Elements of a SystemC Design Platform

GEORGE ECONOMAKOS
Department of Electrical and Computer Engineering
National Technical University of Athens
Zographou Campus, GR-15773 Athens
GREECE

Abstact: - Modern digital design, having to cope with the increasing device and application complexities, is based on high-level textual system specifications. While languages like VHDL and Verilog HDL have been effectively put to use for hardware design, system level hardware/software codesign requires more abstract and more powerful specification languages, like SystemC. However, for SystemC to be effective, it must be integrated in the existing design flow, taking advantage of previous work, avoiding past mistakes and generating the least possible new, offering clear and distinct advantages, implementing the latest research results and helping designers overcome the learning curve. This paper discusses issues of a pure SystemC design platform, states its applicability and its new perspective in system level design and offers tool interoperability solutions. Through such platforms, SystemC can leverage digital design industry from high level hardware design to system level hardware/software codesign.

Key-Words: - SystemC, System Level Design, Behavioral Synthesis, High-level Synthesis, IP-based Design, Design Reuse.

1. Introduction

Over the last ten tears the electronics industry has exploded. Chip fabrication technology continues to mature, with minimum feature size approaching 0.1 μm. With decreases in feature size come added complexities in the design. For example, with the 0.18 μm process of Texas Instruments [1], the equivalent of 20 high-performance microprocessors could exist on the same substrate, with a transistor density of 125 million transistors. This explosion has opened the way for new design techniques and methodologies, like high level and system level design, hardware/software codesign or IP-based SoC design, which raise the level of design abstraction and so, reduce the design space.

While design space reduction supports efficient design space exploration and thus, reduced time to market, design specification techniques have also changed in order to cope with the increasing design complexity. Schematics have been restricted to small or boards level designs and *Hardware Description Languages* (HDLs) have been introduced. During the last years and after extensive standardization efforts, by various institutions and individuals, two languages have prevailed, VHDL [3] and Verilog HDL [2]. Today they are widely spread in both research and industry for simulation, RTL design entry and synthesis, gate level design exchange and tool interoperability and formal verification.

Both VHDL and Verilog are now everyday practice in industrial hardware designs, reaching up to traditional high level synthesis [5, 13]. For system level hardware design or hardware/software codesign, things are not so easy. Usually, system level designs start with a functional model of the system, written in a procedural language like C/C++, from which algorithmic correctness is verified through simulation. This model is manually partitioned into hardware and software components. Software components are transformed through refinement into final implementations. During refinement, elements of the initial system model like code fragments or testbenches, may be reused, and thus time to market is reduced. On the contrary, hardware components must be translated manually from functional C/C++ into RTL VHDL/Verilog for the design to be refined. This translation may introduce difficult to track errors while functional testbenches are useless.

As a solution to this problem, C/C++ modeling and synthesis techniques have been examined thoroughly during the last couple of years

[6]. Two consortiums have been created, the *SystemC* (www.systemc.org) consortium and the *SpecC* (www.specc.org) consortium, proposing corresponding *System Level Design Languages* (SLDLs), each one offering different advantages to end users and tool vendors. Commercial tools have appeared, like the Synopsys CoCentric SystemC Compiler (www.synopsys.com) for behavioral synthesis, CoWare N2C (www.coware.com) for system level design, C Level Design CycleC (www.cleveldesign.com) design methodology, CynApps CynLib (www.cynapps.com) design methodology and Frontier Design A|RT Builder (www.frontier.com) for behavioral synthesis, to name just a few. New research ideas have also been reported [4, 7, 10, 12, 17].

This paper attempts a clear view over previous work, and proposes a configuration for a complete C/C++ based design platform, using the SystemC SLDL. It is concerned with two main problems, behavioral synthesis, with which functional models can be refined into RTL models, and IP libraries, for code reusability. Specifically, it presents a behavioral SystemC preprocessor, which translates behavioral into RTL SystemC, supporting user interaction (the click-and-go application style has been criticized by hardware designers), and an RTL VHDL/Verilog to SystemC translator, which enables reuse of well tested HDL IP cores. Along with commercial tools for RTL synthesis and simulation, this configuration is a complete top-down design platform, which can gradually introduce C/C++ level design in industrial projects, avoiding a new HDL war.

2. SystemC Overview

SystemC was officially introduced almost a couple of years ago [15], as a C++ class library offering hardware semantics to C/C++ programs, (mainly) through concurrent and re-entrant processes and hierarchical modules. Its initial release was targeting RTL models, allowing higher level constructs as well, just like VHDL or Verilog. Later [14], the RTL semantics of the language were clarified and today, SystemC can challenge both HDLs in RTL modeling. The initial release of SystemC included:

- Container classes for hierarchical modules, ports and signals.
- Concurrent and re-entrant processes, contained inside modules.
- A rich set of hardware specific data types.
- A cycle-based simulation kernel, implementing delayed signal assignments and linked with user defined functionality through traditional software development environments.

Since this initial release, the *Open SystemC Initiative* (OSCI) has chosen to distribute SystemC as a public domain, open source product for both research and commercial use. This, combined with the fact that, being a C++ class library, SystemC was easily integrated with common and well known software development environments, has been a great promotion for the language. SystemC has been widely adopted by both industry and research communities.

Additionally, SystemC has been extended mainly focusing on higher levels of abstraction [16] (system level design and hardware/software codesign). The resent (beta) version 2.0 of the language defines new abstract features for process communication and synchronization, inspired from similar constructs of SpecC. Specifically, SystemC 2.0 is a layered approach consisting of the following:

- A general purpose modeling foundation, the SystemC simulation kernel.
- Container classes for events, low level synchronization primitives supporting dynamic process sensitivity (as well as static).
- Container classes for channels, interfaces and ports as a general purpose process communication mechanism.
- Container classes for signals and delayed assignment operators.
- A generalized time model (real-valued, integer-valued, untimed).
- A dynamic thread library (can be used to write a SystemC RTOS).

3. SystemC Based Codesign Methodology

With the introduction of SystemC 1.0 a couple of years ago, the language was adopted for hardware design, in areas where VHDL and Verilog are still leading. With the introduction of version 2.0, SystemC was equipped with system level semantics, beyond traditional HDLs and towards higher abstraction levels. This dual personality of the new SLDL supports a new hardware/software codesign methodology, shown in brief in figure 1.

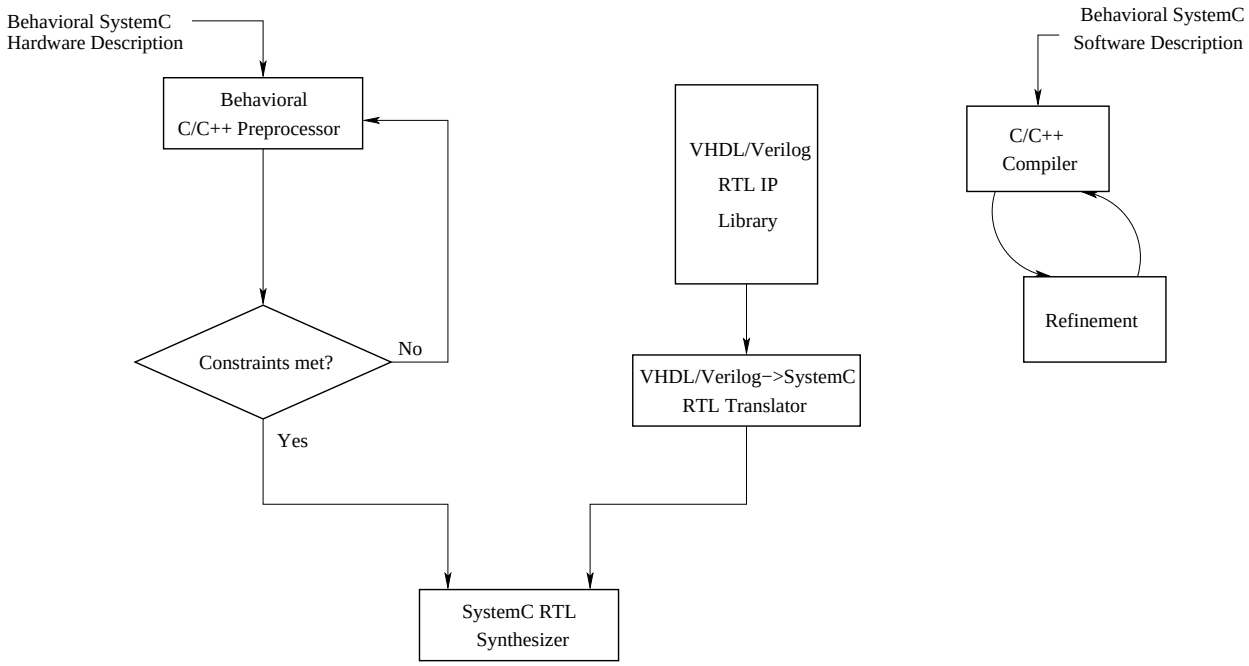

Figure 1. SystemC codesign platform - block diagram

It is common practice today that designs start with a system level model. This is usually written in C/C++, since such a specification is executable and can be used to verify the overall functionality through simulation and custom testbenches. This step can be well hosted under a SystemC environment.

Next, as shown in figure 1, software and hardware components are separated. Each one follows a different design route. However, at each moment, regardless of the progress each design group has made, SystemC allows cosimulation and covalidation, with the original testbenches, generated during system level functional simulation. This is a strong advantage of the proposed methodology and has been found to decrease design time substantially.

The hardware part of the system must first pass through high level synthesis. During the previous years, researchers have made considerable proposals both for general purpose techniques [5, 9, 11] as well as for C/C++ level [4, 7, 10, 12, 17] high level synthesis techniques. Our approach integrates previous proposals while addressing a problem often reported by users of automated high level synthesis tools, limited user interaction. Our high level synthesis step incorporates a SystemC preprocessor, which accepts behavioral SystemC and instructed by appropriate `#pragma` directives, produces RTL SystemC, after applying the requested optimizations. The translation iterates with different user supplied directives until all design constraints are met. The resulting RTL is combined with RTL (or gate level) IP cores to form a complete RTL hardware description, passed to an RTL synthesizer for further optimization. Before that however, another problem calls for solution. Most IP cores today are written in either VHDL or Verilog so, another translator must be written. Previous work [4] has shown the feasibility and the advantages of such a task. In fact, the similarities between SystemC, VHDL and Verilog RTL semantics [14] simplify this translation step.

On the other hand, the software part of the system passes through iterative refinement steps, as traditionally done with everyday software projects. Using the SystemC dynamic thread library, a *Real Time Operating System* (RTOS) core can be written in SystemC, to model are reuse embedded applications.

4. Behavioral Synthesis Preprocessor

Behavioral synthesis or high level synthesis has been an active research field for almost 20 years. A lot of different approaches with interesting and efficient results have been presented all these years [5, 11]. Today, even though it is not considered a completely solved problem, because a lot of NP-complete (or NP-hard) subproblems are involved, it is mature enough to offer industrial level tools and become part of practical design chains.

However, one of the problems designers find difficult to overcome with automated design environments, is the low level of interaction and control over the au-

tomated process. This is particularly true for behavioral synthesis. Hardware designers are not familiar with a press-and-go user interface, which accepts a behavior (much like a software algorithm) and produces an architecture.

The presented design platform offers solution to this problem by proposing an iterative preprocessing design technique. The advantages of this approach are the following:

- The designer can selectively apply the required optimizations by using the corresponding `#pragma` directives.
- The designer can iteratively apply different optimizations, either substituting or supplementing one another until the required constraints are all met.
- The step wise refinement of the design allows better understanding of the result of each individual optimization.
- Expressing both the initial as well as the refined models in the same language (SystemC) allows fast refinement validation through simulation and minimizes errors introduced by manual model translations.
- Knowing the semantics of the RTL subset of SystemC the designer can understand the implementation that will result from each RTL description.

5. RTL Library Translator

During the last years, the diversity of solutions offered by the EDA industry has somehow decelerate tool acceptance. The many different design entry methods, specifically the different design specification languages, the incompatible library formats and the different methodology approaches are just a few of them. A typical example is the "HDL Wars" of the last decade, with VHDL and Verilog (as well as their synthesizable subsets) rivaling for general acceptance. Now that the two HDLs have "seized fire" and taken their places, C/C++ SLDLs seem to prepare for battle against both each other as well as HDLs.

A way to prevent this "war" is to define clear semantics for each language and each application domain. Whenever the application domains of two or more languages overlap, equivalent semantics should be defined, that is semantics with the same implementation but different syntax.

This was the inspiration for the approach of the presented design platform, when dealing with RTL IP core integration. On one hand, a number of HDL based IP management tools and libraries exist and a lot of knowledge resides in them for any new approach to ignore them. On the other hand, writing IP cores in SystemC offers the advantage of cosimulation with the initial testbenches (generated during system level functional validation), at lower abstraction levels (RTL), minimizing errors and delays in the design cycle. So, VHDL and Verilog translators have been used to bring existing HDL cores into the SystemC world.

The key issue in the translation was the corresponding RTL semantics of each language. Using equivalent semantics [14] helped generate equivalent models. As an example, consider VHDL. The issues that had to be resolved and their solutions were mainly the following:

- **Hierarchy:** The VHDL `entity` hierarchy was directly translated into a SystemC `SC_MODULE` hierarchy.
- **Processes:** All VHDL processes were directly translated into SystemC `SC_METHOD`s.
- **Data types:** Corresponding data type have been used for replacements, for example `sc_bv` for `bit_vector`, `sc_lv` for `std_logic_vector`, etc.
- **Signals and ports:** VHDL signals were translated into SystemC `sc_signal`s and ports into `sc_port`s, with the same flow direction.
- **Combinational logic:** Replacement of equivalent operators was performed.
- **Sequential logic:** Register and latch inference statements were found in both VHDL and SystemC and replacement took place. For example, the following code fragments are equivalent and they both infer a register:
  - **VHDL**

```
entity ff is
  port (clk: in bit;
        q1: in bit;
        q2: out bit);
end ff;

architecture infer of ff is
begin
  process(clk)
```

```
  begin
    if rising_edge(clk) then
      q2<=q1;
    end if;
  end process;
end infer;
```

– **SystemC**

```
SC_MODULE(ff) {
  sc_in<bool> clk;
  sc_in<bool> q1;
  sc_out<bool> q2;

  void infer();

  SC_CTOR(ff) {
    SC_METHOD(infer);
    sensitive_pos << clk;
  }
};

void ff::infer()
{
  q2.write(q1.read());
}
```

A similar approach was taken for Verilog HDL.

6. Implementation Details

The proposed platform consists of language translators. Since such programs are utilized in many different application domains, there exist a number of tools which support development. One of the most efficient is the SUIF compiler system [8] (`suif.stanford.edu`).

SUIF is a research compiler. I is design to facilitate experimentation and development of new compiler algorithms, ranging from high level transformations to conventional data flow optimizations. The core system includes a parallelizer that can automatically find parallel loops in Fortran and C programs and generate parallelized C code. It provides all features necessary for parallelization like data dependence analysis, reduction recognition, symbolic analysis, unimodal transformations and data flow optimizations. While the system is not as robust as commercial compilers, it is capable of compiling standard benchmark suites.

In the proposed design platform SUIF has been used to build:

- A SystemC front-end, on top of the normal C front-end supplied by SUIF, for the behavioral synthesis preprocessor.
- VHDL and Verilog front-ends for the IP translators.
- An RTL SystemC back-end, on top of the normal C back-end supplied by SUIF, for both the behavioral preprocessor and the IP translators.
- Specific behavioral transformations (specific scheduling and register allocation algorithms) as separate SUIF passes.

7. Experimental Results

A number of small scale designs have been performed with the proposed platform. They consist of hardware components only and the results of behavioral synthesis are comparable with the ones found using the Synopsys Behavioral Compiler [13]. Simulation time (pre and post synthesis) is comparable with commercial HDL simulators. A number of projects are under development (for Ethernet and 3G mobile communications) for which specification, system validation and RTL generation required less than a week to complete.

8. Conclusion

In this paper, an integrated system level design platform, based on the SystemC specification language, has been presented. It has been found that SystemC, equipped with precise and strict RTL semantics, is offering an efficient design methodology. It can support and automated industrial hardware/software codesign projects. The question however is not how well SystemC is performing, but if it can be adopted by designers. Towards this end some problems are still open. First of all, there is a need for a *single* SLDL, no matter what its name will be, and there is no need for more “Language Wars”. Next, the new design language must be supported by tools that take advantage of previous work and that can work together with the existing design flows. Also, since SystemC is addressing higher abstraction levels, a corresponding design platform must promote user interaction, which can end up as advanced user awareness of the automated design process. Finally, designers must have enough time to feel comfortable with the new methodology and climb the learning curve of the new design paradigm. Under these circumstances, SLDLs can leverage industrial design automation into the next millennium.

References:

[1] R. J. Baker, H. W. Li, and D. E. Boyce. *CMOS: Circuit Design, Layout, and Simulation*. IEEE Press, 1998.

[2] J. Bhasker. *A Verilog HDL Primer, Second Edition*. Star Galaxy Publishing, 1999.

[3] J. Bhasker. *A VHDL Primer, Third Edition*. Prentice Hall, 1999.

[4] G. Bollano, P. Garino, M. Turolla, and M. Valentini. SystemC's impact on the development of ip libraries. In *IP Europe Conference*. CMP, 2000.

[5] G. De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, 1994.

[6] G. De Micheli. Hardware synthesis from c/c++ models. In *Design Automation and Test in Europe Conference and Exhibition*, pages 382–383. ACM/IEEE, 1999.

[7] G. Economakos, P. Oikonomakos, I. Panagopoulos, I. Poulakis, and G. Papakonstantinou. Behavioral synthesis with SystemC. In *Design Automation and Test in Europe Conference and Exhibition*, pages 21–25. ACM/IEEE, 2001.

[8] R. S. French, M. S. Lam, J. R. Levitt, and K. Olukotun. A general method for compiling event-driven simulations. In *32nd Design Automation Conference*, pages 151–156. ACM/IEEE, 1995.

[9] D. Gajski, N. Dutt, A. Wu, and S. Lin. *High-Level Synthesis*. Kluwer Academic Publishers, 1992.

[10] A. Ghosh, J. Kunkel, and S. Liao. Hardware synthesis from c/c++. In *Design Automation and Test in Europe Conference and Exhibition*, pages 387–389. ACM/IEEE, 1999.

[11] Y.-L. Lin. Recent development in high level synthesis. *ACM Transactions on Design Automation of Electronic Systems*, 2(1):2–21, 1997.

[12] L. Semeria and G. De Micheli. Resolution, optimization, and encoding of pointer variables for the behavioral synthesis from c. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(2):213–233, 2001.

[13] Synopsys. *Behavioral Compiler User Guide Version 1999.10*, 1999.

[14] Synopsys. *Describing Synthesizable RTL in SystemC*, 2001.

[15] Synopsys, CoWare, Frontier Design and others. *SystemC Version 1.0 User's Guide*, 2000.

[16] Synopsys, CoWare, Frontier Design and others. *Functional Specification for SystemC 2.0*, 2001.

[17] K. Wakabayashi. C-based synthesis experiences with a behavior synthesizer, "Cyber". In *Design Automation and Test in Europe Conference and Exhibition*, pages 390–393. ACM/IEEE, 1999.