The SATURN Approach to SysML-based HW/SW Codesign

Abstract. The main obstacle for the wide acceptance of UML and SysML in the design of electronic systems is due to a major gap in the design flow between UML-based modeling and SystemC-based verification. To overcome this gap, we present an approach developed in the SATURN project which introduces UML profiles for the co-modeling of SystemC and C with code generation support in the context of the SysML tool suite ARTiSAN Studio®. We finally discuss the evaluation of the approach by two case studies.

Introduction

For a wider industrial applicability of UML, it is essential to close the gap between UML-based modeling and the execution of the models for their verification. The SATURN project developed an efficient approach to close this gap by providing code generation from UML/SysML to synthesizable SystemC and C/C++. For this we defined a SATURN methodology covering the entire design flow from co-modeling via co-simulation to co-synthesis. This chapter introduces the SATURN UML 2.0 profiles for synthesizable SystemC and C. The profiles are defined to customize the SysML tool suite Artisan Studio® for SystemC/C co-modeling.
Based on these profiles, we also customized the code generation capabilities of Studio to generate synthesizable SystemC for simulation including makefiles for C/C++ program compilation and scripts for design flow automation. A co-simulation is realized by means of the QEMU software emulator for the execution of the native software on the target operating system. After co-simulation, SystemC can be synthesized to VHDL as input for the ISE/EDK framework to configure the FPGA. The OS image which includes the SW executable is finally loaded to the target CPU for a complete system configuration. The remainder of this chapter is structured as follows. The next section discusses related work. Section III gives an overview of Artisan Studio® before Section IV introduces the SATURN approach. In Section V we present the evaluation by two industrial case studies. Finally, Section VI closes with a conclusion.

Related Work

Several approaches for UML profiles for hardware modeling were presented in the last years. As such, Kangas et al. published several articles in the context of FPGA synthesis [11]. The UML MARTE profile defines the modeling of a specific set of HW components in [16]. Most recently, we can also find approaches for more specific hardware profile like IP-XACT based UML profiles [2][3][20] and we can identify approaches dedicated to SystemC UML profiles like Riccobene et al. [19]. The OMG UML profile for SoC also falls into the latter category as it is very much oriented towards SystemC [17]. However, it lacks details and does not come with a standard metamodel implementation. Both approaches cover complete SystemC and do not address properties for synthesizable SystemC [14].

Different methodologies cover communication and time at different abstraction levels. Methodologies, such as SystemC-AMS [13] and the synthesizable RTL subset [14], use specific signal channels for communication. The OSSS+R methodology [5] raises the level of abstraction of communication by means of shared objects. All these approaches have also raised the level of abstraction of time handling from strict-timed discrete event (DE) to timed-clocked. However, they still handle time in too many details for virtual platform and system-level specification. The growing interest in the development of virtual platforms determined the development of the TLM-2.0 OSCI standard [15]. In terms of time, TLM-2.0 is more abstract than time-clocked approaches, but still needs to assign a time dimension to transactions to enable platform models with more than one master (i.e. multi-processor platforms) and simultaneous transactions. Although untimed modelling is beyond the scope of TLM-2.0, other works point out that transaction level virtual platforms can also be built as untimed functional models, what is called pure Programmer’s View or PV level [4]. Moreover, a productive electronic system level (ESL) design methodology requires system-level specifications at the most abstract untimed level. Several works have taken up this challenge in SystemC,
such as SystMoC [12], SystemC-H [18], and HetSC [8]. The latter is applied by the SATURN approach.

**ARTiSAN Studio®**

Artisan Studio® is an all-in-one integrated development tool suite which provides systems and software modeling and component based development targeted for technical systems. It is an ideal tool for complex mission-critical systems and software engineering. Artisan Studio provides comprehensive support for the leading industry standards, including OMG SysML, OMG UML and Architectural Frameworks. Studio delivers an integrated collaborative development environment – allowing systems and software engineering teams together. Artisan Studio is highly scalable and suitable for use on small and large technical projects with a proven multi-user repository providing a stable, robust working environment to ensure both the high availability of model information, while securing all valuable data. Engineering teams using Studio’s powerful suite of tools can model systems and software, document legacy systems and generate new code with complete control. Geographical distribution of teams can create a multitude of issues for product development. With Artisan Studio sharing a full model is simplified and allows for control from a common repository, increasing benefits in time and process simplification that is difficult with file based systems. However, one model can still be split into logical packages, by means of configuration management ‘sandboxes’ to meet the needs of the project and team dynamics.

Artisan Studio® supports SysML and UML methodologies. It is also powerful and easy to use customization to meet the needs of particular industry, company and individual preferences. Ergonomic profiling can be used to create an industry domain specific profile for the tool, allowing custom views and tools to meet specific needs. Using Artisan Studio® and extending and adding to the UML and SysML profiles project specific information can be captured using an industry standard. Explorer panes, icons and diagrams are included to display specific elements and the user interface is modified to include new context menu items and behavior and the APIs are used to extract model data for analysis.

Artisan Studio® comes with the Automatic Code Synchronizer (ACS) which is an innovative and unique round-trip engineering tool that generates and synchronizes C, C++, C#, Ada, SPARK Ada and Java code. Driven by UML designs within Artisan Studio, the ACS speeds development by automatically generating software system code. It then ensures the UML design and code remain synchronized - immediately ready to support ongoing development, maintenance, enhancement and integration tasks. This design and development integration allows designers, developers and testers to work together more readily. As well as the typical UML class and relationship information, the Automatic Code Synchronizer also uses dynamic information from UML - such as state diagrams – to generate code logic. The resultant code is also instrumented to animate the appropriate diagram within
Artisan Studio®, when the application is executed either on the host or the target. Code instrumentation lets users interact with the live application using model-level debug capabilities. This simulation approach builds in quality at a very early stage, allowing low cost error correction during design, rather than more costly fixes later in the development process. The ACS technology is user configurable and using the Transformation Development Kit it is possible for users to build their own code generators for domain specific languages such as SystemC.

SysML Based HW/SW Codesign

As given in Fig. 1, the SATURN design flow starts with Artisan Studio® which is customized by SATURN UML profiles for SystemC/C co-modeling. ACS has been configured for code generation for SystemC/C co-simulation. ACS generates SystemC models for simulation as well as interface software for full system mode co-simulation with the QEMU SW emulator. The C code compilation is taken by the generated makefiles. The additionally generated scripts implement the design flow automation like the OS image generation. We additionally support the co-simulation with other simulators like Simulink by means of the EXITE ACE co-simulation environment, e.g., for testbench simulation. After co-simulation we integrated SystemC compilers, currently Agility and SystemCrafter, to generate VHDL which is synthesized for FPGA configuration. The same OS
image including the application SW which was executed under QEMU can be finally loaded to the microcontroller of the FPGA for a configuration of the complete system.

**SysML based HW/SW Co-modeling**

For SystemC-based modeling, SATURN defined three UML profiles. i.e., for (i) Synthesizable SystemC, (ii) for Synthesis-Specific definition, and (iii) for C. The first one is based on SysML and summarized in Table 1. It assigns stereotypes with SystemC specific constraints to SysML objects like blocks, parts and ports. Graphical symbols are inherited from SystemC OSCI drawing conventions. A separate tool-specific UML profile is implemented for synthesis. For the Agility SystemC compiler, we defined an Agility profile with <<ag_main>>, <<ag_blackbox>>, <<ag_add_ram_port>>, <<ag_constrain_ram>>, <<ag_constrain_port>>, and <<ag_global_reset>> as stereotypes.

<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>«SC Module»</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Example operations</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Example (in name_ : sc_module_name)</td>
</tr>
<tr>
<td>Interface</td>
<td>&lt;&lt;SC Interface&gt;&gt;</td>
<td>Interface</td>
<td>⬆️</td>
</tr>
<tr>
<td>Port</td>
<td>&lt;&lt;SC Port&gt;&gt;</td>
<td>Port</td>
<td>⬆️</td>
</tr>
<tr>
<td>Primitive Port</td>
<td>&lt;&lt;SC In&gt;&gt; / &lt;&lt;SC Out&gt;&gt;</td>
<td>Port</td>
<td>⬆️</td>
</tr>
<tr>
<td>Prim. Channel</td>
<td>&lt;&lt;SC Signal&gt;&gt; / &lt;&lt;SC Fifo&gt;&gt;</td>
<td>Property, Connector</td>
<td>← →</td>
</tr>
<tr>
<td>Process</td>
<td>&lt;&lt;SC Method&gt;&gt; / &lt;&lt;SC Thread&gt;&gt;</td>
<td>Action Node</td>
<td>«SC Method» run</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>«SC Clock»</td>
</tr>
</tbody>
</table>

Table 1. SATURN UML Profile for SystemC.
Additional C stereotypes had to be defined for SW integration with SystemC. Here, we simply apply \texttt{<<Processor>>} on SysML blocks to define the CPU target platform by its architecture and operating system. A reference to software executables is introduced by \texttt{<<Executable>>}.

In application we start co-modelling with a SysML Block Definition Diagram (BDD) as shown in Fig. 2. The BDD is based on UML structured classes and describes the composition of a whole system. Each block in a BDD represents either a SystemC module (HW) or a Processor component (SW). As the SATURN profile is targeting at synthesizable SystemC code generation, we support the composition of blocks with different multiplicity.

In a next step the system architecture is defined by a set of Internal Block Diagrams (IBDs) which compare to classical SystemC block diagrams as given in Fig. 3. Though the content of Fig. 2 and Fig. 3 are both hardly readable they give an idea of principle BDD and IBD structures of a PL bus example with SystemC and C parts.

![Figure 2. Block Definition Diagram (BDD)](image-url)
**SATURN Code Generation**

One goal of SATURN is automatic code generation from SysML-based models. For this, we implemented our code generator by means of the ARTiSAN’s Transformation Development Kit (TDK). ACS loads UML/SysML Model into the Dynamic Data Repository (DDR), which stores it in an internal representation. A Dynamic Link Library (DLL) generates the code files. Each time the User Model is modified, ACS detects the changes, updates the DDR, and regenerates the code. For reverse engineering, ACS can also be triggered by modifications in the generated code.
Co-Simulation

SATURN comes with a uniform verification framework covering a heterogeneous set of simulators supporting different types of execution platforms:

- Simulators for hardware and test environments like SystemC, Simulink, Dymola,
- Instruction Set simulators and software emulators for full system emulation including the operating system like QEMU,
- Hardware-in-the-Loop for real-time integration of existing hardware.

Though currently only the coupling of SystemC and QEMU is supported, the final goal is the generic integration of several execution platforms including Matlab/Simulink with EXITE ACE as shown in Fig. 4.

HetSC for Architecture Exploration

Current embedded systems are composed of an increasing number of components in order to implement a huge amount of different functionalities covering a wide variety of behavioural semantics at different levels of abstraction. In this context, HetSC was defined as a system specification methodology for concurrent HETerogeneous embedded systems in SystemC [7][6]. HetSC makes a clear separation between the computation and the communication aspects of the system,
supporting the creation of formal executable specifications of the system since HetSC is based on the ForSyDe formal metamodel [10]. Additionally, HetSC provides mechanisms required by specific model of computations (MoCs) what enables the creation of heterogeneous system specifications; each different component that composed the system can be described under different MoCs. The support of heterogeneity facilitates software generation and hardware synthesis since each system components are syntactically and/or semantically closer to the platform entities. Therefore, the application of HetSC in addition to validating the system's behavior provides a link to system implementation [7].

The HetSC methodology is covered by the SATURN methodology in early design stages where the designer identifies the different concurrent entities that composed the system. Once the functional architecture has been decided, the designer can explore different behavioral semantics in order to characterize the system. As such, HetSC supports different approaches to build untimed concurrent models, which significantly reduce simulation time and specification complexity. Additionally, HetSC supports fast and automatic generation of embedded software implementations over different embedded RTOS-based platforms through SWGen [7].

For SystemC model refinement in Artisan Studio®, an additional UML profile for HetSC has been defined. The stereotypes of this profile represent a set of HetSC channels that implements the semantics of different Untimed MoCs. In untimed systems, there is only a partial order and a causality relation among events. In addition, the computation and the communication take an arbitrary and unknown amount of time. The application of the MoCs on the creation of the system specification guarantees specific properties that determine the system behavior, e.g., determinism, protection against deadlocks.

In order to represent processes networks (PNs), HetSC provides two kinds of channels. \texttt{uc_inf_fifo} represents a channel with an unlimited buffering capacity whereas the channel \texttt{uc_fifo} has limited this capacity. The first one implements the semantics of the Kahn Process Network MoC and the second one the Bounded Kahn Process Network MoC. In order to create systems with the semantics of the Synchronous Data Flow (SDF) MoC, HetSC provides the channel \texttt{uc_arc_seq}. Finally, it covers three different rendezvous channels: \texttt{uc_rv_sync}, \texttt{uc_rv_uni} and \texttt{uc_rv} for the Communicating Sequential Process (CSP) MoC. They specify three different cases of communication. The first one implements the synchronization only between asynchronous concurrent elements. The second one implements synchronization with unidirectional data communication and the last one synchronization with bidirectional data transfer.
Evaluation

The SATURN approach was evaluated in two case studies by two industrial partners:

- An IEEE 802.16e base station for broadband wireless communications undertaken by INTRACOM Telecom
- Smart Camera for Automatic License Plate Recognition (ALPR) undertaken by Thales Security Solutions and Services

IEEE 802.16e Base Station

For the evaluation of SATURN we have selected the last part of the base station Tx PHY chain of an IEEE 802.16e system [9]. This consists of the IFFT, the Cyclic Prefix and the Preamble Insertion blocks. The first was mapped to HW and the last two ones to SW.

As a target platform, the Virtex 5 ML510 [21] development board was used, which consists of a dual PowerPC and Xilinx FPGA Virtex-5 XC5VFX130T. Modeling of the system was initially done in Artisan Studio®. The approach that we followed was a top-down modeling of the system:

We started with creating the Block Definition Diagram (BDD) of the system. The top-level block contained the blocks PPC (software part) and FPGA (hardware part). The FPGA block included:

1) The FFT block, which carried out the main bulk of the computations for the transformation as well as the direct communication from/to the Block RAM allocated for this purpose.
2) The Block RAM to transfer data from/to the hardware as well as for the twiddle coefficients used in FFT and various other configuration parameters such as the size of the FFT and the mode (inverse FFT / direct FFT).
3) The PLB (Processor Local Bus) with the respective components (plb_cpu_transactor, plb_ctrl_status_transactor) which handles the communication from software to the BRAM through the BRAM.

In a second step, we designed the Internal Block Diagram (IBD) of the top-level block which included HW and SW parts and their communication. Thereafter, the FPGA’s IBD was created. The subsystem was composed of:

a) The FFT Peripheral.
b) The dual port Block RAM.
c) The PLB bus and the three transactors for the Block RAM, the FFT and the PowerPC
d) The global clock Clk of the system.
Blocks a), b) and the CPU interface were connected on the PLB through PLB Transactors. The fact that the FFT and the PLB were connected to the two ports of the BRAM allowed us to have a unified memory map with the BRAM address space visible to the CPU as well as access to the BRAM directly from our FFT peripheral. The FFT’s two control registers `ctrl` and `status` were also mapped on the memory map of the CPU so they are accessible from software.

In the next step, we created the FFT activity diagram using the SATURN profile. We have used an SC_METHOD sensitive to the `clock` and to the `ctrl` register to model the FFT peripheral.

After modeling the system and the hardware processes, we generated the SystemC code for the FFT peripheral through ACS/TKD in Artisan Studio®. Subsequently, VHDL for the FFT was generated from the respective SystemC code through the Agility Compiler.

The code generation was used to produce SystemC simulation traces which enabled us to have an early testing environment. We wrote self-contained test benches in SystemC where we applied ACS/TKD to generate the remaining environment. Thereafter, we ran extensive simulations to make sure that our FFT peripheral worked correctly in all modes. We compared test vectors taken from the real system with the output of the SystemC simulation. In a second step of simulation we used the VHDL which was generated from Agility with manually written test benches in VHDL to ensure the correctness of the SystemC to VHDL translation. After importing the FFT we were able to synthesize and place & route the design through XST and ISE. After tuning of the timing constraints to cater for multi-cycle paths created by Agility, the design was downloaded to the Xilinx Virtex-5 ML510.

On the software side we implemented the state machine in C to control the hardware. Currently, the software application code is written outside Artisan Studio and co-simulation was carried out through EXITE ACE. The SATURN profile generated the drivers to access the registers and the BRAM from the software. The drivers relied on Linux operating system and used the `/dev/mem` node (which represented the whole memory of the system) and `mmap` in order to transfer data from/to the memory mapped BRAM and FFT registers.

Therefore, we had to build a Linux kernel image for the PPC 440 in order to bring-up the system and execute our user space application with the SATURN generated drivers. In order to create a kernel image we needed:

1. A DTS file which contains the memory map for the system that we had created through EDK BSB. In order to generate the DTS file we used the `device tree generator` addon of EDK.
2. A file system containing the user space application built for the PPC440.
3. The standard Linux kernel. After the generation of the file system and the DTS file we were ready to build the Linux kernel.
We finally downloaded the Linux image to the PowerPC and the bit file to the FPGA. After booting Linux we could run the user space application and we could verify that data were provided from SW to the FPGA and correctly passed through the FFT. They were then transferred back to SW and processed through the Cycle Prefix and the Preamble Insertion running on the PPC.

**Smart Camera**

In a second case study, we have applied the SATURN tool suite and methodology to design FPGA blocks for an Automatic License Plate Recognition (ALPR) system which is an image-processing technology for license plate identification. This technology is used in various security and traffic applications, such as in the access control system for car parks as given in the following figure.

![Figure 4. License Plate Recognition for Airport Parking](image)

The system uses illumination such as Infra-red and a camera to take the image of the front or rear of the vehicle, then image-processing software analyses the images and extracts the plate information. This data is used for enforcement, data collection, and can be used to open a gate if the car is authorized or keep a time record on the entry or exit for automatic payment calculations.

The license plate detection is mainly a sequence of filters designed to detect regions with a high density of small elements which is characteristic of text. For our studies in the context of SysML, we focused on the implementation of morphological operators which are heavily used in the plate detection stage of the ALPR algorithms covering the image acquisition from a video bus, automatic image contrast correction, automatic exposure time. They are designed as a pixel
pipeline which makes computation on the fly and loops back to the camera control.

Like the previous case study, this case study also followed the lines of the SATURN design starting with the SysML tool suite for modeling and cosimulation after code generation. The co-simulation is used for the communication between the main processor and FPGA. In contrast to the previous application, the smart camera is an embedded system composed of a complete board equipped with a PowerPC or ARM and a separate FPGA for the image processing which has some dedicated external memory controlled by the main processor. The Virtex 5 FPGA was configured by Xilinx ISE and the Linux image was separately loaded to the PowerPC.

Conclusions

In this chapter, we presented the SATURN approach to SysML-based HW/SW co-modeling, -simulation, and -synthesis by means of Artisan Studio®. The two case studies demonstrated the applicability for industrial designs. They also indicated a few subjects for improvements which are planned to be resolved until the end of the SATURN project. We applied Artisan Studio® with the SATURN profiles in replacement of a traditional C++ IDE to write SystemC code. The module composition is defined as a Block Definition Diagram. However, in the first step the definition of members and methods was a little bit slower compared to a traditional IDE as it requires going through a lot of windows pop-ups and tabs. Then, Internal Block Diagrams was used to define the connections of the module's ports. They allow drawing the wiring visually, which is an improvement over the normal coding of connections since it is faster and leads to less errors in connecting ports. Once the system's structure was defined, the behavior and logic had to be coded as usual in a textual editor. Here for efficient capturing, an adequate support of user friendliness, refactoring abilities and code parsing is mandatory for an UML/SysML editor in order to compete with current C++ IDEs. The previous benefit of the graphical wiring is mitigated by the time lost for the definition of the behavior without this support as, from our experience, the complexity of behavior and signal timings is higher than structure definition. When iterating through the verification process, most of the changes affect the behavioral logic whereas the module's structures are more stable. The graphical modeling suffers as usual from its lack of compactness and precision compared to textual languages. Although it gave a comprehensive overview of the network-like relations, it partly lacks semantics. One has to explore the graphical editor tabs associated to any item in order to exactly know how it is defined. In contrast, a textual language is quite more synthetic and gives all details. Another important point is that, when VHDL is not directly generated, the SystemC based approach requires an additional SystemC to VHDL compiler. Thus, a modeling environment with integrated SystemC and VHDL code generation would be appreciated to support the existing SystemC-VHDL tool chain.
The major benefit of the SATURN environment is the ability to simulate both
the SystemC modules and the target operating system of the embedded platform.
This is due to the extension of QEMU to exchange data with the SystemC mod-
ules. It shortens the verification time as it allows debugging hardware software in-
terfaces faster than usual: it can be tested by the emulator. Additionally, it allows
to easily moving between the different hardware platform and operating system
from within the modeling environment to explore different HW/SW alternatives.

In the UML/SysML based approach, a real collateral benefit is having ready to
use diagrams of the modules for the documentation. It usually requires drawing
them with a generic drawing tool or an UML editor which is not adequately inte-
grated into the design flow. With SATURN tools this time is moved from the
specification or post-design phase to the design phase. However, the time saving
compared to classical drawing tools depends on the user friendliness and support
of the individual UML editor, on the number of times a design is changed, and on
how many diagrams are really needed for the documentation. Regardless to this,
the main gain here is having always up-to-date and precisely defined diagrams in
the documentation.

Acknowledgments The work described herein is supported by the ICT Pro-
ject SATURN (FP7-216807).

References

with UML MARTE. In: Proc. DATE Workshop on Modeling and Analysis of Real-
Time and Embedded Systems with the MARTE UML profile, 2008.
Approach for Automatic SPIRIT IP Integration. In: Proc. 5th Int. UML-SoC DAC
Workshop, Anaheim, USA, 2008.
2006.
eling Seamless Refinement, and Synthesis of a JPEG 2000 Decoder. In Proc. of
DATE’08. April 2008.
ferent Models of Computation in SystemC, Annual ACM IEEE Design Automation
Conference Proceedings of the 43rd annual conference on Design automation. 2006
Electronic Embedded Systems in SystemC.
ACM Transactions on Design Automation of Electronic Systems, Special Issue on
Demonstrable Software Systems and Hardware Platforms, V.12, Issue 3, N.22. 2007-
08.


