A Platform-Based Design Framework to Boost Many-core Software Development

Guilherme Madalozzo[1], Marcelo Mandelli[1], Luciano Ost[2], Fernando G. Moraes[1]
[1] FACIN - PUCRS – Av. Ipiranga 6681, 90619-900, Porto Alegre, Brazil
[2] University of Leicester, Department of Engineering, Leicester, UK
{guilherme.madalozzo, marcelo.mandelli}@acad.pucrs.br, luciano.ost@leicester.ac.uk, fernando.moraes@pucrs.br

***Abstract*—Embedded software engineers are dealing with complex and large software codes, which will continue to grow. To achieve a cost-effective design, concomitant hardware and software development is required during early design phases. This paper presents an open-source platform based design framework that combines different ADLs and simulators aiming at improving embedded software productivity, targeting future many-core embedded systems. The proposed approach adopts three models: RTL-VHDL level; RTL-SystemC coupled to ISSs; PBD (Platform Based Design) using OVP. The software (operating system and user applications) is the same for both models. Therefore, the OVP modeling allows fast software validation and debuggability. With the SystemC-ISS, it is possible to accurate estimate performance and energy consumption. The low-level model enables, besides area estimation, the validation of low-level protocols, as the communication protocol, network interface or flow-control mechanisms between routers. Results evaluate execution time, simulation time, and the number of executed instructions for several benchmarks using the proposed approach. The OVP model presents in average five times faster than the RTL-SystemC model, and the RTL-SystemC up to 155 times faster than the RTL-VHDL model.**

***Keywords*—MPSoC; NoC; virtual platform; multi-level; PBD.**

I. INTRODUCTION AND RELATED WORKS

Embedded software complexity (billions of object code instructions) and cost (up to 50% of the whole system [1]) is increasing dramatically during the last decades. The resulting complexity and cost is determined by compound software stack design, involving operating system (OS) porting and analysis, driver development, among others. A significant fraction of emerging embedded software projects is directly dependent on the hardware that it is deployed. In traditional design flow, the software stack is validated after the hardware development. In the underlying approach, the occurrence of errors requires redesign, which increases development costs and time-to-market, thereby making it less suitable for emerging large-scale systems.

Dealing with the complex integration of hardware and software stacks requires efficient modeling and simulation capabilities for concomitant software and platform development. PBDs (Platform Based Design) provide efficient means on which software functionality and target platform architecture can be designed and verified together at early stages of the design flow [2]. Virtual Platforms (VPs) are used as PBD, decreasing development software time since tests are made at the system level. Such simulators abstract away low-level details to boost development cycle, enabling the design exploration of various architectural and software alternatives before it goes down to the RTL/gate-level implementation.

The preferred exploration level usually defines the adoption of such simulators. While PBDs requires quasi-cycle accurate simulators [3], software development demands high simulation speeds (e.g. 100 MIPS) [4][5]. With such conflicting requirements, it is difficult to cover all modeling and simulation needs inherent to platform and software design space exploration with one single simulator.

Works [2][6][7] presents platforms using PBDs method and [8][9][10] using EDKs (Embedded Development Kits). In [2], MPSoC platforms are described in OVP but no comparisons with other ADLs (e.g. SystemC) were made. In [6] an OVP processor integrates the TLM-SystemC platform achieving a speed up to 40 times when compared to an RTL simulation. An analytic method to verify whether the platform meets real-time application requirements is described in [7]. Works propose EDKs to automate FPGA-based MPSoC design and emulation. The work described in [8] presents a PBD model that reduces the MPSoC design complexity using the LavA framework. The work in [9] presents a platform with an SDK (Software Development Kit) able to execute different programming models at an abstract level of hardware. To achieve power and area evaluation, the literature contains works that present MPSoCs in register transfer level. In [10], Authors present an RTL distributed memory platform for design space exploration of MPSoCs. In [5], the Authors present a GALS NoC approach modeled in RTL and TLM-SystemC. Authors in [11] present a deadline evaluation in an RTL model MPSoC.

To overcome aforementioned conflicting requirements, this paper contributes by proposing a platform based design framework to improve many-core software development processes, including OS and application coding, verification and performance analysis from functional to software timing behavior properties.

Our *contribution* distinguishes itself from all previous works mentioned by combining fast and accurate modeling and simulation capabilities in one single software development flow, including: cycle-accurate model (VHDL and RTL-SystemC/ISS) and approximated-time model (OVP, which uses C language). The proposed framework comprises three platform descriptions, which combines different modeling techniques and simulators, targeting fast and accurate development of software targeting emerging multiprocessor embedded systems.

The interoperability between the three platform models is guaranteed through a well-defined hardware abstraction layer (HAL) and a unified software description (i.e. OSs, applications, communication model). In this direction, target software stack can be modified and executed onto the OVP-based platform model until the point where its functionality is validated. The same code can then be executed in a still fast but clock-cycle accurate RTL SystemC-ISS model, which allows assessing lower performance figures (e.g. application execution time). Finally, RTL-VHDL model can receive the target software as input to profile the power figures, e.g., the average switching activity of adopted CPU architecture.

II. MPSOC ARCHITECTURE MODEL

This work adopts a NoC-based MPSoC model with a set of processing elements (PE), with a single processor to avoid complex clustered architectures. There are local memories, acting as scratchpad

memories, with no shared memory. The adoption of this memory organization reduces the NoC traffic, using message passing as communication model.

Besides these characteristics, the management of the MPSoC resources is a key feature to ensure scalability. Management functions include application mapping, monitoring, QoS actions (task migration, communication priorities, scheduling priorities). To improve system performance and management, the present work adopts a clustered distributed management architecture [12].

In a clustered MPSoC, each PE may have distinct roles: Global Manager Processor (GMP), Local Manager Processor (LMP) and Slave Processor (SP). The LMP is responsible for managing the cluster, executing functions as monitoring, task mapping and verification of deadlines. The GMP manages the overall system, also executing all functions of the LMP. The SPs execute user's applications. Each SP runs a simple operating system, which enables the communication between PEs and multitask execution.

Fig. 1 presents a simplified view of a 6x6 MPSoC instance, with four 3x3 clusters. Each PE contains a MIPS-like processor (Plasma) connected to a private memory, a DMA module, a Network Interface (NI), and the NoC router. There is an external memory connected to the GMP, named Task Repository, which contains all applications' tasks that will be executed in the system.

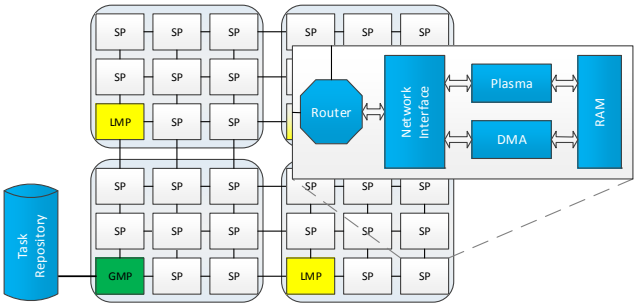


Fig. 1 – Architecture for a clustered NoC-based MPSoC.

At design time, all clusters have the same size. At runtime, clusters may borrow resources from neighbor clusters to map applications' tasks when the cluster has all resources (SP) executing tasks. When resources become available, task migration is used to move tasks to the source cluster, to minimize application fragmentation and reduce the communication energy consumption.

III. MPSoC Modeling

This section presents the hardware models used to describe the MPSoC at different abstractions levels. In the sequel, the unified software model is presented, enabling designers to execute applications with different hardware models.

A. Hardware Model

To evaluate different aspects of the design, three distinct models of the MPSoC architecture presented in Section II are available: (*i*) RTL-VHDL; (*ii*) RTL-SystemC coupled to ISSs; (*iii*) abstract model, using Open Virtual Platform (OVP).

Fig. 2 presents the main pros and cons of each model. The RTL-VHDL model provides clock-cycle accurate simulation and power and area reports. However, software debuggability and simulation time are the main drawbacks on using this model. The clock-cycle RTL-SystemC model was derived from the VHDL model, enabling to reduce the simulation time. The OVP model sacrifices accuracy to provide to the developers higher software debuggability and faster simulation time than the previous models. An important feature of the OVP model is flexibility. It enables to explore different pre-defined hardware modules, such as processors, peripherals, and memories.

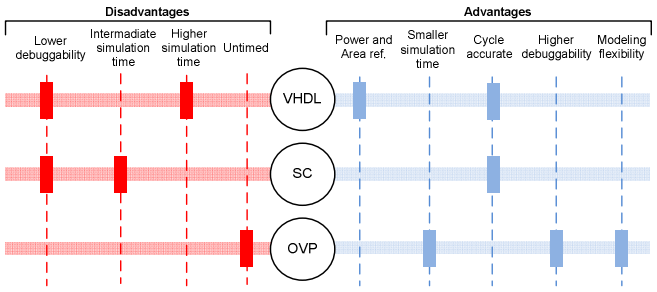


Fig. 2 – Pros and cons of multi-level modeling architecture.

1) RTL-VHDL

The designer may parameterize several architectural features of the MPSoC. The local memory implementation may target ASIC (65 nm technology) or FPGAs (Xilinx Block RAMs) devices. At the NoC level, it is possible to select the buffer depth, the routing algorithm, the arbitration policy, among other parameters.

The VHDL model was successfully implemented in FPGAs (3x3 instance). The FPGA prototype contains the MPSoC and three additional modules: (*i*) MAC Ethernet communication interface with the host; (*ii*) control unit; (*iii*) DDR2 memory controller. The host sends the applications' codes to a DDR2 memory, which acts as the task repository. Next, the host may send commands to the MPSoC to start the execution of users' applications, or to request debug information. The control unit is responsible for controlling the access to the external memory or the MPSoC.

The 65nm ASIC implementation, using the memory generator of the design kit, required roughly one mm^2 for each PE. The MPSoC worked correctly after the back-end simulation.

2) RTL-SystemC

This model has the same structure of the RTL-VHDL model. The NoC, DMA, NI, and memory modules were rewritten in RTL-SystemC. The processor is modeled using a clock-cycle accurate instruction set simulator (ISSs) wrapped in a SystemC module. This approach does not take advantage of SystemC language structures, such as *sc_fifo*. The clock-cycle accuracy was checked in two ways. Initially, waveform traces related to the injected traffic into the NoC were compared. Then, the time to execute different tasks was compared. Both verification methods demonstrate the equivalence between the RTL-VHDL and the RTL-SystemC. There is a small difference in the execution time (<1%), due to pipeline stalls and arithmetic instructions. Such issue is discussed in the results section.

The VHDL model has as the main advantage the fact to be synthesizable, allowing to captures accurate area, frequency and power performance figures. Debug facilities include waveforms and assertions, targeting hardware development, not software development. The RTL-SystemC model enables to simulate larger systems accurately, in a reasonable simulation time. Some gains in debuggability are achieved, e.g. by inserting debug coded in the ISS.

3) OVP Implementation Model

OVP [4] is a virtual platform and modeling framework proposed by Imperas, aiming to accelerate the development of embedded software, specifically for SoCs and MPSoCS. The framework contains three main components: (i) APIs that enable to model hardware components in C language; (ii) library of free open-source CPUs and peripheral models; (iii) OVPsim simulator. OVPsim is a dynamically linked library, which supports the simulation of bus-based multiprocessor platforms. OVPSim relies on dynamic binary translation that increases simulation speed.

Fig. 3 presents the OVP platform architecture, with the interconnection of the OVP processor with the OVP NoC. The OVP NoC is implemented in C (OVP APIs *ppm* and *bhm*), with the same routing algorithm and arbitration policy of the RTL NoC. Two dedicated memory spaces are reserved for the NoC: *reg_bank* and *buffer_proc*. *The reg_bank* area stores the outcoming packets. The *buffer_proc* receives the incoming packets. Callback functions are executed on every read or write access to the defined address area corresponding to the NoC.

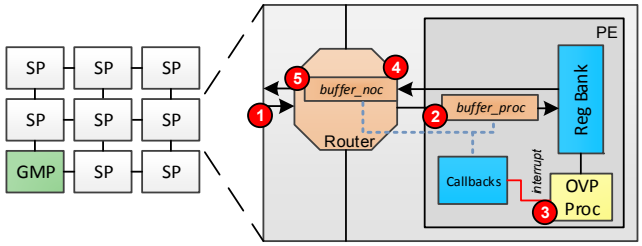


Fig. 3 – OVP Platform Architecture.

When a given router receives a packet (number 1 in Fig. 3), this packet is stored in the *buffer_proc* (number 2). A callback function interrupts the processor (number 3) after the complete reception of the packet. An interrupt is raised, and an ISR (Interrupt Service Routine) is called to read the packet. The router executes a callback function when the processor writes in the register bank. This callback function stores the packet in the *buffer_noc* (number 4). The router injects the packet into the NoC once the packet is stored in this buffer (number 5).

B. Software Model

As mentioned previously, the platform provides a multi-level abstraction. The hardware designer can take the design decisions executing the software with the required model (e.g. RTL models report power and area while OVP results energy consumption). However, the software designer can validate and evaluate the software faster, with no changes in the different hardware models.

Distinct operating systems (µkernel) run in GMP, LMPs, and SPs. The GMP and LMPs processing elements do not execute users' tasks. The µkernel of the LMPs executes management tasks, as NoC monitoring, task mapping, task migration, reclustering. The µkernel of the GMP execute all functions of the LMP plus access to external devices (e.g. task repository), and selection of which cluster receives a new application. The µkernel of the SPs has two primary functions: scheduling, responsible for multi-task execution; and inter-task communication. The inter-task communication occurs through an MPI-like API, with two main functions: *send*() and *receive*(). A non-blocking *send*() insert messages in communication queues while a blocking *receive*() read from the communication queues.

The GMP is responsible for initializing the resources of the MPSoC. When the simulation starts, the GMP initializes the clusters, sending to the LMPs the region they will manage. Once an LMP knows the region it controls, it sends a message to all SPs of the region with its address. Applications are mapped at runtime. When there is a request to execute a new application, the GMP selects the cluster to execute this application. Then, the GMP reads from the application repository the description of the application and transmits it to the LMP responsible by the cluster. When the LMP receives the application description, it starts the task mapping process. Firstly, the initial tasks (those tasks without dependences to other tasks) of the application are mapped. When an initial task communicates with a non-mapped task, a task allocation request is transmitted to the LMP, which executes the mapping heuristic. Each SP may execute a parameterizable number of tasks simultaneously.

The hardware abstraction layer is responsible for managing the instructions of the µkernel that depend on the hardware modeling. User applications are described in C, being the same regardless the hardware modeling. According to Yoo and Jerraya [14], HAL is the software that is directly dependent on the underlying hardware. HAL APIs give, to the operating system and software application, an abstraction of the underlying hardware architecture. Thus, the adoption of an HAL simplifies OS porting to new hardware abstractions.

IV. RESULTS

This Section evaluates and compares the platform models. The comparison occurs in two stages. First, the low-level clock-cycle accurate platforms, RTL-VHDL and RTL-SystemC, are compared. Then, the PBDs models, RTL-SystemC and OVP are compared. As mentioned, RTL-SystemC meets low-level (e.g. quasi-cycle accurate model) and high-level (e.g. debuggability and fast simulation compared to RTL-VHDL model) modeling characteristics.

Applications are modeled as task graphs, App=<T,C>. The set of the application tasks, $T=\{t_1, t_2, t_n\}$, corresponds to the graph vertices. The set $C=\{(t_i, t_j, w_{ij})\} \mid (t_i, t_j) \in T$ and $w_{ij} \in N^*\}$ represents the communication between tasks, corresponding to the graph edges. Five applications are used as benchmarks: MPEG, with 5 tasks; Dijkstra, with 6 tasks; DTW, with 10 tasks; Fixed-Based, with 15 tasks; VOPD (synthetic), with 12 tasks.

An automated process generates a simulation scenario. A simulation scenario contains the hardware description, as well as the compiled software.

A. Low-Level Models Evaluation

This Section presents the first comparison, simulation time and simulated instructions of the low-level models, RTL-VHDL and RTL-SystemC. The platform modeled is a 4x4 NoC-based MPSoC, with a manager processor (GMP), and 15 processors (SPs), which execute a multitask operating system. Two applications are evaluated in this experiment: MPEG and VOPD, with 5 and 12 tasks respectively. The first scenario (SC1) executes 2 MPEG and 1 VOPD instances, totalizing 22 tasks. The second (SC2) and third (SC3) scenarios contain 44 and 88 tasks, respectively.

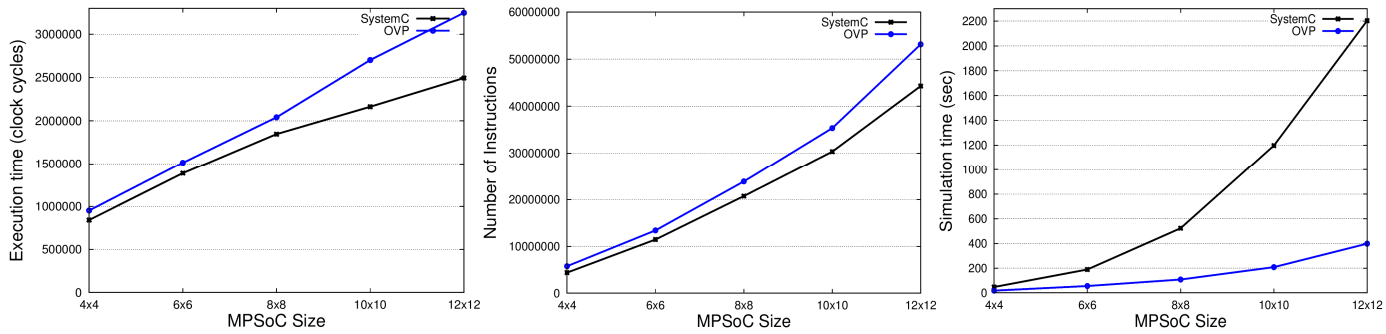
Table I evaluates the simulation time (8-core Xeon processor, 32 GB RAM) for the three scenarios. A speedup of two orders of magnitude is observed using the RTL-SystemC simulation. Table II evaluates the execution time required to execute all applications of each scenario, and Table III presents the total number of instructions executed by all processors in the platform. Such results demonstrate that the RTL-SystemC and RTL-VHDL models have the same behavior. There is no HAL for these two models. The differences observed in the Tables are due to simplifications made in the ISS, not reflecting the real operation of some instructions (e.g. multiplication and division instructions).

TABLE I - SIMULATION TIME (IN SECONDS) FOR RTL MODELS

Scenarios	*VHDL*	*SystemC*	*VHDL/SystemC*
SC1	2,425	19	127
SC2	4,407	37	119
SC3	7,932	51	155

TABLE II - EXECUTION TIME (IN CLOCK-CYCLES) FOR LOW-LEVEL MODELS

Scenarios	*VHDL*	*SystemC*	*VHDL/SystemC*
SC1	265,015	265,228	0.998
SC2	526,660	528,524	0.996
SC3	1,050247	1,055,543	0.995



(a) (b) (c)

Fig. 4 - Comparison between the SystemC and OVP models. (a) execution time in clock cycle; (b) number of instructions simulated; (c) simulation time in seconds.

TABLE III - NUMBER OF EXECUTED INSTRUCTIONS FOR LOW-LEVEL MODELS

Scenarios	*VHDL*	*SystemC*	*VHDL/SystemC*
SC1	422,757	423,120	0.999
SC2	834,335	836,335	0.997
SC3	1,662,311	1,664,941	0.998

B. *High-Level Models Evaluation*

The previous Section demonstrates the accuracy of RTL-SystemC model in terms of executed instructions and execution time, using the RTL-VHDL as reference. In this Section, RTL-SystemC is the reference model to evaluate the behavior of the OVP model, for large MPSoCs. Considering that RTL-VHDL showed accurate results compared to RTL-SystemC, this Section will not apply comparative evaluations efforts with RTL-VHDL model. There are five scenarios evaluated the RTL-SystemC and OVP models with different MPSoC size (4x4, 6x6, 8x8, 10x10 and 12x12). Four benchmarks are evaluated in this experiment: MPEG, DTW, Dijkstra and Fixed Based.

Fig. 4(a) presents the execution time, in clock cycles, for all scenarios. For MPSoCs up to 64 PEs, the difference between RTL-SystemC and OVP is approximately 10%. For larger systems (up to 144 PEs), this difference reaches 25%. At the end of the simulation, both RTL-SystemC and OVP report the number of executed instructions per PE.

Fig. 4 (b) presents the total number of simulated instructions for all scenarios. The difference for the larger MPSoC is 17%. Even if the same workload is applied for both simulations, the number of instructions varies due to differences in the mapping, number of tasks per processor, and in the traffic in the NoC.

Fig. 4 (c) presents the simulation time for all evaluated scenarios. Simulation time is the real time required to finish the simulation. Note that the number of tasks for each scenario is approximately equal to the number of PEs, corresponding to a light workload. Even with this light workload, the OVP simulation has a speedup of five times compared to the SystemC simulation. The execution time presents an error of 10 to 20% depending on the system size, which is an acceptable error for high-level models.

V. CONCLUSIONS AND FUTURE WORKS

This paper proposed a PBD method to software development flow in NoC-based MPSoCs modeling. With the complex task of integrating hardware and software using models in low-level (e.g. RTL-VHDL), we present a virtual platform modeled in OVP. This abstraction level enables the hardware component reuse, fast simulation, and software validation at early design stages, generating estimations related to the performance (e.g. execution time). The RTL-SystemC is effectively clock-cycle accurate, with a speedup of two orders of magnitude compared to RTL-VHDL model. Moreover, the PBD method proved to be an instruction accurate model.

Future works include: (*i*) estimate power at abstract model (OVP), considering static and dynamic consumption, evaluating the error between RTL-VHDL and OVP; (*ii*) evaluate other ADLs to modeling platforms, as ArchC using the SystemC simulator; (*iii*) execute tests with known benchmarks.

ACKNOWLEDGMENTS

The Author Fernando Moraes is supported by CNPq - projects 472126/2013-0 and 302625/2012-7, and FAPERGS - project 2242-2551/14-8. The authors would like to thank Imperas Software Ltd. and Open Virtual Platforms for their support and access to their models and simulator.

REFERENCES

[1] International Business Strategies, Inc. (IBS), 2013.

[2] Rekik, W; Ben Said, M; Ben Amor, N. “Virtual Prototyping of Multiprocessor Architectures Using the Open Virtual Platform”. In: ICCAT, 2013, 6p.

[3] Binkert N.; at al. “The gem5 simulator”. ACM SIGARCH Computer Architecture News, v.39 (2), 2011, 7p.

[4] OVP 2014, Available at: www.ovpworld.org/technology_ovpsim.php

[5] Lemaire, R.; Thuries, S.; Heiztmann, F. “A flexible modeling environment for a NoC-based multicore architecture”. In: High Level Design Validation and Test Workshop, 2012, pp. 140-147.

[6] Zhang, D.; Zeng, X.; Wang, Z.; Wang, W.; Chen, X. "MCVP-NoC: Many-Core Virtual Platform with Networks-on-Chip support". In: ASICON, 2013, pp. 28-31.

[7] Indrusiak,L. “End-to-end Schedulability Tests for Multiprocessor Embedded Systems based on Networks-on-Chip with Priority-Preemptive Arbitration”. Journal of Systems Architecture, v.60(7), 2014, pp.553-561.

[8] Meier, M.; Engel, M.; Steinkamp, M.; Spinczyk, O. “LavA: An Open Platform for Rapid Prototyping of MPSoC”. In: FPL, 2010, pp.452-457.

[9] Benini, L.; Flamand, E.; Fuin, D.; Melpignano, D. "P2012: Building an Ecosystem for a Scalable, Modular and High-Efficiency Embedded Computing Accelerator". In: DATE, 2012, pp.983-987.

[10] Busseuil, R.; Barthe, L.; Almeida, G.; Ost, L.; Bruguier, F.; Sassatelli, G.; Benoit, P.; Robert, M.; Torres, L. “Open-Scale: A Scalable, Open-Source NoC-based MPSoC for Design Space Exploration”. In: ReConFig, 2011, pp.357-362.

[11] Paulin, P.; Pilkington, C.; Langevin, M.; Bensoudane, E.; Gagné, V.; Nicolescu, G. “Parallel Programming Models for a Multiprocessor SoC Platform Applied to Networking and Multimedia”. In VLSI, 2006, pp.667-680.

[12] Castilhos, G.; Mandelli, M.; Madalozzo, G.; Moraes, F.G. “Distributed Resource Management in NoC-Based MPSoCs with Dynamic Cluster Sizes”. In: ISVLSI, 2013, pp. 153-158.

[13] Hu, W; Tang, X; Bin Xie; Chen, T; Wang, D. “An Efficient Power-Aware Optimization for Task Scheduling on NoC-based Many-core System”. In: CIT, 2010, pp. 171–178.

[14] Yoo, S; Jerraya, A. "Introduction to hardware abstraction layers for SoC". In: DATE, 2003, pp.336-337.