Simulation-Based Verification of the MOST NetInterface Specification Revision 3.0

Andreas Braun, Oliver Bringmann
FZI Forschungszentrum Informatik
Systementwurf in der Mikroelektronik
Haid-und-Neu-Str. 10-14
D-76131 Karlsruhe
Andreas.Braun@fzi.de

Djones Lettnin, Wolfgang Rosenstiel
Universität Tübingen
Wilhelm Schickard-Institut für Informatik
Sand 13
D-72076 Tübingen

***Abstract*—Design and specification errors are hard to find in the traditional automotive system design flow. Consequently, these errors may be detected very late e.g. in a hardware prototype or even worse in the final product. In order to allow the verification of distributed embedded systems in early design phases, this work proposes a flexible and efficient virtual prototyping approach in order to check the consistency of system specifications. Our virtual prototyping approach has been applied to the Media Oriented Systems Transport (MOST) specification revision 3.0 and verifies the influence of two newly specified algorithms, namely Ring Break Diagnosis and Sudden Signal Off detection, with respect to numerous network configurations. In total we have verified the specification using more than 10^5 automatically generated network configurations. The overall costs for network modelling and verification compared to cost-expensive error detection and correction at later design phases have been significantly reduced.**

I. Introduction

Over the last years, the requirements of modern infotainment systems used in the automotive industry have increased rapidly. Many vehicles are now equipped with radio, cell phone, CD changer, navigation systems and other comfort functions. These devices must interact with each other to provide controllability via a central human machine interface without disturbing the driver. However, the bandwidth of the controller area network (CAN) is not sufficient for audio and video signals. The answer of the automotive industry to this gap is the standardization of the Media Oriented Systems Transport (MOST) [13].

The MOST-Bus system provides high bandwidth up to 150 MBit/sec., and it is developed for transferring streaming and controlling data especially for infotainment systems. Like other modern communication systems (e.g., FlexRay [14]), it has a high complexity to provide different needs of the connected devices. This leads to enormous design and verification costs in the well known V-model. Multiple iterations during the development process are required resulting in an increasing time-to-market. By the fact that statistically more than forty percent of the errors in the overall development process are located in the specification [1], the cost and time-to-market can be reduced significantly by using early verification techniques.

[1]This work was supported in part by the ARTEMIS project CHESS under grant 01IS09003A.

In order to verify the functional correctness, fault and timing behaviour of a system, formal and simulation-based verification can be applied. The basic advantage of formal verification using model checking is that it provides completeness. For instance, model checking tools such as Uppaal [3] and SPIN [2] are able to verify real time properties and system protocols, respectively. However, the main limitation of model checking approaches is the state space explosion (i.e., memory problems) due to the design complexity. Therefore, the classical formal techniques are still too labour-intensive to be widely applicable for the verification at system level. On the other hand, the simulation-based approach can handle large systems. Functional behaviour (e.g., time constraints) can be checked using defined assertions. It is also possible to assess dependability by the use of fault injection in early phases.

In this paper we present a new simulation-based verification approach linking the MOST system requirements and virtual prototyping. Our new approach is suitable for automated verification of system configurations as well as system functionalities. This novel verification technique is evaluated on two newly specified algorithms taken from the MOST Specification Revision 3.0, namely Ring Break Diagnosis (RBD) and Sudden Signal Off (SSO) detection. The virtual prototype is based on SystemC [15], which provides an accurate simulation of the timing and functional behaviour at system level. Additionally, distributed embedded systems and systems-on-chips (SOC) can be modelled and simulated in order to test concurrency and timing aspects.

The remainder of this paper is organized as follows: Section II discusses the related work. Section III describes our proposed verification approach. In Section IV, we introduce the modelling of the automotive MOST applications. Section V presents our experimental results. Finally, Section VI concludes the paper and describes a brief outlook on future work.

II. Related Work

There are several works in the verification of system level designs. By Kim [9] a simulation-based verification approach of the FlexRay communication controller is proposed. The communication data is transmitted bit per bit over the virtual bus. This leads to insufficient simulation performance for an adequate coverage of the design space and a high modelling effort. [7] gives an overview on simulation performance and modelling effort referred to the level of abstraction. Verification coverage and test automation is not considered in the approach proposed by Kim. [8] describes a network on chip simulation and verification platform. In this approach, the abstraction level is raised to increase simulation performance. Automated verification is supported by network traffic generation but it suffers in case of automatic platform configuration. In order to test a new configuration, the design has to be manually modified. Assessment of the verification quality by coverage is not considered. Habibi [6] describes a verification approach for transaction level models (TLM) on a PCI bus example. The verification process is supported by automated test case generation. The platform can be automatically configured with a various number of masters and slaves. He considers coverage and also includes assertion-based verification by using the property specification language (PSL) [20].

Our work is very similar to the approach proposed by Habibi. We improved the assertion-based verification by using finite linear time temporal logic (FLTL) [21], an extension on LTL with time bounds to consider timing constraints. Our simulation model uses an abstract level of TLM for accelerating the simulation. The platform can be configured by various parameters e.g. the number of network devices and device configuration. The test cases are also generated automatically and the code coverage is assessed during our proposed simulation-based verification approach for distributed embedded systems.

III. MOST Verification by Virtual Prototyping

MOST is a computer networking standard and it is developed by the MOST Cooperation. It was specially developed to interconnect multimedia components in automobiles. The MOST specification defines all seven layers of the ISO/OSI Reference Model. Normally the devices are connected to each other by a ring topology, but star configurations and double rings for critical applications are also possible. A MOST network consists of exactly one TimingMaster (TM) and 1 up to 63 TimingSlaves (TS). The TM is responsible for the generation and transport of the system clock. Every device itself consists of an intelligent network interface controller (INIC) and an external host-controller (EHC). The INIC is responsible for the data link layer of the ISO/OSI Reference Model, the EHC covers the higher layers of the reference model.

By the specification revision 3.0 [13] a new MOST generation is introduced. The bandwidth is increased up to 150 MBit/sec., data transport is exclusively done via a fiber optic transmission medium. New functionality such as the ring break diagnosis and the sudden signal off detection is provided. These new functionalities are integrated to the NetInterface as embedded software. The expression NetInterface stands for the entire communication section of a device, the physical interface, the INIC and the network services. The algorithms of the NetInterface are specified by state diagrams and additional textual explanation of the behaviour and functionality. Our main goal for the verification process is to simulate the specified algorithms and to check the properties and requirements taken from the specification document by using virtual prototyping in early development phases.

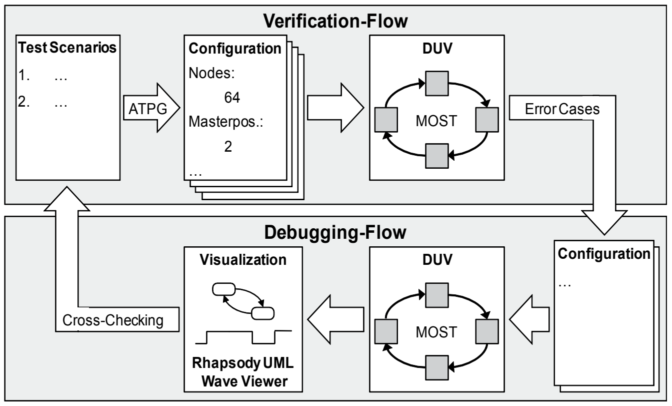

Fig. 1. Verification Flow

Our verification flow for verifying the NetInterface is shown in Fig. 1. The first step is to define several test scenarios such as error free or ring break scenario. Out of these scenarios an optional number of test cases are generated automatically by using an automatic test pattern generation (ATPG). These test cases are executed and checked automatically by the test bench to guarantee automated and fast verification. If an error occurs during automated testing, the identification (ID) of the failing test case is stored for later analysis. In order to simplify analysis, the test bench provides analysis functions such as tracing and co-simulation for visualization. During the verification runs these functions (e.g., tracing) are disabled in order to accelerate the verification process. In the analysis mode every test case is considered exclusive. The tracing produces files in a text and a value change dump (VCD) format. Additionally, a co-simulation shows the dynamic behaviour through visualization and highlighting of abstract unified modelling language (UML) models of the system. This analysis phase allows determining whether the failure is an error in the specification or in the test bench. By using this approach, erroneous behaviour, live- and deadlocks, timing errors, worst cases and other unwanted behaviour can be detected in different network configurations.

The test bench itself consists of different elements. The main component is the design under verification (DUV), which includes the basic MOST model and the specified algorithms of the NetInterface. For checking specified properties a SystemC temporal checker monitors the DUV. For determining the code coverage of the specified algorithms the profiling options of the compiler are used (refer to Section III-D).

The following section describes in detail the integrated components for the verification. The scenarios, the input parameters for the test bench and the verified algorithms of the NetInterface are described in Section IV.

A. MOST NetInterface Model

The NetInterface (see Fig. 2) is modelled in the system description language SystemC. The SystemC modelling language is a set of library routines and macros implemented in

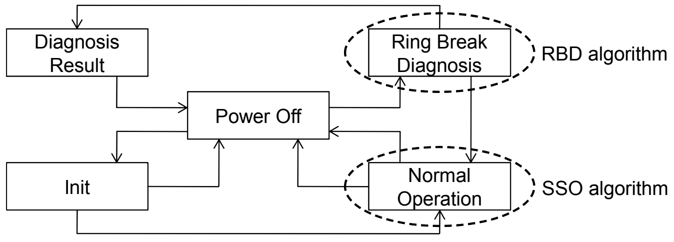

Fig. 2. Top Level Statechart of the NetInterface

C++. It is known as a system description language especially for TLM. TLM is a high-level approach for abstract modelling of communication among modules. SystemC provides accurate simulation of the timing and functional behaviour. This allows a fast and abstract modelling at system level, so that it is possible to implement new systems directly from their specification without any knowledge of the chip architecture. Additionally, distributed embedded software in the form of state machines or even C code could be integrated very fast. SystemC together with C/C++ provides full functionality for software integration in contrast to using VHDL and Verilog. The connection between the devices of a network is modelled by TLM techniques. This increases simulation speed and ensures faster verification of the model.

The MOST system model itself implements two main device forms, the TM and TS device. These devices are modelled as SystemC modules. Each of the modules consists of an abstract INIC. The EHC is replaced by a small environment simulation (see Fig. 3). The specified NetInterface algorithms are encapsulated in these modules so that it is possible to instantiate an configurable amount of TM and TS devices. The NetInterface algorithm in a device is triggered by using a given time parameter. The NetInterface trigger and other timers used by the algorithm are calculated in a timer function which can start, stop and indicate a timer overrun.

Every device is connected via SystemC channels in a ring topology. A device or a channel delay is optional. The data transfer and synchronization to a TM within devices is managed by several functions. If a device is locked to a signal transported via the fiber optic medium, it can read and write to the bus. If a device is unlocked, it can neither read nor write data exchanged via the bus. By using fault injection techniques it is possible to simulate device breakdowns, ring breaks and excessive attenuation on the transmitting medium. Therefore, a so-called saboteur is integrated to the channel [19].

The configuration of the simulation platform is done by reading XML files. By reading this file the network simulation is instantiated, configured, started and the channels are interconnected automatically. To prevent an unlimited runtime in the case of deadlocks and livelocks, the simulation time is bounded by an additional value. The XML file itself depends on the test case and defines several configuration parameters that are listed below:

- *Number and position of TM and TS*
- *NetInterface trigger time for each device*
- *Sample rate for frame generation*

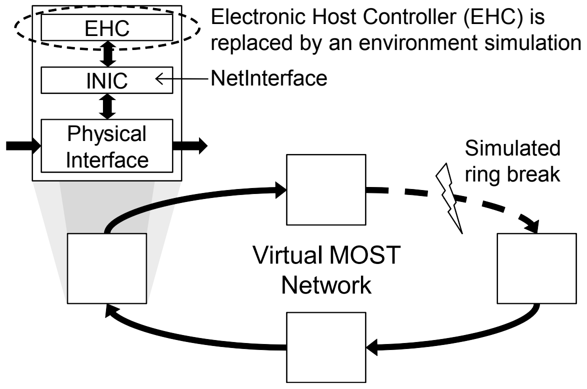

Fig. 3. Simulated MOST Ring

- *Fault injection locations, times and events*
- *Timer values of the integrated timers for each device*
- *Several events for the environment simulation*
- *Maximum simulation time*

The simulation performance of the NetInterface model depends on the number of instantiated virtual devices and the level of abstraction used for modelling. Usually a higher level of abstraction accelerates the simulation, but this also limits the number of detectable errors. So it is important to find a good trade-off between simulation performance and the level of abstraction. The NetInterface algorithm is directly implemented as a function of the device model. The function itself is triggered in realistic time steps but the execution time of the software is set to zero. Commands by the NetInterface are executed immediately and no latency is modelled for them. The data transfer is done via TLM interfaces which directly transmit uncoded parts of the specified frames. This accelerates the simulation speed significantly in comparison to bit per bit transmission e.g. as realized in register transfer level (RTL), but it leads to a limited timing accuracy. The locking process of the MOST phase-locked loop (PLL) is reduced to a simple frame counter in the model. If a device receives three stable frames its status is changed to lock. Another decisive abstraction is the neglecting of higher layers of the communication model. Events normally generated by higher layers are simulated using a small environment model included in the devices.

B. Automatic Test Pattern Generation

Test case generation can be subdivided into two main techniques: random and partition testing. By using random testing the input space is searched randomly. The constrained and weighted randomization improves the probability for generating corner cases. However, there is no assurance that these cases are sufficient. The only way to assess the generated cases is to cover the input space. In this sense double test cases can be avoided and the number of test cases can be limited by a coverage requirement. Partition testing searches the input space deterministically. This assures that the created test cases have a higher quality referred to redundancy, failure sensitivity, representativeness and efficiency. But partition testing is more expensive than simple random testing because of the automation difficulty. The aspects of random vs. partition testing are discussed in [23] and [24] in more detail.

In our verification approach we used a simple ATPG based on random testing. The generator is based on the SystemC verification library (SCV). The SCV provides a pseudo random generator with data introspection, constraint and weighted randomization. The inputs of the ATPG is the defined test scenarios, for randomization of the timer parameters two different distributions can be chosen, equal and Gaussian distribution. Furthermore several key parameters can be chosen to generate a demanded number of test cases. E.g., the parameter variations are used as a multiplicator for the defined possibilities, the parameter ring break event determines whether the injected ring breaks are static or dynamic. Static means that the ring break is injected before the simulation starts without calling it off during simulation. An example for a scenario definition is given in Table I.

The generated case data is stored in XML files and could be loaded as configuration data. The failure injection events are also stored by these files and no randomization during runtime is used. One disadvantage of this realization is that the size of the XML file increases. The main advantages are that the automated verification is accelerated and it is possible to assess the test case before simulation.

C. Verification of Temporal Properties at System Level

The temporal properties are formalized based on the state charts from the NetInterface specification. In order to verify temporal properties, the C/C++ language does not support any mechanisms for temporal properties or assertions in software modules. Therefore we integrate an existing SystemC temporal checker (SCTC). The SCTC supports checking of temporal properties either in PSL [20] or FLTL [21]. An example for the property definition in FLTL is shown in Equation 1. Thereby a requirement referred to the time difference between two events is described, one thrown by a TM and the other by TS.

$$\begin{aligned} &F(TM.timer == active) -> \\ &F[Timeout](TS.timer == active) \end{aligned} \tag{1}$$

D. Coverage Analysis

Coverage analysis can be subdivided into three main coverage forms [18], functional coverage, parameter coverage and code coverage. The Functional coverage is done by the use of the SCTC together with the distinction of the different scenarios, because every scenario is referred to only one defined functionality (e.g. a ring break scenario results in a specified behaviour). Parameter coverage would be useful to assess the generated test cases and to reduce the verification time, but no suitable tool was available. The code coverage is assessed by using the GNU profiling options and additional open source tools, this results in a branch coverage test which is accepted as the minimum criteria [17]. Unused transitions of the NetInterface state chart specification can be detected.

E. Tracing Approach

To analyze and assess simulation runs tracing is essential. Therefore a central record class was developed which is interfaced to all relevant data of the DUV by using a generic notification approach. This class is responsible for monitoring and storing all variable and state changes of every instantiated device together with timestamps. At the end of the simulation all the information is written to files. Two different formats are provided, one is a VCD format which can be visualized by a wave viewer e.g. GTKWave and the other format stores data in plain text. During automated verification the tracing can be disabled to accelerate simulation.

F. Co-Simulation

Co-simulation is provided for better analysis of single test cases. It shows the dynamic behaviour by visualization and highlighting the abstract UML models of the NetInterface. Beside the visualization it provides read access to the same variables and states monitored by the tracing class. The co-simulation depends on the UML tool Rhapsody. It provides break points and a step by step mode. The data transfer between the test bench and Rhapsody is event triggered by state or variable changes, this ensures sufficient simulation speed.

IV. MOST Design Application

In the following two verification examples RBD and SSO are described in detail. Both algorithms are a part of the whole NetInterface algorithm (see Fig. 2). During simulation phase each possible number of devices from 2 up to 64 was tested. The sample rate was set to 44.1 or 48 kHz. Faults are either injected at the beginning or during the simulation. The timers are set to a value of their specified range.

Depending on the tested scenario the diagnosis info, status in result state and the time difference for entering the result states are checked by the SCTC for all instantiated devices. Additionally, the relative ring position referred to the TM which is determined by the algorithm and used for error location or as an address is checked. The verification results are explained in Section V.

A. Ring Break Diagnosis Application

The RBD algorithm was introduced to detect different kinds of errors (shown in Table II) and their location before system start up. In case of errors the system is lead to the NetInterface state *Diagnosis Result* to analyze the detected error. If no error is detected the system would change its state to *Normal Operation*.

The main goal of this verification is to check the textual function description against the state chart description of the specified algorithms. Therefore six scenarios have been defined, error free, ring break, excessive attenuation, multi master, all slave and combination. In the case of the first five scenarios the expected results are well defined. The combination case is not exactly defined in the specification, but with the proposed verification flow it was possible to determine the behaviour of the algorithms in the case of multiple errors. For test case generation each scenario was treated equal, approximately 25,000 cases have been generated for each scenario. That makes up to 150,000 different configurations for

TABLE I
EXAMPLE OF A DEFINED RBD SCENARIO

Number of Devices	Number of TM	Number of Ring Breaks	Number of Exc. Attenuation	Sample Time [kHz]	Distribution	Ring Break Event	Exc. Attenu-ation Event	Variations	Total Cases
2, 3, 63, 64	1	1	0	44.1, 48	gaussian	static	static	250	2000
4 – 62	1	1	0 – 1	44.1, 48	equal	static	st., dyn.	10	2360
2, 64	1	1, 2	0	44.1, 48	gaussian	dynamic	static	250	2000
60 – 64	1	2 – 4	0	44.1, 48	gaussian	st., dyn.	static	50	1500

TABLE II
RING BREAK DIAGNOSIS RESULTS

Diagnosis Result	Diagnosis Info	Description
No error	-	No error detected. Transition to Normal Operation.
Position detected	Relativ Position	Ring break or excessive attenuation detected. The result indicates the rel. position.
Diagnosis failed	-	Number of TimingMaster is not equal to one or no valid rel. ring position is available.

the whole RBD verification. The maximum system simulation time was set to 3500 msec. and 8 timers are configured for each device.

B. Sudden Signal Off Detection Application

The SSO algorithm was introduced to guarantee a suitable shutdown sequence either by a normal shutdown caused by the PowerMaster (PM) or in the case of errors. These errors can be: an emergency shutdown caused by low voltage or a ring break or excessive attenuation. In all cases the cause of the shutdown is saved for determining the reason and the location. Afterwards the state of the NetInterface is changed to *Power Off*.

Normally, shutdown by the PowerMaster is initiated by a request and answer process of all network devices. This depends on functionality integrated to higher layers of MOST. By the fact that these layers are neglected to improve simulation performance, initiation is done by the simulation environment.

Five scenarios have been defined and tested: shutdown by PM, sudden signal off, excessive attenuation, emergency shutdown and combination. The combination scenario is not specified by the SSO description as well, but it could also help to increase the knowledge about the algorithms behaviour. For each scenario 30,000 test cases have been generated this makes a total sum of up to 150,000 test cases. The maximum system simulation time was set to 800 msec., so that the SSO verification was less time consuming compared to the RBD verification. For the SSO detection 8 timers are configured for each device. To accelerate the simulation the devices start the simulation directly in NetInterface state *Normal Operation*.

V. EXPERIMENTAL RESULTS

For verification of the RBD and SSO algorithms, approximately 150,000 test cases where generated and executed for each algorithm. As execution platform two DELL PE2950 III Quad-Core Xeon E5320 boards with 1.86 GHz clock frequency have been used. Altogether eight cores were available for parallelization of the virtual prototyping approach. This was sufficient to test the whole cases for the RBD verification within 5 days. The SSO algorithm was verified faster, because the algorithm is less complex and the worst case execution time of the algorithm is shorter referred to the RBD algorithm. The verification results of the SSO algorithm were performed in 3 days. In comparison to the simulation time the most time was spent for error analysis, explanation and description during the verification process. Therefore the developed analysis functions of the model have become very useful. The analyzed errors which have been found relating to the verification of the specification are listed and described in the following. Table III and IV give a short overview of the used scenarios referred to the analyzed and listed errors.

1) *(RBD) Wrong error location*: Referred to the specification the RBD algorithm should detect the position of a ring break if one exists. In some cases the algorithm was not able to locate the correct position.
2) *(RBD) Wrong relative ring position*: During RBD the algorithm is also used for determining the relative ring position referred to the TM. In NetInterface state *Normal Operation* the position can be used as an address. In case of an error the relative position is used for error location.
3) *(RBD) Multiple TM after diagnosis*: In NetInterface state *Normal Operation* a functional ring can only have one TM. That is also expected for NetInterface state *Diagnosis Result*.
4) *(RBD) Bad calibration of timer DiagSignal*: The overrun of the timer DiagSignal indicates that no signal at the devices input is available (ring break). This leads the algorithm to states for ring break analysis. The problem here is that other timers did not correlate to DiagSignal. The result is an unexpected behaviour.
5) *(SSO) Wrong error location*: The SSO algorithm also has the ability to locate different errors. At a certain number of devices, dependent on the timer configuration, the algorithm was not able to locate the error position right.
6) *(SSO) Wrong error reasons*: Under certain circumstances the SSO algorithm stores two error reasons at different locations. This effect is caused by the locking behaviour of the MOST devices.
7) *(SSO) Unpermitted self restart*: The unpermitted self restart was a really unexpected error. It means that the simulation presented the possibility that the MOST network was able to restart the ring during shutdown,

TABLE III
SIMULATION RESULTS (RBD)

Scenario	Test Cases	Error Cases	Error
Error Free	26240	-	-
St. Ring Break	17480	134	1)
		48	4)
Dyn. Ring Break	8040	1543	1)
		23	4)
St. Exc. Attenuation	17480	-	-
Dyn. Exc. Attenuation	8040	-	-
Multi Master	26200	685	2)
		913	3)
All Slave	24240	314	2)
		479	3)
Combination	26480	-	-

TABLE IV
SIMULATION RESULTS (SSO)

Scenario	Test Cases	Error Cases	Error
Shutdown by PM	31480	-	-
Sudden Signal Off	29120	14	7)
St. Exc. Attenuation	22000	78	5)
		240	6)
		2	7)
Dyn. Exc. Attenuation	8440	212	5)
		756	6)
Emergency Shutdown	29120	34	7)
Combination	32400	8	7)

which leads the devices to NetInterface state *Init*.

In addition to the located errors the simulation has helped finding and understanding unknown effects and worst cases. One important effect to know is that the nodes react on each other in the communication direction of the ring. Under consideration that there is a worst case reaction time of each device depending on the NetInterface algorithm trigger, this reaction time can sum up to a significant time effecting the whole timer calculation.

VI. CONCLUSION AND FUTURE WORK

The growing complexity of modern automotive networks leads to an increasing number of potential error cases which can firstly be detected at a real hardware prototype or even worse in the final product. This results in time consuming redesigns or changes in the initial specification. In this work, we have presented a verification approach for the upcoming MOST specification revision 3.0 by virtual prototyping. By automatization of the test bench various characteristics of the network can be analyzed and verified with minimal modelling and verification effort. We have verified the functional and timing behaviour of the MOST network under consideration of test scenarios with and without failure injection. As a result, several specification errors of the newly specified algorithms Ring Break Diagnosis and Sudden Signal Off were discovered and reported to the MOST cooperation. The MOST cooperation has already taken the reported errors into account in the final revision 3.0 of the specification. The proposed verification approach can also be applied to other network technologies and is easy to insert in the automotive design flow. This can help to reduce verification costs and error detection in early development phases. Future work will focus on improving our coverage analysis for better assessment of the verification. One approach is to develop a coverage driven test pattern generation to find a sufficient number of different test cases.

ACKNOWLEDGMENT

The authors would like to thank the MOST Cooperation for supporting the described work.

REFERENCES

[1] M. Kieviet. *IEC 61508 Umfassende Sicherheit für Maschinen und Anlagen*. http://www.innotecsafety.de.
[2] Bell Labs. http://spinroot.com/spin/whatispin.html.
[3] K. G. Larsen. *Formal Methods for Real Time Systems: Automatic Verification & Validation*. ARTES '98.
[4] D. Grosse and R. Drechsler. *Formal verification of LTL formulas for SystemC designs*. ISCAS '03: In Proceedings of the International Symposium on Circuit and Systems.
[5] J. Jimenez, E. Fernandez, J. L. Martin, U. Bidarte and A. Zuloaga. *Simulation environment to verify industrial communication circuits*. IECON '02: IEEE 28th Annual Conference of the Industrial Electronics Society.
[6] A. Habibi and S. Tahar. *Design and Verification of SystemC Transaction-Level Models*. TVLSI '05: Transaction on Very Large Scale Integration Systems.
[7] S. Pasricha, N. Dutt and M. Romdhane. *Using TLM for Exploring Bus-based SoC Communication Architectures*. ASAP '05: 16th IEEE International Conference on Application-Specific System, Architecture Processors.
[8] S. Chai, C. Wu, Y. Y. Li and Z. Yang. *A NoC Simulation and Verification Platform Based on SystemC*. CSSE '08: International Conference on Computer Science and Software Engineering.
[9] W. S. Kim, H. A. Kim, J. H. Ahn and B. Moon. *System-Level Development and Verification of the FlexRay Communication Controller Model Based on SystemC* FGCN '08: Second International Conference on Future Generation Communication and Networking.
[10] D. Lettnin, P. K. Nalla, J. Ruf, T. Kropf and W. Rosenstiel. *Verification of Temporal Properties in Automotive Embedded Software*. DATE '08: Design, Automation and Test in Europe.
[11] A. Habibi, A. Gawanmeh and Sofiene Tahar. *Assertion Based Verification of PSL for SystemC Designs*. SOC '04: International Symposium on System-on-Chip.
[12] D. Lettnin, M. Winterholer, A. Braun, J. Gerlach, J. Ruf, T. Kropf and W. Rosenstiel. *Coverage Driven Verification applied to Embedded Software*. VLSI '07: IEEE Computer Society Annual Symposium on ISVLSI.
[13] MOST Cooperation. http://www.mostcooperation.com.
[14] FlexRay Consortium. http://www.flexray.com.
[15] Open SystemC Initiative. http://www.systemc.org.
[16] T. Kropf. *Introduction to Formal Hardware Verification*. Springer-Verlag '98.
[17] P. Liggesmeyer. *Software Qualität: Testen, Analysieren und Verifizieren von Software*. Spektrum Akademischer Verlag '09.
[18] W. K. Lam. *Hardware Design Verification:Simulation and Formal Method-Based Approaches*. Prentice Hall Computer '08.
[19] S. Misera, H. Vierhaus and A. Sieber. *Fault Injection Techniques and their Accelerated Simulation in SystemC*. DSD '07: 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools.
[20] Accelera. *Property Specification Language (PSL)* Version 1.1 '04.
[21] J. Ruf, D. W. Hoffmann, T. Kropf, and W. Rosenstiel. *Simulation-guided property checking based on multi-valued AR-automata*. DATE '01: Design, Automation and Test in Europe.
[22] H. Moinudeen, A. Habibi and S. Tahar. *Model Based Verification of SystemC Designs*. IEEE '06: North-East Workshop on Circuits and Systems.
[23] J. W. Duran and S. C. Ntafos. *An Evaluation On Random Testing*. IEEE '84: Transactions on Software Engineering.
[24] E. J. Weyuker and B. Jeng. *Analyzing Partition Testing Strategies*. IEEE '91: Transactions on Software Engineering.
[25] S. Srinivasan and N. Vijaykrishnan. *Transaction level error susceptibility model for bus based SoC architectures*. ISQED '06: 7th Internation Symposium on Quality Electronic Design.