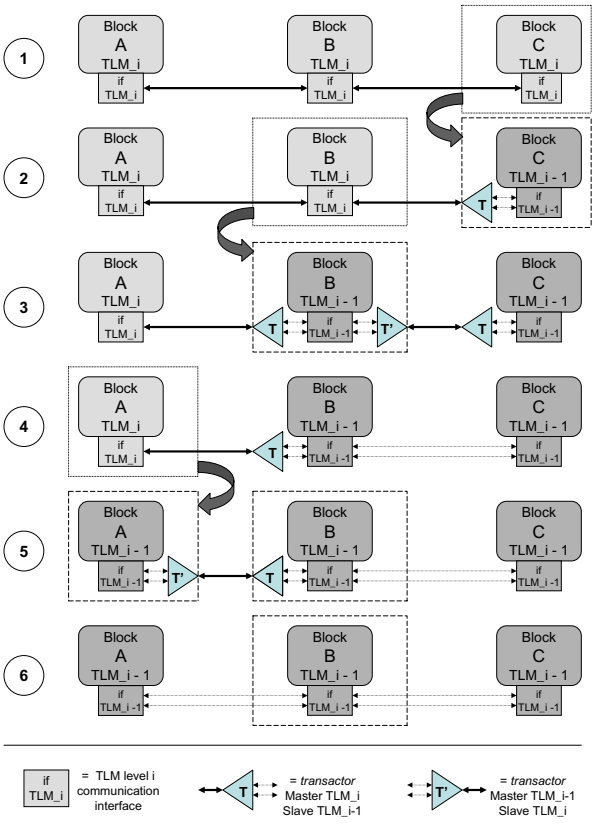

Figure 6. Substitution principle example.

this is possible using transactor *T* that bridges gap between different abstraction levels of B and C.

This substitution principle allows on one side to use at any simulation step block description mostly suited to that simulation needs, with more or less details. On the other side it reduces verification task complexity by a "divide et impera" approach, giving a chance to verify even very complex systems that would not be treatable if taken as a whole.

Third row of Figure 6 illustrates verification and substitution step concerns block B. Note that transactor *T*' is inserted for B-C blocks communication. This allows application of modularity substitution and *properties re-use* verification techniques as explained in next section. By using the *properties translation* technique, instead, T' insertion can be avoided, jumping to row four.

As step by step all system blocks undergo the same procedure (fifth and sixth rows of Figure 6), system level verification is achieved in a compositional way, by verifying and debugging each component refinement, and equivalence of interconnects separately.

In SoC design flow, some system parts (i.e., SW side components) do not need to be refined. Their initial high level implementation does not change during the whole design flow. SoC design becomes therefore a hybrid representation where transactors bridge gaps between components implemented at different levels of abstraction interacts.

```
// TLM level i write property:
always ((a ≥ LOW && a ≤ HIGH && d == DATA )
         → eventually (a_status == SUCCESS))


// TLM level i-1 (refined) write property:
always ((r.a ≥ LOW && r.a ≤ HIGH && r.d == DATA
         && r.type == WRITE)
         → eventually (response.status == SUCCESS))
```

Figure 7. Property refinement example.

3.3. Properties re-use

Matching API's of reference block with API's of its refined version is the starting point of properties re-use technique. Figure 5 shows an example of this mechanism.

Exactly the same property is expected to be checkable both in reference block and in the refined block thanks to transactors usage, since interfaces are kept after the refinement step (i.e., see rows 1-2 of Figure 6 for block C refinement). However, the twofold transactor employment is also the main drawback cause of this approach, since it increase simulation time.

Properties re-use technique is always applicable following substitution principle methodology. In fact, a complete system simulation is performed before taking out transactors (fourth row of Figure 6).

Simulation time can be reduced by verifying system design without transactors and translating properties set, as explained in the next Subsection.

3.4. Properties refinement

The initial set of properties is translated for the refined block at a lower abstract level. Figure 7 shows an example of property translation. The first property holds if a simple *write()* statement belonging to TLM level_i description successfully executes (see Figure 5). Note that function parameters and returned value constitute the I/O block elements. Corresponding refined property expresses the same meaning even if it acts on a different block pattern.

In the simple example above one property matches another property. However, a more complex property could be translated in a set of more refined properties. In that case, the first holds if and only if all the refined properties hold.

Translation process transforms the initial property set into a refined property set and it follows rules dependent on architectural choices (communication protocol, etc.) made on a refinement step. That process could be (semi)automatically made following the TLM standard [11] that imposes specific translation rules.

Therefore, equivalence between two blocks is restricted to a set of formal properties and its quality strictly depends on properties completeness as well as properties checking methodologies [12, 13].

4. Case Study

The key concepts expressed in this paper have emerged by analyzing the *face recognition system*, a case study de-

signed by STMicroelectronics (Central R&D of Agrate-Milano, Italy), for the SYMBAD project [14]. SYMBAD has led to an integrated design and verification methodology for re-configurable systems.

Main problems found in face recognition design flow are the following:

- No implementation and verification modularity. API's have been re-implemented at every design flow step.
- No verification modularity. Almost all system refinement steps could be checked by simulation and simulation could start only after the whole system refinement.
- PSL properties formulation effort spent only for last step of verification.

The standard TLM proposed by OSCI Work Group has been applied in real case study above.

Table 1 summarizes statistics of Face Recognition System code concerning total number of SystemC *SC_MODULE*, processes, *SC_PORT* and code rows at every abstraction level. Note that at every refinement step both *SC_MODULE* and *SC_PORT* numbers grow, as well as code rows, due to implementation hierarchy and HW/SW partitioning.

Level	*SC_MODULE*#	Proc.#	*SC_PORT*#	C. Rows#
1	15	14	14	5125
2	17	16	18	5977
3	18	16	19	6134
4	21	23	19	8619

Table 1. Face Recognition System features.

All implementation problems concerning modularity have been overcome by using OSCI *convenience* and *TLM interfaces*. A standard extension has been mandatory as its first release does not completely cover higher TLM level.

As SystemC language *flavor* is currently under discussion inside IEEE PSL standardization committee (IEEE-1850), to address SystemC description verification we follow essentially the approach described in [7] to perform PSL→SystemC checker translation.

Properties *re-use* and *refinement* techniques have been applied to check *div*, *distance* and *root* blocks. A set of 19 properties has been created for golden model (TLM level 3). First, all properties have been re-used in RTL system description, measuring system simulation time delay. Then, properties have been manually translated and checked in the system without transactors. Table 2 summarizes simulation times (in seconds) comparing the presented techniques (third and fourth columns) in respect to the original system with no properties (second column). Third and fourth rows point out real cost in terms of time delay spent adopting properties re-use approach to avoid their translation effort.

Properties check increases very marginally simulation time: 4.8% at TLM level_3, 5.6% at RTL and 3.8% at RTL with transactors. On the contrary, simulation time remarkably grows (29.7%) adding transactors to RTL implementation. However, in our opinion, transactors utilization approach is acceptable as it saves properties translation effort and avoids potential translation mistakes.

Level	Face (s)	Face + PSL re-use (s)	Face + PSL translation (s)
TLM level_3	0.205	0.215	-
RTL	18.272	-	**19.296**
RTL + transactors	23.692	**24.593**	-

Table 2. Face Rec. System simulation times.

5. Concluding Remarks

PSL utilization in a TLM-based verification flow for SoC designs has been analyzed in this paper. The substitution principle, a compositional approach that saves time and effort in the design implementation and verification allowing the *properties re-use* technique has been proposed. This approach and the *properties refinement* alternative have been applied on a real case study and the obtained experimental results have been compared. Even if the former increases system simulation time, it is better since it avoids tedious and expensive properties translation work and its mistakes risk.

References

[1] J. Krasner. *"Embedded Software Development Issues and Challenges"*. Embedded Market Forecaster, July 2003.

[2] A. Ziv. *"Functional Verification and the SoC Challenge"*. International Seminar on Application-Specific Multi-Processor SoC, MPSOC 2003.

[3] http://www.systemc.org

[4] L. Cai, D.Gajski. *"Transactional Level Modeling: An Overview"*. Proc. of IEEE Int. Conf. On Hardware/Software Codesign & System Synthesis, 2003, pp. 19-24.

[5] C.Norris Ip,S.Swan. *"A Tutorial Introduction on the New SystemC Verification Standard"*. Proc.of IEEE DATE, 2003.

[6] D.S. Brahme, S. Cox, J. Gallo, M. Glasser, W. Grundmann, C. Norris Ip, W. Paulsen, J.L. Pierce, J. Rose, D. Shea, and K. Whiting. *"The Transaction-Based Verification Methodology"*. Technical report #CDNL-TR-2000-0825. Cadence Berkeley Labs, August 2000.

[7] A.Dahan, D. Geist, L. Gluhovsky, D. Pidan, G. Shapir, Y. Wolfsthal, L. Benalycherif, R. Kamdem, and Y. Lahbib. *"Combining System Level Modeling with Assertion Based Verification"*. Proc. of IEEE ISQED, pp. 310-315, 2005.

[8] T.Grotker, S. Liao, G. Martin, S. Swan. *"System Design with SystemC"*. Kluwer Academic Publishers, 2002.

[9] F. Ghenassia, et al. *"Using Transactional Level Models in a SoC Design Flows"*. Kluwer Academic Publishers, 2003.

[10] A. Donlin *"Transactional Level Modeling: Flows and Use Models"*. Proc. of IEEE CODES+ISSS, 2004.

[11] A. Rose, S. Swan, J. Pierce, and J.M. Fernandez. *"Transaction Level Modeling in SystemC"*. TLM library white paper, www.systemc.org.

[12] S. Katz, O. Grumberg, D. Geist. *"Have I Written Enough Properties? A Method of Comparison Between Specification and Implementation"*. Proc. of the 10th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods, pp. 280-297, 1999.

[13] F.Fummi, G.Pravadelli, A.Fedeli, U.Rossi, F.Toto. *"On the Use of a High-level Fault Model to Check Properties Incompleteness"*. Proc. of ACM/IEEE International Conference on Formal Methods and Models for Co-Design (MEMOCODE), Le Mont Saint Michel, France, 24-26 June, 2003

[14] M. Borgatti, A. Capello, U. Rossi, G.L. Lambert, I. Moussa, F. Fummi and G. Pravadelli. *"An Integrated Design and Verification Methodology for Reconfigurable Multimedia System"*. Proc. of DATE, vol. 3, pp. 266-271, 2005.

[15] A. Haverinen, m. Leclercq, N. Weyrich, and D. Wingard *"SystemC based Communication Modeling for the OCP Protocol"*. White Paper, v.1.0, October 14, 2002.

[16] Accellera, *Property Specification Language Reference Manual*, v1.1, June 9, 2004.