Closing the Gap between UML-based Modeling, Simulation and Synthesis of Combined HW/SW Systems

Fabian Mischkalla, Da He, Wolfgang Mueller
University of Paderborn, C-LAB
Fuerstenallee 11, D-33102 Paderborn, Germany

***Abstract*—UML is widely applied for the specification and modeling of software and some studies have demonstrated that it is applicable for HW/SW codesign. However, in this area there is still a big gap from UML modeling to SystemC-based verification and synthesis environments. This paper presents an efficient approach to bridge this gap in the context of Systems-on-a-Chip (SoC) design. We propose a framework for the seamless integration of a customized SysML entry with code generation for HW/SW cosimulation and high-level FPGA synthesis. For this, we extended the SysML UML profile by SystemC and synthesis capabilities. Two case studies demonstrate the applicability of our approach.**

I. Introduction

Current embedded systems and SoC design has to cope with first-time-right requirements in order to shorten the time-to-market. This rapidly increasing complexity of such systems requires the introduction of new innovative design methodologies with design automation at higher levels of abstraction. In the last years, we can identify two major approaches in the area of systems specification and simulation. One is the introduction of system level design languages (SLDL) like SystemC and SpecC. The other is the introduction of UML to electronic systems design, mainly for early design steps. There exist only very few approaches, which combine UML and SystemC for modeling and verification of such systems. However, this actually is mandatory for a seamless design flow from UML-based requirement specification and modeling to simulation and synthesis.

In this paper we present an approach to bridge the gap between UML-based modeling and SystemC-based simulation. We propose a methodology, which covers the entire design flow of HW/SW comodeling, -simulation, and -synthesis. For comodeling, we customize a graphical SysML [1] design environment, i.e., Artisan Studio. For customization, we present a set of domain specific UML profiles on top of SysML capturing synthesizable SystemC and C. Additionally, for efficient SystemC/C cosimulation we retargeted the code generator for TLM-based SystemC/QEMU code generation, where the QEMU processor emulator is connected to a memory-mapped SystemC TLM bus via a specific transactor. Through this we can apply QEMU as powerful and fast instruction set simulator with support of multiple instruction set architectures, like ARM or PowerPC, which may run different (real-time) operating systems. After cosimulation, the SystemC part is further synthesized to VHDL for FPGA configuration and the compiled SW binaries are loaded to the target CPU.

The paper is organized as follows. In Section II, we give an overview of related work. Section III introduces the individual steps of our methodology. Section IV presents two case studies with experimental results before Section V closes with a conclusion.

[1] SysML^{TM} is an OMG standard profile for UML defined for general Systems Modelling

II. Related Work

A comprehensive overview of UML for electronic systems design can be found in [1]. They introduced two basic categories for model-to-code relationships: 1-to-many and 1-to-1. In our approach, we address the 1-to-1 relationship in order to define an efficient UML capture and a clear semantics for the later code generation.

A 1-to-1 SystemC code generation from UML was first investigated in [2], who presented several benefits when combining UML/SysML and SystemC, like a common and structured environment for system documentation/specification. The introduction of several software-oriented constructs in SystemC 2.0, like Interface Method Calls, additionally motivated the combined application of SystemC and UML. In this context, efforts at Fujitsu [3] pushed the development of the OMG UML Profile for SoC, and STMicroelectronics introduced their UML/SystemC profile [4] soon after.

As an alternative for direct code generation, some approaches take the UML model as an XMI (XML Metadata Interchange) file for translation to their target like [5], [6], [7]. A UML-based multiprocessor SoC design framework for FPGA synthesis is proposed by KOSKI [8], which provides a UML profile for orthogonal application and architecture modeling at system level, an automated architecture design space exploration, and automatic code generation based on platform libraries.

All approaches target at the generation of SystemC or C/C++ executable models from UML for early functional and performance verification. However, in contrast to our

978-3-9810801-6-2/DATE10 © 2010 EDAA

approach, [2], [3], [4], [5], [6] do not support HW/SW comodeling and [8] does not consider the execution of native (real-time) operating systems. For such operating systems, we apply and integrate the QEMU processor emulator via a transactor for TLM-based systems. QEMU supports the execution of a variety of operating systems and hardware architectures. Additionally, we consider SystemC for simulation and synthesis in order to arrive at a seamless design flow from modeling to synthesis.

III. Methodology for UML-Based HW/SW Codesign

Our methodology extends generally a classical SystemC to FPGA flow by HW/SW cosimulation and a UML front-end as shown in Figure 1. The focus is on (i) a seamless integration of different commercial EDA tools and (ii) bridging the gap between UML-based comodeling, cosimulation, and a final FPGA implementation.

The flow starts with a UML/SysML capture, like ARTiSAN Studio, which is extended by comodeling capabilities. This supports the application of UML for requirements of functional and non-functional properties without changing the design tool and paradigm. After entering the model, a 1-to-1 code generation is applied. The HW and SW components are automatically translated to SystemC or C. SystemC is compiled to simulation platform dependent binaries. Black box SW components, which refer to external C sources, are cross-compiled for the target architecture.

The cosimulation is performed by a simulator and a software emulator that are synchronized by communication patterns given in the generated code. On the one hand, we apply Modelsim as a mixed-language simulator of SystemC and VHDL in order to check later SystemC synthesis results. On the other hand, QEMU executes the native operating system and software binaries based on the specific hardware architecture. For our Xilinx Virtex-II targets, for instance, we setup QEMU for PowerPC 405 emulation with a linux operating system. After successful cosimulation, a tool, like the Agility SC Compiler [9], is applied for SystemC to VHDL synthesis. The generated VHDL is the input for the later (Xilinx) FPGA synthesis tool chain. In parallel, the already compiled software binaries are directly forwarded for download to the individual target processor, which coexecutes with the FPGA.

In the next paragraphs we present the UML profiles for customizing the UML capture. Thereafter, we give more details on code generation and the individual design steps like comodeling, cosimulation and synthesis.

A. UML Profiles

In order to extend the comodeling capabilities of ARTiSAN Studio, we apply the standard extension mechanism of UML and define a set of profiles [1]. Since Studio is already based on the UML profile SysML, we have defined three profiles for (i) synthesizable SystemC, (ii) synthesis extensions, and (iii) C integration on top of SysML.

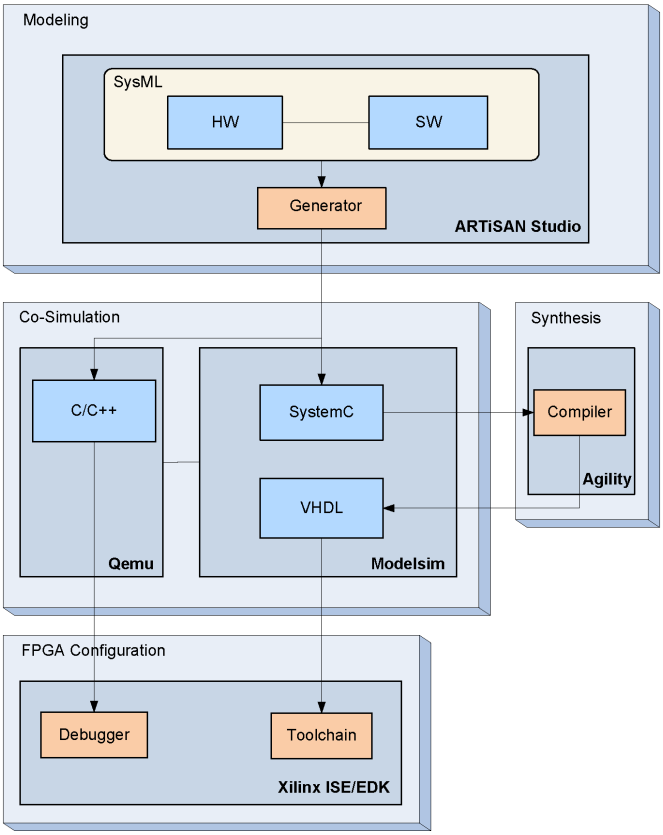

Figure 1. Design Flow

1) UML Profile for Synthesizable SystemC:

The SystemC profile maps the SystemC synthesis subset defined by OSCI [10] to SysML. For this, we assign stereotypes as listed in Table I with SystemC specific constraints to SysML stereotypes. Additionally, graphical symbols from the SystemC drawing conventions for interfaces and ports are integrated.

Table I
Profile for synthesizable SystemC

SystemC	Stereotype	UML Metaclass
Module	≪sc_module≫	Class
Interface	≪sc_interface≫	Interface
Port	≪sc_port≫	Port
Prim. Port	≪sc_in≫ / ≪sc_out≫	Port
Signal	≪sc_signal≫	Property, Connector
Fifo	≪sc_fifo≫	Property, Connector
Process	≪sc_method≫ / ≪sc_thread≫	Action
Main	≪sc_main≫	Operation
Clock	≪sc_clock≫	Class

A SystemC design may include a set of hierarchically structured modules. For that, we assign ≪sc_module≫ to SysML blocks and parts. The communication between SystemC modules is realized via ports, which need interface classes. Correspondingly, we specified ≪sc_interface≫ and ≪sc_port≫. The stereotypes ≪sc_in≫ and ≪sc_out≫ refine SysML flowports to the SystemC primitive ports. They may be bound to primitive channels like ≪sc_signal≫ or

≪sc_fifo≫. Behavior is captured by action nodes stereotyped by ≪sc_thread≫ and ≪sc_method≫. We additionally define ≪sc_main≫ as an simulation entry point and ≪sc_clock≫ for SystemC clocks though both functions are not synthesizable. However, they are mandatory for simulation.

2) UML Profile for Synthesis Extension:
As previously indicated, our methodology applies a SystemC compiler to close the gap between TLM-based SystemC and RTL. Such a compiler typically defines tool specific synthesis extensions. These extensions are macro statements and tagged by ag_, which stands for the applied Agility SC Compiler. As they are just for synthesis, they are typically deactivated for simulation by default. Table II shows the mapping of the synthesis extension stereotypes to UML elements.

Table II
PROFILE FOR SYNTHESIS EXTENSIONS

SystemC	Stereotype	UML Metaclass
Main	≪ag_main≫	Class
Module	≪ag_blackbox≫	Class, Property
Attribute	≪ag_add_ram_port≫	Property
Attribute	≪ag_constrain_ram≫	Property
Prim. Port	≪ag_constrain_port≫	Port
Prim. Port	≪ag_global_reset≫	Port

The ≪ag_main≫ stereotype identifies a SystemC module as a top level block for synthesis. With ≪ag_blackbox≫ it is possible to generate an instantiation of a VHDL entity with appropriate control/data ports. Then, the internal behavior can be linked with an existing netlist of an IP core. The compiler supports also references to native implementations of memories. By default, ≪ag_constrain_ram≫ declares an array as a single-port RAM for read/write access. As memory, like BlockRAM of the Virtex FPGA series, is sometimes configurable as multi-port memory, the stereotype ≪ag_add_ram_port≫ enables the implementation of RAM with true dual port behavior. An asynchronous global reset is defined by ≪ag_global_reset≫.

3) UML Profile for C:
For true HW/SW comodeling with SysML, it is necessary to support basic capabilities for software integration as listed in Table III. Here CPUs are mainly characterized by their architecture and the installed OS. We support this by ≪cpu≫, which indicates a SysML block as a processor with additional properties for the underlying architecture and OS. The ≪executable≫ stereotype is used to refer to an instance of a C application. Through this, we also support software reuse and can easily link the model to arbitrary binaries. This is managed by an existing or automatically generated makefile.

Table III
PROFILE FOR C

C	Stereotype	UML Metaclass
CPU	≪cpu≫	Class
Process	≪executable≫	Action

B. Code Generation

One goal of our approach is automatic code generation from SysML based models. For this, we implement our code generator by means of ARTiSAN's template based code generation. This technology is applied for code generation backend in several UML modeling environments. Additionally, ARTiSAN Studio integrates a framework for model-code synchronization, the TDK (Template Development Kit) and Shadow ACS (Automated Code Synchronization).

Figure 2 illustrates the basic scheme how TDK and Shadow ACS cooperate. Shadow ACS loads the *User Model* into the Dynamic Data Repository (DDR), which stores it in an internal representation. Afterwards a Dynamic Link Library (DLL) is applied to generate the code files. Each time the *User Model* is modified, Shadow ACS detects the changes, updates the DDR and regenerates the code. For reverse engineering, Shadow ACS can also be triggered by modification in the generated code. The DLL is produced by Studio's TDK framework. For this, TDK runs through a *Generator Model*. The *Generator Model* is composed of transformation rules written in a proprietary description language, i.e., SDL Template Language. This language provides various constructs, through which UML elements and properties of the DDR can be retrieved and processed.

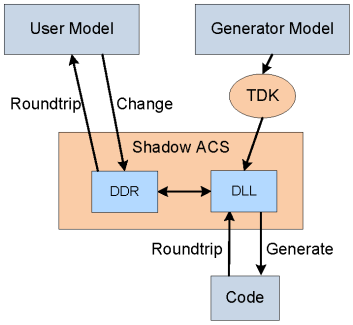

Figure 2. ARTiSAN Studio Code Generation

For SystemC code generation, we thus have to implement a *Generator Model* along the mapping rules of our profiles. Then the final code generator goes through each SysML block, which are divided into three categories. Regular SysML blocks define cosimulation environment definitions. For communication and synchronization between QEMU and SystemC, the code generator generates QEMU plugin for address mapping. For each block stereotyped by ≪sc_module≫ the code generator creates a class declaration with a sc_module inheritance and fills the skeleton with statements on the basis of block properties and operations. SysML parts, ports, and connectors of an Internal Block Diagram (IBD) are translated to module instantiations and port bindings. If the block additionally refers to an SysML Activity Diagram, a SystemC process for each action node is generated. Each SysML block stereotyped by ≪cpu≫ points to an QEMU executable. The action nodes of the associated Activity Diagram refer to software processes for which a makefile is generated that compiles and links the individual modules or libraries to an executable.

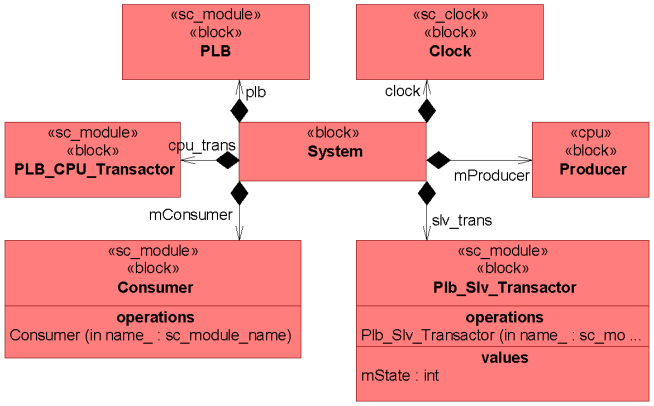

Figure 3. Block Definition Diagram

C. Design Steps

Based on our profiles and the code generation, we now outline some details on the application and the individual sequential design steps.

1) Comodeling:

Comodeling starts with the specification of a SysML Block Definition Diagram (BDD), which is based on the UML class diagram. In a BDD, users define system components in form of SysML blocks as well as properties and operations. A hierarchical relationship is indicated by the composition between blocks. Figure 3 shows a BDD example with a top level block `System`, a CPU `Producer`, and several SystemC modules.

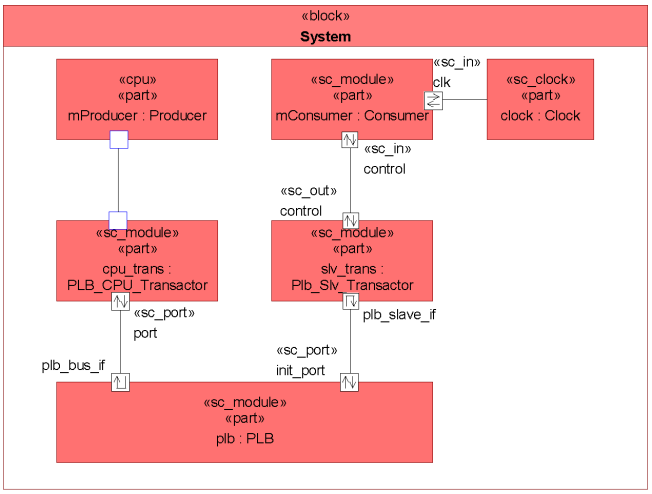

Figure 4. Internal Block Diagram of `System` block

The system and communication architecture is represented by an IBD. SysML parts are instances of the blocks that are defined in an BDD. Figure 4 shows the IBD of the top level block `System`. The parts are connected via SystemC ports while the SystemC/CPU connection is represented by a native SysML port.

A SysML Activity Diagram defines the dynamic behavior of a SystemC module with one action node for each SystemC process. The action itself is specified as plain sequential C++ code. We implemented this solution as it has been shown in industrial studies that it is more efficient to code SystemC behavior descriptions in textual form rather than as activity or state machine diagrams. In the activity diagram the input events represent SystemC signals or ports, to which the processes are sensitive. Figure 5 shows the activity diagram of the `Consumer` block.

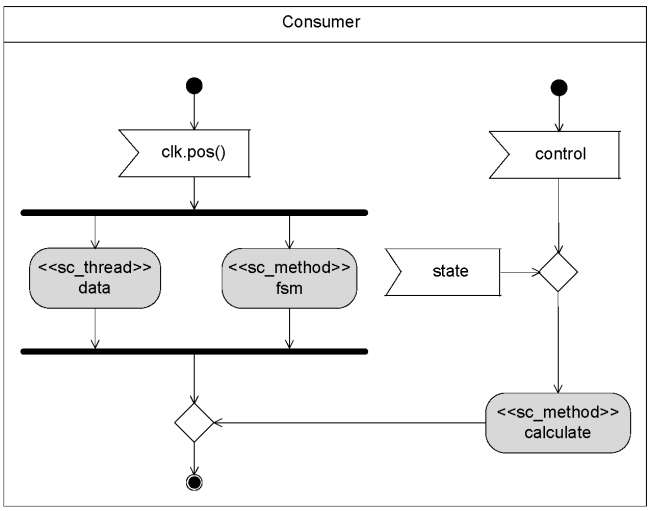

Figure 5. Activity Diagram of `Consumer` block

2) Cosimulation:

After code generation, C binaries on QEMU and SystemC are cosimulated. The main concern here is to deal with communication and synchronization between the simulator and the emulator in terms of autonomous processes. For synchronization, we apply message queues from the boost library, which supports sending and receiving messages and synchronized memory allocation between processes.

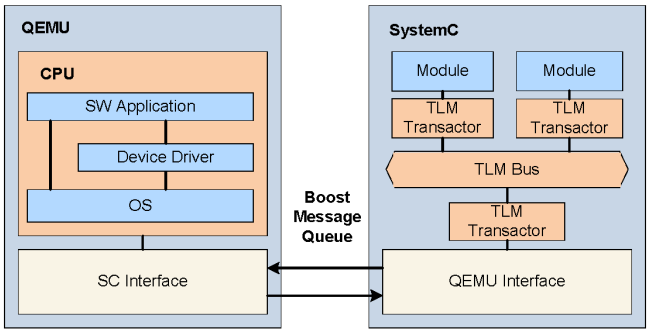

Figure 6. Cosimulation environment architecture

The diagram in Figure 6 outlines the architecture of our cosimulation environment based on an asynchronous communication paradigm. As such, SystemC and QEMU run concurrently until the SW application initiates an IO access, which synchronizes both processes by means of a blocking receive call. If the IO operation is finished, SystemC and QEMU are decoupled and run again until next IO. The IO

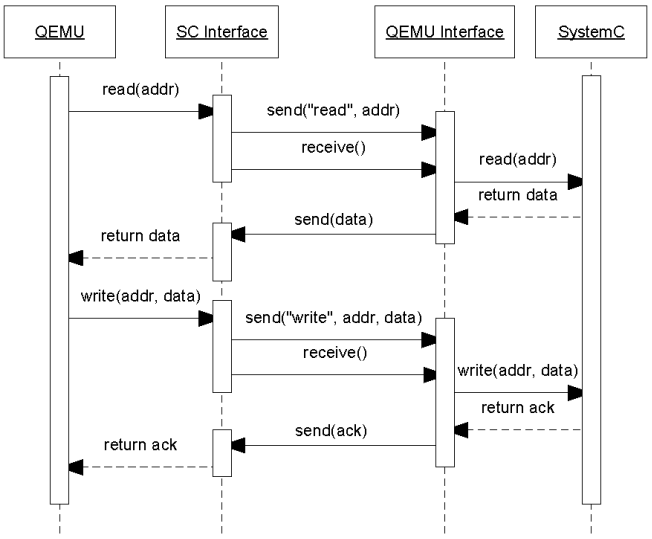

Figure 7. IO operations in cosimulation

operations are provided by automatically generated OS user mode device drivers.

Figure 7 illustrates the sequential flow of read and write IOs. Each IO issued by the software application invokes a corresponding callback function inside QEMU. The callback basically implements a transactor as the `SC Interface` and subsequently sends a write/read request including data/address information to SystemC. Thereafter, a blocking receive call halts QEMU until it receives the expected ack/data from SystemC. In the meantime the `QEMU Interface` receives the request and forwards it to SystemC. The ack/data is returned to QEMU as soon as SystemC simulation is replying.

In our current cosimulation environment, we apply a TLM bus model for architecture simulation on the SystemC side. This bus model is part of the GREENBUS framework from GREENSOCS [11]. GREENBUS provides us with a generic bus framework supporting different abstract simulation modes as known in TLM. These are Programmer View (PV), Bus Accurate (BA) and Cycle Accurate (CC) simulation mode. Additionally, it can be customized towards a specific bus protocol to simulate more architecture specific behavior. Currently Processor Local Bus (PLB) and PCI Bus protocol are available.

By means of inter-process communication between SystemC and QEMU, the synchronization mechanism yields to a major overhead in cosimulation. We have resolved this by introducing a burst transfer technique to minimize the number of sent/received messages and speed up the simulation performance. In the case of a burst length of 8KB, a block of 8KB data is transferred as one message (size is 8192 bytes) instead of 8192 messages (size of each is 1 byte).

3) Synthesis and FPGA Configuration:

To address a more systematic design process, our methodology covers the synthesis of TLM-based SystemC to RTL implementation in VDHL. After a one-click synthesis process, the flow applies a FPGA tool chain, e.g., Xilinx's ISE/EDK Design Suite. EDK supports a complete implementation process for SoC design including HW and SW. We can finally download the generated hardware bitstream with Chipscope Pro from the EDK. For existing software, the flow applies Xilinx's internal debugger XDB to load applications and OS.

IV. Experimental Results

We have evaluated our approach by two case studies. The first one has the focus on comodeling and cosimulation to indicate the performance of our cosimulation concept. The second case study implements an image processing object detection algorithm to demonstrate the design flow in real application and the straightforward refinement to FPGA.

A. TLM Master/Slave

In the first experiment we compare the cosimulation performance based on a point-to-point data transfer scenario under different TLM simulation modes. Additionally, the impact of increasing burst length of transactions is investigated. The example is composed of a HW component (DDR SDRAM) and a SW application, which invokes IO operations. Additionally, we installed linux (kernel version 2.6.29) on QEMU emulating a PowerPC 405 at 100MHz. We modeled this case study in ARTiSAN Studio and generated the code. Figure 8 shows our cosimulation result with the data throughput. It indicates that in all three simulation modes the performance is significantly improved when the burst length increases. It also demonstrates the difference between PV, BA and CC; PV mode is found much faster than BA and CC. In comparison with the average execution time of the same example on a real target platform (Xilinx Virtex-II Pro), the PV simulation mode with burst length larger than 4KB achieves approximately the same result.

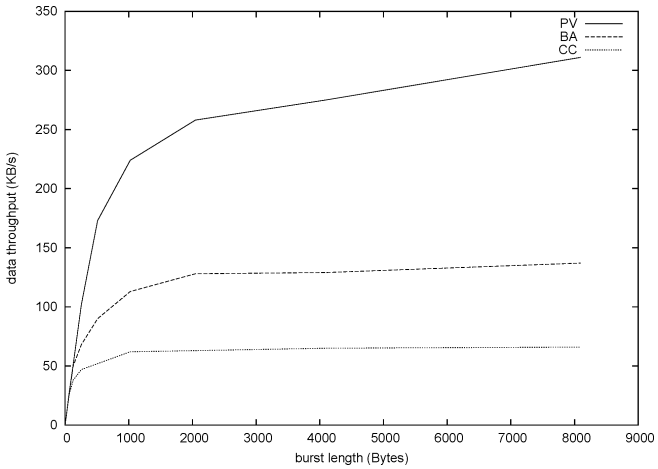

Figure 8. Cosimulation result of point to point data transfer

B. Video Pipeline

The second example demonstrates the benefit synthesizable SystemC code generation from UML/SysML for simulation and synthesis .

Table IV
SIMULATION METRICS OF VIDEO PIPELINE

	Erosion			Dilation			Labeling		
	Time (ms)	Speed-Up	LoC	Time (ms)	Speed-Up	LoC	Time (ms)	Speed-Up	LoC
SystemC TLM	7625	12.20	424	8203	13.19	410	33260	13.22	1063
SystemC RTL	103671	0.89	1621	110922	0.92	1585	453494	0.97	16580
VHDL	93027	1	1606	102048	1	1542	439890	1	15770

The case study implements an image processing video pipeline specification from the Embedded Video Detection (EmViD) project [11]. The system is composed of three HW blocks. It first reads 80x60 pixel images in binary format from memory and performs two sequential morphological operations, i.e., erosion and dilation. These are filters to minimize noise influence. In a second step, all image objects, which remain as coherent regions, are labeled and added with additional coordinates and size information. The generated SystemC components are cycle-accurate and have two communication interfaces. Both, the input and output interfaces are realized by applying the Open Core Protocol (OCP). Each module use an internal block RAM to store the image stream and intermediate results. The system is finally implemented on an Xilinx University Programm (XUP) developer platform board, which is composed of a Xilinx Virtex-II-Pro FPGA with a peripheral connection for a video camera.

For verification, we defined a SystemC simulation testbench, which provides stimuli in form of strict binary images. Table IV shows the results, which we measured during simulation of different refinement steps. For this, we take the host CPU time in milliseconds where each component processes 1000 images of 4800 bit. The host architecture was an Intel(R) Core 2 Duo CPU running with 2.66GHz. To avoid non deterministic results through a possible internal schedule between processors we have disabled one CPU. We see that the TLM based SystemC components run nearly 13 times faster than corresponding VHDL implementations. The intermediate SystemC RTL format is provided by the SC Compiler to validate the generated RTL against the previous defined C++ testbench. The lines of code (LoC) give an overview of the increasing complexity from SystemC to VHDL.

V. CONCLUSION

We introduced a set of UML profiles for comodeling of HW/SW systems and a code generation scheme for cosimulation support. This closes the gap between UML/SysML-based modeling and simulation and significantly automates the error-prone step to manually transfer UML models to SystemC and to setup cosimulation environments. Our two case studies have demonstrated that our flow and the applied tools can manage design of greater complexity providing a considerably fast verification environment. They indicate a significant improvement by automatic generation of the simulation environment. Once the model is implemented just very few steps have to be taken to arrive at a FPGA implementation covering HW and SW application.

However, we can also identify a major drawback which is due to applied commercial tools. In our current environment, error messages which are generated in later design steps can hardly be linked to objects and classes in the UML/SysML model. Nevertheless, the support of Studio for code/model synchronization partly helps to avoid this problem at least in the first code generation.

ACKNOWLEDGMENTS

The work described in this paper was funded by the European project SATURN (FP7-216807). We greatly appreciate the cooperation with the SATURN project partners.

REFERENCES

[1] Y. Vanderperren, W. Mueller, and W. Dehaene, "Uml for electronic systems design: a comprehensive overview," *Design Automation for Embedded Systems*, vol. 12, no. 4, pp. 261–292, 2008.
[2] M. Pauwels *et al.*, "A design methodology for the development of a complex system-on-chip using uml and executable system models," in *System Specification & Design Languages*. Springer US, 2004.
[3] Fujitsu, "New soc design methodology based on uml and c programming languages," *FIND*, vol. 20, no. 4, pp. 3–6, 2002.
[4] E. Riccobene, A. Rosti, and P. Scandurra, "Improving soc design flow by means of mda and uml profiles," in *Proc. 3rd Workshop in Software Model Engineering*, 2004.
[5] R. Boudour and M. T. Kimour, "From design specification to systemc," *Journal of Computer Science*, vol. 2, no. 2, pp. 201–204, 2006.
[6] K. D. Nguyen, Z. Sun, P. Thiagarajan, and W.-F. Wong, "Model-driven soc design via executable uml to systemc," in *IEEE RTSS'04*, 2004.
[7] W. Tan *et al.*, "Synthesizable systemc code from uml models," in *UML for Soc Design, DAC 2004 Workshop*, 2004.
[8] T. Kangas *et al.*, "Uml-based multiprocessor soc design framework," *ACM Trans. Embed. Comput. Syst.*, vol. 5, no. 2, pp. 281–320, 2006.
[9] Agility compiler. [Online]. Available: www.msc.rl.ac.uk/europractice/software/mentor.html
[10] OSCI, *SystemC Synthesizable Subset (draft 1.1.18)*, 2004.
[11] Greensocs. [Online]. Available: www.greensocs.com