Impact of Description Language, Abstraction Layer, and Value Representation on Simulation Performance

Wolfgang Ecker
Infineon Technologies AG
IFAG COM BTS MT SD
81726 Munich, Germany
Wolfgang.Ecker@infineon.com

Volkan Esen,
Lars Schönberg,
Thomas Steininger,
Michael Velten
Infineon Technologies AG
TU Darmstadt - MES / BTU Cottbus
Firstname.Lastname@infineon.com

Michael Hull
Infineon Technologies AG
University of Southampton
mh102@ecs.soton.ac.uk

Abstract

In recent years other verification features than simulation performance such as robustness and debugging gained increasing impact on simulation language and tool selection. However, fastest model execution speed is still priority number one for many design and verification engineers. This can be seen in the continuously growing interest in virtual prototypes and transaction level modeling (TLM).

As part of the ongoing re-work modeling language strategies and the world wide introduction of TLM, a detailed analysis of the impact of description languages, abstraction layers and data types on simulation performance is of high importance. For the presented analysis, we considered five designs that have been modeled in VHDL, Verilog, SystemVerilog, and SystemC, using different value representations and coding styles, covering the abstraction levels from functional to behavioral to RTL.

This paper presents our evaluation environment and several interesting findings of our analysis. The most important results are as follows: We found that HDL tool/language/abstraction selection of RTL models impacts on the execution speed with a factor of 4.4. We found that Verilog is on average 2x faster than VHDL for RTL models. We found that SystemC results in 10x slower RTL models than HDLs and surprisingly results in 2.6x slower TLM[1] PV models than SystemVerilog. And we found finally that on average over all analyzed aspects SystemVerilog models are executed fastest.

1 Introduction

During the last years simulation performance had a continuously decreasing impact on simulation language and tool selection. Other items as verification aspects, robustness, or debug increased importance [1]. Furthermore, the performance difference of different tools and languages has been estimated as quite low.

However, simulation time is still the main focus of design engineers and managers. The main reason is that simulation is still the workhorse for verification despite the upcoming of formal methods, emulation, and FPGA prototyping. Unfortunately, simulation time increases at least linear with the circuit complexity that according to Moore's law increases exponentially over time. The need for simulation of embedded software puts even more burden on simulators since the interpretation of SW by an HDL model needs to be performed. This burden is further increased by constraint random simulation that reduces effort for building test cases by requiring more simulation runs.

[1] TLM defines the styles PV (programmer's view), PVT (programmer's view with timing), and CA (cycle accurate).

Also the impact of simulation performance on the overall verification process comes to the fore.

- Engineers' waiting time until a regression run returns results is directly impacted by simulation speed. The classical "night time slot" for regression has to be met to avoid further delay.
- Engineers' waiting time in the debug-modify-compile-simulate loop is double impacted by simulation speed: Higher simulation speed shortens both the time until a specific state in simulation is reached and the time until the simulation result of a modification is made available to the engineer.

The most obvious sign for need of simulation speed is the upcoming interest and usage of behavioral models for making executable specifications and for fast software verification.

The result of analyzing impact of description language, abstraction layer and data types on simulation performance is summarized in this paper.

It is organized as follows: After referencing related work, we present the test analysis environment and the model alternatives used for analysis. Finally, analysis results are presented, key conclusions highlighted, and discussed.

2 Related Work

Several comparisons between different modeling languages and modeling styles have been published already, however most of them completely ignore performance issues:

The comparison in [2] is just restricted to a listing of different language features in VHDL, Verilog and SystemVerilog and what equivalent they have in the other languages (if any). A SystemVerilog centric comparison can be found in [3]. In [4] a comparison is made between VHDL and Verilog concerning issues like basic language features, learning curve, support of reuse concepts and applicability to different abstraction layers. A similar comparison is done in [5] which also includes the topic of data types, library concepts, testbench features and verboseness. A comparison of the different language features of VHDL, C++ and SystemC and their correspondence to the different abstraction layers is done in [6].

Some papers discuss simulation performance but are restricted to just a few modeling languages and usually just one modeling alternative. SystemVerilog is not considered at all:

A simulation tool related analysis of Verilog models was described in [7]. The use of profiling, VCD dump reduction, careful use of optimization flags, and two-state simulation were named as alternatives to dramatically improve simulation speed. A general claim on simulation speed was made in [8]. Cycle

978-3-9810801-2-4/DATE07 © 2007 EDAA

simulation is claimed to be 10-50x faster than event driven simulation, while behavioral models are claimed to be 5-10x faster than RTL. Detailed analysis of modeling styles and languages is missing in both papers.

A tool comparison with only VHDL models and without detailed analysis of the impact of modeling styles has been published in [9]. Also a comparison of Verilog tools and VHDL tools compared to the fastest Verilog tool is described in [10]. Verilog is claimed to be 4-5x faster than VHDL but for an undefined modeling style.

A comparison between VHDL and C including simulation performance is made in [11], a similar comparison between VHDL and ADA in [12]. In [13] the differences concerning simulation time between VHDL and SystemC models are being discussed. The impact of LSE optimization for SystemC models is described in [14].

In all papers only a small subset of modeling styles is considered.

Our contribution to the field of language comparison is as follows. We created five different designs in four different languages (VHDL, Verilog, SystemVerilog and SystemC), using several abstraction layers and different data types.

Those models are then used in a regression environment in order to get representative information concerning performance differences between languages, abstraction layers and data types. In addition we use simulators of several EDA vendors in order to minimize the effect of tool dependency.

3 Analysis Environment

3.1 The Reference Designs

We used five different models for our evaluations: An 8 bit CPU core with memory, a 12 bit CPU subsystem consisting of CPU, memory and IO devices, a systolic array for concurrent stream processing, an FFT, and a switch. These models have been selected since they cover a wide variety of architecture alternatives (FSM, datapath, memory, communication) and modeling needs (arithmetic, bit manipulation, data flow, control flow).

We did not use a detailed analysis on single language feature basis because we found out that models like these did not produce realistic results, even if a weighted statistic was applied to rate the result. The reason might be that many effects of combining constructs were not captured.

Also a wide range of coding style alternatives was used. For example, we used flat and hierarchical descriptions, dataflow and control flow descriptions, as well as table driven descriptions and algorithmic descriptions.

Synthesizing the models resulted in 5k gates to 100k gates, not considering the memories. This gives the models the size of a smaller to medium sized IP block.

Different testcases covering all aspects of the models as well as a wide range of simulation times were written for each model. Execution times were up to 10000 CPU seconds providing a quite realistic benchmark in terms of simulation time.

3.2 Overall Structure of the environment

Code for design and testbench as well as assembly code resides in a hierarchical file structure under clear case version control.

All tests, i.e. code and tool options are specified in XML. They are interpreted by a test handler. The following steps are executed per testcase:

- Copy code to empty directory in order to eliminate side effects as much as possible
- Perform compilation according to the sequence of files and the tool options specified in the test
- Elaborate model, when needed as separated step and measure system and user time
- Execute simulation and measure system and user time
- Write measured time to a data base; the measured time is qualified with the test description (test case, language, optimization mode, trace mode, etc.)
- And finally clear directory

A script for generating a survey on languages and modeling in a tabular way helps to get a general overview. Specific scripts help to relate execution time under certain aspects and to do statistics.

3.3 The Tools

In order to eliminate tool dependency of measured simulation performance, several EDA tools were used to execute the models.

All tools were executed in full optimization mode. Object visibility was not required and no signal trace was performed. The selected simulation mode allows focusing on languages and abstraction. It provides the fastest execution and it reflects a typical tool setting for regression simulation with self checking testbenches.

The following tools were used:

- Synopsys “vcs2005.06-SP2”
- Mentor “modelsim6.2e”
- Cadence “ncsim5.4”
- OSCI 2.1v1 SystemC kernel (for SystemC models only); optimization –O3 for gcc was used; when using further optimizations the compilation crashed

In this paper, we give no single tool related information due to NDA agreements with EDA companies. The impact of tool capabilities on the result is discussed in a tool anonymous way.

3.4 Measurement

All tools were executed on dual processor (3 GHz Intel Xeon with 4GB RAM) workstations running enterprise linux under LSF control, a state of the art application scenario. Measurements were repeated 5 times to equalize impact of workload. CPU time and system time of elaboration and simulation were added in order to form a representative value for execution time.

4 Model Alternatives

As already proposed in the RASSP taxonomy [15], we consider value representation as an independent model alternative. Orthogonal to value representation, we take a mix of functional, structural, and timing abstraction as second independent model alternative. We also approximate the modeling style by a model classification from ARM [16] which primarily specifies TLM abstraction.

4.1 Selected Languages

As already mentioned, we evaluated VHDL, Verilog, SystemVerilog, and SystemC. The following dialects were used:

- VHDL'87 because this language dialect is still standard in our company. VHDL'93 is not used due to several major portability issues.
- Verilog'01, however mainly in a Verilog'95 compatible style. No portability issues between the two standard versions were found yet.
- SystemVeriog P1800, the current IEEE standard. Several restrictions had to be considered.
- SystemC 2.1.v1.

4.2 Time abstraction and description styles

The model's time representation and description style cannot be treated separately. We give an overview in this subsection.

4.2.1 Functional Model (un-timed)

Functional models reflect only functionality of the design. Neither architectural nor timing aspects are covered. The modeling concept is like programming in a sequential programming language. Models might have been built in the programming language C as well. We made two different characteristic models:

- Flat functional model, i.e. using only one thread without any function calls, as special coding for optimized execution.
- Hierarchical functional model, i.e. using function calls that reflect functional design partitioning but not necessarily architectural partitioning.

External communication as required for IO instructions was mapped to files. Feedback loops in IO devices were not considered for modeling; however they might have been modeled with an IO buffer in the model.

Functional models are expected to be the fastest models. They have been built to set a base line. The two functional model alternatives described above were modeled in all languages.

4.2.2 Behavioral Model

Behavioral models reflect the classical CPU, memory, and IO device (peripheral) partitioning. No further partitioning is performed, i.e. CPU, memory, and IO devices are internally modeled as single thread.

Timing in general is only approximated. No special effort has been made to create timing accurate models.

For consideration of execution timing, the model is annotated with constructs to delay the sequential execution stream, e.g. in VHDL with `wait for Tdelay`. Similarly timing is considered in subprograms encapsulating the communication.

The following ways for modeling communication have been analyzed:

- Signal based communication without handshake, i.e. special wait and delay statements have been inserted to guarantee correct communication.
  This communication style mimics hardware-like communication that abstracts real timing to simulation iterations that do not consume simulation time (e.g. in VHDL delta cycles).
- Signal based communication with handshake, i.e. handshake guarantees correct communication, no further insertion of delay is required except to allow for signal update.
- Method based communication, i.e. method calls ensure correct communication. This style was implemented in SystemC and SystemVerilog only; SystemC models use OSCI TLM communication while SystemVerilog models use interface based communication with export and import of tasks.

All communication methods and models were implemented using the following timing accuracy:

- Un-timed modeling close to the OSCI PV abstraction. If any, only simulation cycle iterations without simulation time advance were executed.
- Propagation delay based modeling close to the OSCI PVT abstraction. Delay was specified in terms of units of simulation time.
- Clock related modeling close to the OSCI CA abstraction. Timing was considered by relating execution progress to subsequent clock edges.

Altogether 9 different styles of behavioral modeling were analyzed. VHDL and Verilog supported only 6 different styles whereas SystemC and SystemVerilog support all 9 styles.

4.2.3 RT-Model

In case of VHDL and Verilog, the RT models followed the IEEE synthesis standards. VHDL, Verilog, and SystemVerilog models were synthesizable with a commercial RTL synthesis tool.

The SystemVerilog models were not just a Verilog model compiled with a SystemVerilog compiler. Several SystemVerilog specific constructs as always_ff, priority_if, or interfaces have been used.

SystemC models mimic the VHDL coding style for synthesis. SystemC RT models were not synthesized with a tool.

Two basic RT modeling styles were analyzed:

- A two process model with one process modeling the registers and one process modeling the glue logic. The model was especially tuned for execution speed.
- A structural model with several design hierarchies. The 8 bit model was partitioned in a very fine granular way. Partially RT gate models were instantiated. The 12 bit model was partitioned in a more coarse granular way, i.e. the leave elements were muxes, registers, ALUs, and FSMs. Special care has been taken to cover a wide range of modeling styles. So e.g. one FSM was implemented via table-look-up technique, another via boolean equations.

The two RT model alternatives described above were modeled in all languages (2 per language).

4.2.4 Summary of time abstraction and description styles

The classical HDLs VHDL and Verilog were analyzed in 10 different time abstraction and description style modeling alternatives, SystemVerilog and SystemC in 13 different alternatives.

All time abstraction and description style alternatives were analyzed with a set of different value representations. They are discussed in the next subsection in more detail.

4.3 Value Representation

Value representation is mostly omitted when modeling languages and styles are compared. In this study we took a strong focus on value representation as well, also motivated through the intermediary results. Generally, we distinguish 3 classes of values.

4.3.1 Abstract Un-encoded Values

Abstract un-encoded values do not possess explicit mapping to vectors or arrays of bit. So, the value of a number is represented but its encoding (e.g. one's complement signed, two's complement signed, unsigned) is not. Abstract values are represented in a packed way which allows the simulator to handle them in one step if the size of the CPU word is sufficient. Available language representations are:

- VHDL: integer, enumeration, boolean
- SystemC: int, uint, enum
- Verilog and SystemVerilog support abstract, but not un-encoded values. Independent from their semantics, bits can be set explicitly. For that reason, analysis and special handling in simulators is needed to map those values to a CPU word.

4.3.2 Bit Values

Bit values represent only the logic state of one line or one port. Bit values are supported by VHDL (bit), SystemVerilog (bit) and SystemC (sc_bit).

Bit values may be composed, for example to vectors. A composition of bit values may go hand in hand with a higher level semantic representation, e.g. a vector of 12 bit may carry the semantics of a positive or negative number.

- Vectors of bit without additional semantics are available only in VHDL (bit_vector) and SystemC (sc_bv). More complex operations than boolean operations (such as arithmetic operations) must be implemented explicitly.
- Vectors of bit with arithmetic operations are available only in VHDL and SystemVerilog. VHDL provides additional standard packages for that purpose. SystemC natively does not support arithmetic operations on bit vectors. The representations for vectors with arithmetic operations are:
    - VHDL: signed and unsigned types from package ieee.numeric_bit.
    - SytemVerilog language built-in support of signed and unsigned interpretation.

4.3.3 Meta Values

Meta values represent not only logical information but also simulation related or hardware related information. So, 'X' represents among others un-initialized values or simulation conflicts, 'Z' represents high impedance values.

The following representations have been evaluated:

- Since VHDL does not possess a built-in logic type with meta values the data types defined in the package ieee.std_logic1164 is used.
    - Both, the unresolved and resolved alternative were analyzed. Arithmetic operations were explicitly modeled in the model.
    - Signed and unsigned types supporting numeric operations on std_logic are available as standardized versions from the package ieee.numeric_std.
    - In order to identify the impact of an optimized implementation, we also analyzed a self defined logic type with meta values 'X' and 'Z'.
- Verilog possesses a built-in logic type with meta values.
- For SystemVerilog, we did not analyze the Verilog type wire/reg (which is available because of upward compatibility) but the type logic. The type logic is an extended digital 0/1-logic with the meta values 'X' and 'Z' as well. Arithmetic operations are also language inherent in SystemVerilog. We analyzed both the signed and the unsigned interpretation.
- For SystemC, we used the types sc_logic and sc_lv. For arithmetic operations we again used model specific implementations because no arithmetic support for these types is provided in SystemC.

4.3.4 Summary

In summary we analyzed the impact of 9 different value representations in VHDL, 1 value representation in Verilog, 4 value representations in SystemVerilog and 3 value representations in SystemC. All model alternatives are implemented for all time abstraction and description styles. The resulting number of models and testcases is discussed in the next section.

4.4 Overall testcases

When adding up all our modeling and language alternatives, we will end up with 191 model alternatives as shown in the next table.

	Description	Values	All together
VHDL	10	9	90
Verilog	10	1	10
SystemVerilog	13	4	52
SystemC	13	3	39

This will result in 955 different models considering our five different designs.

For simulation we execute 5 testcases for each design to avoid execution of specific code and 5 repetitions of each testcase to eliminate LSF impact on the results. Together roughly 20k test cases simulation runs will have to be executed. Unfortunately, we were not able to run all models in all environments.

5 Results and Conclusions

The findings are first presented for modeling styles, tools, and languages separately. Then accumulated results are summarized. For all tool independent comparisons, we used the fastest execution time amongst all tools.

5.1 Performance Impact of Modeling Styles

No surprise, our observations matched with the general claim that functional description with abstract un-encoded values, behavioral models with bit models, and RTL models with logic values roughly have an execution time of 1 vs. 10 vs. 100 (we measured on average 1 vs. 8,6 vs. 87,0). However we detected some interesting details:

- Hierarchical models on average showed a 1.5x slower execution speed than flat models. This points to weak tool based optimization.
  In detail, we found out that the hierarchical Verilog RTL models were on average 1.6x (1.3x to 1.7x) slower than the dual process/task models. Surprisingly, this effect could not be observed in VHDL (here the difference was on average just 1.06) even though VHDL prevents optimization of propagation of 0-delayed events through its simulation cycle.

  Further on, we found out that hierarchical sequential elements were 1.5x (1.3x to 1.8x) slower than their flat counterparts.
- The biggest difference in execution speed concerning the different value representations was approximately a factor of 2. Analyzing the single impact, abstract un-encoded values (when available) improved execution speed on average by a factor of 1.5 over bit values. Bit values in turn improved execution speed on average by a factor of 1.4 over standard simulation logic. The standard simulation logic types finally improved execution speed over user defined simulation logic types in VHDL by a factor of 1.2 on average.
  Surprisingly VHDL's unresolved std_ulogic executed slightly slower than std_logic and VHDL's bit based numeric type, numeric_bit.signed, executed up to 4.0x slower than numeric_std.signed. The trend for this observation held for all tools and was on average 2.7x.
- Object based PV TLM models executed roughly 2x slower than method based PV TLM models. That is less than expected because object based PV TLM models require a much higher effort for event handling and thread synchronization and an additional decoding step for message decoding.
- Functional models executed 4.1x faster than method based PV TLM. This shows a relatively small remaining speedup factor for SoC simulation which is left beyond the currently propagated PV style.

5.2 Performance Impact of Model Language

Performance of modeling languages is directly impacted by the optimization implemented in the executing tools. So, this comparison gives a today's view on existing environments which is an important criterion for today's decision. When the underlying engines are optioned differently in the future, this observation may change.

5.2.1 SystemC

SystemC showed to provide the slowest execution amongst all evaluated modeling languages. Surprisingly, SystemC functional models executed 2x slower than HDL models and SystemC method based PV models executed 2.6x slower than SystemVerilog models. The restrictions for using C++ optimizations for SystemC models may be the reason but at the moment we do not see how this situation might improve in the near future.

The RTL performance of SystemC models is 10x slower than those of HDL RTL models. This excludes SystemC to be an alternative RTL synthesis language compared to standard HDLs.

Furthermore, we detected a non negligible performance penalty from embedding SystemC in EDA's HDL simulators over using the OSCI SystemC kernel stand alone. Even more restricted optimization and further hooks for debugging might be the reason for that observation. This means, the performance benefit of using SystemVerilog, Verilog, or VHDL over SystemC would be even greater when we consider the SystemC runtimes of the commercial EDA tools.

5.2.2 Verilog

Verilog models showed on average 1.9x (1.5x to 2.5x) faster execution than VHDL models. Verilog RTL models executed on average 1.8x (1.5x to 2.0x) faster than VHDL models. In addition to a higher freedom in model optimization from language side and a four-state vs. a nine-state type, a higher R&D investment in Verilog tools and their optimization may be the reason for these results.

5.2.3 SystemVerilog

SystemVerilog models using two-state types executed on average 2.0x (1.4x to 2.4x) faster than VHDL (four-state models on the other hand are slightly slower than Verilog models). This shows that a richer set of language features (as typing) and stronger checking is not necessarily a reason for slower execution (as often mentioned when VHDL was compared to Verilog).

Considering that several abstraction levels are not possible using plain Verilog (e.g. bit representation, method based TLM) SystemVerilog produces the best overall results.

5.3 Tool Observations

As already stated earlier, we do not provide specific information concerning one tool. So, we summarize just some tool related items:

- As an extreme case, for one functional model we detected a factor 10 speed difference between two commercial simulators. This is one strong indicator that execution speed should be analyzed as part of the tool evaluation process.
- Each commercial tool optimized one language best. In about 95% of the cases this difference was more than factor 1.8.
- All commercial tools generally showed a faster execution speed for Verilog RTL than VHDL RTL. The speed of SystemVerilog models – as well as the currently supported language standard – differed strongly.
- We found several different interpretations of languages, bugs and tool crashes even more than 15 years after VHDL'87 and more than 10 years after Verilog'95 standardization. Unexpectedly, VHDL showed more problems than Verilog
- For that reason, huge effort was spent to make models executable on all tools. Unfortunately, this was not possible for some VHDL models and several SystemVerilog models.

5.4 Accumulated Impacts

All observations presented before focus on single modeling, language, or tool aspects. Since these aspects are not independent their individual effects do not stack directly. Instead the overall effect is usually slightly smaller.

Two overall impacts shall be presented here:

- The VHDL coding style recommended by VSIA for re-use is to use std_logic types as interface objects. However these kinds of models executed up to 2.5x slower than not std_logic based models.
- The fastest HDL RTL model (using the fastest tool and most efficient coding) executed 4.4x faster than its slowest

counterpart (whereas user defined simulation logic and numeric_bit types were not considered in the comparison).

These observations suggest that there is still a lot of space for performance optimization of HDL simulators and a lot to do for EDA R&D teams. Improvements in the following fields promise non negligible improvements:

- In-lining optimization and signal optimization at hierarchy boundaries.
- Avoidance of unnecessary execution by optimized task ordering
- Optimization of two-state execution, especially the VHDL numeric_bit package.
- Support of all C/C++ compiler switches for code linked to EDA tools
- More efficient mapping of vectors of bit to CPU words.
- Most important for SystemC, more efficient thread switching

6 Summaries and Outlook

In this paper we presented the current state of our analysis of different coding styles, modeling languages, and tools with respect to simulation performance. Major findings were:

- Tool/language/abstraction impact on RTL execution speed of factor 4.4
- Verilog RTL executed on average 2x faster than VHDL
- The execution speed of SystemC RTL models was 10x slower than HDL RTL models and surprisingly SystemC TLM models executed 2.6x slower than SystemVerilog TLM PV models.
- On average over all models and all modeling styles, SystemVerilog showed the best execution speed. This however holds for one commercial tool only.

We have also shown our analysis framework that we will continuously use to evaluate the evolution of computation power and EDA tools.

In the future, we plan to extend the evaluation in the following directions.

- First, we want to complete the missing models, especially in the behavioral domain, and add functional C models as a reference for the highest achievable execution speed of models.
- Next, we plan to consider the impact of testbench features as assertions, code coverage, functional coverage, and constraints.
- Further on, we intend to analyze the impact of language mix on the performance.
- Last but not least, we aim at measuring the impact of signal trace on the overall execution speed.

References

[1] Tom Fitzpatric: "Functional Verification Technology and Methodology Backgrounder". Available from: http://www.mentor.com/products/fv/techpubs

[2] Stephen Bailey: "Comparison of VHDL, Verilog and SystemVerilog". Available from: http://www.mentor.com /training_and_services/tech_pubs.cfm

[3] SystemVerilog: "Is this the merge of Verilog and VHDL": Proceedings of the SNUG Boston 2004.

[4] Peet James: "Synergy Between VHDL & Verilog", Proceedings of VIUF Spring '95, Users Meeting

[5] Douglas Smith .: "VHDL & Verilog Compared & Contrasted - Plus Modeled Example Written in VHDL, Verilog and C", 33th DAC. 1996.

[6] Robertas Damaševièius: "A subset based comparison of major design languages", ISSN 1392 – 124X INFORMACINĖS TECHNOLOGIJOS IR VALDYMAS, 2004, Nr.1(30)

[7] Rajesh Bawankule: "Speed up Verilog Simulations by 10-100x without spending a penny". Proceedings of the DVCon2003

[8] Phil Dreike, James McCoy: "Co-Simulating Software and Hardware in Embedded Systems". Embedded Systems Programming. http://www.embedded.com/97/feat9706.htm

[9] Röhm, W.: "Latest Benchmark Results of VHDL Simulation Systems", Proceedings of the EURODAC'95.with EUROVHDL'95.

[10] John Sanguinetti: "Simulation Speed in HDLs", http://www.angelfire.com/in/rajesh52/papers.html

[11] Matthias Bauer, Wolfgang Ecker, Michael Gasteier, Manfred Glesner: "Evaluation of Sequential VHDL and C for System Description and Specification". Proceedings of the VUIF'96 Fall Meeting.

[12] Wolfgang Ecker, Joerg Boettger. "Evaluation of Ada'95 and VHDL for System Level Modeling ", Proceedings of the VIUF '97 Spring Meeting.

[13] Mario Steinert: " Using SystemC for Hardware Design Comparison of results with VHDL, Cossap and CoCentric, Proceedings of the ESNUG2002

[14] David A. Penry, David I. August: "Optimizations for a Simulator Construction System Supporting Reusable Components", Proceedings of the 40th DAC,

[15] Rapid Prototyping of Application Specific Signal Processors (RASSP), http://www.eda.org/rassp/

[16] ARM technical paper, Taking Design to the System Level, Chris Lennard, Davorin Mista, April 2005, http://www.arm.com/pdfs/ARM%20ESL%20final%20checked%20(3)JC.pdf

[17] Cliff Cumming, Lionel Benning: "SystemVerilog 2-State Simulation and Verification Advantages", Proceedings of Boston SNUG 2004