SyCE: An Integrated Environment for System Design in SystemC*

Rolf Drechsler Görschwin Fey Christian Genz Daniel Große

Institute of Computer Science
University of Bremen
28359 Bremen, Germany
{drechsle, fey, genz, grosse}@informatik.uni-bremen.de

Abstract

We present an integrated system design environment for SystemC, called SyCE. The system consists of several components for efficient analysis, verification and debugging of SystemC designs. The core tools are 1) ParSyC, a parser for SystemC designs that has also some synthesis options, 2) CheckSyC, a verification tool for formal equivalence checking, property checking and generating checkers for simulation or synthesis, 3) DeSyC, a tool for automatic debugging and error location in netlists, and 4) ViSyC, a visualization tool for schematic and source code view supporting cross-probing and annotation of simulation and debugging results. The tools fully support hierarchy and interact tightly. Designs can be described at different levels of abstraction.

1. Introduction

The design complexity of today's circuits and systems requires modeling at a high level of abstraction. Until recently, mainly *Hardware Description Languages* (HDLs), like VHDL or Verilog, were used to describe the hardware on the *Register-Transfer-Level* (RTL). Considering ASICs of more than 10 million gates and a HDL to gate ratio of approximately 1:10 to 1:100, i.e. one line of HDL code on the RTL corresponds to 10 to 100 gates in the netlist, the HDL description consists of several hundred thousand lines of code.

To cope with these designs the abstraction level is raised. One very promising way to do so is based on descriptions in C-like languages (see e.g. [9]). In this context SystemC is a very powerful language that allows to describe circuits and systems at different levels of abstraction, i.e. from transaction-level down to the gate-level [8, 10].

In the recent past many approaches have been presented for individual tasks in the system design phase, e.g. tools for synthesis [13] and verification [12, 2]. But for successful circuit and system design there is a high demand to cover all aspects of the flow. Especially design understanding by visualization and debugging to locate errors should be aided by tools. To address all of these issues different tools have to be tightly integrated.

In this paper we present the integrated system design environment SyCE that has been developed over the past few years at the University of Bremen. SyCE is the first integrated environment for SystemC design that allows to design, verify and debug systems. The SystemC description itself, but also the debugging results, can be visualized. The design environment consists of four main components: a parser (including logic synthesis options), a verification tool, a debugging tool, and a visualization component. The project started three years ago and in the meantime first results on the individual tools have been published in [7, 5, 4, 3, 6]. Here, for the first time the complete design environment is described and the latest features are highlighted.

Based on SyCE, it is now possible to design, verify and debug within one framework. First experimental studies show very promising results.

2. Overall Flow of SyCE

The overall flow is described in Figure 1. The input consists of a SystemC description and (if the verification tool is used) a set of properties can be specified.

In a first step the parser *ParSyC* reads in the SystemC description and transforms it to an internal representation. The parser – as shown in the figure – is the core of SyCE and generates the information required by the other tools. *ParSyC* is an improved version of the tool presented in [4]. It produces an easy-to-process representation of a SystemC design in form of an *Abstract Syntax Tree* (AST). The tool PCCTS (Purdue Compiler Construction Tool Set) [11] was used to build the parser. PCCTS enables the description of the syntax of SystemC in the form of a grammar, provides facilities for AST construction and finally generates a parser. In consecutive translation steps the AST of the SystemC description is transformed into an internal representation. The design can be described at different levels of abstraction. Dependent on the consecutive steps various operations can be performed. *ParSyC* can also generate a flat netlist and by this can be used for logic synthesis.

*Parts of this work have been supported by BMBF VALSE-XT.

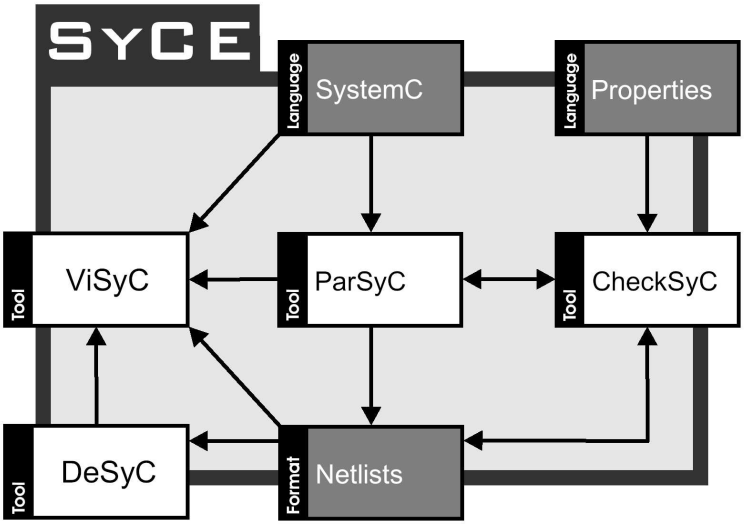

Figure 1. Overall flow of SyCE

Besides synthesis verification has become one of the most important aspects in system design. So far, there is not one single technique that covers all demands [1] and for this, the verification tool *CheckSyC* [5, 6] provides different verification approaches that include formal techniques and simulation approaches based on checkers. Simulation techniques are known to be "less powerful" regarding the completeness, but due to the complexity of the systems it happens that a formal proof fails. In this case simulation is a good "fall back" position. Here, it is important to notice that our simulation environment is based on the same property specification as the formal property checker. By this, the designer uses the same specification scenario for different approaches. This underlines the unified system view in SyCE. For verification of the design the tool *CheckSyC* offers different formal and simulation based approaches, i.e. equivalence checking and property checking as well as checker generation. In all cases, if a faulty behavior is observed, one or more counter-examples (or traces in the sequential case) are generated and given to the debugging tool.

DeSyC is used for debugging on the gate-level. A netlist generated by *ParSyC* and the counter-examples derived by *CheckSyC* are the input. Verification tools usually only provide a set of traces that exhibit the malfunction, i.e. a set of counter-examples. Then, debugging has to be carried out – often manually – using a simulator. *DeSyC* aids this time-consuming task by automatically localizing candidate error sites. As a result only a small portion of the circuit has to be inspected to correct the error. The most important features of *DeSyC* are:

- Fast simulation based diagnosis using counter-examples
- Diagnosis for time-dependent errors
- Handling of hierarchical circuits

The basic diagnosis technique in *DeSyC* is path-tracing and uses the improvements proposed in [3]. The tool automatically identifies circuit locations that might be the reason for the faulty behavior. The resulting set of error candidates is handed to the visualization tool.

For visualization the tool *ViSyC* has been developed that makes use of the visualization engines from Concept Engineering[1]. *ViSyC* allows to easily navigate through complex designs. But following the description above, *ViSyC* also provides the visualization of debugging results. This helps the designer to easily identify reasons for bugs.

ViSyC is a tool for schematic and source code view supporting cross-probing and annotation of simulation and debugging results. The tool can be used "stand-alone" by directly using the information generated from *ParSyC* or in combination with *DeSyC* as outlined above, i.e. the diagnosis results are simple to understand and presented in an easy-to-read way.

In the following we mainly concentrate on the "stand-alone" features, since the use in combination with *DeSyC* essentially describes a special case, i.e. some parts of the design are highlighted and the parts not relevant for debugging are not shown.

The tool supports a rich set of features, like:

- detail hiding
- source code browsing
- interactive design exploration

[1] www.concept.de

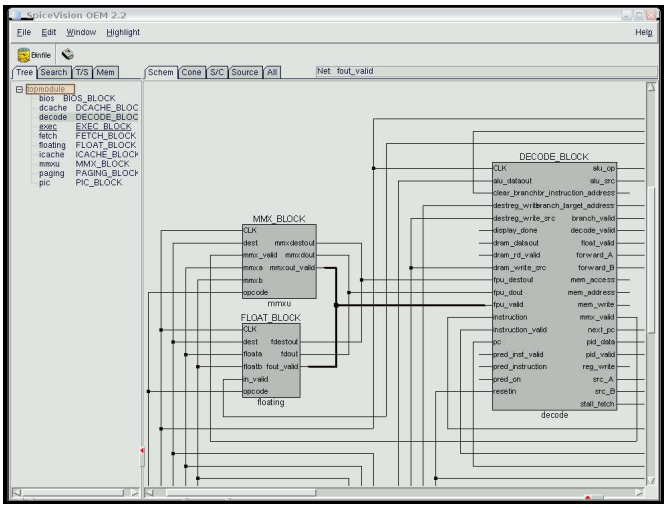

Figure 2. Schematic view

- cross-probing between schematic view and SystemC source code
- critical path fragment navigation
- object search

ViSyC implements system exploration via an interactive GUI, that connects several schematic views with the corresponding source code fragments. Each schematic view represents a kind of abstraction level, where each level is a superset of the less abstract one. Each abstraction level of the specification is mapped to one of the schematic levels. We adopted the method of data hiding from SystemC, that encapsulates modules to get a higher level of abstraction than allowed in RTL designs.

An example can be seen in Figure 2. The schematic view of a SystemC RISC-CPU is shown. The CPU source code is included in the official SystemC package and freely available. The right part of the window displays the high-level view of module instances and their connections. Single components can be highlighted through selection. Information about the selected component is displayed in the information bar on top of the schematic. Module declarations are displayed in the tree window, next to the schematic. The tree window enables hierarchical access to all module declarations and their submodules. Selecting submodules is possible through expanding subtrees. All selected components can be displayed in another view by dragging them into another window. Thus the RISC structure can be visualized arbitrarily in the range from high-level to gate-level view.

3. Summary

In this paper we outlined common problems in the hardware design flow, handled by HDL-based tools. We introduced the integrated SystemC environment SyCE, that addresses high abstraction levels and covers important aspects of a continuous design flow.

References

[1] R. Drechsler. *Advanced Formal Verification*. Kluwer Academic Publishers, 2004.

[2] F. Ferrandi, M. Rendine, and D. Scuito. Functional verification for SystemC descriptions using constraint solving. In *Design, Automation and Test in Europe*, pages 744–751, 2002.

[3] G. Fey and R. Drechsler. Efficient hierarchical debugging for property checking. Technical report, DDECS, Bremen, 2005.

[4] G. Fey, D. Große, T. Cassens, C. Genz, T. Warode, and R. Drechlser. ParSyC: An efficient SystemC parser. In *Workshop on Synthesis And System Integration of Mixed Information technologies (SASIMI)*, pages 148–154, 2004.

[5] D. Große and R. Drechsler. Checkers for SystemC designs. In *Second ACM & IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, pages 171–178, 2004.

[6] D. Große and R. Drechsler. *CheckSyC*: An efficient property checker for RTL SystemC designs. In *IEEE International Symposium on Circuits and Systems*, 2005.

[7] D. Große, R. Drechsler, L. Linhard, and G. Angst. Efficient automatic visualization of SystemC designs. In *Forum on Specification and Design Languages*, pages 646–657, 2003.

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

[9] R. G. (moderator). IEEE design and test roundtable on C++-based design. *IEEE Design & Test of Comp.*, pages 115–123, 2001. May-June.

[10] W. Müller, W. Rosenstiel, and J. Ruf, editors. *SystemC Methodologies and Applications*. Kluwer Academic Publishers, 2003.

[11] T. Parr. *Language Translation using PCCTS and C++: A Reference Guide*. Automata Publishing Co., 1997.

[12] J. Ruf, D. W. Hoffmann, T. Kropf, and W. Rosenstiel. Simulation-guided property checking based on multi-valued ar-automata. In *Design, Automation and Test in Europe*, pages 742–748, 2001.

[13] Synopsys. *Describing Synthesizable RTL in SystemC^{TM}, Vers. 1.1*. Synopsys Inc., 2002. Available at http://www.synopsys.com.