Hardware-software co-simulation for medical X-ray control units

Bruno Kleinert
Universität Erlangen-Nürnberg
Martensstraße 3
91058 Erlangen, Germany
bruno.kleinert@cs.fau.de

Gholam Reza Rahimi
Universität Erlangen-Nürnberg
Martensstraße 3
91058 Erlangen, Germany
rahimi.r.g@gmail.com

Marc Reichenbach
Universität Erlangen-Nürnberg
Martensstraße 3
91058 Erlangen, Germany
marc.reichenbach@cs.fau.de

Dietmar Fey
Universität Erlangen-Nürnberg
Martensstraße 3
91058 Erlangen, Germany
dietmar.fey@cs.fau.de

ABSTRACT

In this paper we present our solution to master the complexity of product adaption cycles of a medical X-ray control unit. We present the real hardware and software platform and our mapping of it to a virtual X-ray control unit, implemented as our hardware-software co-simulation. To reduce complexity for hardware developers, we developed our own XML-based abstract system description language which is mapped onto instantiations of parameterizable SystemC template modules. We verified the correctness of our virtual X-ray control unit by co-simulating unmodified software to hardware components, which we implemented in our system description language from the specification of the real system. Due to reduced complexity of our virtual X-ray control unit, it can be used as a time and cost saving test platform for future hardware and software adaption cycles.

Keywords

Hardware-software co-simulation, virtual machines, QEMU, SystemC

1. INTRODUCTION

In this paper we present a simulation environment to master complexity of adaption cycles of an X-ray control unit used in a medical device. The current state of the art of our established industry partner to develop such devices happens with little computer-assistance. Boards, wired interconnections, sequences and logical functions of software and hardware are frequently available only as drawings.

In this paper we present our simulation of the control unit with the virtual machine [2] QEMU [1] and the hardware simulator SystemC [4], which we connected to each other. We managed to lower complexity by developing our own abstract XML-based system description language (SDL) that is interpreted by our SystemC-based hardware simulator.

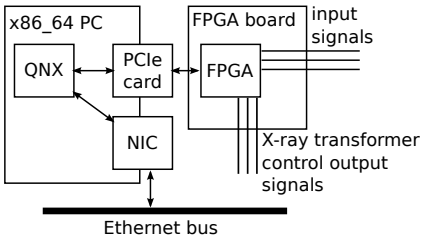

Figure 1: X-ray control unit architecture overview.

The resulting hardware-software co-simulation executes unmodified software from our industry partner. We implemented functionality of the hardware in our own SDL and co-simulated it to our extended QEMU.

This paper is organized as follows. In Section 2 we present the real X-ray control unit. Section 3 presents our simulation environment we developed as a virtual copy of the real control unit. The correctly working simulation of the virtual control unit is presented in Section 4, while related projects and works are discussed in Section 5. Section 6 concludes this paper and Section 7 presents future improvements to the current state of our work.

2. SYSTEM OVERVIEW

In this section we present the current state of the real X-ray control unit, built from an x86_64 PC and an FPGA board.

Components of the control unit are depicted in Figure 1. The PC executes application software. An Ethernet network interface card (NIC) connects the PC to other system components. The FPGA board is connected to the PC via PCI Express.

2.1 Software component

SIMUTOOLS 2015, August 24-26, Athens, Greece

DOI 10.4108/eai.24-8-2015.2261148

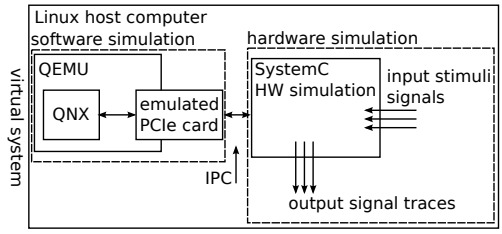

Figure 2: Virtual control unit.

As shown in Figure 1, application software can use the NIC and the connection to the FPGA via QNX device drivers. Via the Ethernet connection, control software transmits data and control commands. For development purposes, a telnet connection can be established to a QNX command line interface. We use this interface and a test program that allows read and write accesses to registers in the FPGA (see following Section 2.2 for details).

The test program communicates with the FPGA and hence allows to verify a correctly operating connection and operation of the FPGA. This is also applicable to our virtual control unit.

2.2 Hardware component

The shown PCI Express card serves as an adapter for a cable that connects the FPGA to a PCI Express card slot on the mainboard of the PC. From the PC it is accessed as a memory mapped device.

The FPGA is programmed to process safety signals and generates control signals for the transformer of the X-ray source. Hundreds of simple finite state machines (FSM) and logic processing gates build a complex system.

Memory mapped registers serve for purposes to obtain system status information or to send control data. New data for the PC is signaled via MSI-X interrupt requests (IRQ) or alternatively via pin-based routed IRQ. An interrupt service routine (ISR) in QNX consumes the newly available data. When the ISR has finished, the read data is typically made available to application software through system call interfaces of QNX.

3. SIMULATION ENVIRONMENT

As mentioned in Section 1, a challenge in our work is the hardware-software co-simulation of the software and hardware components, as presented in Section 2. In the following Sections, we present our SystemC-based hardware simulator and our hardware-software co-simulation.

Figure 2 presents our virtual control unit. The QEMU and SystemC Linux processes running on a Linux host computer are shown. QEMU and SystemC exchange data via an emulated PCI Express connection, implemented using inter process communication (IPC) techniques. Software and hardware simulation are separately highlighted by dashed boxes.

Our XML-based SDL and virtual control unit are presented in the following Sections.

3.1 System description language

Our SDL allows the use from primitive up to complex building blocks, i.e., from binary logic gates up to complete state machine blocks. The language is designed to be extensible and puts no upper limit on the complexity of building blocks. All building blocks form a library. From this library, developers use elements in their system descriptions. In our concrete case, these are the basic FSMs and logic components, that are used to implement the complex signal processing network of the real X-ray control unit.

For hardware simulations, input stimuli patterns are necessary. For our simulator, stimuli patterns are also described in an XML-based language.

3.2 Hardware simulation

We developed a hardware simulator, that is able to interpret our SDL as input and instantiates parameterizable SystemC template modules with parameters extracted from system descriptions.

We map elements in our SDL to parameterizable SystemC modules. To use a building block in our SDL, an equivalent parameterizable SystemC template module has to exist. All these template modules build a library of available mappable modules in our SDL. The mapping and instantiation is done during the start of our simulator when the XML tree of a system description is traversed.

The design of our SDL and hardware simulator makes time-consuming re-compilations of SystemC unnecessary, as the parameterizable SystemC template modules are compiled together with our simulator.

3.3 Hardware-software co-simulation

We extended QEMU by an emulation of the real PCI Express card. The FPGA is connected by PCI Express to the PC via memory mapped registers.

Figure 3 shows our implementation of the coupling between QEMU and SystemC. Between QEMU and SystemC Linux processes, data to read, write or to signal an IRQ is exchanged by three POSIX named pipes. One to transfer a datum from QEMU to SystemC, one to read a datum from SystemC, and one to signal an IRQ from SystemC to QEMU. Newly available data is signaled via POSIX signals to the according process. A POSIX signal handler in a receiving process is executed to read new data from its input named pipe. The information transmitted through the write and read named pipes is a register address and the corresponding data, while the IRQ number is sent through the IRQ named pipe.

When a register in the SystemC process changes, the simulated hardware sends the new datum through the read named pipe and triggers an IRQ by writing the IRQ number to the IRQ named pipe and send a POSIX signal to the QEMU process. If there is an ISR installed in the guest operating system, it will be executed to handle the IRQ.

In contrast to QEMU-SystemC [7], our lightweight extension of QEMU allows the use of kernel-based virtual machine

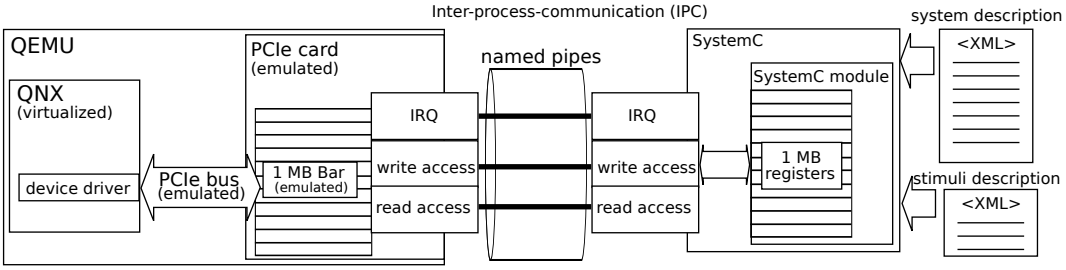

Figure 3: Overview of our hardware-software co-simulation.

(KVM) [3], which results in increased execution performance of guest systems.

4. RESULTS

We could successfully execute unmodified test software from our industry partner to write and read arbitrary registers of the emulated PCI Express card.

We verified our hardware-software co-simulation by a counter state machine in our SDL. One register to start the counter, one to stop it and one to read the current value of it.

5. RELATED WORK

FAUmachine [5] is targeted at fault tolerance testing of guest software [6]. It can not utilize virtualization performance enhancements of CPUs at the time of this writing, while our interests include also high performance. The integrated VHDL simulator does not meet the requirements of the hardware we want to simulate at the time of this writing.

Another existing hardware-software co-simulator is QEMU-SystemC [7]. We tried to use QEMU-SystemC but it terminated unexpectedly in several scenarios, required a particular outdated version of the GCC compiler suite to work as expected. Attempts to make it work for our needs turned out to be more time consuming than implementing our own solution. Furthermore, QEMU-SystemC terminates unexpectedly when KVM for performance increases is enabled.

6. CONCLUSION

We presented our contribution to master the complexity of product adaption cycles of a medical X-ray control unit. We presented the real control unit that is built from two components, an x86_64 PC to which an FPGA board is connected. While the PC executes application software, the FPGA processes safety input signals and controls the transformer of the X-ray source.

For future development of this control unit, we developed our virtualized control unit. We mapped logical functions, implemented for the FPGA, to our hardware simulator based on SystemC. We mapped QNX and application software unmodified to the virtual machine QEMU. To speed up guest system execution, we configured QEMU to utilize the Kernel Virtual Machine of Linux, which was not applicable to other co-simulators. We extended QEMU to exchange data with SystemC via inter process communication.

We developed our own abstract system description language, interpreted by our hardware simulator based on SystemC. To simulate systems, our language is mapped to pre-compiled parameterizable SystemC template modules.

We verified the correctness of our hardware-software co-simulation by using software from our industry partner. It proofed that our connection between QEMU and SystemC works like in the real X-ray control unit.

By our simulation of the X-ray control unit, less lines of code need to be written to model a hardware system on an abstract level in comparison to a SystemC model. Software and hardware can be tested with our virtual control unit.

7. FUTURE WORK

In the current state of our hardware-software co-simulation, the simulated times of QEMU and are not synchronized thoroughly. In future, we want to develop an adaptive interface to keep simulated times synchronized dynamically.

8. REFERENCES

[1] F. Bellard. QEMU, a Fast and Portable Dynamic Translator. In *USENIX Annual Technical Conference, FREENIX Track*, pages 41–46, 2005.

[2] R. P. Goldberg. Survey of virtual machine research. *Computer*, 7(6):34–45, 1974.

[3] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. kvm: the Linux virtual machine monitor. In *Proceedings of the Linux Symposium*, volume 1, pages 225–230, 2007.

[4] P. R. Panda. SystemC - a modeling platform supporting multiple design abstractions. In *System Synthesis, 2001. Proceedings. The 14th International Symposium on*, pages 75–80, 2001.

[5] S. Potyra. *Transparente und hochperformante VHDL-Cosimulation im Kontext der virtuellen Maschine FAUmachine*. PhD thesis, Universitätsbibliothek der Universität Erlangen-Nürnberg, 2013.

[6] S. Potyra, V. Sieh, and M. D. Cin. Evaluating fault-tolerant system designs using FAUmachine. In *Proceedings of the 2007 workshop on Engineering fault tolerant systems*, page 9, 2007.

[7] T.-C. Yeh, G.-F. Tseng, and M.-C. Chiang. A fast cycle-accurate instruction set simulator based on QEMU and SystemC for SoC development. In *MELECON 2010 - 2010 15th IEEE Mediterranean Electrotechnical Conference*, pages 1033–1038, April 2010.