Introduction Virtual Platforms

MICROELECTRONIC SYSTEMS DESIGN RESEARCH GROUP

Matthias Jung (2012)

This document is intended as a quick introduction to Virtual Platforms to acquire the necessary background knowledge for the laboratory tasks.

Motivation

Today's companies have to deal with complex architectures (e.g. multi-core systems) and a constant pressure to deliver their products quickly because of many competitors on the market. The old-established design-flow procedures have a performance problem due to the high complexity of modern systems. New development tools and approaches for Electronic System Level (ESL) design are needed to fulfil these requirements. In the past the software was developed after the hardware from various teams or a rapid prototype was created by means of e.g. a Field Programmable Gate Array (FPGA). [1]

To decrease the Time-to-Market (TTM), costs and efforts, it is necessary to develop software and hardware more concurrently and supporting the collaboration of the different teams. An effective approach for this issue is called Virtual Prototyping (VP).

Virtual prototypes are high-speed, fully functional software models of physical hardware systems which can include processors, Application-Specific Instruction Set Processors (ASIPs), Application-Specific Integrated Circuits (ASICs) or even a whole System on Chip (SoC). It is easier to test the product because the virtual prototype provides visibility and controllability over the entire system. There are helpful and powerful debugging mechanisms for virtual prototypes which are almost unthinkable on a real hardware system. This leads to a higher quality of the product and a lower supporting effort. [2]

Virtual prototypes are high-speed, fully functional software models of physical hardware systems.

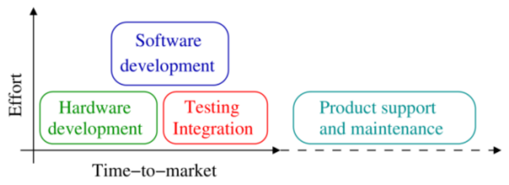


Figure 1: Traditional Flow

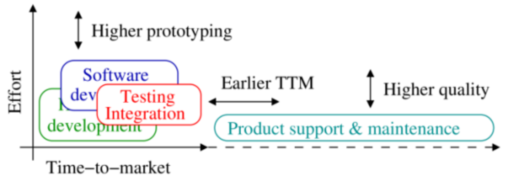


Figure 2: Virtual Prototyping Flow

Figure 1 shows the traditional flow. Figure 2 shows the fusion of hardware development, software development and testing which leads to a decreased TTM, less effort, better quality, less costumer support, and finally to reduced costs. Case studies showed that it is possible to deliver more competitive products up to 6 months faster. [2]

Virtual Prototypes are well suited for early software development. However, they are also reasonable for Design Space Exploration (DSE) because they are easy to modify. With the platform tools it is even possible to include Register Transfer Level (RTL) modules which lead to the possibility to simulate them together with functional high-level software models to save simulation time.

Design Space Exploration is the investigation of different implementation variants regarding their optimal solution. [3]

Advantages of virtual prototyping: [2]

- Visibility and controllability over the entire system
- Start software development 9 to 12 months in advance of the hardware availability
- Quickly correction of specification errors prior to Implementation
- Identification and correction of software bugs in hours rather than days
- Early Design Space Exploration
- Integration of new software on first silicon within a day
- Enable development and collaboration at worldwide locations
- Removed physical hardware dependency from the supply chain
- Improved development cycles by 30% to 50%
- Reuse (parts of) prototype for future work

To build a virtual prototype it is necessary to choose a language which covers the fields of system specification, hardware design and software development like SystemC.

SystemC

SystemC is a modelling language for systems containing hardware as well as software components. It is maintained by Accellera and it was ratified as standard IEEE 1666 in 2005. It is mainly used for high-level system modelling which leads to a shorter simulation time in contrast to other Hardware Description Languages (HDLs) like VHDL or Verilog. But it is also possible to model components on RTL level.

SystemC is maintained by Accellera. *www.accellera.org* Former: Open SystemC Initiative (OSCI)

SystemC extends the programming language C++ with classes and macros to an effective event-driven simulation kernel. These extensions provide the principles of parallelism and synchronisation. [4]

TLM2 - Transaction Level Modelling

Transaction level modelling is used to model communication between SystemC components by using function calls. The emphasis is more on the functionality of the data transfers and less on their actual implementation. Instead of modelling data transfers between different modules pin and cycle accurate, so-called transactions are used. Whole blocks of data are transferred as a reference which leads to a significant speedup of the simulation time. [5]

TLM2 is also an Accellera standard.

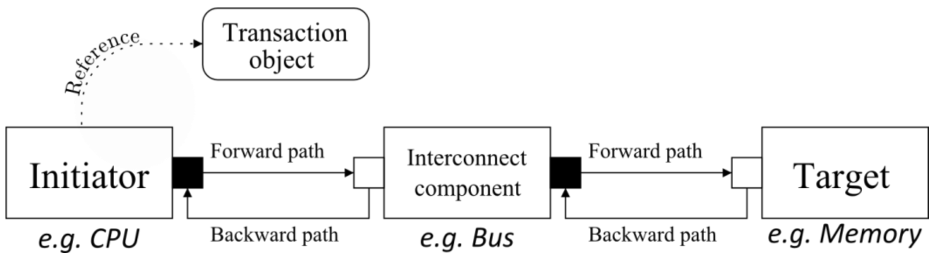


Figure 3: References to a transaction object are passed along the forward/backward path

In TLM, an initiator ■ (e.g. processor) is a module that initiates new transactions, and a target □ (e.g. memory) is a module that responds to transactions initiated by other modules. A transaction consists of a C++ object, called Generic Payload (GP), and a timing annotation which is passed between initiators and targets using function calls. Instead of passing the whole transaction object, only a reference is passed, which leads to the mentioned speedup. A module that merely forwards transactions (e.g. bus or Network on Chip - NoC) without modifying their content is known as an interconnect component. An interconnect component would have both, a target □ and an initiator socket■ . [5] [6]

TLM 2.0 is communication centric and focused on memory mapped bus modelling. The base protocol consists of reading or writing a bunch of bytes to or from a specified address. The GP of a transaction has following properties:

More information about TLM can be obtained from the Doulos Website: *www.doulos.com*

- **Command:** whether writing or reading
- **Address:** the address to/from which we write/read the data bytes
- **Data:** a pointer to the data bytes
- **Byte enables:** possibility to ignore some bytes
- **Streaming:** send a bunch of bytes to the same address
- **Response status:** whether the transaction worked correctly or not

Virtual Platforms with Synopsys Virtualizer

The problem of traditional hardware development is the slowness of the RTL simulation and the lack of configurability and visibility to analyse performance. In the traditional software development (compare Figure 4) the visibility and controllability over the entire system is often missed as well.

Synopsys *Virtualizer* is a graphical environment for designing, exploring, optimizing and debugging system architecture by means of virtual prototypes. The whole hardware system, the compiler toolchain, the software and the debugging tools are now located inside the developer's desktop PC (compare Figure 5). The developer can analyse the performance of the system, study the throughput, find bottlenecks or have a look at the task scheduling and therefore has the possibility to optimise, for instance the bus and memory architecture. [7] It is even possible to model hardware control elements like buttons or a touchscreen as shown in Figure 6.

Synopsys bought the Company *CoWare* with their Virtual Platform tool *Platform Architect* in 2010

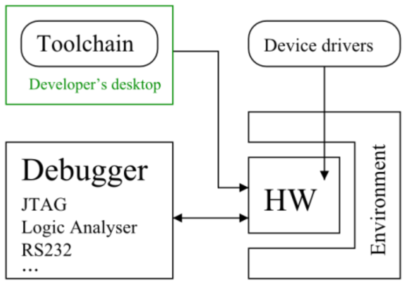


Figure 4: Traditional software development and debugging

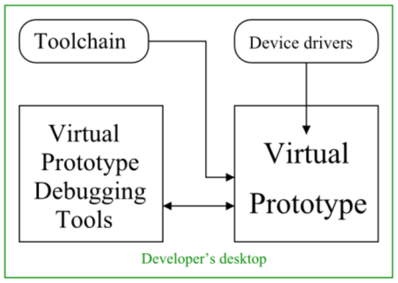
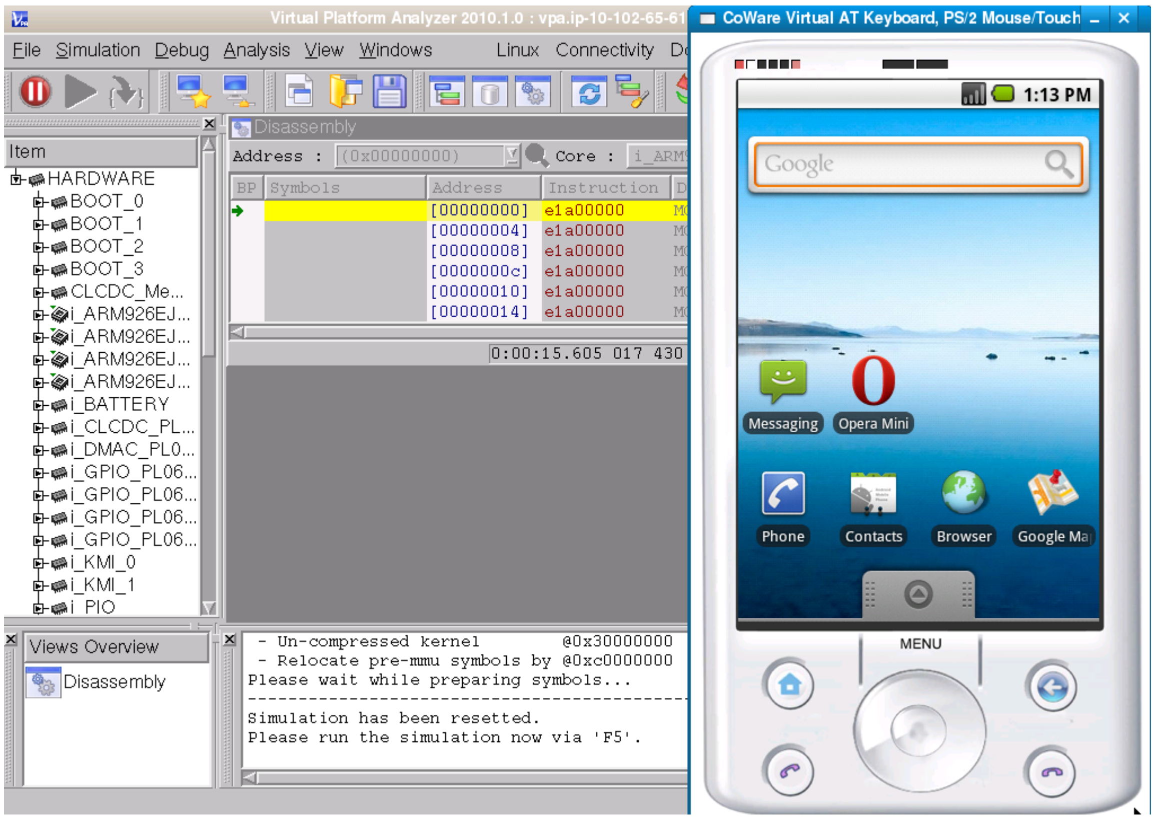


Figure 5: Virtual Platform software and hardware development and debugging

Virtualizer comes with a large library of generic SystemC components (e.g. clock generator, timer or memory) and also with cycle-accurate SystemC TLM processor support packages (PSPs) for widely-used ARM processor families. It is possible to include the SystemC model of a self designed ASIP from Synopsys *Processor Designer* or to integrate pure SystemC or VHDL components.

A fast TLM2 processor model can simulate 250 million instructions per second.



```
BP  Symbols   Address      Instruction
              [00000000]   e1a00000
              [00000004]   e1a00000
              [00000008]   e1a00000
              [0000000c]   e1a00000
              [00000010]   e1a00000
              [00000014]   e1a00000
0:00:15.605 017 430
```

```
 - Un-compressed kernel              @0x30000000
 - Relocate pre-mmu symbols by @0xc0000000
Please wait while preparing symbols...
-------------------------------------------
Simulation has been resetted.
Please run the simulation now via 'F5'.
```

Even complex hardware devices like smartphones can be modelled by means of virtual platforms

The website www.tlmcentral.com is a exchange platform for TLM2 Models

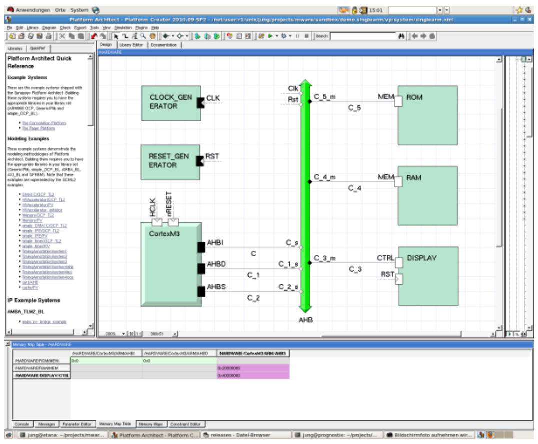
Figure 6: Virtual cellphone demo platform realised with Synopsys Virtual Platform

Tools of Virtualizer

The Synopsys Virtualizer consists of different tools which support each other:

Platform Creator:

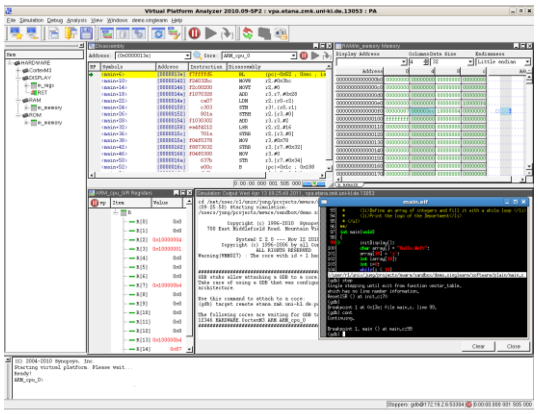
Platform Creator is the tool in which you assemble and configure your system. You can choose components from a large library and connect them by drag-and-drop. If you want to connect two components with different port types, a transactor that translates between these different protocols, is inserted. Whole bus frameworks can be generated automatically and with a powerful TCL (Tool Command Language) interface it is possible to generate platforms with scripts dynamically.



You can start Platform Creator from the command line with the **`pct`** command

Platform Analyzer:

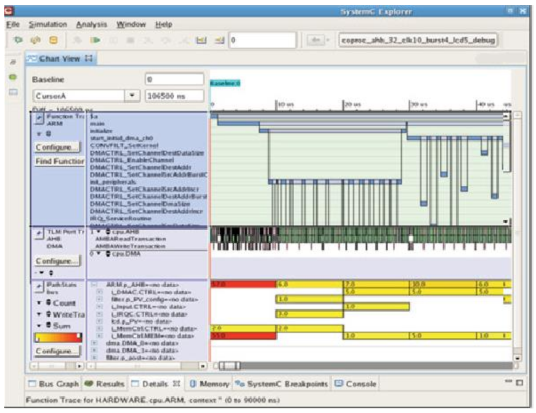
Platform Analyzer serves as a powerful software debugging tool. You have the possibility to analyse contents of registers and memories or you can single-step through the assembler program. There is an interface for the Gnu Project Debugger (GDB) thus you can use your usual GDB debugger as with real hardware. There is a TCL interface to interact with the simulation, too.



You can start Platform Analyzer directly from Platform Creator.

SystemC Explorer:

With SystemC Explorer it is possible to analyse registers, ports and bus transactions. It is a powerful virtual-hardware debugging tool that provides graphical transaction tracing and statistical analysis views that enable you to identify performance bottlenecks, determine their root-cause and examine the sensitivity that system performance may have, to individual or combined parameter settings. [7]



You can start SystemC Explorer directly from the Platform Creator.

Different Abstraction Levels of Virtual Platform Simulation

In general, four major timing abstraction levels are distinguished, namely:

Lowest simulation speed

- **Cycle-Accurate** (CA) models – as the name implies- behave precisely like real hardware. Obviously this results in slow simulation speed.

- **Approximately-Timed** (AT) models still provide almost exact timing of signals, but allow simplification of internal behaviour to achieve faster simulation speed.

- **Loosely-Timed** (LT) models accelerate the simulation even more, but of course at the expense less of timing accuracy.

- **Untimed** models are completely decoupled from real timing and allow the maximum speedup of simulation.

Highest simulation speed

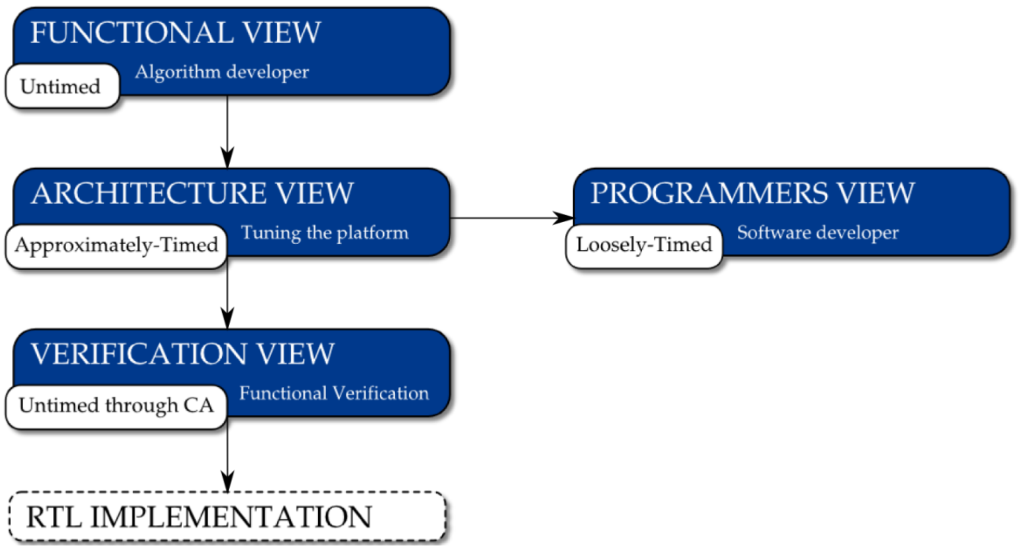


Figure 7: The four common abstraction levels and their typical applications [5]

The typical application options of the different abstraction levels are shown in Figure 7. Untimed and LT models are used if only the function behavior is needed, AT models if the simulation needs to be fast but precisely timed.

Untimed models can be developed with C++ or SystemC. AT and LT models are usually modeled with TLM. CA models can be developed in VHDL or SystemC.

Extremely convenient is the possibility to mix up models of different abstraction level. Therefore slow CA models need only to be used where they are really essential, as shown in the next section.

Combining VP Simulation Techniques with VHDL Simulation

There are various models for different kinds of abstraction available for VP simulations. It is possible to use a CA model, which complies with the RTL implementation, or an AT transaction level model with only rough timing annotations. This leads to the trade-off between accuracy and simulation speed.

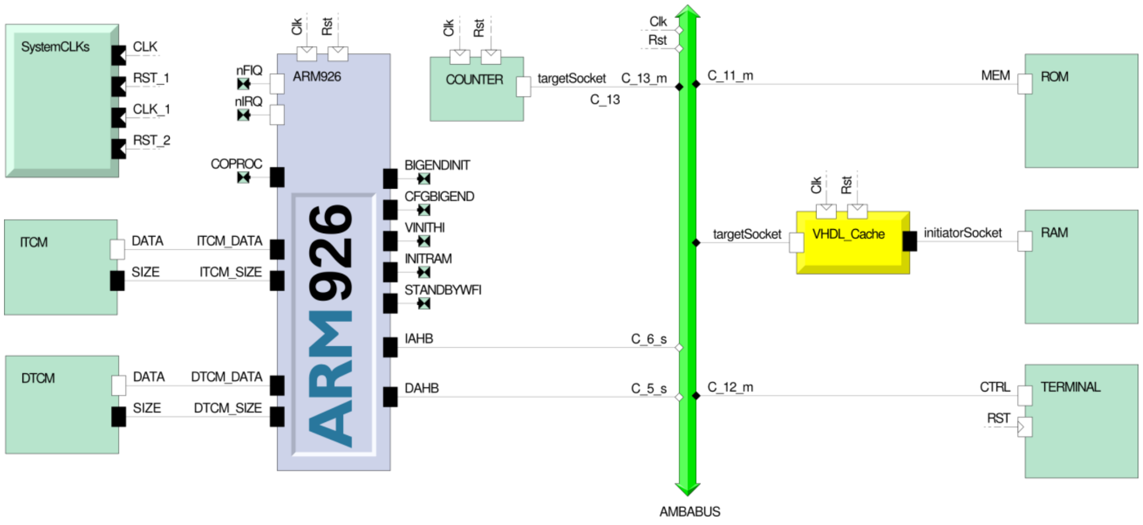


Figure 8: Virtual Platform with AT SystemC components in combination with a CA VHDL component

A common simulation practice is to build a TLM based AT platform with only one single CA model as shown in Figure 8. In other words, the virtual platform is used as a realistic test bench for the CA model. In this case the simulation is relaised as a co-simulation between SystemC and a HDL simulator like ModelSim, as shown in Figure 10 and Figure 11.

To translate between the transaction level and the CA level so called *transactors* are inserted, as shown in Figure 9.

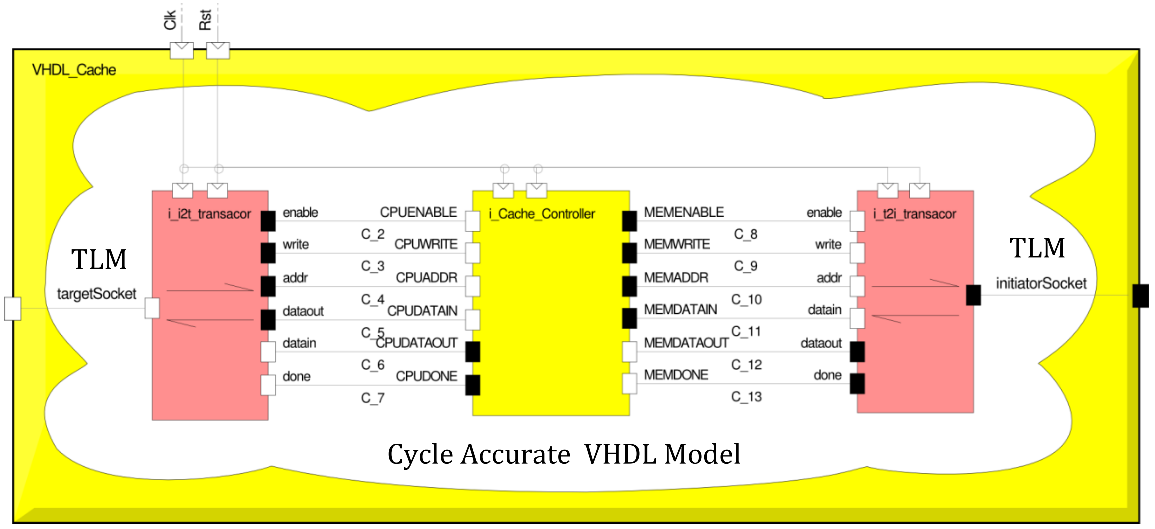


Figure 9: Transactors translate between transaction level and CA level

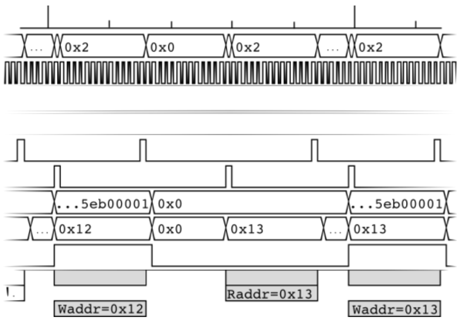


Figure 10: Debug external VHDL Signals together with TLM2 Transactions in SCExplorer

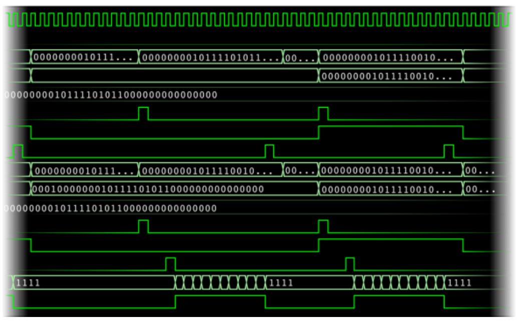


Figure 11: Debug external and internal VHDL Signals with Modelsim

References

[1] Matthias Jung, *Design and Evaluation of a Virtual Multi-ARM Platform with Real-Time Operating System*. Kaiserslautern, June 2011.

[2] Synopsys Inc. (2010) Synopsys Virtual Prototyping Solution. [Online]. http://www.synopsys.com/Systems/VirtualPrototyping/Pages/default.aspx

[3] Martin Holzer, *Design Space Exploration for the Development of Embedded Systems*, PhD thesis ed., Technischen Universität Wien, Ed. Wien, 2008.

[4] S. Liao, G. Martin, and S. Swan T. Grötker, *System Design With SystemC*, Kluwer Academic Publishers, Ed., 2002.

[5] John Aynsley. (2008, June) Doulos TLM 2.0 Tutorial. [Online]. http://www.doulos.com/knowhow/systemc/tlm2/tutorial__1/

[6] Open SystemC Initiative. (2009, July) OSCI TLM-2.0 LANGUAGE REFERENCE MANUAL. [Online]. http://www.systemc.org

[7] Synopsys Inc. (2010) Platform Architect. [Online]. http://www.synopsys.com/Systems/VirtualPrototyping/Pages/default.aspx