Specification of Adaptive HW/SW Systems in SystemC

Fernando Herrera*, Eugenio Villar*,
Philipp A. Hartmann†
*University of Cantabria, Spain
†OFFIS, Germany

Abstract

This paper proposes a SystemC-based specification methodology of adaptive embedded systems to be implemented on a platform including one or more processors, thus supporting the execution of embedded software, and digital hardware with capabilities of partial dynamic reconfiguration (DRHW). For it, it proposes the collaboration of two specification methodologies: HetSC and OSSS+R. The main issues for the integration of these specification methodologies are addressed. This includes how to install and use them together, which is the structure of the specification, how adaptivity is specified for SW and DRHW implementation, and the syntactical and semanticual issues related to the MoC interface implicit in their connection. [1]

1 Introduction

From the early versions to the last update of the International Roadmap for Semiconductors (ITRS) [ITR07], the need for the evolution of the design methodology towards a new framework centered on a system-level specification methodology has been pointed out. It has motivated an important research effort to overcome challenges like finding a unified system-level specification language and methodology [SV07]; improving the software and hardware productivity gap [ITR07]; and minding features like heterogeneiity and adaptivity [ÅRH+06], which have to have some explicit deal at the specification level.

Fortunately, some of these challenges are being overcome. SystemC has become the IEEE1666 standard and gained strength as unified specification language. In addition, as well as features for system-level and hardware specification, it provides facilites which have enabled the development of several extensions, like the HetSC methodology [HV07], for improving the support of heterogeneity, and OSSS+R[SON06] to enable the abstract specification of DRHW.

[1]Work co-funded by the European Commission within the Sixth Framework Programme as part of the ANDRES project (IST-5-033511).

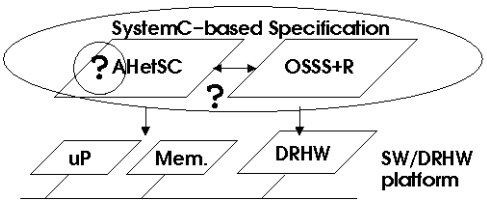

Figure 1: Main issues of this paper.

However there are still open issues, such as how to integrate in SystemC abstract HW and SW specification methodologies supporting adaptivity; the support of a full HW/SW implementation flow from the resulting specification methodology; and the interpretation, modelling and implementation of software adaptivity. As reflected in Fig. 1, this paper addresses these issues while proposing the integration of two specification methodologies: OSSS+R and HetSC. This integration yields a specification methodology supporting an implementation flow able to target part of the adaptative functionality to embedded software, while the other part is implemented as DRHW. The paper first reviews the relevant related work (section 2) and then, the OSSS+R and HetSC methodologies (section 3). Then, different aspects of their integration are introduced: which infrastructure need to be installed and how to use it (section 4.1), the structure of the specification (section 4.2), the way adaptivity is specified (section 4.3), and the syntactical and semantical issues of the connection (section 4.4), which requires an analysis of the underlying MoC(s) considering adaptivity. Finally, section 5 gives the main conclusions and future lines of this work.

2 Related Work

Several frameworks for system-level heterogeneous specification have been proposed. Ptolemy [BLL+07] and Metropolis [DDM+07] are among of the most know ones. Both are based on Java, although are different in their conception. Ptolemy is a component-based methodology supporting hierarchical heterogeneity since the execution semantic is distributed in a hierarchy of *director* classes. However, Metropolis supports amorphous heterogeneity by means of a class-based infrastructure which enables an orthogonalization of computation and communication. Another important approach is SysML (Systems Modelling Language) [Wei08], a UML2.x profile to provide a domain-specific modelling language. This language must rely on an implementation language for its simulation.

In a SystemC context, several research activities have improved the support of heterogeneous specification. For instance, SystemC-H [PS04] is a methodology that proposes a general extension of the SystemC kernel for the support of different MoCs. The extension includes a solver for each MoC and requires the modification of the SystemC library, which is not always

978-1-4244-2265-4/08/$25.00 © 2008 IEEE

worthy for the untimed MoCs supported. SysteMoC [FHT06] focuses on providing a methodology with the ability to extract and analyze the MoC employed in the SystemC design as a prerequisite for further design activities. In order to achieve this, the SysteMoC library provides support for a basic MoC called *Funstate*. A set of works have extended SystemC capabilities for supporting analog models. They provide specific solvers and synchronization layers to communicate with the plain SystemCpart of the specification. Likely, the most known work is SystemC-AMS[VGE04], developed by the OSCI SystemC-AMS working group. Other relevant works on analog domain are SystemC-A [AJK] and SystemC-WMS [OBC06].

Some recent works have focused on the support of abstract modelling of hardware in SystemC. In [SWT+04] an OO SystemC specification of the hardware partition is generated from an initial UML specification. The OO SystemC specification feeds a synthesis tool called *OOAS*, which generates synthesizeable SystemC code. In *GNOS03*, a set of facilites, which are partly a set and partly a superset of SystemC, enables OO specification of hardware, constraining the usage of features like pointers. Such specification can be also synthesized by means of a tool called *Fossy* [Fos]. These methodologies did not provide support for modelling and design of DRHW.

There have been several proposals enabling modelling and profiling of DR hardware architectures in SystemC. In [PMC03], a methodology for modelling dynamically reconfigurable blocks in SystemC is presented. These blocks can be associated to different *contexts* during simulation time and, using context swithching and active running times, an effective design space exploration (DSE), considering bus traffic is done. This approach requires a transformation of the source code. *Perfecto* framework [LH05] is able to provide area figures as well as time performance estimation. This framework handles the slice as the basic area unit and each DR component in the framework, called DRLC, takes a specific amount of them. In [SRHT07], an extension of the *Virtual Processing Components* (VPC) framework for supporting the simulation and modelling of reconfigurable systems is proposed. A common issue of these proposals is that they are not provided with support of a synthesis flow which eliminates the need for manual refinement.

3 Previous Work

This work proposes the collaboration of two basic methodologies, called HetSC and OSSS+R. These specification methodologies have a main advantage with respect to the methodologies presented in section 2. Each one already support an implementation flow, for embedded software generation, and for synthesis of DR hardware, respectively. This makes good candidates for the generation of a specification methodology able to target SW/DRHW platforms. This is a partial result of the ANDRES project [AND]. This project aims the development of a SystemC-based framework for the design of Adaptive Heterogeneous Embedded Systems (AHES) with a strong formal foundation given by ForSyDe meta-model [Jan04].

3.1 HetSC

HetSC [HV07] is a methodology for enabling heterogeneous specification of complex embedded systems in SystemC. Untimed and synchronous MoCs are suported over the strict timed DE MoCs of SystemC. In ANDRES, HetSC is used for specifying embedded software. Providing support for specification under several abstract MoCs (e.g. KPN, PN, CSP, SR, etc.), HetSC enables a more intuitive and safer design of the concurrent software.

The HetSC methodology defines a set of specification rules and coding guidelines for each specific MoC making the design task more systematic. Then useful properties for concurrent software, such as determinism, deadlock protection, etc are achieved. The support of some MoCs requires new specification facilities with specific semantics and abstraction levels. The HetSC library, associated to the HetSC methodology, provides such a set of facilities, covering the deficiencies of the SystemC core language for heterogeneous specification. In addition, some facilities of the HetSC library help to detect and locate MoC rule violations and assist in debugging concurrent specifications. The HetSC methodology has an associated software design flow called SWGen [SWG], put into practice in [FHSV03]. SWGen is a library-based methodology which enables the automatic generation of embedded software without modifying the original HetSC code.

In [AND] several tasks were pointed out for integrating the HetSC methodology into the ANDRES design flow. Two of them, the extension of HetSC for supporting adaptivity and the improvement of the connection with the other SystemC-based ANDRES methodologies, are in the scope of this paper. In [HVG+08], the connection of HetSC and SystemC-AMS methodologies enabled the connection at the specification level of software and analog hardware models. Now, this work will complete the former two objectives.

3.2 OSSS+R

OSSS+R [SON06] is a SystemC based modelling library providing high-level language constructs enabling application-driven modelling of (self-)reconfigurable hardware systems. OSSS+R keeps a well-defined synthesis semantics which enables that OSSS+R designs can be automatically mapped to platforms supporting dynamic partial reconfiguration.

Based on OSSS [GNOS03], in OSSS+R object-orientation is used as an adequate abstraction mechanism for dynamically reconfigurable hardware. The concept is based on the assumption that changing functions of parts of a hardware system resembles the

use of polymorphism in object-oriented software design [SON06]. Considering a digital hardware system consisting of a static and a dynamically reconfigurable part, it is obvious that the interface between the two parts needs to be fixed. However, the implemented functionality of the reconfigurable hardware may change. Hence, the key idea of OSSS+R is to model the reconfigurable area of a hardware system as a polymorphic object with a fixed interface, the *recon object*. The fixed *interface* is defined by a base class, while its possible variants belong to different subclasses. During run-time,different variants of the adaptive object can be configured and used. To handle the management of different object configurations and to ensure persistence OSSS+R introduces *Named Contexts*. A context represents all relevant information of an object, including its current type and state. From the designer's point of view, a context and a C++ pointer are used in a similar way, thus objects can be assigned to it and methods can be arbitrarily called. However, an automatically instantiated infrastructure ensures that a context is enabled (i.e. configured) only if it is accessed. Its state is automatically saved and restored during consecutive reconfigurations. Because a context can be accessed concurrently, incoming requests are serialised using a built-in scheduler. Contexts hide the complexity of configuration management and state preservation and enable the designers to use adaptivity transparently.

In ANDRES, OSSS+R is being enabled with the support of automatic hardware synthesis for DR architectures [HOS+07], by extending the capabilities of *Fossy* tool. This is a distinguishing capability respect to any of the hardware methodologies mentioned in section 2.

4 Interoperability

4.1 Installation and Usage

Since, both HetSC and OSSS+R are library-based specification methodologies, they can cooperate in a library-based framework as shown in Figure 2. Both

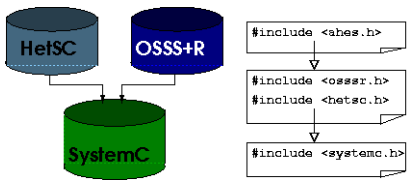

Figure 2: HetSC and OSSS+R libraries.

libraries include new specification facilities based in SystemC ones. This enables a decoupled way to extend SystemC code, which enables the building of a framework integrating contributions from different parties, like in the current case. Thus, in order to install and use HetSC and OSSS+R features in a SystemC-based specification, the SystemC library must be installed before the HetSCand OSSS+R libraries. In order to use

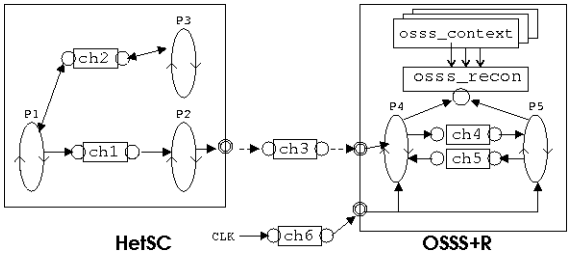

Figure 3: Specification structure.

their specification facilities, OSSS+R and HetSC main headers must be included at least once for each compilation unit. In the ANDRES framework these headers are internally called by a single header file (*ahes.h*). At specification level, any number of compilation units (thus separated compilation) are supported.

4.2 Specification Structure

In Fig. 3, the basic module hierarchy and concurrency structure of a specification making use of HetSC and OSSS+R facilites has been depicted. For it, the graphical representation of SystemC specification facilities defined by HetSC [HV07] has been used.

On the left hand side of Fig. 3, the HetSC part is basically a process network (PN) of SystemC processes (P_1, P_2 and P_3) of SC_THREAD type, the most general class of SystemC processes. They are connected by means of channel instances (like ch_1 and ch_2), which are strongly associated to the MoC. In this paper, the focus is on the usage of untimed MoCs of HetSC to specify the SW part. The SWGen methodology is able to automatically produce SW code preserving a concurrency structure similar to the SystemC one. For instance, in HetSC ch_2 can represent a bidirectional blocking rendezvous channel (uc_rv) and ch_1, a blocking bounded fifo channel (uc_fifo). Then, this part connects a CSP MoC with a Bounded KPN untimed MoC [HV07].

On the right hand side, the OSSS+R part is a PN composed of SystemC clocked processes ($SC\_CTHREAD$), like P_4 and P_5 These are triggered by a triggerer event, which is transfered from a source (the clock) by means of a channel $ch6$ of sc_signal type. These processes are connected by means of channels $ch4$ and $ch5$, of sc_signal type. These channels can be used for data transfer. The semantic of the sc_signal lets the persistance of the read value during each clock cycle. Aditionally, they serve for synchronization, by informing a process whether it must go on computing in the next cycle or not. The structure also includes the access of several clocked processes to a reconfigurable object.

Regarding to the module structure, in any case, OSSS+R obligues to enclose each reconfigurable object in a module. SystemC also forces HetSC to enclose the concurrent structure at least within a module. Fig. 3, proposes a structure of at least a module for each part.

This is convenient since it makes clear the specification structure and easier a later HW/SW partition process and the location of HW/SW interfaces. In any case, more hierarchical levels would be possible.

4.3 Adaptivity

HetSC and OSSS+R collaboration enables the system-level specification of adaptive HW/SW systems. This section explains how adaptivity is specified by using HetSC and OSSS+R methodologies. They provide an interpretation in a SystemC-based framework from a general and formal concept of *Adaptivity*. In time, HetSC and OSSS+R adaptative structures have their corresponding interpretation in the implementation plane (Fig. 4).

The formal description of *Adaptivity* is provided by ForSyDe and goes beyond the objectives of this paper. Nevertheless, some basic lines are given to reflect how hardware and software adaptivity models provided by HetSC and OSSS+R source from a basic common understanding. ForSyDe supports adaptivity by means of the *adaptive process* (AP). An AP is a process where the inner functionality directly relating process input and output ForSyDe signals can be changed during run-time by means of a special adaptation input ForSyDe signal. Moreover, ForSyDe gives an additional resource-aware nouance to adaptivity, in the sense that an AP is tied to some form of *actual computation resource*. Several functionalities allocated to the same AP will be computed by using such computation resource in an exclusive manner. In this sense, an ANDRES model using APs is a constrained model, which is a result of a first refinement step from the specification model [HOS+07]. The AP leaves still some freedom degree for the notion of computation resource in the different implementation domains involved, as will be seen.

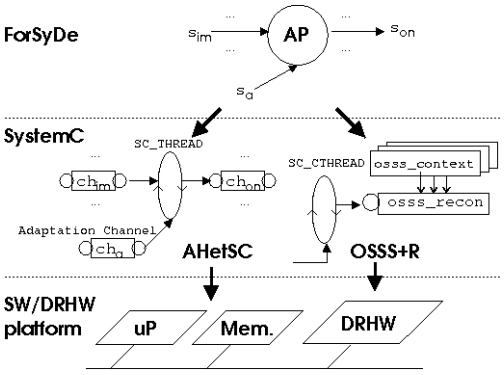

Figure 4: Adaptivity in HetSC and OSSS+R.

Fig. 4 depicts OSSS+R and HetSC interpretations using, as Fig. 3, the HetSC graphical representation. The system-level modelling of adaptive hardware is provided by OSSS+R. OSSS+R supports the specification of an AP thanks to the *recon_object* and *named_context* facilities. In effect, any clocked process accessing different *named_contexts* attached to a given *recon_object* is a process changing its funcionality during run-time, specifically at different clock cycles. The computation resource notion of the AP in OSSS+R is also clear, since two clocked processes cannot execute at the same cycle. At an implementation level, each *recon object* is implemented as a piece hardware area whose functionality can be changed during run time without perturbing or changing the functionality of the rest of the hardware area.

The system-level modelling of adaptative software has required the extension of HetSC. Adaptivity has been incorporated to HetSC in a general way following the ForSyDe formalism. The AP is modelled just as a SystemC process with some specific features. Since in HetSC, channels are the only way to communicate processes, the adaptation input is just the read of an input channel access (ch_a). The values read from this channel modify the internal functionality which relates values read from input channels with the values written to output channels within the adaptive SystemC process. For instance, the HetSC implementation of a mode-based AP, requires the adaptation input to be implemented as read accesses to an input channel which transfers data of a simple countable data type (enumerated, integer, etc.). The value read is interpreted as the *mode* which will determine which code path is executed to relate values read from input channels with values written to output channels. Additionally, the extension of a basic asumption of the HetSC general methodology has been required. Previously, HetSC assumed that a channel only transferred values or data tokens and no kind of references. However, in order to support the modelling of a *function-based* AP, the possibility to transfer function pointers through an adaptation input channel has been contemplated. Then, the SystemC process implementing the *function-based* AP internally uses this pointer to internally call a function which will directly relate values read from the input channel with values written to the output channels. The specific types of output and input channels, including the adaptation channel, depend on the MoC the adaptive structure is immersed in. This affects *when* adaptation takes place. Moreover, the detail of time information associated to such *when* depends on whether a timed, synchronous or untimed MoC is being considered.

This general SystemC-based modelling of adaptivity in HetSC enables an open software implementation semantic. As was mentioned, the specific meaning of *actual computation resource* is not specified by ForSyDe. Neither it is done by the SystemC adaptive process of HetSC. For instance, it is not defined if the *actual computation resource* of a software implementation of a SystemC AP has to be a complete Turing machine (a processor, a DSP, etc) or a software context (a set of variables, register values, etc). Moreover, in the latter case, it would not be defined if such context would have to be a *heavy* or a *light* context

(proper of SW processes and SW threads). Currently, the SWGen methodology targets only single-process single-processor architectures. Thus SWGen currently maps SystemC processes to threads of the same process and processor. Because of this, an adaptive process in SystemC becomes an adaptive thread and the input, output and adaption channels are implemented by means of inter-thread communication mechanisms. This means that the *thread context* is the shared resource among the different functionalities attached to the adaptive thread. However, the flexible interpretation of adaptivity in HetSC enables further extensions, which in time support further extensions of the SWGen methodology, to target embedded systems on platforms based on perative systems supporting processes (like *Embedded Linux*) and multiprocesor architectures.

4.4 Connection

In this section we analyze the syntactical and semantical issues of the connection of HetSC and OSSS+R structures. Fig. 3 shows a convenient way to connect both parts when each one allocates its own module: by means of a channel ($ch3$). This structure let a clear location of HW/SW interfaces at specification level. This will facilitate the later development of an implementation flow.

This OSSS+R-HetSC connection, as well as representing a DRHW/SW connection, also represents a HetSC MoC - OSSS+R MoC connection. In terms of the HetSC methodology this involves using a border process (BP) or a border channel (BP) to solve such connection. It was already mentioned which MoCs does HetSC supports. Also that this paper focuses on the usage of HetSC untimed MoCs. In this context, it is necessary an analysis of the underlying MoC which a OSSS+R specification abides.

To the effect of MoC analysis, it can be asumed that modules, ports and exports have no involvements in the simulation semantic, hierarchical information can be been neglected. Part of the analysis of the MoC in OSSS+R MoC has been already done in section 3. Such description is actually the description of a clocked-synchronous (CS) MoC. Moreover, it could be precised as a timed-CS MoC, since each clock event transfered through the $ch6$ channel is placed in different SystemC time stamps.

The introduction of adaptivity in the OSSS+R MoC by means of the *recon_object* and the *named_context* needs some additional considerations. Reconfigurable objects add an additional synchronization mechanism between clocked processes determined by the internal scheduler, since they guarantee an exclusive access at each cycle. Strict time information for reconfiguration times confirms the timed character of the OSSS+R MoC. There is no explicit data transfer among processes by means of the *recon_object*. However, it must be taken into account that a named context is not side effect free. If every named context were accessed only by one process, there is neither effective data transfer nor state changes involvements among clocked processes. However, if two processes try to access the same named context in the same cycle, there would be a potential non-determinism. Since SystemC semantic does not state which process should enter first, then the specification would admits at least two different execution paths, potentially affecting in a different way the context state and finally producing different outputs. However, in OSSS+R case, the internal scheduler of the recon object provides an additional semantic to the SystemC-one which makes the specification predictable. In the HetSC case, the usage of side effect free functions facilitate the preservation of determinism. Basically, it prevents a kind of indirect shared variable communication mechanism among processes, a basic HetSC rule.

Summarizing, apart from the involvements of adaptivity, it can be concluded that OSSS+R follows a timed clocked synchronous MoC, directly relying over the SystemC discrete event (DE) MoC. Therefore, the connection depiced in Fig.3 is an untimed/timed-CS connection. This kind of connection was studied in [HVG+08]. There, two possible solutions were proposed. One, based on BPs, could be applied to Fig. 3. Then, P_2 would be a BP which access an untimed MoC channel ($ch1$) in one side, while on the other it access a *sc_signal* channel ($ch3$). The other solution, would be more convenient for this case. There, $ch3$ would be a BC presenting an interface proper of a channel of an untimed MoC of the HetSC part (usually blocking access methods) and an interface proper of a timed-CS MoC, usually the non-blocking access semantic of the *sc_signal* interface methods.

5 Conclusions

In this paper, the integration of HetSC and OSSS+R SystemC-based specification methodologies has been proposed for the system-level specification of AHES to be implemented in platforms based on architectures provided with embedded processors and dynamically reconfigurable hardware. To solve this task, how adaptivity is interpreted in OSSS+R and HetSC methodologies has been explained, which has involved the extension of HetSC to support this concept. The main syntactical and semantical issues in the connection of these methodology have been also addressed. It has been concluded that OSSS+R basically implements a timed-clocked synchronous MoC directly relying over de DE strict-timed MoC of SystemC. These structures can be well connected to untimed parts by means of BPs or BCs which internally solve semantic conflicts when empty buffer or buffer overflows conditions appear. In future work, the usage of more complex connections in the shape of *converter channels* (able to keep unchanged after a refinement process and to make implicit data type convertion) and the definition of the design flow will be faced.

References

[AJK] H. Al-Junaid and T Kazmierski. An Analogue and Mixed-Signal Extension to SystemC. Available at `http://eprints.ecs.soton.projectac.uk/10644`.

[AND] ANDRES. `http://andres.offis.de`.

[ÅRH+06] K. Årzén, A. Robertsson, D. Henriksson, M. Johansson, H. Hjalmarsson, and K. H. Johansson. *Conclusions of the ARTIST2 Roadmap on Control of Computing Systems.* ACM Special Interest Group on Embedded Systems Review, 3(3), July 2006.

[BLL+07] C. Brooks, E.A. Lee, X. Liu, S. Neuendorffer, Y. Zhao, and H. Zheng. *Heterogeneous Concurrent Modeling and Design in Java (Vol 1&2:Introduction to Ptolemy II).* EECS Dpt, Univ. of California, Berkeley, UCB/EECS-2007-7, January 2007.

[DDM+07] A. Davare, D. Densmore, T. Meyerowitz, A. Pinto, A. Sangiovanni-Vincentelli, G. Yang, H. Zeng, and Q. Zhu. *A Next-Generation Design Framework for Platform-Based Design.* In *DVCon 2007*, February 2007.

[FHSV03] F. Fernandez, F. Herrera, P. Sanchez, and E. Villar. *Embedded Software Generation from SystemC.* In *SystemC: Methodologies and Applications.* Kluwer. March, 2003.

[FHT06] J. Falk, C. Haubelt, and J. Teich. Efficient Representation and Simulation of Model-based Designs in SystemC. In *Proc. of FDL'06*, Darmstad, September 2006.

[Fos] Fossy. `http://fossy.offis.de`.

[GNOS03] E. Grimpe, W. Nebel, F. Oppenheimer, and T. Schubert. *Object-Oriented Hardware Design and Synthesis Based on SystemC 2.0.* Kluwer, 2003.

[HOS+07] A. Herrholz, et al. ANDRES-ANalysis and Design of runtime REconfigurable heterogeneous Systems. In *Proc. of DATE'07*, Nice, April 2007.

[HV07] F. Herrera and E. Villar. *A Framework for Heterogeneous Specification and Design of Electronic Embedded Systems in SystemC.* ACM Transactions on Design Automation of Electronic Systems, 12(3):22, 2007.

[HVG+08] F. Herrera, E. Villar, C. Grimm, M. Damm, and J. Haase. *Heterogeneous Specification with HetSC and SystemC-AMS: Widening the Support of MoCs. In SystemC*, in *Embedded Systems Specification and Design Languages.* Springer, 2008.

[ITR07] International Technology Roadmap for Semiconductors, 2007. Design. Available in `http://www.itrs.net/reports.html`.

[Jan04] A. Jantsch. *Modelling Embedded Systems and SoCs.* Morgan Kaufmann, June 2004.

[LH05] C.F. Liao and P.A. Hsiung. *A System-based Performance Evaluation Framework for Dynamically Reconfigurable SoC.* In *VLSI Design/CAD Symposium.* Aug. 2005.

[OBC06] S. Orcioni, G. Biagetti, and M. Conti. *SystemC-WMS: Mixed Signal Simulation based on Wave Exchanges.* In *Applications of Specification and Design Languages for SoCs.* Kluwer, 2006. `http://www.deit.univpm.it/systemc-wms`.

[PMC03] A. Pelkonen, K. Masselos, and M. Cupak. System-Level Modeling of Dynamically Reconfigurable Hardware with SystemC. In *Proceedings of the 10th Reconfigurable Architectures Workshop, RAW2003*, 2003.

[PS04] H.D. Patel and S.K. Shukla. *SystemC Kernel Extensions for Heterogeneous System Modeling: A Framework for Multi-MoC Modeling.* Springer, July 2004.

[SON06] A. Schallenberg, F. Oppenheimer, and W Nebel. OSSS+R: Modelling and Simulating Self-Reconfigurable Systems. In proc. of FPL'06. August 2006.

[SRHT07] M. Streubuhr, C. Riedel, C. Haubelt, and J. Teich. *System Level Modelling and Performance Simulation for Dynamic Reconfigurable Computing Systems in SystemC.* In *Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*, Aachen, Germany, March 2007.

[SV07] A. L. Sangiovanni-Vincentelli. Quo Vadis SLD: Reasoning about Trends and Challenges of System-Level Design. *Proceedings of the IEEE*, 95(3):467–506, March 2007.

[SWG] SWGen. `http://www.teisa.unican.es/SWGen`.

[SWT+04] C. Schulz, M. Winterholer, Schweizer T, T. Kuhn, and W. Rosenstiel. *Object Oriented Modelling and Synthesis of Systemc Specifications.* In *In Proceedings of Asian Pacific DAC, ASP-DAC'04*, 2004.

[VGE04] A. Vachoux, C. Grimm, and K. Einwich. *Towards Analog and Mixed-signal SoC Design with SystemC-AMS.* In DELTA'04. Perth, Australia. 2004.

[Wei08] Tim Weilkiens. *Systems Engineering with SysML/UML.* 2008.