Chapter 8

NONINTRUSIVE HIGH-LEVEL SYSTEMC DEBUGGING*

Frank Rogin\textsuperscript{1}, Erhard Fehlauer\textsuperscript{1}, Steffen Rülke\textsuperscript{1}, Sebastian Ohnewald\textsuperscript{2}, and Thomas Berndt\textsuperscript{2}

\textsuperscript{1}Fraunhofer-Institut für Integrierte Schaltungen IIS
Branch Lab Design Automation
Zeunerstr. 38, 01069 Dresden, Germany
\{frank.rogin; erhard.fehlauer; steffen.ruelke\}@eas.iis.fraunhofer.de

\textsuperscript{2}AMD Saxony LLC & Co. KG / Dresden Design Center (DDC)
Wilschdorfer Landstr. 101, 01109 Dresden, Germany
\{sebastian.ohnewald; thomas.berndt\}@amd.com

Abstract We present a nonintrusive high-level SystemC debugging approach to be used with SystemC v2.0.1 and GNU debugger GDB. Our approach is integrated into an industrial design flow and enables developers to debug designs at high-level working with signals, ports, events, and processes. Thus, one gets quick and concise insight into static structure and dynamic behavior of the design without the burden of gaining detailed knowledge of the underlying SystemC simulation kernel. Only minor transparent changes to SystemC kernel source code are required, whereas there is no need to touch the flow or the designs. Practical experiences show promising results.

Keywords High-level Debugging, validation, system level design, SystemC, GDB

1. Introduction

System level design methodologies promise to address major challenges in modern System-on-Chip (SoC) designs. System level design embraces various abstraction levels, different components (IP, SW/HW), diverse tools,
and methodologies which further complicate design comprehension. Studies revealed that today often more than 50% of design time is spent to verify a complex design that means to identify, understand, localize, and correct errors [3].

Many system level languages and design environments were proposed over the last years, e.g. [2, 10]. One of the most popular languages of this type is SystemC [11]. It has become a de facto standard in industry and in the academic field. SystemC provides concepts such as object-orientation, concurrency, and high-level modeling.

Currently SystemC does not comprise debugging aspects. It solely defines functions to trace module level variables and signals. Traced values are written to file during simulation, and analyzed with standard tools afterwards. Standard C++ debuggers are applied to analyze a functions local variables during simulation run. Unfortunately, both debugging approaches operate on very low abstraction level. Especially, standard C++ debuggers do not understand specific SystemC constructs. Besides, SystemC maps modules onto individual threads of execution which leads to nonlinear execution sequences. This makes predicting which module will be active next extremely difficult.

As working at appropriate abstraction levels is an essential means to understand designs and to fix bugs quickly, several commercial and research tools have been developed dealing with high-level SystemC debugging. Some of the available commercial solutions and academic prototypes are listed and assessed below.

MaxSim Developer Suite [1] comprises a block level editor, and simulation, debugging, and analysis tools. It addresses architectural analysis as well as SystemC component debugging at low level and at transactional level. ConvergenSC System Verifier [12] targets SystemC system level design and verification. It utilizes a simulation kernel which is specially adopted to fit SystemC needs. Its integrated debugger offers SystemC specific commands supporting breakpoints and SystemC QThreads at source level. CoCentric System Studio [13] supports SystemC design, simulation and analysis at system level, and partly synthesis from behavioral and RT level. It utilizes standard C++ debuggers (e.g. GDB), i.e. it does not handle SystemC constructs in a specific way.

[7] presents a method to extract structural data from SystemC designs automatically, and to pass it to a commercial visualization tool using an application programming interface (API). The SystemC kernel has been modified to interface to the API. [9] uses SystemC simulation results to create Message Sequence Charts to visualize SystemC process interaction at a high level. Filters cut out parts of interprocess communication in order to reduce information complexity. [4] applies the observer pattern [8] to connect external software to the SystemC simulation kernel. This general method facilitates loose coupling and requires just minimal modifications of the SystemC kernel.