Code Generation for QEMU-SystemC Cosimulation from SysML

Da He, Fabian Mischkalla, Wolfgang Mueller

University of Paderborn/C-Lab, Fuerstenallee 11,
33102 Paderborn, Germany
{dahe, fabianm, wolfgang}@c-lab.de

Abstract. With the continuously increasing complexity of modern electronic systems, the engineers are facing more and more challenges (e.g., time-to-market) in the hardware and software co-development. In this article we present a SysML-based modeling approach for HW/SW combined systems and a code generation scheme to automatically provide the QEMU-SystemC based virtual simulation environment.

1 Introduction

Due to the increasing complexity of electronic systems Model Driven Engineering (MDE) based design techniques like Systems Modeling Language (SysML [6]) are becoming more and more interesting for combined system documentation and early testing. However, there is still a big gap from abstract UML specifications to executable models, i.e. virtual prototypes, for dynamic verification and synthesis.
In [1], [2], [3], [4] and [5], we developed an efficient HW/SW codesign approach based on SysML by covering the design flow (shown in the Figure 1) from modeling to implementation via retargetable code generation. For modeling, we defined additional UML based extensions on top of SysML, which support: (i) target SW integration, (ii) convenient SystemC modeling, and (iii) SystemC based high-level synthesis.

This paper focuses on code generation for the automatic configuration of the HW/SW cosimulation infrastructure composed of QEMU and SystemC. We implemented the retargetable code generation scheme in ARTiSAN Studio® (a UML/SysML development suite for model engineering) to generate C/C++ code for target SW binaries, synthesizable SystemC for platform components like interconnects, memories, and HW accelerators as well as other utilities such as self-managed makefiles and scripts for complete design flow automation. We address QEMU in its full system mode as a CPU emulator for different processor architectures running the target SW binaries along with an operating system.

2 HW/SW Codesign and Code Generation

Our HW/SW codesign methodology focuses on system architecture modeling in terms of component-oriented view. For this, we use SysML for structural modeling. As such we are able to define individual system components, their interfaces, and interconnections. The entire system model can be specified by SysML front-end editors like ARTiSAN Studio®. Based on our custom annotations to SysML, we map SystemC execution semantic to the specification model. This 1-to-1 relationship provides efficient and automated code generation as well as round-trip capabilities by common UML tools. In our codesign methodology, a specification model consists of three different types of components: (i) software executables, (ii) processors, (iii) platform specific interconnect and hardware components.

Figure 2 gives an overview of the code generation for the automatic configuration of a QEMU-SystemC cosimulation environment. Each HW component is translated into a SystemC module. Such components can be either predefined components from external libraries or explicitly modeled by the user. In the first case, legacy SystemC code is referenced from libraries, such as for TLM Bus and TLM Transactors as shown in Figure 3. For the latter, a new SystemC module with a header and an implementation file is generated. For each SW component referencing external source the code generator generates a register address map extracted from the model, user space driver functions and makefiles to cross-compile it to a target SW binary. The register address map is generated as C macros in a header file containing the definition of base address, high address of the device as well as offsets of each internal registers. The user space driver functions provides software developers easy access to the hardware devices via generic byte/half-word/word read as well as write operations. A target processor platform based on a dedicated ISA (Instruction Set Architecture) and an OS (Operating System) image is specified by processor models that are allocated from these binaries. Finally, automatically generated shell scripts compile this bundle to an executable image including OS and application software binaries.
A processor component communicates over so-called TLM interconnects with other system components. For this communication, a QEMU Plugin (cf. Figure 3) containing all accessible device declarations via the QEMU standard API call (cpu_register_io_memory) is automatically generated. The QEMU Plugin is implemented in form of a runtime library and is loaded into the QEMU core dynamically. As the QEMU plugin is based on the information about the modeled system architecture in terms of devices and their register address map, it may differ from design to design. In order to automate this configuration, we developed an ARTiSAN Studio® code generator to extract the information from the specification model and to generate the plugin. The automatic code generation significantly reduces the configuration effort for setting up the entire QEMU-SystemC based cosimulation environment.

3 QEMU-SystemC based Cosimulation

With code generation, the complete cosimulation environment is automatically configured and can be launched by a simple button click. Due to the support of multiple target architectures by QEMU, we can easily select them in the specification model. Currently, Linux and Android are supported for PowerPC405 as well as ARM926EJ-S in form of OS library elements.

The diagram in Figure 3 outlines the detailed infrastructure of our cosimulation framework. The QEMU emulator and SystemC simulator are running as two separate processes which are synchronized by means of inter-process communication by shared memory. On the SystemC side the TLM Transactor (on top of the TLM Bus in
Figure 3) transforms data from the shared memory to TLM transactions, so that they can be forwarded by the TLM bus and processed further by other individual SystemC modules. On the QEMU side, we use the aforementioned QEMU plugin to connect the QEMU core to the SystemC process.

Our cosimulation framework is based on an asynchronous communication paradigm, which means that the SystemC simulator and QEMU emulator run concurrently until SW applications from QEMU invokes an I/O operation (read or write operation), which exchange data between both simulators by blocking calls. After the I/O access is finished, QEMU and SystemC resume concurrent execution until the next I/O operation. During cosimulation the QEMU emulator acts as a master that can initiate I/O operations while the SystemC part only reacts as a slave. The synchronization between QEMU and SystemC must be currently ensured by the application design itself as the applied QEMU is untimed.

Figure 4 shows more details of the sequential flow of read and write I/O operations. Each I/O operation in the SW application invokes a corresponding callback function that is implemented in the QEMU Plugin. The callback functions then send the data to the TLM Transactor of the SystemC process and wait for the acknowledgement/data via blocking receive calls. While the QEMU is waiting, the SystemC continues processing the transaction and returns the acknowledgement/data thereafter. In case of read operation, the data are simply retrieved via plugin and transactor from SystemC.

4 Conclusion

In this paper, we introduced a flexible MDE based HW/SW codesign approach to close the gap between comodeling, cosimulation, and implementation via automatic code generation. The code generation includes C/C++ for target SW, synthesizable SystemC for HW components and self-managed scripts for design flow automation. The code generation framework automatically configures the QEMU-SystemC cosimulation framework. The implemented tool environment was successfully validated in the SATURN project by several partners who applied it for several industrial case studies [3].

References

5. EU Project: SysML bAsed modeling, architecTUre exploRation, simulation and syN-thesis for complex embedded systems (SATURN), www.saturn-fp7.eu