System Co-Verification of Concurrent Designed Heterogeneous Sub-Systems

Suad Kajtazovic, Christian Steger* Markus Pistauer†
{kajtazovic, steger}@iti.tugraz.at m.pistauer@cisc.at

Abstract

This paper focuses on the integration of concurrent developed sub-systems and the automatic setup of a distributed, heterogeneous verification platform to be used with today's popular simulators from the EDA industry for analog, digital and mixed signal simulation. The sub-systems are mostly developed in different description languages to gain best system performances and the required functionality. For the verification of the system, the system integration requires either translating of foreign sub-systems in a specific language, which causes lost of performances or using of cosimulation techniques to bridge the gap between the sub-systems. A novel methodology presented in this paper is applied in an application for automatic generation of heterogeneous, distributed cosimulation frameworks. It is explained by an example taken from the automotive industry, whereby analog, digital and software components are simulated.

1 Introduction

Verification plays an important role in the design of microelectronic systems. A designed system can be verified using simulators. However, microelectronic embedded systems become more and more complex. They can contain both digital and analog components. Most ASICs do not only contain mixed-signal components but also MOEMS (MEMS/MOEMS: Micro Electro Mechanical/Optical Systems) such as pressure sensors, acceleration sensors etc. Complex systems are designed mostly separate using different modeling languages. The sub-systems are probably optimized but to verify the functionality of the system they need to be integrated into one simulation environment. The verification of a system is the next problem. There are three possibilities to verify such as system in one environment:(1) Homogeneous verification by translation of sub-system descriptions into a target language, (2) Homogeneous verification by enhancement of the provided simulator to support required description languages, and (3) Heterogeneous verification, whereby a simulator is used for each description language. Here, cosimulation interfaces are required for the data exchange.

Translating a sub-system description into a target language often produces loss of performances. The enhancement of simulators to support most of the used description languages is desired by many EDA tool vendors. Design and integration of concurrent developed

*Graz University of Technology, Institute for Technical Informatics, Inffeldgasse 16/1, A-8010 Graz, Austria
†CISC Semiconductor Design+Consulting GmbH, Lakeside B07, A-9020 Klagenfurt, Austria

sub-systems within an automatically generated cosimulation framework is a main issue discussed in this work. The automation in system verification can decrease system design. Especially design steps for setting-up a verification platform can be automated. In this paper we present essential steps in a system design, which can be automated to decrease the system design time as well as a method for the cosimulation data-exchange between involved simulators. One important topic in this work is the generation of cosimulation interfaces.

2 Related work

In past years the automatic generation of cosimulation interfaces has been researched intensively. Some of the cosimulation tools related to this work are presented here.
Jerraya et al. [ACFP96] describes CoSim, a tool for automatic generation of VHDL-C interfaces. It is integrated into the COSMOS codesign tool, which allows a system specification in SDL and supports an automatic translation to the corresponding VHDL and C code. CoSim uses a so called VCI compiler to generate the VHDL-C interface. From an interface-description, the VCI compiler generates the needed VHDL entity and the C part of the interface. Data exchange between both sides is realized using UNIX pipes.
A multilanguage distributed cosimulation platform can be created automatically by the MCI (Multilanguage Cosimulation Interface) [HLV+98] tool. The MCI tool uses a configuration file to generate interfaces to the common cosimulation bus automatically. All involved sub-systems are connected to this bus. It serves as a common backplane for data-communication. The MCI tool configures an untimed, event-controlled cosimulation, without synchronization of involved simulators. It also generates interfaces for VHDL, C, SDL and Matlab.
DCB (Distributed Cosimulation Backbone) [BW02] is based on the HLA (High Level Architecture) method for the generation of distributed cosimulation interfaces. DCB serves as common interface for different simulator types. Each simulator can be connected via ambassadors to the DCB backbone. Ambassador controls the data exchange between DCB backbone and connected simulator. DCB supports both synchronous and asynchronous simulation. Therefore, rollbacks are possible.

2.1 Summary

In contrast to tools described above, our methodology uses generation of channel-based layered cosimulation interfaces, which are inserted into the system. Moreover, the inserted interfaces control the cosimulation with an asynchronous mechanism. The implemented synchronization method enables cycle-accurate cosimulations. The interface generation is not only focused on the interfaces between two models but on the whole system design. To solve problems, which occur when integrating sub-systems developed in different languages, we tried at first to describe the top-level of a system using a semantic that is language independent. We present a complete solution for generating a verification platform both for homogeneous and for heterogeneous configurations.

3 Design methodology

In this approach we used the IP (intellectual property) library to design a system. The IP library provides verified and reusable models. Moreover, it enables us to define rules important for creating of a verification platform.
The most important factors, which depend the creation of a verification platform, are the system description at the top-level, available cosimulation environment and their geographical configuration. The generation of a cosimulation platform can be automated. It has to be considered at three design levels: (1) System design level - A system is described using an intermediate description that enables integration of sub-systems described in different languages, (2) Language level - The system has been enhanced and modified to meet the requirements of the target simulation environment, and (3) Simulator level - The verification platform has been generated. Involved simulators communicate via integrated cosimulation interfaces. The dataflow and simulation of whole system is controlled by a synchronization mechanism. The language level and simulator level are steps that are processed

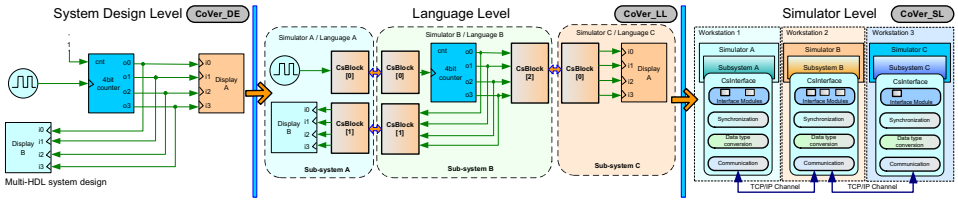

Figure 1: Design levels for a co-verification

automatically. Figure 1 depicts different levels in a system design, which are considered for automatic generation of a verification platform.

3.1 System design level

A system at the system design level (*CoVer_DE*) can be fully described by definining blocks, their parameters and connections between ports of the blocks (models). A behavioural description of a model at this level is not necessary since we are building our system by using IP blocks provided from an IP-library. Each block contains a given number of ports and parameters. The block has a hierarchical structure that can contain sub-blocks and sub-connections. This additional information for the system integration as well as the hierarchical structure of the system is described using XML (Extensible Markup Language). Independent of used description language, XML enables us to describe the system's hierarchical structure and its mapping. It also allows us to set model parameters. Moreover, it is used for the model description and for the graphical representation.

3.2 Language level

At the language level (*CoVer_LL*) the system has been described with the IP modules and enhanced with interface modules, which are necessary to connect modules described with different languages. At this level the XML description serves as an outline to generate the system description in target languages of used IP modules. As depicted in figure 1 the modules are grouped by their language. For each language group a top-level module has been created. To connect two modules, which are written in two different languages an interface-module pair has been inserted between them. The functional behavior of the system is not affected by inserting of the interface-modules. Moreover it enables a distributed parallel cosimulation, which can increase the simulation speed.
The interface module transfers signals from one simulator to the other simulator. It uses an open simulator interface to establish a connection to the simulator and to the signals that should be transferred to other simulator. A cosimulation interface between VHDL and C has been evaluated by many published projects. Mostly it is based on FLI (Foreign Language Interface) [Tec01] provided by VHDL, which is used to build a wrapper written in C. Signals connected to the entity of the interface module are available via the simulator's

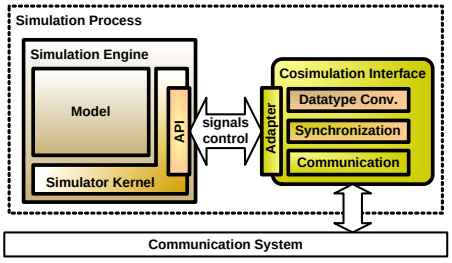

Figure 2: A generic interface module connected to the simulation engine.

open interface. These signals are converted to a data format adequate for data transfer and they are transferred to other interface module in an interface pair. An interface module cares about the transfer of signals in both directions and proper conversion of signal values. Furthermore it synchronizes the simulator, on which he is attached with other simulators in a cosimulation.

3.3 Simulator level

The language groups build independent entities, which are simulated on the language specific simulators. At the simulator level (*CoVer_SL*) the generated entities are adapted to the target simulation environment. Using open interfaces provided by simulators, different language groups are connected via interface modules. The interface modules are responsible for synchronizing involved simulators, for data conversion between different languages and for data transfer. One of the important tasks in a cosimulation environment is to synchronize involved simulators. The communication overhead and the cosimulation speed are directly affected by the used synchronization mechanism. A decentralized conservative

synchronization method based on the BSP (Bulk Synchronous Parallel) [Cal95] model has been used to synchronize simulators. Converting data types is the next step done by the interface modules. This step is required to convert values from one data format to another. The data type conversion is predefined during the code generation phase at the language level. Communication is based on channels, which implements different protocols such as TCP/IP, UDP, UNIX-Pipes etc. A peer-to-peer connection is used to connect two simulators. The network topology is created by inserted interface-modules. No common interface backplane has been used in order to reduce the communication overhead. The backplane benefits the control of the entire cosimulation process at a central place and the dynamically seamless coupling of simulators. The major drawback of the backplane approach is the centralized communication topology that represents a performance bottleneck. The generated model sources are distributed to the proper locations and simulators are configured to simulate the parts of the designed system.

4 Experimental example

An example of the code generation of a heterogeneous system using the methodology described in section 3 is presented in this section. The system controls the power needs of the automotive electromechanical loads and the charging of the battery and prevents that the battery gets completely discharged at any time. The microcontroller developed in SystemC as a state-machine represents a software part of the system. All other components are coded in VHDL-AMS. The concurrent developed sub-systems are integrated using the schematic editor in our application framework. Figure 3.a depicts a schematic overview of the APMS system at the system design level. To overcome the problems of the integration

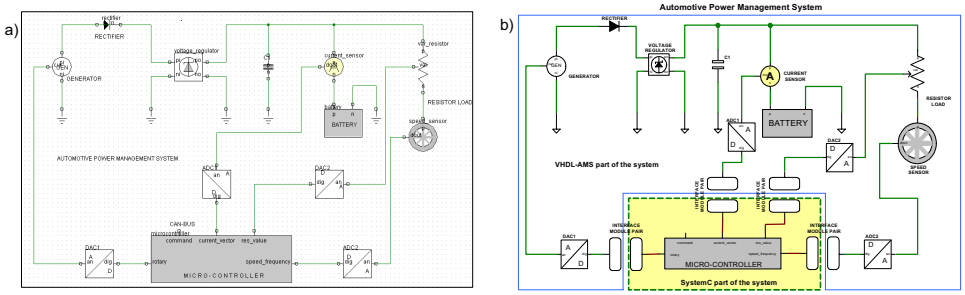

Figure 3: System overview at the system design level (a) and at the language level (b).

of models written in different languages, four CsBlock pairs have been inserted between the microcontroller unit and four converters. With the insertion of CsBlocks, the APMS system has been subdivided into two language/simulator groups. Next step is generating the top-level description of the generated language/simulator groups. In our example we have two language groups: SystemC and VHDL-AMS. The SystemC system consists of the microcontroller and four SystemC-interface modules. Respectively the VHDL-AMS sub-system

consists of VHDL-AMS components and four VHDL-AMS-interface modules. Figure 3.b depicts the system overview at the language level with inserted interface modules.

4.1 Experimental results

We applied our methodology on this example very successful. To allow a comparison between a homogeneous and heterogeneous simulation we developed the MCU unit in VHDL and in SystemC. The computed data of both simulations are identical, which verifies the correctness of the inserted interfaces and the used synchronization method. Describing of the top-level using XML semantic confirms that XML fulfil the requirements at this abstraction level. Inserting interface modules between functional modules described using different languages is more flexible than creating wrappers around the foreign modules, which have to be used in a specific language environment. The applied synchronization technique can handle multiple simulator connections and it supports cycle-accurate cosimulation. The presented solution enables parallel, distributed cosimulation or even simulation to increase the simulation performance.

5 Conclusion

In this paper we presented a novel methodology for system verification of concurrent developed sub-systems. A complete solution has been successfully validated with an example from the automotive industry. We developed an application framework, which is based on the described methodology. Currently it supports a system design in SystemC, VHDL/AMS, SaberMAST languages. Future work in our project will be the evaluation of the used methods in other complex cases and the enhancement of the developed application to support other HDLs and EDA tools.

References

[ACFP96] A.A.Jerraya, C.A.Valderrama, F.Nacabal, and P.Paulin. Automatic generation of interfaces for distributed c-vhdl cosimulation of embedded systems: an industrial experience. Technical Report Proceedings of IEEE, June 1996.

[BW02] Braulio Adriano de Mello and Flávio Rech Wagner. A Standardized Co-Simulation Backplane. *SOC Design Methodologies*, pages 181–192, 2002.

[Cal95] Radu Calinescu. Conservative discrete event simulations on bulk synchronous parallel architectures. Technical Report PRG-TR-16-95, Computing Laboratory, Oxford University, 1995.

[HLV+98] F. Hessel, P. LeMarrec, C. Valderrama, M. Romdhani, and A. Jerraya. MCI-multilanguage distributed co-simulation tool, 1998.

[Tec01] Model Technology. *Modelsim: Foreign Language Interface*. Model Technology, Portland, USA, 5.5f edition, August 2001.