SystemC-VHDL co-simulation and synthesis in the HW domain

Massimo Bombana, Siemens MC S.p.A., Italy
Francesco Bruschi, Politecnico di Milano, Italy

Abstract

Embedded systems design requires the development of complex HW modules to cope with the most stringent timing constraints of the specifications. This implies the need to update and enrich HW design methodologies to face abstraction and novel requirements. Here we will present some results of design practice of HW modules in this context. Co-simulation and synthesis are combined in this approach to achieve higher abstraction levels in the design, to improve validation and re-use of previous designs and human experience. The proposed methodology is embedded in a SystemC based design flow. The SystemC-VHDL co-simulator tool is also based on a SystemC/C++ front-end developed to support the co-simulation between VHDL and SystemC. The prototypal state of the adopted tools increase the novelty and interest of the approach.

1 Introduction

Embedded systems are increasingly used in various application domains, ranging from telecom to automotive, from domotics to avionics. Design tasks face increasing demands of advanced functionalities of devices, and tight integration with available IPs, components and platforms. This triggers an expansion of the design methodology also for HW components, giving relevance to issues like re-use, quick prototyping, HW/SW integration. Such new approach should allow the development of programmable IP-oriented devices (FPGAs) and the introduction on the market of novel EDA toolsets, operating at high levels of abstraction and producing automatic implementations.

In the telecom domain the design of customised interfaces between standard busses and proprietary modules sitting, for example, on boards for mobile communication systems, is one of the crucial issues and bottleneck to success. SystemC is nowadays considered one of the languages that will allow an increase of abstraction for HW modelling and will ease the integration of SW modules in the context of a formalized system level design methodology. One of the features that this language offers is the possibility to mix various levels of model abstraction during the design flow. The designer can model an item of the system through a pure functional untimed specification, and then check its interaction with other elements of the system in the early design phases. New applications are seldom designed from scratch: more often in industrial practice several modules are reused from previous versions or implementations. This paves the way to a methodology that mixes not only abstraction levels but also languages. In fact the large majority of the databases of reusable designs in our industrial design centers are written in VHDL, while much effort is being spent on introducing SystemC as the system level modelling language. Model validation becomes a crucial issues in such context.

In this paper we focus on the analysis and formalization of a design methodology able to mix abstraction levels and languages for HW development, combining behavioural synthesis and SystemC/VHDL cosimulation. In section 2 we will describe the adopted design flow, underlining the integration of various existing commercial tools into a unified design flow. Requirements and constraints are analysed and solutions evaluated. In section 3 we describe a design practice involving the design of elements of different complexity in this design flow. An assessment of the design experience is summarised in section 4. Finally conclusions and plans for future work are described in section 5.

2 Methodological approach and design flow

The main goal of our activity consisted in defining, applying and evaluating a tool flow that allows cosimulation of VHDL modules and SystemC code, in a methodological framework including specifically high-level modelling, simulation and synthesis. The task is accomplished by satisfying the following requirements:

- possibility of modelling the application at RT and/or behavioural SystemC level, mixing VHDL and SystemC modules, both in the model itself and in the test bench;
- feasibility of applying synthesis, either from the RT or from the behavioural level of abstraction, both for VHDL and SystemC modules;
- possibility of simulating each representation and to cosimulate mixed representations (for example of SystemC models and VHDL test benches);

¦ using a flow that is based on commercially supported tools and exploiting the available expertise at the human level.

The test bench and the elaborated module are then provided to Modelsim and Synopsys DLL for final simulation. This design step produces cosimulation

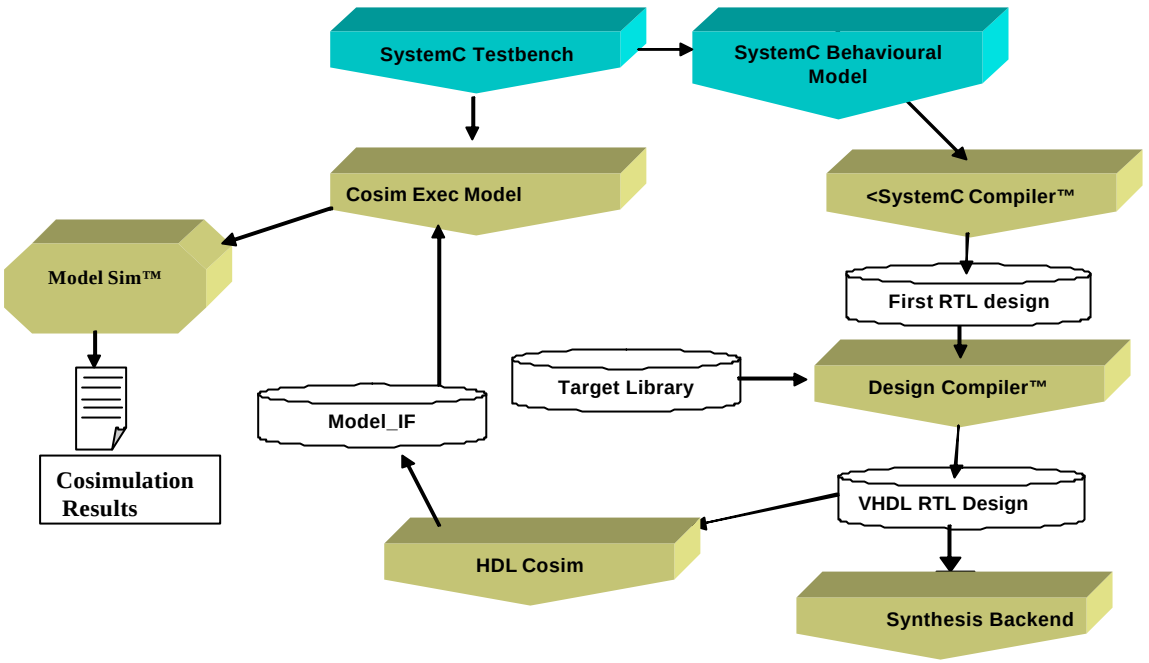

Figure 1 HW design flow for synthesis and cosimulation.

The complete and formalized design flow is shown in Figure 1. The design activity starts with the modeling phase at behavioral level, where SystemC (specifically the SystemC subset for synthesis) is used for a high level description of the application. The test bench is also specified in SystemC at the same level of abstraction. The system consisting of model and test bench is simulated in order to verify the satisfaction of the functional requirements and to identify the compliance with design constraints and specs at this level.

After system level verification, the synthesis step is performed using SystemC Compiler or Design Compiler, according to the selected level of abstraction that was adopted in the previous phase. Specifically using SystemC Compiler a VHDL netlist at RT level is generated. In order to check the results of the synthesis step, when some degree of manipulations have been performed after synthesis by the designer (in form of corrections or additional manual coded modules), this level of the design is simulated again at this level of abstraction, including also the low level design details. In order to perform this step without translating the test bench manually from SystemC into VHDL (manual operation that can introduce errors in the design), the cosimulation step is performed. At first the test bench is given to an intermediate module (HDL Cosim), that creates an environment around the VHDL description to allow its connections with the SystemC environment. All these translation procedures are transparent to the final user. traces that are visually inspected to verify correctness.

The actual state of the cosimulator allows using different simulators available on the market, including Scirocco and ModelSim. In our case we used ModelSim since this is the tool we usually adopt for VHDL simulation.

2.1 Design issues: Integrability into an EDA environment

Reuse of existing designs and valorisation of human expertise are key factors in design centers in industry. The tool is part of a commercial toolset fully compatible with our design flow.

A main issue concerns the fact that different abstraction levels are applied in the previous flow: it may happen that the test bench belongs to a different level in comparison with the VHDL model. This may introduce a mismatch in some signal types. For instance, this happens for bool vs. std_logic type of ports. Typically the interfaces of the behavioural modules contain ports of type bool, that is a more abstract and generic type with respect to the resolved ones. The synthesis step, in turn, generates VHDL entities that have interfaces made of std_logic ports (this is reasonable for a RT description). So, a manual file correction is needed in order to specify that the SystemC interface generated must match bool types to sc_logic ones. Different-level adaptations of this kind can be automatically produced. This effort will be justified when the proposed design flow will generate a relevant demand in the designers' community.

2.2 Input specification style and comparison with standard simulators

All the SystemC constructs are supported by the co-simulator. This implies that specifications syntax need not to be tailored to any specific language subset. VHDL specifications also follow the standard format, according to the used VHDL simulator. However the synthesis tool is constrained by the synthesis subset, both for VHDL and SystemC. This issue finally imposes to adopt the synthesis subset for the entire flow (except for the testbench).

A few manipulations are necessary at the moment in the generation of the SystemC test bench, like for instance in the include sections and in the name that instantiates the module. These are trivial issues that will likely be corrected in the future releases of the tools.

3 Design Practice

The first set of design tasks using the combination of behavioural synthesis and cosimulation had the goal to provide a methodological assessment and the identification of the optimal conditions for the measurements, in terms of different parameters tuning. For this goal SystemC behavioural specifications of two simple devices were used.

We applied the formalized design flow (figure 1), including synthesis, high level SystemC simulation and low-level SystemC/VHDL cosimulation. The traces derived from the simulations at at the two abstraction levels, i.e. before and after the generation of the VHDL RTL netlist, were compared. This allowed to check the correctness of the design task consisting of automatic synthesis and manual code adaptations and to verify the lack of trivial bugs in the new toolset applying SystemC/VHDL cosimulation.

SystemC tracing features were used to generate waveforms. They were put into a graphical format using free waveform viewers. A preprocessor flag controls the switching among the simulation of the full SystemC model and the cosimulation (see Figure 2).

The tracing routine makes use of this attribute as shown in the same code sample.

After a reset phase of 100 cycles, the simulation runs for 1000 cycles.

The trace obtained instantiating a counter allows to verify that the device exhibits the expected behaviour, with the counter incremented at each clock cycle.

According to the flow depicted in Figure 1, synthesis is then performed. After this phase, three files are available.

```
#ifndef COSIM
F = sc_create_vcd_trace_file("waves");
#else
F =
sc_create_vcd_trace_file("waves_cosim")
;
#endif
sc_trace(F,clock,"clock");
sc_trace(F,reset,"reset");
sc_trace(F,out,"out");
```

Figure 2 Code sample from the testbench

The first one provides some information on the interfaces generated for cosimulation. The most relevant one in this case is the type of the ports, that is written as bool (see text in **Figure 3**). This type was inserted manually and substitutes the type **sc_int**. This operation is due to the combination of synthesis and simulation that we are experimenting. If VHDL and SystemC modules were interfaced directly at the same (for example RT) level, this manipulation would have not been necessary.

```
.INPUT_PORTS
 1  clk (STD_L) [=] bool (clk)
 2  reset (STD_L) [=] bool (reset)

.OUTPUT_PORTS
 1  out_port  (STD_L_V(3 downto 0))
sc_int (out_port)
```

Figure 3 Generated interface definition

The second file represents a wrapper for the VHDL modules, created by the cosimulation environment. These interfaces incapsulate the VHDL code, allowing communication with the SystemC blocks. They are implemented in SystemC and transparent to the final user.

Finally, the third file contains the VHDL code that has been generated by the synthesis flow from the SystemC specification. This is an RT level VHDL description and the blocks defined in this net will be cosimulated with the original SystemC test bench.

The fact the obtained cosimulation trace is behaviourally similar to the fully SystemC simulation suggests that the two simple models are functionally equivalent.

A more complex testing was applied to a PCI-like Interface to be used in GSM Base Station Controllers (BSCs) This application has been described elsewhere [x]. We only remind that the reconfigurability features of the PCI (plug'n'play) are not implemented; the mapping of the device in the address spaces of the bus is hardwired; (it is obviously parametric in the model, but it cannot be changed at run time);

Starting from these requirements, a functional decomposition of the specification is performed, modelling different blocks. Further steps of the synthesis flow will produce FPGA implementation.

This real life application is characterized by a much larger number of signals to be checked and by a more complex control flow, generating different sections of behaviour to be controlled separately. As an example of the traces produced from the behavioural SystemC description, refer to figure 4 for the example of a master write PCI transition. Here the correct arbitration phase can be observed (control signals *req/gnt*), followed by the address phase and by the data burst phase (signals *frame,ad,be*).

Note the presence of a second clock, at a higher frequency than the PCI bus one. This overclocking was needed by the interface because the constraints given by the synthesisable subset handled by SystemC Compiler force to add some otherwise redundant clock cycles in the behavioural code. The overclock ratio has been kept minimal in order to allow fpga synthesis. These remarks are related to the synthesis aspect and not to the cosimulation.

From this design practice a relevant design experience was gained not only on the cosimulation task but also on the synthesis process which is strongly interleaved to our methodological approach. Specifically, the following points where highlighted in which the diversity of abstraction level can introduce additional design steps, that can be addressed at this time only in a manual (not automatic) way:

1) Stimuli generation: the different behaviour of the SystemC description and of the synthesised VHDL model could impair cosimulation behaviour. To exemplify this problem, one could think of the reset behaviour: the SystemC test bench could work even in absence of a correct reset phase, while this is not the case for the VHDL synthesised description.
2) Signal compatibility issues: as described earlier, the manual adaptation of the signal types during the interface generation could cause some subtle unexpected synthesis behaviour (at least, it is not exactly known how this signal semantics translation is performed by the tool).

Moreover we will develop examples of middle complexity and at the same abstraction level.

4 Evaluation of the methodology

Part of the novelty of the described approach lies in the fact that commercial tools operating at this level have been only recently made available. Moreover SystemC and VHDL cosimulation is a novel approach that allows better reliability for the functional correctness of mixed specifications and implementations.

Some improvements are necessary to provide acceptable solutions to the limiting factors highlighted during this design practice and reported in the previous section.

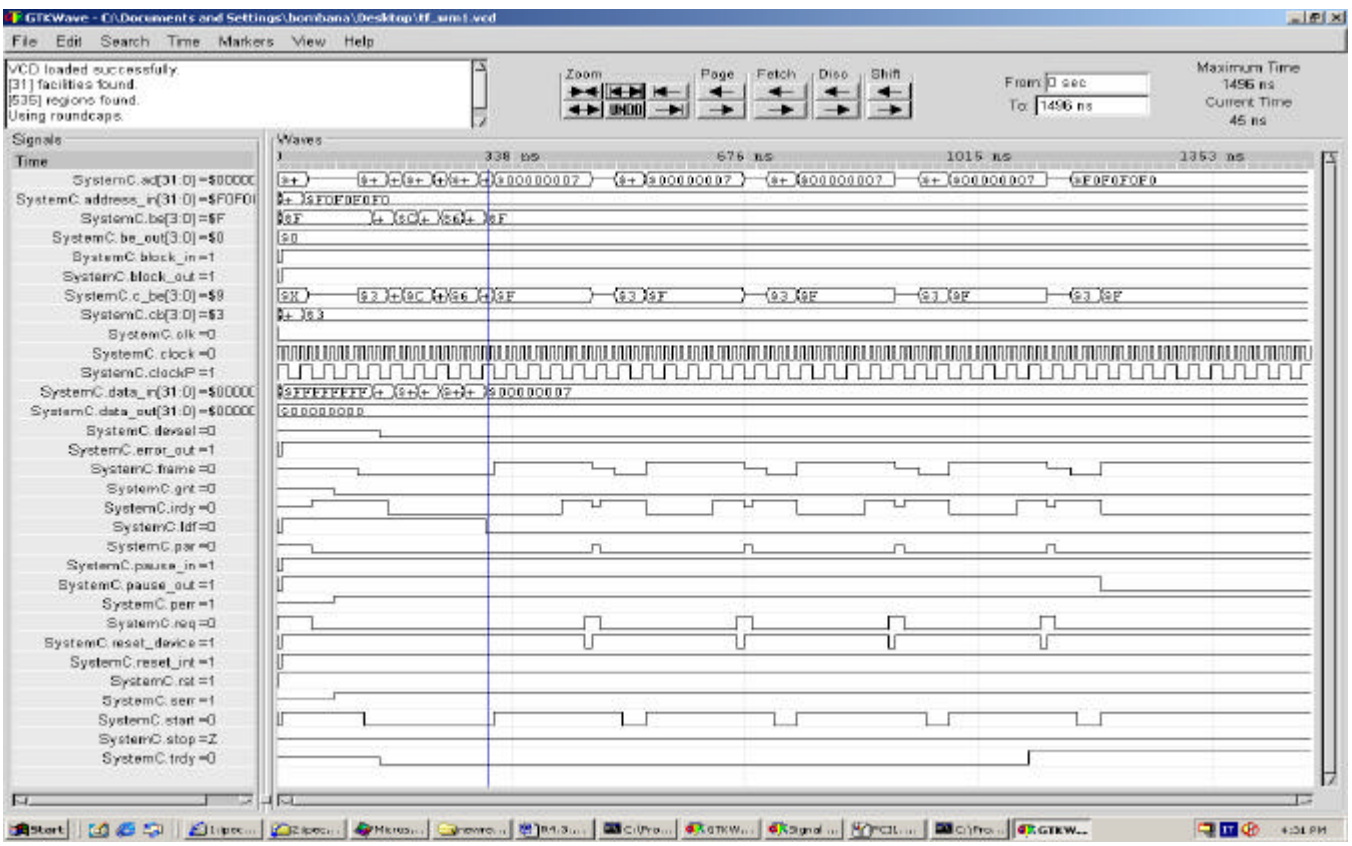

Figure 3 Cosimulation traces

We see a clear advantage in using this tool inside a methodology strongly biased towards synthesis (as in our case). Other approaches (mixing VHDL and SystemC models at the same abstraction levels) are also very relevant for good design practice, but not considered here. In order to cope with the synthesis-oriented market segment, it would be relevant to implement strategies that would reduce the level of manipulation required by the user in order to adapt, for instance, the types of the ports depending on the adopted abstraction level. Other priorities should include a complete debugging of the toolset, and its full integration into System Studio.

From the results of our experiments, we believe that the innovative aspects of the toolset address the co-simulation needs in a satisfactory way. Moreover the user interaction with the tool is good.

A final remark on the market potential of this design approach concerns the fact that it depends strongly on the acceptance of SystemC as a HW description language in a large numbers of design centers and industry labs. Since VHDL is already well established for RT level design, it seems reasonable that SystemC will at least address more abstract design levels, including system level design and then involving interaction with SW modules. In this perspective the cosimulation tool should also address a tighter link with synthesis and how to provide co-simulation for more abstract models and design cases. From this point of view, the success of SystemC as a language is also depending strongly on the good performance and market success of the synthesis flow from abstract SystemC specifications.

Performance comparisons with pure RTL simulations are not feasible. In fact, the behavioural abstraction level of the testbench could improve the global speed, but the computations performed to simulate the RTL part can act as a bottleneck for the global figure.

However the goal of this approach was not the performance boost of the simulation, but the analysis of the possibility to simulate together models written at different levels of abstraction and with different languages.

5 Conclusions and future work

As SystemC gains popularity as a specification and design language, the possibility of simulating models written in different languages will become more and more important. The availability of design tools that can simulate new design modules, written in SystemC, together with existing ones, modelled with the traditional HLDs, will possibly allow a smooth transition towards the use of this design language, thus broadening the set of design teams that can take advantage from it. In this paper we reported a set of design experiences with the tools today available that allow cosimulation. Even if the maturity level of these tools isn't full yet, the results of the tests we performed were satisfactory. Some major improvements that should be addressed regard the integration in a design flow that contemplates synthesis from high level behavioral models.

6 References

[1] SystemC Version 2.0 User Guide
[2] http://www.systemc.org/index.html
[3] Synopsis, "CoCentric SystemC HDL Cosim User Guide", 2001
[4] A. Allara, M. Bombana, P. Cavalloro: Requirements for synthesis oriented modelling in SystemC, Proc. FDL 2001, Lyon, 3-7 September 2001
[5] PCI Special Interest Group, PCI Local Bus Specification, Production Version, Revision 2.1, June, 1, 1995