A SoC design flow based on UML 2.0 and SystemC

Sara Bocchio, Elvinia Riccobene, Alberto Rosti, and Patrizia Scandurra

1 STMicroelectronics, AST Agrate Lab R&I, Italy
{sara.bocchio, alberto.rosti}@st.com

2 Università di Milano, Dip. di Tecnologie dell’Informazione , Italy
riccobene@dti.unimi.it

3 Università di Catania, Dip. di Matematica e Informatica, Italy
scandurra@dmi.unict.it

Abstract. In this paper we show how to improve the system level design flow for System-on-Chip currently used at STMicroelectronics exploiting the use of lightweight modeling methods, like UML, to be used as higher system-level languages operating in synergy with some other lower level languages, like SystemC. Such an improvement it was possible through the definition of a UML 2.0 profile for SystemC. High-level functional UML models or platform independent models (PIMs), as conceived by the current Model Driven Architecture (MDA) vision, can be “refined” down to platform specific models (PSMs) written in the UML SystemC profile for the final implementation. A case study based on a 802.11.a physical layer application mapped on a hardware software platform is presented, in order to validate the UML-SystemC generation flow.

1 Introduction

The conventional design process starts from a functional executable model, usually done in a software programming language (like C/C++) or by modeling tools such as Simulink, capturing the system behavior from the requirements, usually written in natural language. The partitioning between hardware and software follows to decide the final destination of the design components. The software parts are compiled for the target processing elements, whereas the hardware part is initially expressed in an hardware description language at RTL, and then refined to a structural representation using the logic synthesis which produces a netlist that can be mapped on a library of standard cells or hard-macro blocks (blocks for which the layout is already available) for the final implementation. This structural representation is used to produce the final physical layout. However, there is still a large gap between the functional executable model level and the implementation level. In addition, since the verification of an integrated hardware and software system takes place after both components are completed, errors discovered in the verification phase are often uncorrectable.

We claim that this gap can be reduced by leveraging the joint capabilities of UML and SystemC to provide a modeling framework for systems in which high-level functional models can be “refined” down towards an implementation model directly connected to the final physical implementation (see [9]).
As shown in Fig. 1, the UML may improve the SoC design flow essentially in three ways: (i) the UML in a platform-independent manner can be adopted at system functional executable model level to describe the specification, like Simulink; (ii) the UML profile for SystemC can be used for the hardware description at the abstraction layers (functional untimed/timed, transactional, behavioral, bus cycle accurate) on top of the RTL layer; (iii) UML profiles tailored for programming languages like C/C++, Java, etc. can be used, instead, for the software parts. Moreover, the availability of UML profiles for other languages (JHDL, SystemVerilog, etc.) currently used in the SoC design and the definition of suitable PSM bridges - as conceived by the current MDA vision [6] - among profiles may allow to move from the description of a system in a given language to the description of the same system in another language at the same level of abstraction or lower.

Fig. 1. New SoC Design flow

2 A UML 2.0 profile for SystemC

The UML profile for the SystemC language is based on the UML 2.0 specification [7] and on the SystemC 2.0 specification [5]. The profile definition is organized in four parts, reflecting the architecture of the SystemC definition as built on top of the standard C++. The complete UML profile definition for SystemC is described in [10].

1. **The SystemC core layer - structure and communication** - defines stereotypes of the core layer of SystemC which can be used in various UML structural diagrams to represent the structural and communication building blocks of a SystemC specification like modules, interfaces, ports and channels.

2. **The SystemC core layer - behavior and synchronization** - defines stereotypes of the core layer of SystemC which lead to a variation of the UML method state machines, called SystemC Process State Machine (see [8]), to allow high level specification of the SystemC processes functionality in modules and channels.
3. **The SystemC core layer - data types** - defines a UML class library to represent the set of SystemC data types.

4. **The SystemC layer of predefined channels, interfaces and ports** provides concepts for the layer of the predefined channels, interfaces and ports of SystemC. These concepts are implemented both as a class library, built applying the basic group of stereotypes of the SystemC core layer (or the basic SystemC profile), and as a group of standalone stereotypes that specialize those of the SystemC core layer (the extended SystemC profile).

### 3 Tool Support

In order to develop our SoC design flow, we decided to rely on tools supporting UML 2.0 with the standard extension mechanism of UML profiles, and (possible) model transformations. Using such UML tools allows modeling the system by means of a diagrammatic UML representation, then to refine it in a UML-like SystemC model and to perform code generation in SystemC. Our current implementation is based on the Enterprise Architect tool [3], but any other tool supporting UML 2.0 can be also used and easily customized since our profile has been defined relying only on the standard UML 2.0 specification instead of tool’s “implementations” of UML 2.0.

We added to the Enterprise Architect the capability to generate complete SystemC code from the UML models for both structural and behavioral aspects. This feature can be obtained either by customizing the C++ code generation templates of the UML tool or by exporting the model in the standard XMI (XML Metadata Interchange) format and generating the SystemC code from it.

### 4 HW/SW co-design

Our approach to SoC design through UML allows modeling the hardware and software parts together in a unique UML environment. The hardware architecture is modeled by constructs from the UML profile for SystemC, so it can also be refined down to the register transfer level, modeling the details of the implementation. The application can be modeled in software at two levels of abstractions: transactional or instruction level. When modeled by transactions, it works as a library encapsulated in a SystemC module. Processes are associated to the software functional description to sustain its concurrent activity within the system, whereas communication is implemented by transactions that model the interactions with the hardware architecture. This level of abstraction allows testing the correctness of the application on a high level description of the hardware, focusing on the communication and on its performances. The application can be also modeled at the instruction level by integrating an instruction set simulator of the target processing unit within the SystemC environment, to execute the compiled application code together with the SystemC model that implements the hardware. In this case both hardware and software models provide cycle accurate simulation.
5 Case Study

We provide an application example related to a system that implements an 802.11.a physical layer transmitter and receiver described at instruction level. Therefore from one side the C/C++ application code will be encapsulated as a library functions in a UML class. This class provides through ports the I/O interface of the software layer to the hardware system. From the other side a ISS encapsulation in UML is also provided, in order to represent all the elements of the system. The UML encapsulation of the ISS is built by the UML profile for SystemC, in order to generate a SystemC wrapper for the ISS and to allow an hw/sw cosimulation at transactional or cycle accurate level. The application code generated by UML diagrams – EA tool already provide this feature - will be executed by the LX ISS.

Fig. 2. The StdBus module

The system is composed by a VLIW processor developed in ST, called LX [4], with a dedicated hardware coprocessor that implements an FFT operation. The processor acts as a master to the hardware module and the memory components, where code and data are stored. The communication is realized by a system bus: in particular we use On Chip Configurable Network [1], a transactional level cycle accurate SystemC model available at [2]. All the blocks of this system are modeled in UML class diagrams using our profile, in order to automatically generate the SystemC code. As an example, Fig. 2 and Fig. 3 show the class diagram of the StdBus model and the state machine of the SC_THREAD stdbus_process.
In Fig. 4 the UML composite structure diagram of the overall system is shown. It describes the hierarchical structure of the platform. Several UML object diagrams can describe different configuration scenarios. This separation allows generating different parametric platform instances starting from the same model, i.e. structure diagram.

The design has been imported from the C++/SystemC reference description into EA, empowered by the UML 2.0 profile for SystemC, then the code generation has been tested to cross-check the generated C++ code and the original code.
Fig. 4. The Platform internal structure

Acknowledgements

We would like to thank Marcello Coppola, Andrea Council, Stephane Curaba, Andrea Giorgi and Francesco Papariello for their precious work.

References