Development of embedded PC and FPGA based systems with virtual hardware

Wojciech M. Zabołotny[a]

[a]Institute of Electronic Systems, Warsaw University of Technology, ul. Nowowiejska 15/19, 00-665 Warszawa, Poland

ABSTRACT

This paper discusses a methodology available to develop digital systems based on both: embedded computer and tightly coupled FPGA, using the virtual hardware. This approach allows to test design concepts and even to develop some parts of firmware and software before the real hardware is built, or to allow multiple developers to work simultaneously when only limited number of prototype devices is available.

The aim of this paper is to show different available methods, providing the hardware-software co-simulation for development of such digital systems, with emphasis on the open source solutions, and to discuss their applicability.

Keywords: FPGA, Embedded PC, virtual hardware, hardware-software co-simulation

1. INTRODUCTION

Simulations are widely used today to support development and debugging of electronic systems. Typical simulators like QEMU,[1] or Virtual Box[2] (I'll call them the "software level simulators" or shortly "SW simulators") can emulate the target platform providing instruction-accurate simulation of the software running on it. Therefore they allow to test the user software or even the components of the operating system. The SW simulators are optimized for simulation of software, even though they are able to provide functional simulation of the hardware creating the emulated platform.

The opposite side of the development activity is the simulation of user designed digital hardware. Such hardware is usually described with hardware description languages (HDL) like Verilog or VHDL, and can be simulated with "hardware level simulators" (I'll call them shortly "HW simulators"). Those simulators are optimized for precise simulation of hardware components. For example the simulators like GHDL,[3] or Icarus Verilog[4] (to name two open source simulators) or ModelSim[5] (the commercial one), are able to provide the "cycle accurate" Register Transfer level (RTL) simulation of the hardware.

Contemporary electronic systems are built usually as tightly coupled: user designed specialized hardware, the embedded system used to control that hardware and to process and retransmit the acquired data, and software running on that embedded computer. Testing of all listed parts separately is difficult, or even impossible. Without possibility to perform the joint simulation of both - hardware and software components, the development becomes expensive and time consuming.

In typical development cycle we have to design the hardware prototype, basing on some assumptions regarding possible software solution. The work on software part may be started when the hardware specifications are ready, but thorough testing of the interactions between the hardware and software parts must be delayed, until the hardware prototype is ready.

If the results of tests show the need for significant changes in hardware design, it is necessary to prepare next prototype and subject it to tests. Sometimes the above step must be repeated a few times, depending on the design complexity, and on skills of the development team.

One possible solution of the above problem would be to prepare the versatile hardware with reach set of functions and communication interfaces, containing the big FPGA chip tightly connected with the embedded PC. Such versatile platform could be used as "the first prototype" implementing both hardware and software components of the designed system. After successful testing, the second prototype reduced to really utilized functions and interfaces may be built.

Further author information: (Send correspondence to W.M.Z.)
W.M.Z.: E-mail: wzab@ise.pw.edu.pl, Telephone: +48 22 234 7717

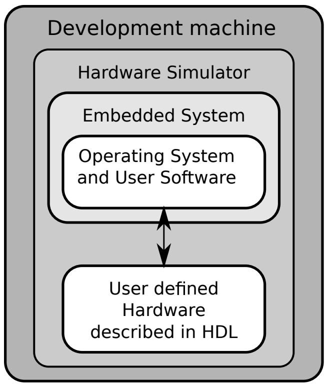

Development machine
Hardware Simulator
Embedded System
Operating System and User Software
User defined Hardware described in HDL

Figure 1. Simulation of the system using only the hardware simulator.

In fact such approach is often used, when vendor manufactured evaluation kits (for example see Ref. 6) are used for initial development, and the final hardware still significantly mimics the architecture of the development board.

However such approach has also significant disadvantages. The platform, versatile enough to be used as prototype for complex systems, will be expensive. Additionally in contemporary, fast changing electronic market the life of such platform can be quite short. When a new family of FPGA chips dominates the market, the usefulness of the platform based on the old family decreases.

Therefore approach based on joint simulation of software and hardware components of the designed system - i.e. hardware-software co-simulation is needed. In this paper I'll describe available solutions, with emphasis on open source tools, and evaluate their applicability.

2. USE OF HW SIMULATOR FOR THE WHOLE SYSTEM

The simplest, but naive approach would be to use the HW simulator to simulate the whole system - including the embedded system and the newly developed hardware (see Fig. 1).

Generally it can be done, but the performance of such simulation is usually very poor, as the hardware level simulators are not suited for simulation of such complex systems as embedded PC. The RTL simulation analyzes the state of all logic gates and registers in the simulated system, which is not needed to provide required accuracy of SW simulation.

In Ref. 7 there is described a simulation of OpenRisc CPU running very simple C program. Even displaying of the simple welcome message takes 40 seconds of simulation time, so simulation of any serious OS running on such platform would be too slow to be useful.

Such approach can be probably used for systems containing FPGA chips connected to simple microcontrollers, but it is useless to investigate systems containing bigger embedded computers, running under control of the true Operating System.

Maybe the situation could be improved by providing the higher (not RTL) level model of the CPU and the embedded system, but this would require further investigations.

The real solution would be to avoid simulation of the embedded PC and the software in the HW simulator.

In some scenarios it is possible to run the software part of the system on the development machine itself, as it will be described in the next section.

3. HW SIMULATOR WORKING WITH SOFTWARE RUNNING ON HOST MACHINE

In this approach, we use the hardware level simulator to emulate the user designed hardware, while the user software is running directly on the development machine (see Fig. 2).

The simulation environment is created by substitution of original libraries, providing access to the hardware in final system, with special libraries implementing the same API from the user side, but communicating with the HW simulator emulating the user designed hardware.

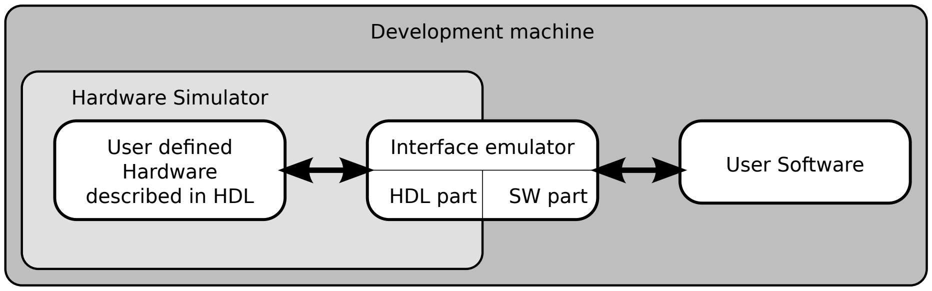
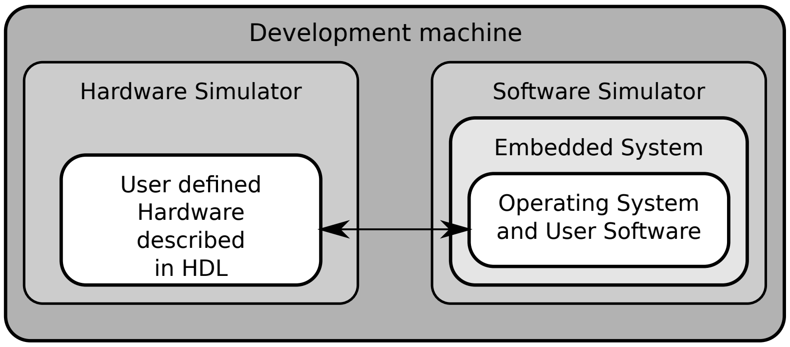


Figure 2. Simulation of the system using the user software running on host machine, and user hardware simulated by hardware simulator, connected via emulated bus.



Development machine
Hardware Simulator
User defined Hardware described in HDL
Software Simulator
Embedded System
Operating System and User Software

Figure 3. Co-simulation using two simulators: HW simulator and SW simulator.

I've used successfully used this approach when developing the firmware for Data Contrator Card (DCC) board for RPC trigger in CMS experiment in LHC.[8] The software used to control and test the DCC board during development was written in Python. It could communicate either with the real VME interface or with special module, providing access to the emulated local bus in hardware simulated with GHDL[3] simulator.

The module emulating the accesses to local bus provided functions allowing to perform the read access, the write access, and to advance the simulation time in the simulator for user defined time. The module was published as open source in the Usenet.[9] The method appeared to be very useful, however it may be applied only to relatively simple software, as it does not allow to emulate behavior of the whole embedded system cooperating with our hardware (e.g. the behavior of device drivers, performance of data transfers and so on). To extend possibilities of such testing, I have also prepared another module,[10] allowing to connect the VHDL described system, simulated in GHDL, to the emulated serial port (the ptmx device) in the Linux based development machine.

In most cases however the development includes testing of both - hardware and low level components of the Operating System like device drivers, and then instead of running the user software directly on the development host, we need to combine two simulators.

4. CO-SIMULATION USING BOTH HARDWARE LEVEL AND SOFTWARE LEVEL SIMULATORS

Let's assume, the we have an open source software simulator, able to simulate our target embedded PC in real time (e.g. QEMU) and open source hardware simulator, able to simulate the hardware described in HDL (e.g. GHDL). It seems, that we should be able modify them to work in parallel, providing both: fast simulation of the software components working on our target platform - provided by the SW simulator, and cycle accurate simulation of the user-defined hardware - provided by the HW simulator (see Fig. 3).

It seems, that simple extension of the concept described in section 3 should be sufficient. Unfortunately the problem is more complex, as the synchronization of both simulators is not easy.

When the only interaction, between the part simulated in the SW simulator and the user defined hardware, is reading of the registers or memories and writing to them, the synchronization is relatively simple. We can run the HW simulator only when the SW simulator accesses the user defined hardware, continuing the simulation, until its simulated time (t_{HW}) is equal to the simulated time in SW simulator (t_{SW}). Unfortunately such simple model fails, when the user designed hardware is connected to the external world, and the contents of the HW registers may change due to the event which is not generated by the SW simulator. If $t_{SW} > t_{HW}$, the SW simulator will see those changes with delay. In case of interrupts it will lead to increased interrupt latency, but in some scenarios it may give even disastrous results (e.g. when the program flow depends on the content of a HW register in particular moment of simulated time). The only solution would be to keep both simulators synchronized, but this would dramatically impair performance of the simulations. The problem of synchronization was discussed in many publications and conference presentations[11–15] and different solutions are proposed, but solving of this problem for Register Transfer Level (RTL) simulations seems to be impossible. Therefore the RTL simulations are too slow when combined with the software level simulations, and the only viable approach is to use the Transaction Level Modeling[16] (TLM) instead.

There are some solutions, based on this concept, allowing to simulate the hardware described on TLM level in SystemC language with open source QEMU - QEMU-SystemC[17] (which is a part of a bigger GreenSocs[18] project) or with proprietary simulator in Open Virtual Platform.[19]

5. USE OF SYSTEMC FOR HARDWARE SIMULATION AND DEVELOPMENT

The SystemC[20] language is often used for modeling and verification of the digital electronic systems. However its wide use for synthesis of digital systems in FPGA is limited by the fact that most synthesis tools provided by the FPGA vendors do not support SystemC synthesis “out of the box”. It is necessary to buy additional tools like AutoESL,[21] SystemCrafter[22] or Catapult C[23] at relatively high price.

The alternative could be to generate the SystemC model of the hardware from the HDL description used for synthesis. In fact there are tools[24] aimed on conversion of the Verilog or VHDL sources into SystemC, but it is almost impossible to create to create the TLM model suitable for simulation from the RTL sources prepared for synthesis.

Another possibility could be generation of the HDL code suitable for synthesis from the TLM model used for simulation and verification. Unfortunately the open source tools which are freely available[25,26] are able to convert only the RTL style SystemC sources. To generate the RTL code from TLM SystemC model we need one of the previously mentioned commercial tools.[21–23]

6. MODELING OF USER DESIGNED HARDWARE AS A STANDARD QEMU DEVICE

If we are not going to use the commercial SystemC synthesizers, we need to work with two parallelly maintained descriptions of our hardware - the first one, implemented in HDL, for synthesis, and the second one, implemented in SystemC, for hardware-software co-simulation.

However, if we accept inconvenience of having two, separately maintained descriptions of our hardware, we can go even further, and try to implement the model of our device in QEMU itself (see Fig. 4).

The QEMU simulator offers models of different devices used in multiple emulated hardware platforms. The hardware devices needed to simulate those platforms are implemented in C++ language in the framework of QEMU.

Unfortunately QEMU documentation[27] is rather sparse, and to understand the API used to model hardware, it is necessary to analyze the code of device models provided in the “hw” directory[28] in the QEMU sources.

To evaluate the device modeling API of QEMU as a tool to model user designed hardware, I’ve created a few models of bus mastering DMA devices:

- The encryption accelerator engine[29]
- The Analog to Digital converter[30]
- The Network Interface Card[31]

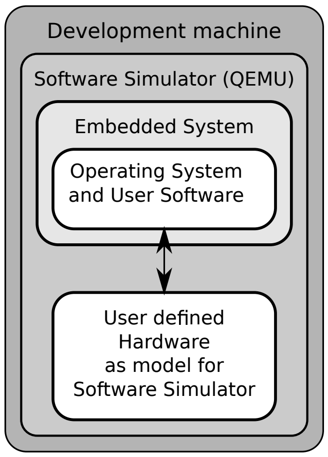

Development machine
Software Simulator (QEMU)
Embedded System
Operating System and User Software
User defined Hardware as model for Software Simulator

Figure 4. Simulation based an a software simulator (QEMU) alone, but supplemented with the model of user hardware, implemented as a QEMU device.

All of the above models have been published as open source designs. They have been successfully used as didactic tools to train the students in development of device drivers, but have also shown good potential for modeling of user design hardware, regarding it's communication with the embedded system. Similar approach has been also used in Ref. 32, where the data acquisition card and its QEMU model are described.

Main problem with modeling of our hardware as QEMU device, except of sparse documentation, is quick rate of API changes. The models written by me and by authors of Ref. 32 were prepared for version 0.14 of QEMU. The current version of QEMU is 1.1, and the device modeling API has significantly changed. There are some attempts to stabilize the modeling API. The first one was introduction of the concept of the new device model qdev.[33,34] The second one was the announcement of the the "rbus" (remote bus), allowing to connect external models, in the last conference presentations[35,36] dedicated to development of the QEMU, Unfortunately the source code of rbus seems to be not widely available yet.

7. CONCLUSIONS

The virtual hardware may be a reasonable option to increase speed and decrease costs of the development of systems consisting of user defined digital part, based on programmable logic, tightly coupled with the embedded system or micro-controller system, running the controlling software.

Unfortunately no perfect solution exists yet, which allows to obtain both real time, instruction-accurate simulation of the software part of the system and cycle accurate simulation of the user defined hardware.

There is a trade-off between the speed and accuracy of the simulation, and there are a few options to chose from:

- RTL simulation of the whole system in HW level simulator (fully open source solution available, cycle accurate but unacceptably slow simulation)
- RTL simulation of the user designed hardware in HW level simulator linked via emulated I/O interface to the software running on development host (fully open source solution available, cycle accurate hardware simulation, but testing of low level software layer, like device drivers and communication performance is not possible)
- Simulation of the software part in SW simulator linked with TLM level HW simulator (currently available for SystemC description of user designed hardware, so either commercial SystemC synthesis tools are necessary, or maintenance of separate HDL and SystemC sources of user designed hardware is necessary)
- Simulation of both software part and user designed hardware in QEMU, where user designed hardware is implemented as QEMU device model (fully open source solution, fast simulation, maintenance of separate device models in C++ and in HDL is necessary)

The QEMU simulator seems to be very promising software level simulator for hardware-software co-simulation systems, but further work is necessary on:

- providing a method to assure stable, well documented API for preparation of device models,
- implementing a method to link QEMU with RTL HW simulators for slower, but cycle-accurate simulations.

REFERENCES

[1] "QEMU open source processor emulator." http://wiki.qemu.org/Main_Page.
[2] "Oracle VM VirtualBox." https://www.virtualbox.org/.
[3] "GHDL Where VHDL meets gcc." http://ghdl.free.fr.
[4] "Icarus verilog." http://iverilog.icarus.com/.
[5] "ModelSim - Advanced Simulation and Debugging." http://model.com/.
[6] "Avnet Spartan-6 / Intel Atom Development Kit." http://www.xilinx.com/products/boards-and-kits/AES-S6NITX-LX75T.htm.
[7] "Openrisc verilog simulation of serial port communication." http://balau82.wordpress.com/2009/12/17/openrisc-verilog-simulation-of-serial-port-communication/.
[8] Zabolotny, W. M., Bluj, M., Bunkowski, K., Gorski, M., Kierzkowski, K., Kudla, I. M., Oklinski, W., Pozniak, K. T., Wrochna, G., and Krolikowski, J., "Implementation of the data acquisition system for the resistive plate chamber pattern comparator muon trigger in the cms experiment," *Measurement Science and Technology* **18**(8), 2456 (2007).
[9] Zabolotny, W., "Bus controller model for VHDL & Python cosimulation." http://ftp.funet.fi/pub/archive/alt.sources/1361.gz (2007).
[10] Zabolotny, W., "Pseudo UART allowing to connect via pseudoterminal to GHDL simulated IP core." http://ftp.funet.fi/pub/archive/alt.sources/2618.gz (2011).
[11] Monton, M., "Checkpointing for Virtual Platforms and SystemC-TLM-2 (PhD Thesis)." http://mariusmonton.com/PhDMarius.pdf (2011).
[12] Struczynski, K., "Software – Hardware co-simulation. SystemC – Qemu (MSc thesis in Polish)," (2010).
[13] Monton, M., "QEMU and SystemC." http://adt.cs.upb.de/quf/quf11/quf2011_02.pdf (2011).
[14] Becker, M., Zabel, H., and Müller, W., "QEMU/SystemC Cosimulation at Different Abstraction Levels." http://adt.cs.upb.de/quf/quf11/quf2011_04.pdf.
[15] Quaglia, D., Fummi, F., Macrina, M., and Saggin, S., "Timing Aspects in QEMU/SystemC Synchronization." http://adt.cs.upb.de/quf/quf11/quf2011_05.pdf.
[16] "Osci tlm-2.0 language reference manual." www.accellera.org/members/download_files/check_file?agreement=tlm_2-0_lrm.
[17] "QEMU-SystemC." http://forge.greensocs.com/en/Projects/QEMUSystemC.
[18] "Greensocs." http://www.greensocs.com/.
[19] "OVP: Fast Simulation, Free Open Source Models. Virtual Platforms for software development." http://www.ovpworld.org/.
[20] "IEEE Standard for Standard SystemC Language Reference Manual," *IEEE Std 1666-2011* , 1–638 (2011). http://standards.ieee.org/getieee/1666/download/1666-2011.pdf.
[21] "AutoESL High-Level Synthesis Tool." http://www.xilinx.com/products/design-tools/autoesl/index.htm.
[22] "SystemCrafter Bringing Hardware and Software Together." http://www.systemcrafter.com/.
[23] "Catapult C Synthesis Overview, Reduce the Time to Verified RTL." http://www.mentor.com/esl/catapult/overview.
[24] "Verilator." http://www.veripool.org/wiki/verilator.
[25] "SystemC to Verilog converter." http://sysc2ver.sourceforge.net/.
[26] "SystemC to Verilog Synthesizable Subset Translator." http://opencores.org/project,sc2v.
[27] "Qemu internals." http://qemu.weilnetz.de/qemu-tech.html.
[28] "QEMU sources - hw directory." http://git.qemu.org/?p=qemu.git;a=tree;f=hw.

[29] Zabolotny, W., "Model of Bus Mastering ADC implemented in QEMU." http://ftp.funet.fi/pub/archive/alt.sources/2602.gz (2011).

[30] Zabolotny, W., "Model of Bus Mastering PCI AES256 accelerator for QEMU." http://ftp.funet.fi/pub/archive/alt.sources/2622.gz (2011).

[31] Zabolotny, W., "Model of BM DMA network card for QEMU." http://ftp.funet.fi/pub/archive/alt.sources/2695.gz (2012).

[32] Pisa, P., "Comedi and uio drivers for pci multifunction data acquisition and generic i/o cards and their qemu virtual hardware equivalents." http://lwn.net/images/conf/rtlws-2011/paper.14.html.

[33] Armbruster, M., "QEMU's device model qdev: Where do we go from here?." http://www.linux-kvm.org/wiki/images/b/bc/2011-forum-armbru-qdev.pdf (2011).

[34] Armbruster, M., "QEMU's new device model qdev." http://www.linux-kvm.org/wiki/images/f/fe/2010-forum-armbru-qdev.pdf (2010).

[35] Williams, J. and Iglesias, E., "Custom Hardware Modelling for FPGAs and Embedded Linux Platforms with QEMU." http://elinux.org/images/9/95/Jw-ei-elc2010-final.pdf (2010).

[36] Williams, J. and Crosthwaite, P., "Dynamic Co-simulation of FPGA-based Linux Systems-on-Chip." http://elinux.org/images/3/3a/Elc2011_williams.pdf (2011).