A UML Profile for SysML-Based Comodeling for Embedded Systems Simulation and Synthesis

Fabian Mischkalla, Da He, Wolfgang Mueller
University of Paderborn/C-LAB, Paderborn, Germany

Abstract—After its wide application in the area of software engineering, UML is still not fully accepted for other engineering domains like for electronic systems design. The main obstacle is due to a major gap in the design flow between UML-based modeling and verification. To overcome this gap, we introduce a UML profile for synthesizable SystemC and C and present its implementation in the context of the advanced SysML modeling environment of ARTiSAN Studio. We demonstrate how to customize Studio for SystemC/C comodeling so that it can serve as a verification and synthesis front-end.

Index Terms—UML, SysML, SystemC modeling, simulation

I. INTRODUCTION

For the efficient application of UML in engineering it is essential to close the existing gap in the design flow from modeling to verification and synthesis. In this paper, we introduce an approach which is based on our UML profile for synthesizable SystemC and was developed in the ICT project SATURN. We have implemented the profile as an extension of SysML in the context of ARTiSAN Studio where we combined it with a code generation for cosimulation and synthesis. In this context, we defined and implemented a methodology covering the entire design flow from high level comodeling to HW/SW cosimulation and FPGA synthesis. This paper introduces the details of our SATURN UML 2.0 profile, composed of three subprofiles: (i) synthesizable SystemC profile, (ii) Agility synthesis profile, and (iii) C profile. These profiles are applied to configure ARTiSAN Studio, in which we also customized the code generator for combined SystemC and C code generation. For cosimulation the generated SW application integrates with the SystemC modules via a memory-mapped I/O. The generated code implements a cosimulation via blocking inter-process communication between SystemC and QEMU, a fast software emulator. QEMU applies dynamic binary translation and supports a variety of instruction sets. Thus, QEMU may also execute the final software as binaries under the original operating system. After cosimulation, we can also easily control the further synthesis from ARTiSAN Studio, which is executed by the SystemC Agility compiler for VHDL generation which is further input to ISE/EDK for final FPGA configuration. The same SW binaries executing under QEMU can be uploaded to the target CPU on the FPGA without further modification (see Fig.1).

Today, there exist several related approaches for the definition of UML profiles for hardware modeling. Kangas et al. published several work for FPGA synthesis [1] and the UML MARTE profile supports the modeling of a specific set of HW components [2]. Most recently, we can also find approaches for more specific hardware profile like IP-XACT based UML profiles as in [3], [4], [5]. However, they all do not consider SystemC specific aspects. For SystemC, there exists OMGs UML for SoC profile [6] and Riccobene et al [7]. Though intended as a standard for general SoC modeling, the OMG standard is very much SystemC oriented. However, both approaches mainly target at SystemC simulation and do not consider HW/SW cosimulation and synthesis.

Fig. 1: SATURN Design Flow

The remainder of this paper is organized as follows. In the next Section we introduce our UML profile. Sections III and IV demonstrate the customization of ARTiSAN Studio and its application to HW/SW comodeling before the final section closes with a conclusion.
II. SATURN PROFILE

Our SATURN profile is defined as a set of three UML subprofiles using the standard extension mechanism of UML, i.e., stereotypes, tagged values and constraints as it is shown in Figure 2. The profiles enable HW/SW comodeling with automatic code generation for cosimulation and synthesis.

Fig. 2: Block Definition Diagram (BDD)

They are based on the OMG standard SysML and have extensions for SystemC, synthesis and C which are presented in more details in the following subsections. The most common SystemC data types (e.g. sc_int, sc_uint, sc_bigint, sc_biguint etc.) and interfaces, as well as ANSI C data types, are predefined and integrated into the profiles. In the communication library we provide several HW IP components like BlockRAM and PLB bus including their SystemC implementation.

A. UML Profile for Synthesizable SystemC

An overview of the profile which maps synthesizable SystemC subset similar to [8] to SysML is shown in the following table. The profile assigns stereotypes with more SystemC specific constraints to SysML objects like blocks, parts, and ports. Graphical symbols for ports and interfaces are inherited from SystemC drawing conventions.

<table>
<thead>
<tr>
<th>SystemC</th>
<th>Stereotype</th>
<th>Metaclass</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module</td>
<td>&lt;&lt;sc_module&gt;&gt;</td>
<td>Class</td>
<td><img src="module.png" alt="Diagram" /></td>
</tr>
<tr>
<td>Interface</td>
<td>&lt;&lt;sc_interface&gt;&gt;</td>
<td>Interface</td>
<td><img src="interface.png" alt="Diagram" /></td>
</tr>
<tr>
<td>Port</td>
<td>&lt;&lt;sc_port&gt;&gt;</td>
<td>Port</td>
<td><img src="port.png" alt="Diagram" /></td>
</tr>
<tr>
<td>Primitive</td>
<td>&lt;&lt;sc_in&gt;&gt;, &lt;&lt;sc_out&gt;&gt;</td>
<td>Port</td>
<td><img src="primitive.png" alt="Diagram" /></td>
</tr>
<tr>
<td>Signal</td>
<td>&lt;&lt;sc_signal&gt;&gt;</td>
<td>Property, Connector</td>
<td><img src="signal.png" alt="Diagram" /></td>
</tr>
<tr>
<td>Process</td>
<td>&lt;&lt;sc_method&gt;&gt;, &lt;&lt;sc_thread&gt;&gt;</td>
<td>Action</td>
<td><img src="process.png" alt="Diagram" /></td>
</tr>
<tr>
<td>Main</td>
<td>&lt;&lt;sc_main&gt;&gt;</td>
<td>Operation</td>
<td>None</td>
</tr>
<tr>
<td>Clock</td>
<td>&lt;&lt;sc_clock&gt;&gt;</td>
<td>Class</td>
<td><img src="clock.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>

Fig. 3: SATURN UML Profile

Fig. 4: UML Profile for SystemC
SysML blocks stereotyped with <<sc_module>> represent the basic architectural units. Inside each block users can specify processes, ports, internal data, local functions and optionally instances of other modules to create a hierarchy. The <<sc_interface>> and <<sc_port>> stereotypes are used for modeling communication between modules. For simple port connection <<sc_in>>, <<sc_out>> and <<sc_signal>> can be applied. The dynamic behavior of blocks are modeled either by <<sc_method>> or <<sc_thread>> stereotypes, which are applied to an action node of an activity diagram. Finally, for simulation purpose <<sc_main>> and <<sc_clock>> stereotypes are introduced.

B. Profile for Agility Synthesis

The Agility SC Compiler [9] that we have applied for the SystemC to VHDL translation provides a wider synthesizable subset compared to the OSCI defined subset [8]. For this, we have to define additional stereotypes in order to cover Agility synthesis specific features. Figure 5 shows the SystemC compiler dependent stereotypes. <<ag_global_reset>> can be applied on an input port of a module to declare a global asynchronous reset for the whole design. By default, the Agility SC compiler generates VHDL ports of type std_logic for single-bit ports and numeric_std_unsigned for multi-bit ports. The application of the <<ag_constrain_port>> stereotype specifies user preferred VHDL port types. By means of the <<ag_constrain_ram>> and <<ag_ram_port>> stereotypes the Agility is capable of synthesizing an array declaration to a hardware-platform specific RAM component with single or true dual ports. The <<ag_blackbox>> stereotype supports IP reuse of the RAM component, where the Agility SC compiler creates an instance of a RAM component and links its implementation with an imported netlist. Finally, the <<ag_main>> stereotype identifies the synthesis entry point.

III. CUSTOMIZATION OF ARTiSAN STUDIO

In the SATURN project we employed ARTiSAN Studio for UML/SysML modeling which is an integrated development tool suite that provides advanced system modeling with customizable code generation capabilities. Due to the fact that our profile is based on standard SysML that comes with the default configuration of ARTiSAN Studio, we defined the previously explained stereotypes and tagged values in a profile package as shown in Figure 7. That screenshot is taken after the UML/SysML and SATURN profiles are loaded into the model.

C. Profile for C

The C stereotypes given in Figure 6 provide means for SW integration as part of HW/SW comodeling. Here, the CPU platform is characterized by its architecture and operating system. We support this by <<cpu>> which is applied on SysML blocks. A software executable for a CPU is also defined by a block, but stereotyped with <<executable>>.

Each subpackage of the Saturn Profile package represents a subprofile in the SATURN profile with the corresponding stereotypes and tagged values. ARTiSAN Studio provides the ability to link user defined stereotypes with UML/SysML meta-class items, e.g., <<sc_module>> is
associated with class/block, so that one particular stereotype can only be applied to the model items, to which the stereotype is linked. The <<sc_module>> stereotype, for instance, can only be applied to class/block.

In order to make the SATURN profile more applicable and user friendly, we extended parts of Studio’s user interface including context menu commands, diagram toolbars, double-click behavior etc. This concept is referred to as an ergonomic profile in ARTiSAN Studio with Visual Basic Script language support. The latter is also for the implementation of profile constraints, through which user input to the model can be simplified and less error-prone.

IV. APPLICATION OF THE PROFILE

The SATURN methodology is based on the seamless integration of a tool chain starting with ARTiSAN Studio as a frontend tool for comodeling and code generation. After import of SATURN profile on top of Studio’s SysML profile, code generation executes by permanently observing modifications of the model and updates the generated code on demand and vice versa. The model covers hardware and software aspects as well as the basic communication structure.

A. Hardware

From architectural modeling, we apply SysML Block Definition Diagrams (BDD) as shown in Fig. 2. A BDD is based on UML structured class diagram, which is for the description of the system’s composition. Each block in a BDD represents either a SystemC module (HW) or a Processor component (SW platform). As our profile is targeting at synthesizable SystemC code generation, we support only the composition with different multiplicities to associate blocks.

For more advanced system modeling of internal organization, we apply SysML Internal Block Diagrams (IBD), which are based on UML2 composite structure diagram. Hereby, IBD parts are defined as instances of SysML blocks. Fig. 8 gives an example of an IBD. For each SystemC module instance users can add either SystemC primitive ports or sc_port/ sc_interface. The connection between SystemC and CPU parts is established by connecting non-stereotyped SysML standard ports.

The dynamic behavior of system design is defined through a SysML Activity Diagram (AD). We support one diagram for each SystemC module. The action nodes in the AD represent SystemC processes, i.e., either SC_METHODs or SC_THREADS. For the sake of synthesizable SystemC, threads can only be sensitive to clock signals, i.e., they are clocked synchronously. The implementation of each process is entered as textual code inside each action node. Figure 9 shows an example of a SC_METHOD implementation. This modeling style comes is due to our experience that textual representation of sequential code is much more effective than 2- or more dimensional graphics. To avoid disambiguities we assume a Petri-Net like token semantic for the AD.
B. Communication Architecture

To arrive at a precise cosimulation of the hardware (SystemC) and the software (C), the SATURN profile applies a TLM-based design of communication architectures. For this we provide a repository with classes of predefined SystemC blocks. Currently, these are for buses, memories, and transactors.

- **BRAM blocks**: These blocks are applied as BRAM memories (single port or true dual port, default size = 2048 Bytes). The internal structure and behavior is not modeled as they refer to existing IP blocks for simulation.

- **PLB block**: In TLM-based designs, software components and hardware components communicate via a bus structure. Our framework provides a predefined bus block and its SystemC implementation for simulation. Currently, this is only the PLB for Xilinx Virtex-II Pro.

If a hardware block is attached to a PLB, a transactor has to be applied to connect the bus and the block. In summary, the following transactors are supported:

- **PLB_SLV_Transactor block**: A general purpose slave transactor IP. The code generator automatically integrates the corresponding SystemC model.

- **PLB_BRAM_Transactor block**: The specific slave transactor IP for attaching a BRAM to the bus.

- **PLB_MST_Transactor block**: A general purpose master transactor IP. Currently, the internal structure and behavior have to be explicitly modeled by the user.

- **PLB_CPU_Transactor block**: The specific master transactor IP to connect a processor block to the PLB.

C. Software

In SATURN profile, blocks can represent any level of the system architecture: the top level environment, the HW/SW components, and the software executables. In order to map software executables to processor components, i.e., blocks stereotyped with <<cpu>>, the SATURN profile applies SysML allocations. For this, Figure 11 shows the SATURN mechanism for mapping a block stereotyped with <<executable>> to a processor instance of the system. In IBDs such an association is indicated by the name of the allocated software executable in the allocatedFrom compartment. Additionally, the allocatedTo compartment of the software block lists the deployment on the processor platform. As shown in the properties of a software block, each executable has a tagged value "directory" linked to the stereotype <<executable>> that refers to the directory of the source code. This provides a flexible interface to integrate arbitrary source code (which also could be generated by any other software environment or code generator) or any UML software component based on the ARTiSAN Studios C Profile.

For the code generation of each executable, our profile refers to a generic driver in user mode and a parameter file containing the address range of each device connected to the bus model. The driver implements read and write access to bus slaves via address memory mapped I/Os of the operating system.

Finally, type and version of the operating system must be specified for each processor instance. Our implementation currently only supports Linux for PowerPC 405 for Virtex-II Pro FPGAs. However, in principles it is possible to provide any operating system and other instruction set architectures as long as they are supported by the QEMU software emulator and the target platform.

V. CONCLUSION

In this paper, we presented our approach for SysML-based HW/SW comodeling by means of the SysML capture ARTiSAN Studio. For this, we introduced a UML/SysML profile for the customization of ARTiSAN Studio for SystemC/C comodeling. As Studio comes with a framework for designing customizable code generators it can be easily adopted to ESL tool chains. In [10] we have demonstrated how to apply it for SystemC/QEMU cosimulation. The latter is an efficient software emulator which can be configured with the target instruction set architecture and operating system so that it is possible to cosimulate SystemC modules with the final SW binaries. This results in an accurate and efficient virtual environment for software evaluation. The same
binaries can be later loaded and executed on the final target CPU. Thus, the introduction of the SysML-based capture is a first step to close the gap to first requirement analysis and system specification which currently moves to software engineering technologies based on UML.

Future work will focus on enhancements towards power and timing analysis, i.e., TLM 2.0 and UPF, and extensions for functional verification.

**ACKNOWLEDGMENT**

The work described herein is partly supported by the BMBF ITEA2 project VERDE (01|S09012H) and the FP7 project SATURN - SyML bAsed modeling, architectTUrE explo Ration, simulation and sYnthesis for complex embedded systems (FP7-216807). We also gratefully acknowledge the fruitful discussions and feedback from our project partners: ARTISAN Software, INTRACOM, Thales T3S, and University of Cantabria.