

# SystemC

By Víctor Fernández and Eugenio Villar, University of Cantabria

## 1. Introduction

SystemC™ is an ANSI standard C++ class library for Electronic System Level (ESL) design [1]. The goal of the language is to provide the system designers and architects with the required facilities for the specification, modeling and design of complex SoCs (Systems on Chip) containing Hardware and Software parts.

The technological advances in nanoelectronics fabrication during the last years have allowed the integration of complete complex electronic systems in only one chip. Those systems are composed of application-specific or programmable hardware parts and of one or several microprocessors running embedded software on a Multi-Processing SoC (MpSoC). The design process for such complex systems can only become affordable by a shifting from traditional methodologies to higher abstraction levels. Traditional Hardware Description Languages (HDLs) such as VHDL and Verilog [2] lack the required features to accommodate the increasing functionality to be executed by the embedded microprocessors on chip. In this scenario, the OSCI (Open SystemC Initiative) was created in 1999 as an independent, non-profit making association dedicated to defining and advancing SystemC™ as an open industry standard for system-level modeling, design and verification. It is composed of a large and growing number of system design companies, semiconductor companies, intellectual property providers, and EDA tool vendors who have joined together to support and promote the language.

The first versions of the language had a strong HW orientation. The C++ classes defined were able to cope with the features needed to model electronic systems, which were not covered by C++ such as time, structural hierarchy, concurrency, communication, a discrete-event simulation mechanism and hardware-oriented data types. These initial versions were improved with additional features supporting system modeling at higher abstraction levels, with HW and SW parts leading to version 2 of the language which was ratified as IEEE Std. 1666™-2005 (IEEE Standard SystemC Language Reference Manual) in December 2005 [3].

## 2. Language Core

SystemC can be structured as shown in Figure 1:

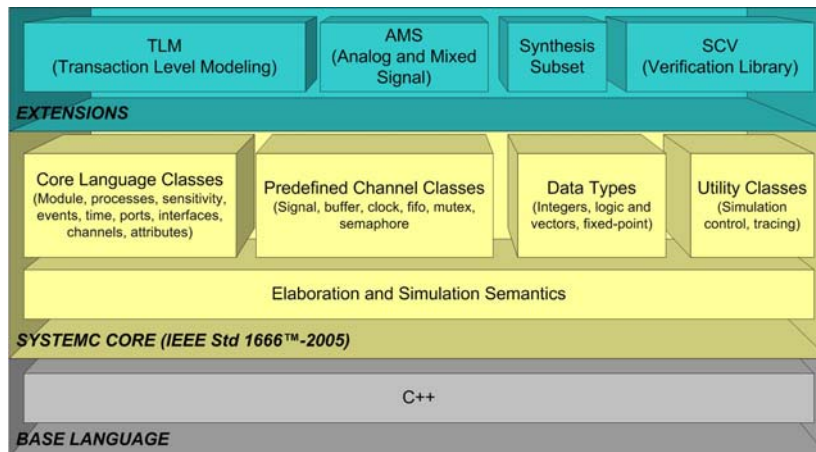


Figure 1. SystemC structure.

The language relies on C++ as it is actually a C++ class library. The middle part of Figure 1 describes what is usually called the language core. It is composed of two layers. The lower layer is a discrete-event (DE), event-driven simulator. Its implementation is undefined in order to allow the development of different simulators with the same functionality but not necessarily the same performance. The upper layer contains a varied set of C++ classes clearly defined in the 1666™-2005 IEEE Standard. Most of these classes provide the semantic, structural, functional, communication and data typing facilities required for HW/SW system modeling and design. As most of them have simulation semantics, they can be seen as the user facilities to program the underlying DE virtual machine. On top of the core part of the language, one can find several language extensions, as yet not standardized, which are being developed by OSCI working groups and which will be briefly described in the next section. One of the main advantages of SystemC is its flexibility. Being a C++ library, the system designer can exploit the full language on top of the SystemC core with his/her own specific semantics for any specific application.

The SystemC C++ class library is open and can be freely downloaded from OSCI [4] including a reference model of the simulator. Library classes are available for the user code by including the *systemc* file in the header. SystemC classes use a private elaboration and implement the simulation kernel whose semantics is the first subject covered by the IEEE standard.

Executing a SystemC application consists of elaboration followed by simulation. During elaboration, the internal data structures are created in order to support the semantics of simulation. The hierarchy of processes, modules, ports and channels is created. The simulation phase is based on an event-driven scheduler, which means that processes are executed in response to the occurrence of events. Events (*sc\_event* class) are notified at specific points in simulation time. The time concept is supported by the *sc\_time* class. Simulation time is an integer value whose physical meaning is defined by the simulation time resolution (*sc\_set\_time\_resolution*).

SystemC descriptions, once the hierarchy is flattened, can be seen as a set of parallel processes. Such processes have static and dynamic sensitivity lists that define the set of events and time-outs that can potentially cause the process to be resumed or triggered. Scheduling of processes is non-preemptive and execution order cannot be fixed directly by the user. The model of computation is based on an evaluate-update mechanism with delta cycles in a similar way to HDLs. Events can be notified for the next delta cycle, for a future time and for the same delta cycle. This last feature has to be used with care because it could cause indeterminism in execution order and results. The simulation of a SystemC description is controlled by a set of utility classes in order to start, stop, finish execution or trace results.

When describing a complex system it is necessary to define structure and hierarchy. To support this feature, the core language provides the *sc\_module* class. Behavior is modeled through processes which are encapsulated within modules. Ports are used to input and output data to the modules. Connections among modules are performed via channels (*sc\_channel* class). Channels define communication methods and have interfaces to connect to ports. This topology allows the desirable property for an ESL language of separating communication from behavior in order to describe and refine both features with maximum orthogonality.

As mentioned above, channels are contenders for communications methods among modules. Users can define their own channels. On the other hand, the core part of SystemC also defines some predefined channels for common usage. These channels are: *sc\_signal* (derived from it are *sc\_buffer*, *sc\_clock* and *sc\_signal\_resolved*), *sc\_fifo*, *sc\_mutex* and *sc\_semaphore*. Communication mechanisms defined by these channels can be easily derived from their names.

A SystemC description can use all C++ types. Moreover, the language provides a set of additional data types to represent values with application-specific word lengths applicable to digital hardware. These include logic and vector types to describe low-level HW signals: *sc\_logic* for describing a single bit with '0', '1', 'Z' and 'X' values, *sc\_lv* as a vector of previous single bits and *sc\_bv* as a vector of single bits with only '0' and '1' values. There are also fixed-point data types with several controls on integer and decimal sizes as well as quantization and overflow modes. Finally, there are word-wide definable integers: *sc\_int*, *sc\_uint* for signed and unsigned integer values at a precision limited by their underlying native C++ representation and their specified word length and *sc\_signed*, *sc\_unsigned* for signed and unsigned integer values at a precision limited only by their specified word length.

For additional information about the language syntax and semantics, apart from the IEEE standard [2], there are several sources which can be recommended [4-6].

### 3. SystemC applications

OSCI has been organizing SystemC user group meetings since 2002 with more than 700 attendees in recent years. In these meetings, data has been collected via surveys which reveal realistic information about SystemC use in several aspects [7]. Two of them are shown in figures 2 and 3.

It can be seen in Figure 2 that SystemC is applied in four main areas: system-level modeling and platform architecture, specification and algorithm design, as a reference model for functional verification and as a virtual platform for software development. The most relevant information derived from the figure is the fact that the language has a tendency to be used to the same extent in all these areas. This demonstrates that the language copes well in all the areas of ESL design where it is being used, which is a desirable property for a language that has the goal of being the only ESL language.

Figure 3 shows another feature in which SystemC is well balanced. Focusing on the left pie chart, the use of SystemC for most applied abstraction levels from system to HW implementation is shown. These levels are: purely functional Untimed Level, intermediate Transaction Level (TL) and Register Transfer Level (RTL) as input to automated synthesis paths. As can be seen in the figure the language use is distributed nearly in the same proportion over these abstraction levels, being applied a bit more intensively at TL. Transaction Level Modeling (TLM) [8] is a modeling methodology for complex HW/SW

systems based on reducing the number of events to be simulated by taking into account only the data transactions among the components instead of the more detailed, clock-based data movements at RTL. In this way, the simulation speed is greatly increased while still being able to take into account the execution times of the HW and SW components.

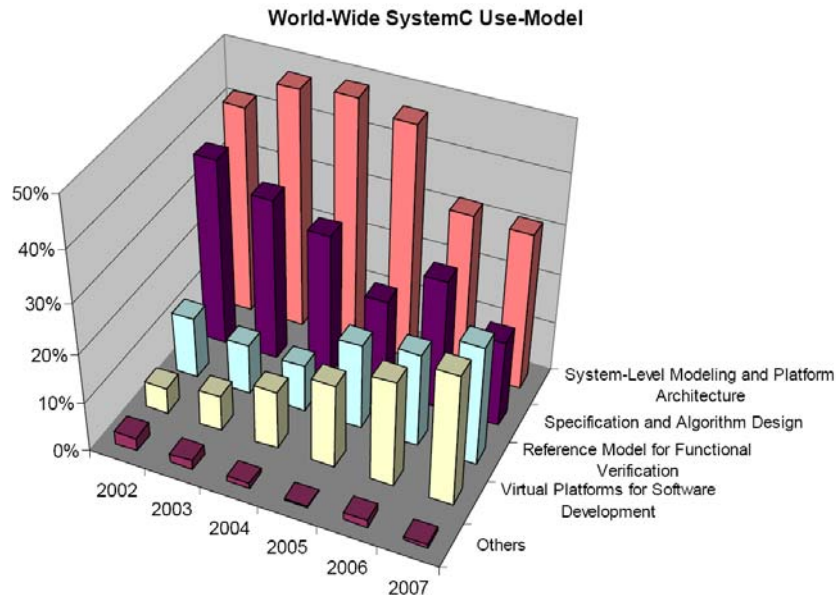


Figure 2. SystemC areas of application

Inside TL three sub-levels have been distinguished: Untimed Level, also known as Programmer's View (PV) because this level is widely used by programmers, Timed Transaction Level, also known as Programmer's View plus Timing (PVT) and the Cycle Accurate Level (CA). These three sub-levels are nearly equally used as can be seen in Figure 3.

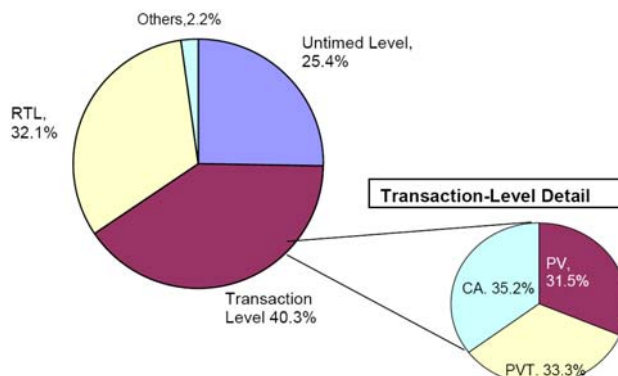


Figure 3. SystemC use at different abstraction levels

SystemC has become the main language supporting commercial ESL tools [9]. Most of them provide support for complex, multi-processing, HW/SW platform modeling and analysis. Some of them provide HW synthesis technology.

#### 4. Future trends

The near future evolution of SystemC is going to be led by two main drivers. On one hand, there are four official technical working groups controlled by OSCI in charge of formally defining language extensions in several selected areas. On the other hand, SystemC has fueled many ESL research activities as the reference language for system specification, modeling, design and verification [10-11].

The following are the working groups currently active inside OSCI.

There is an Analog and Mixed-Signal Working Group (AMSWG) developing a SystemC extension in order to be able to cope with that kind of system parts. Modern SoCs tend to include not only the digital part of the system but also the analog and mixed-signal interface. That is the reason why the development of this language extension is necessary.

Synthesis Subset Working Group (SWG) is in charge of defining a synthesizable subset of SystemC. Syntax and semantics have to be defined in order to be recognized in common by tool developers and final users.

Another key point in system design is the verification flow. In that context, the Verification Working Group (VWG) is responsible for the definition and development of methodology and add-on libraries for verification in SystemC including the SCV library.

One of the most active groups is the Transaction-level Modeling Working Group (TLMWG). It is in charge of the definition and development of the methodology and add-on libraries for transaction-level modeling in SystemC. The intense activity of this group derives from the growing interest in Transaction level as a necessary middle point between system specification and input to the synthesis tools [8].

Last but not least, SystemC provides language support in a wide variety of research activities on electronic system-level design methodologies. Next, there is a brief overview of the most relevant activities with references for additional information. A good overview of the evolution of the field can be found in the Springer series on Selected Contributions from FDL [12]. One of the most active research areas currently is interoperability between UML and SystemC. In this context, UML supports the PIM MDE while SystemC provides the corresponding PSM simulation framework [13-14]. SystemC has been proposed for heterogeneous modeling and specification under several models of computation showing its capability to support the development of frameworks with similar features to those of Ptolemy [15-16]. SystemC has proven to support SW simulation at the source code including abstract models of the RTOS and the microprocessor [17-18]. There is also a clear interest in developing automatic methodologies for synthesizing SW and Hardware-Dependent Software (HdS) from the SystemC specification [19-20]. Another active area is the use of SystemC for performance analysis at system level including power estimation for design-space exploration [21-22].

The main conclusion that can be derived from the previous analysis is that SystemC has proven to be a very good language for ESL design. The language supports both those mature design tasks covered by commercial tools as well as the experimentation of new design methodologies. It is reasonable to expect the language to develop in the future as it contributes to the maturity of the ESL design methodology.

## 5. Links and references

- [1] [http://www.systemc.org/community/about\\_systemc](http://www.systemc.org/community/about_systemc).
- [2] <http://www.accellera.org>.
- [3] <http://www.systemc.org/downloads/lrm>.
- [4] J. Bhasker: "A SystemC Primer", Star Galaxy Publishing, 2002.
- [5] "SystemC Golden Reference Guide", Doulos, 2006.
- [6] D. C. Black and J. Donovan: "SystemC: From the Ground Up", Springer, 2006.
- [7] [http://www.systemc.org/community/user\\_groups/OSCI\\_2007\\_Survey\\_Data\\_Trends\\_Report.pdf](http://www.systemc.org/community/user_groups/OSCI_2007_Survey_Data_Trends_Report.pdf).
- [8] F. Ghenassia, "Transaction-Level Modeling with SystemC", Springer, 2005.
- [9] [http://www.garysmitheda.com/ESL\\_web.pdf](http://www.garysmitheda.com/ESL_web.pdf).
- [10] T. Grötter, S. Liao, G. Martin and S. Swan, "System Design with SystemC". Kluwer Academic Publishers, 2002.
- [11] W. Müller, W. Rosentiel and J. Ruf, "SystemC. Methodologies and Applications". Kluwer Academic Publishers, 2003.
- [12] E. Villar (Ed.): "Embedded Systems Specification and Design Languages", Springer, 2008.
- [13] G. Martin and W. Müller (Eds.): "UML for SOC Design", Springer, 2005.
- [14] E. Riccobene, P. Scandurra, A. Rosti and S. Bocchio: "A SoC Design Methodology Involving a UML 2.0 Profile for SystemC", proc. of DATE'05, IEEE, 2005.
- [15] H. D. Patel and S. K. Shukla, "SystemC Kernel Extensions for Heterogeneous System Modeling: A Framework for Multi-MoC Modeling and Simulation", Kluwer Academic Publishers, 2004.
- [16] 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, Special Issue on Demonstrable Software Systems and Hardware Platforms, V.12, Issue 3, N.22. 2007.
- [17] H. Posadas, J. A. Adámez, E. Villar, F. Escuder and F. Blasco (DS2): "RTOS modeling in SystemC for Real-Time embedded SW simulation: A POSIX model", Design Automation for Embedded Systems, V.10, N.4, Springer, 2006.
- [18] A. Jerraya, A. Bouchhima and F. Pétrot: "Programming models and HW-SW interfaces abstraction for multi-processor SoC", proc. of DAC'06, ACM, 2006.
- [19] J. Chevalier, M. de Nanclas, L. Fillion, O. Benny, M. Rondonneau, G. Bois and E.M. Aboulhamid: "A SystemC Refinement Methodology for Embedded Software", Design & Test of Computers, V. 23, N. 2, IEEE, 2006.
- [20] I. Viskic, S. Abdi and D. D. Gajski: "Automatic generation of embedded communication SW for heterogeneous MPSoC platforms", Proceedings of the 2007 LCTES conference, ACM, 2007.
- [21] R. Damaševičius and V. Štuikys: "Estimation of Power Consumption at Behavioral Modeling Level Using SystemC", EURASIP Journal on Embedded Systems, Volume 2007, Hindawi Publishing Corporation, 2007.
- [22] M. Hassan, E. Okushi and M. Imai: "ASIP array system performance analysis and design space exploration using SystemC", proc. of International Conference on Computer Engineering & Systems ICCES '07, IEEE, 2007.