This is the author's version of the work. The definitive work was published in *Proceedings of International Conference on Computer-Aided Design (ICCAD), pp. 185-189, 2010.* The work is supported in part by the German Science Foundation (DFG), SFB 694.

Symbolic System Level Reliability Analysis

Michael Glaß, Martin Lukasiewycz, Felix Reimann, Christian Haubelt, and Jürgen Teich
University of Erlangen-Nuremberg, Germany
{glass,martin.lukasiewycz,felix.reimann,haubelt,teich}@cs.fau.de

***Abstract*—More and more embedded systems provide a multitude of services, implemented by a large number of networked hardware components. In early design phases, dimensioning such complex systems in terms of monetary costs, power consumption, reliability etc. demands for new analysis approaches at the electronic system level. In this paper, two symbolic system level reliability analysis approaches are introduced. First, a formal approach based on Binary Decision Diagrams is presented that allows to calculate exact reliability measures for small to moderate-sized systems. Second, a simulative approach is presented that hybridizes a Monte Carlo simulation with a SAT solver and delivers adequate approximations of the reliability measures for large and complex systems.**

I. Introduction

In modern means of transport such as automobiles, airplanes, and trains, a multitude of applications that range from safety-critical to comfort applications are implemented as *networked embedded systems*. These systems typically consist of many hardware resources that interact with each other to a large extend. A major threat to the reliability of these components and, thus, to the entire system are (a) *ever shrinking device structures* and (b) their *places of activity*, i.e., hardware components move from dedicated and protected mounting spaces to installation spaces near sensors, moving parts, or even within the engine. The former leads to components being susceptible to high temperatures, cosmic rays, manufacturing tolerances etc. The latter exposes the components to *destructive agents* inducing accelerated aging processes and *stress* that, thus, leads to a significant increase of their defect rate. Together, this results in a growing inherent *unreliability* of embedded system components. On the other hand, customers surveys reveal that *reliability* often constitutes the most important purchase criterion, cf. for example [1]. Thus, an appropriate trade-off between several design objectives like monetary costs, energy consumption, volume consumption, latency, and lifetime reliability becomes mandatory.

Paper Goal. The work at hand gives an introduction into two complementary system level reliability analysis approaches that are tailored to be used within an optimization process of embedded systems during early design phases. The first approach that is based on *Binary Decision Diagrams* (BDDs) allows to calculate exact reliability measures for small to moderate-sized systems. The second approach copes with present scalability issues of BDDs by hybridizing a Monte Carlo simulation with a *Satisfiability* (SAT) solver. The SAT-assisted simulation delivers adequate approximations of the reliability measures for large and complex systems where the BDD-based approach fails.

Outline. The rest of the paper is outlined as follows: Section II gives an overview of the analyses described in this work and their application domain. The used system model is presented in Sec. III. Section IV introduces the BDD-based reliability analysis approach while the SAT-assisted simulation is presented in Sec. V. Experimental results are discussed in Section VI while Section VII concludes the paper and provides references for further reading.

Supported in part by the German Research Foundation (DFG), SFB 694.

II. Overview

The discussed reliability analysis approaches are tailored to the *design space exploration* of embedded systems at system level. The task of design space exploration is to find the set of optimal *feasible implementations* for a given *specification* with respect to several, often conflicting *design objectives*. In the used approach, cf. [2], a set of *applications* is mapped to an *architecture* following the well-known *Y-chart* approach [3]. At the highest level of abstraction in early design phases, focus is put on the capability of the used analysis approaches to investigate thousands of possible design options within a reasonable amount of time while providing acceptable accuracy.

The reliability analysis approaches discussed in this work focus on the consideration of *permanent resource defects*.[1] In particular, a redundant task binding as the used technique to cope with resource defects is assumed. The analysis approaches, moreover, assume that the components of the system are *independent*, i.e., the fault of one component does not influence the reliability function of another component. Of course, functional dependencies, e.g., a component fails as soon as its power supply fails, are respected by the analysis approaches. Although the assumptions may seem stringent at first glance, it should be noted that the analysis is applied at a very early stage in the design process where several necessary design decisions for more detailed investigations may not have been taken yet. Thus, the proposed approaches can be seen as the top of a *hierarchical analysis* approach that is refined during the design process. Consider an automotive network where tasks are executed on *Electronic Control Units* (ECUs). The defect of each ECU is given by a reliability function and because of the remote mounting spaces, it can be expected that the ECUs are independent of each other. In a subsequent design step, the number of cores, the assignment of tasks to cores, and local schedules of each ECU are determined. At the ECU level, the assumption of independent components does not hold anymore [5] and more sophisticated but, however, less scalable analysis approaches need to be applied there, cf. for example [6]. This in depth analysis now delivers a revised reliability function for the ECU that is propagated back to the system level analysis. Thus, the proposed analysis approaches have the ability to increase their accuracy during

[1] Orthogonally, for the investigation and toleration of *soft errors*, several system level mapping techniques and analysis approaches that focus on scheduling have been developed in recent years, cf. for example [4].

the design process. In particular, the analysis approaches are capable of handling arbitrary reliability functions used to model the components. Thus, several important influences on the reliability of the components such as aging or temperature effects can seamlessly be modeled by means of sophisticated distributions such as WEIBULL distribution, lognormal distribution, or linear combinations of these.

III. SYSTEM MODEL

For the reliability analysis that is performed during design space exploration, a model of the available hardware components as well as the tasks that need to be distributed in the system is defined. This graph-based specification consists of the *applications*, the *architecture*, and a relation between these two views, the *mapping edges*:

- The applications are modeled by a graph $g_{\mathsf{t}}(T, E_{\mathsf{t}})$ that represents the functional behavior of the system. The vertices $t_1, ..., t_{|T|} \in T$ denote *tasks* while the directed edges E_{t} model data dependencies between tasks.
- The architecture is modeled by a graph $g_{\mathsf{a}}(R, E_{\mathsf{a}})$ that represents the available interconnected hardware components. The vertices $r_1, ..., r_{|R|} \in R$ represent the *resources*, e.g., ECUs, buses, sensors, or actuators. The edges E_{a} model available communication links between resources.
- The relation between architecture and application is modeled as a set of *mapping edges* E_{m}. Each mapping edge $m_1, ..., m_{|E_{\mathsf{m}}|} \in E_{\mathsf{m}}$ is a directed edge from a task to a resource. A mapping $m = (t, r) \in E_{\mathsf{m}}$ indicates that a specific task t can be executed on a resource r.

From the specification of the system that includes all design alternatives, an *implementation* has to be deduced. This implementation corresponds to the actual embedded system that will be implemented and consists of two main parts:

- The *allocation* $\alpha \subseteq R$ is a subset of the available resources and represents the resources that are actually used and implemented in the embedded system.
- The *binding* $\beta \subseteq E_{\mathsf{m}}$ is a subset of the mapping edges in which each task is *bound* to *at least one* hardware resource that executes this task at runtime:

$$\forall t \in T : |\{m | m = (t, r) \in \beta\}| \geq 1 \tag{1}$$

The *task instances* that result from this *multiple binding* enables a designer-transparent investigation of redundancy at task level that allows to avoid costly resource replications, cf. [7].

Definition 1: A binding is called *feasible* if it guarantees that all data-dependent tasks are executed on the same or adjacent resources to ensure a correct communication:

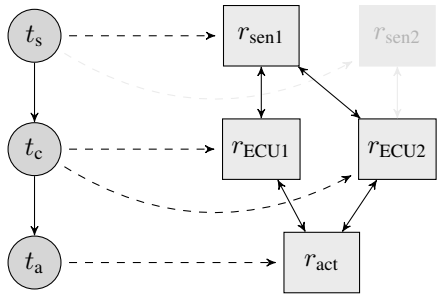
$$\begin{aligned} \forall (t, \widetilde{t}) \in T : \exists\, m = (t, r), \widetilde{m} = (\widetilde{t}, \widetilde{r}) \in \beta : \\ (r = \widetilde{r}) \vee ((r, \widetilde{r}) \in E_{\mathsf{a}}) \end{aligned} \tag{2}$$

Thus, an implementation ω is defined as a pair (α, β), with a *feasible* implementation requiring a feasible binding for the given allocation, cf. Fig. 1.

To enable the reliability analysis of a given implementation, the components are annotated with properties in the specification. The lifetime reliability of each resource $r \in R$ is modeled as a *reliability function* $\mathcal{R}_r : \mathbb{R}_0^+ \rightarrow \mathbb{R}_{[0,1]}$ that returns the probability of the lifetime τ_{LT} of the resource being greater than a certain time τ:

$$\mathcal{R}_r(\tau) = \mathcal{P}[\tau_{LT} > \tau] \tag{3}$$

It holds that $\mathcal{R}(0) = 1$ and $\mathcal{R}(\infty) = 0$, i.e., a resource provides correct service at time 0 and will eventually fail.



(a) application (b) architecture

Fig. 1. A feasible implementation of a simple system specification. The application (a) consist of a sensor task t_s, a control task t_c, and an actuator task t_a. The architecture (b) consists of two different sensors suitable to carry out the sensor task, two ECUs suitable to execute the control task and an actuator that carries out the actuator task. The possible mapping of tasks to resources is depicted by the dashed edges with resources that are not allocated and mappings that are not activated being shown in gray. The feasible implementation itself includes a multiple binding of the control task t_c to both ECUs r_{ECU1} and r_{ECU2}. This implementation allows to tolerate the defect of one of the two allocated ECUs.

IV. BDD-BASED RELIABILITY ANALYSIS

In this section, the BDD-based reliability analysis approach as proposed in [7] is introduced in two steps: First, it is described how the *structure function* φ [8] of the implementation can be generated from the system model and represented by *Binary Decision Diagrams* (BDDs) [9]. BDDs are an efficient representation of Boolean functions by means of a directed acyclic graph with one root and two sinks, the 0 and 1 sink. Traversing the BDD from the root to the sink determines if the Boolean function evaluates to 0 or 1 based on the path, determined by the assignment of the variables that are represented as vertices. After the structure function is generated, φ can efficiently be evaluated to determine the reliability of the system in terms of the *Mean-Time-To-Failure* (MTTF).

A. The structure function φ

To model the behavior of the system under the influence of defects, the structure function $\varphi : \{0,1\}^{|\alpha|} \rightarrow \{0,1\}$ with the Boolean input vector $\boldsymbol{\alpha} = (\boldsymbol{r_1}, \ldots, \boldsymbol{r_{|\alpha|}})$ is determined. This Boolean function indicates a system providing correct service by evaluating to $\varphi = 1$ and a system failure by evaluating to $\varphi = 0$, respectively. For each allocated resource $r \in \alpha$, a binary variable $\boldsymbol{r}$ is introduced with $\boldsymbol{r} = 1$ indicating a correct service and $\boldsymbol{r} = 0$ a resource defect, respectively. For a given implementation $\omega = (\alpha, \beta)$, φ is determined by

$$\varphi(\boldsymbol{\alpha}) = \exists \boldsymbol{\beta} : \psi(\boldsymbol{\alpha}, \boldsymbol{\beta}) \tag{4}$$

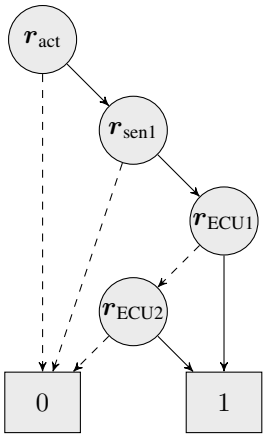


Fig. 2. The structure function φ of the implementation in Fig. 1, encoded as a Binary Decision Diagram (BDD). The regular edges correspond to the variable of the corresponding node being 1, while the dashed edges correspond to the variable being 0, respectively.

with $\psi(\boldsymbol{\alpha}, \boldsymbol{\beta})$ being the *characteristic function* of the implementation that is given by:

$$\psi(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \bigwedge_{t \in T} \left[\bigvee_{m=(t,r)\in\beta} \boldsymbol{m} \right] \wedge \tag{5a}$$

$$\bigwedge_{m=(t,r)\in\beta} \boldsymbol{m} \rightarrow \boldsymbol{r} \wedge \tag{5b}$$

$$\bigwedge_{(t,\widetilde{t})\in E_{\mathrm{t}}} \bigwedge_{\substack{m=(t,r),\\ \widetilde{m}=(\widetilde{t},\widetilde{r})\in\beta}} \boldsymbol{m} \wedge \widetilde{\boldsymbol{m}} \rightarrow C(m, \widetilde{m}) \tag{5c}$$

Here, $\boldsymbol{\beta} = (\boldsymbol{m_1}, \ldots, \boldsymbol{m_{|\beta|}})$ is a vector of Boolean variables with a variable $\boldsymbol{m} = 1$ encoding the assumption that a task instance m is active and, thus, capable of providing correct service or inactive, i.e., $\boldsymbol{m} = 0$. $\psi(\boldsymbol{\alpha}, \boldsymbol{\beta})$ encodes the following: Term (5a) states that at least one instance $m = (t, r) \in \beta$ of each task $t \in T$ is necessary for a working system. Term (5b) implies that any task instance relies on a non-defect resource. Furthermore, if two instances of data dependent tasks are assumed to contribute to a working system, they must be able to communicate properly. Term (5c) uses the *communication function* $C : \beta \times \beta \rightarrow \{0, 1\}$ shown in Eq. (6) to indicate whether this communication is feasible or not:

$$C(m, \widetilde{m}) = \begin{cases} 1, & \text{if } \begin{smallmatrix} m=(t,r), \widetilde{m}=(\widetilde{t},\widetilde{r}): \\ r=\widetilde{r} \ \vee \ (r,\widetilde{r})\in E_{\mathrm{a}} \end{smallmatrix} \\ 0, & \text{else} \end{cases} \tag{6}$$

For determining the reliability of the implementation with respect to resource defects, knowledge about the actual set of tasks that implements the system that provides correct service is not necessary. In fact, to perform a correct reliability analysis, it is sufficient to know that there *exists* a set of task instances that guarantee a system that provides correct service for a given set of properly working resources. This is achieved by applying *existential quantification*[2] to the characteristic function $\psi(\boldsymbol{\alpha}, \boldsymbol{\beta})$ that results in the desired structure function $\varphi(\alpha)$, cf. Fig. 2.

[2] $\exists y : \psi(\boldsymbol{x}, \boldsymbol{y}) = \psi(\boldsymbol{x}, \boldsymbol{y})_{|y=1} \vee \psi(\boldsymbol{x}, \boldsymbol{y})_{|y=0}$

B. Evaluating φ

In order to quantify the reliability of an implementation, its *reliability function* $\mathcal{R}$ has to be determined. Using a specific SHANNON-decomposition as proposed in [10], the probability $\mathcal{P}$ of the system providing correct service at time τ is determined. This decomposition scheme can be applied to the BDD directly and is defined as follows:

$$\mathcal{P}(\tau, \varphi) = \mathcal{R}_r(\tau) \cdot \mathcal{P}(\tau, \varphi_{|\mathbf{r}=1}) + (1 - \mathcal{R}_r(\tau)) \cdot \mathcal{P}(\tau, \varphi_{|\mathbf{r}=0}) \tag{7}$$

This function determines the probability of a structure function φ to evaluate to 1 at a given time τ. The function $\mathcal{R}_r : \mathbb{R}_0^+ \rightarrow \mathbb{R}_{[0,1]}$ is the reliability function of a single resource r and returns the probability of this component to provide correct service at time τ. To derive the reliability function $\mathcal{R}$ of the entire implementation, the structure function φ has to fulfill the following condition:

$$\varphi(\boldsymbol{\alpha}) \geq \varphi(\widetilde{\boldsymbol{\alpha}}) \text{ , if } \forall i \in \{0, \ldots, |\alpha|\} : \mathbf{r_i} \geq \widetilde{\mathbf{r}}_\mathbf{i} \tag{8}$$

In other words, a resource that provides correct service can only improve the overall system performance, but not lead to a system defect. Since this condition is trivially fulfilled by the presented approach to generate φ, the desired reliability function $\mathcal{R}_\varphi : \mathbb{R}_0^+ \rightarrow \mathbb{R}_{[0,1]}$ of the implementation is given by:

$$\mathcal{R}_\varphi(\tau) = \mathcal{P}(\tau, \varphi) \tag{9}$$

Based on the reliability function of the system, several reliability-related measures like the *Mean-Time-To-Failure* (MTTF) or the *Mission-Time* (MT) can be derived. For example, the MTTF is calculated as follows:

$$\text{MTTF}(\varphi) \ = \int_0^\infty \mathcal{R}_\varphi(\tau) d\tau \tag{10}$$

In system level design space exploration approaches, the MTTF is typically used as the design objective that quantifies the reliability of an implementation.

This BDD-based reliability analysis approach delivers exact results, but requires the storage of the BDD in the main memory. As BDDs may grow exponential in the number of variables in the worst case, the scalability of this approach may become a serious issue. Approaches to cope with this problem based on *early quantification* are, e.g., proposed in [11], [12]. However, for large systems, a simulative approach, as presented in the next section, allows to efficiently analyze these systems while delivering appropriately approximated results.

V. SAT-ASSISTED SIMULATION

Given the fact that BDD-based approaches may fail due to outsized BDDs in case of large and complex systems, this section presents a complementary reliability analysis approach based on the hybridization of a *Monte-Carlo simulation* and a *SAT solver* [13] termed *SAT-assisted simulation* [12]. While the explicit encoding of the BDD requires lots of memory, the simulation is based on iteratively carried out simulation runs and, thus, allows to trade off the memory for runtime. This enables an analysis of large and complex systems. However, the simulation will only deliver approximations while the BDD-based approach allows to determine exact reliability measures.

Although using a Monte-Carlo simulation may seem to be a low hanging fruit at first glance, each simulated fault raises the need for a *feasibility test*. This test needs to determine whether the system provides correct service or whether it failed with respect to a given set of resource defects. The test can be reduced to the problem of finding a feasible implementation for a given platform that only contains the resources that provide correct service and the mapping possibilities given by the multiple bound tasks. In particular, the problem of finding a feasible implementation for a platform is shown to be $\mathcal{NP}$-complete, cf. [14]. Given the fact that the accuracy of the simulation is typically related to the number of performable simulation runs, this test becomes the bottleneck of the Monte-Carlo simulation.

As a remedy, a state-of-the-art SAT solver based on a backtracking algorithm is used for the feasibility test within the simulation. This enables to efficiently perform hundreds of simulation runs, even for large and complex systems where known exact methods fail. For the SAT solver, the system function needs to be provided in *Conjunctive Normal Form* (CNF). If the system function is not directly given in CNF like in [11], several efficient techniques are known to transform any Boolean function into CNF, cf. [15].

The iterative SAT-assisted simulation approach works as follows: First, for the overall system encoded in $\psi(\boldsymbol{\alpha}, \boldsymbol{\beta})$, a set Γ_ψ of N times-to-failure is determined by N independent simulation runs:

$$\Gamma_\psi = \bigcup_{i=0}^{N} \text{mcs}(\psi(\boldsymbol{\alpha}, \boldsymbol{\beta})) \tag{11}$$

The function mcs : $\{0,1\}^{\{0,1\}^n} \rightarrow \mathbb{R}^+$ carries out one simulation run based on a given characteristic function $\psi(\boldsymbol{\alpha}, \boldsymbol{\beta})$. The function *mcs* is outlined in Alg. 1.

Given the times-to-failure Γ_ψ, the desired reliability function of the system is approximated as follows:

$$\mathcal{R}(\tau) \approx \frac{|\{\gamma | \gamma \in \Gamma_\psi \wedge \gamma > \tau\}|}{N} \tag{12}$$

Algorithm 1 mcs($\psi(\boldsymbol{\alpha}, \boldsymbol{\beta})$) - SAT-assisted Monte-Carlo simulation. First a set Γ of times-to-failure in ascending order is computed that contains a specific time-to-failure γ for each allocated resource $r \in \alpha$ of the structure function using the function timesToFailure : $2^R \rightarrow 2^{(R \times \mathbb{R}^+)}$, cf. line 1. The time-to-failure of each resource r is determined by using inverse transform sampling $\gamma = \mathcal{R}_r^{-1}(\text{rnd})$ based on the reliability function $\mathcal{R}_r(\tau)$ of the resource and a random number rnd $\in \mathbb{R}_{[0,1]}$. For each element $(r, \tau) \in \Gamma$ and with respect to the order of Γ, the characteristic function ψ is incrementally extended with a negated component variable $\neg r$ using conjunction, cf. line 3. This corresponds to the resource r being faulty. The SAT solver is invoked using the function sat : $\{0,1\}^{\{0,1\}^n} \rightarrow \{0,1\}$ that returns *true* if the characteristic function can be satisfied, i.e., if the overall system provides correct service. If the overall system failed, cf. line 4, the time-to-failure of the resource that failed last corresponds to the overall system time-to-failure and is returned in line 5.

Require: $\psi(\boldsymbol{\alpha}, \boldsymbol{\beta})$
Ensure: Γ is an ordered set
1: Γ := timesToFailure(α)
2: **for** $(r, \gamma) \in \Gamma$ **do**
3: $\quad \psi(\boldsymbol{\alpha}, \boldsymbol{\beta})$:= $\psi(\boldsymbol{\alpha}, \boldsymbol{\beta}) \wedge \neg \boldsymbol{r}$
4: $\quad$ **if** $\neg$ sat($\psi(\boldsymbol{\alpha}, \boldsymbol{\beta})$) **then**
5: $\quad\quad$ **return** γ // time-to-failure
6: $\quad$ **end if**
7: **end for**

VI. Experimental Results

This section puts focus on the complementary application of the exact BDD-based analysis approaches and the SAT-assisted simulation with respect to the scalability vs. the accuracy of the techniques. The comparison is performed using the common BDD-based approach as presented in this work that is proposed in [7], the BDD-based approach that makes use of early quantification as is proposed in [12], and the SAT-assisted simulation that discussed here and proposed in [12]. A detailed discussion of the presented results is given in [12].

Testsuite. For the comparison, a testsuite containing various system level design specifications is prepared: (a) 7 real-world specifications from the data-streaming as well as the automotive domain are chosen. The complexity of the real-world test cases ranges from about 50 tasks with 30 available resources up to about 250 tasks with about 1000 available resources. (b) 8 synthetic test cases are generated. The complexity of the synthetic test cases ranges from 50 tasks with 25 available resources to 150 tasks with 75 available resources. For each of the 15 test cases, 10 implementations of different complexities with respect to the BDD sizes are generated: Using very few resources with marginal task redundancy creates implementations of low complexity, whereas using many resources with a high amount of task redundancy results in implementations of high complexity. The result is a testsuite of 150 test cases that covers a broad variety of test instances ranging from simple examples, over moderate, up to hard and highly-complex real-world test cases. The experiments are carried out on an Intel Pentium 4 3.00 GHz Dual Core machine with 1.5 GB RAM. The number of simulation runs for the SAT approach is set to 2000.

Scalability. From each complexity class, 3 examples and the corresponding runtimes of the analysis approach are depicted in Table I. The early quantification approach outperforms both other approaches in terms of runtime and allows to analyze also several of the hard examples. In particular, it failed to analyze 18 of the 150 test cases. The common BDD-based approach shows the worst scalability, i.e., it failed to analyze 95 test cases, and, except for a few cases where early quantification induces too much overhead, requires longer runtimes than the early quantification approach. The best scalability is provided by the SAT-assisted simulation that successfully analyzed all testcases. However, this is bought by a significantly longer runtime in cases where a BDD-based approach succeeds. Thus, SAT-assisted simulation should be replaced by BDD-based analysis whenever possible to take advantage of the lower runtime and exact results.

Accuracy. The accuracy of the SAT-assisted simulation is investigated in Table II. The relative error and the deviation of the results delivered by the SAT-assisted simulation are calculated using the 132 testcases where a BDD-based analysis

TABLE I
RUNTIMES OF THREE RELIABILITY ANALYSIS APPROACHES: (A) BDD-BASED AS PRESENTED IN THIS WORK, (B) BDD-BASED USING EARLY QUANTIFICATION (EQ), AND (C) THE SAT-ASSISTED SIMULATION. FOR EACH CLASS OF TESTCASES, THREE EXAMPLES ARE SELECTED. ALL TIMES ARE GIVEN IN MILLISECONDS (MS).

	BDD-based [7]	BDD-based + EQ [12]	SAT-assisted simulation [12]
simple	14	**13**	868
	10	24	800
	36	**13**	1,832
moderate	230	**72**	7,688
	19,721	**27**	1,816
	–	**169**	5,500
hard	–	**3,008**	62,236
	–	**42,494**	55,960
	–	–	**53,192**

TABLE II
ACCURACY OF THE SAT-ASSISTED SIMULATION. THE RELATIVE ERROR IS CALCULATED BASED ON 132 OUT OF 150 TESTCASES WHERE THE EXACT RESULTS ARE DELIVERED BY THE EARLY-QUANTIFICATION APPROACH GIVEN IN [12].

# simulation runs	relative error in %	deviation in%
500	4.01	2.08
2000	1.51	1.02
4000	1.18	0.94

delivered an exact result. For the simulation, 500, 2000, and 4000 simulation runs have been performed. While the relative error when performing 500 simulation runs is rather high with about 4% and a deviation of about 2%, the accuracy when carrying out 2000 runs improves significantly and results in a relative error of only about 1.5% with a deviation of about 1%. Given the techniques are tailored to the system level reliability analysis in early phases of the design process, 2000 simulation runs deliver a reasonable accuracy. Increasing the number of simulation runs to 4000 results in a marginal enhancement of the accuracy at the expense of a doubled runtime.

VII. CONCLUSION AND FURTHER READING

The work at hand introduces two complementary system-level reliability analysis approaches tailored to a utilization in early design phases of an embedded system. A formal analysis based on *Binary Decision Diagrams* (BDDs) is proposed to determine the exact reliability of a system implementation for small to moderate-sized problems. This analysis technique is capable of automatically taking arbitrary system structures into account and deriving the overall lifetime reliability of the system based on arbitrary reliability functions of the system components. The second approach copes with the scalability issues of BDDs by using a simulative approach that hybridizes a Monte Carlo simulation with a SAT solver. This efficient approach delivers appropriate approximated results where known exact methods fail.

More experimental results for the BDD-based reliability analysis can be found in [7]. An application of the analysis in the design of automotive networks and self-healing systems is given in [11] and [16], respectively. The integration of *majority voters* and the consideration of *graceful degradation* is presented in [17] and [18]. The consideration of *data-redundancy* in networked embedded systems is proposed in [19]. The presented techniques are available within the open source framework JRELIABILITY [20] under LGPL.

REFERENCES

[1] O. Wyman, "Auto & Umwelt 2007: Kundenerwartungen als Chance für die Hersteller," 2007.

[2] M. Lukasiewycz at al., "Combined System Synthesis and Communication Architecture Exploration for MPSoCs," in *Proc. of DATE '09*. IEEE Computer Society, 2009, pp. 472–477.

[3] D. Gajski et al., *High-level synthesis: introduction to chip and system design*. Kluwer Academic, 1992.

[4] P. Pop et al., "Design optimization of time-and cost-constrained fault-tolerant embedded systems with checkpointing and replication," *IEEE Trans. on VLSI*, vol. 17, no. 3, pp. 389–402, 2009.

[5] Srinivasan et al., J., "The Impact of Technology Scaling on Lifetime Reliability," in *Proc. of DSN '2004*, 2004.

[6] Gu at al., Z., "Application-specific MPSoC reliability optimization," *IEEE Trans. on VLSI*, vol. 16, no. 5, p. 603, 2008.

[7] M. Glaß et al., "Reliability-Aware System Synthesis," in *Proceedings of DATE '07*, 2007, pp. 409–414.

[8] A. Birolini, *Reliability Engineering - Theory and Practice*. Berlin, Heidelberg, New York: Springer, 4th edition, 2004.

[9] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *Trans. on Comp.*, vol. 35, no. 8, pp. 677–691, 1986.

[10] A. Rauzy, "New Algorithms for Fault Tree Analysis," *Reliability Eng. and System Safety*, vol. 40, pp. 202–211, 1993.

[11] M. Glaß et al., "Symbolic Reliability Analysis and Optimization of ECU Networks," in *Proc. of DATE '08*, 2008, pp. 158–163.

[12] ——, "Towards scalable system-level reliability analysis," in *Proc. of DAC '10*, 2010, pp. 234–239.

[13] M. Davis et al., "A machine program for theorem-proving," *Comm. of the ACM*, vol. 5, no. 7, pp. 394–397, 1962.

[14] C. Haubelt, J. Teich, R. Feldmann, and B. Monien, "SAT-Based Techniques in System Design," in *Proc. of DAC '03*, 2003, pp. 1168–1169.

[15] G. Tseitin, "On the complexity of derivation in propositional calculus," *Studies in constructive mathematics and mathematical logic*, vol. 2, no. 115-125, pp. 10–13, 1968.

[16] M. Glaß et al., "Symbolic Reliability Analysis of Self-healing Networked Embedded Systems," in *Proc. of SAFECOMP '08*, 2008, pp. 139–152.

[17] F. Reimann et al., "Symbolic Voter Placement for Dependability-Aware System Synthesis," in *Proc. of CODES+ISSS '08*, 2008, pp. 237–242.

[18] M. Glaß et al., "Incorporating Graceful Degradation into Embedded System Design," in *Proc. of DATE '09*, 2009, pp. 320–323.

[19] M. Lukasiewycz et al., "Exploiting Data-Redundancy in Reliability-Aware Networked Embedded System Design," in *Proc. of CODES+ISSS '09*, 2009, pp. 229–238.

[20] JReliability, "The java-based reliability library," http://www.jreliability.org/, Version 1.2.