This is the author's version of the work. The definitive work was published in *Proceedings of International Workshop on Reliability Aware System Design and Test (RASDAT '10), pp. 17-22, 2010.* The work is supported in part by the German Science Foundation (DFG), SFB 694.

Lifetime Reliability Optimization for Embedded Systems: A System-Level Approach

Michael Glaß, Martin Lukasiewycz, Christian Haubelt, and Jürgen Teich
Hardware/Software Co-Design, University of Erlangen-Nuremberg, Germany
Email: {glass, martin.lukasiewycz, haubelt, teich}@cs.fau.de

***Abstract*— This paper presents an automatic reliability-aware system-level design methodology to tolerate hardware defects caused by manufacturing tolerances as well as destructive agents and aging processes at the places of activity of the system components. This is achieved by (1) integrating the capability of a redundant placement of software tasks in an automatic design process to cope with the hardware defects, (2) providing an automatic lifetime reliability analysis to trade off the arising costs in favor of the achieved reliability increase, and (3) proposing a software architecture for the runtime phase. Real-life case studies from the automotive domain illustrate the effectiveness of the proposed techniques.**

***Keywords*— Embedded Systems, Lifetime Reliability, Dependability, Analysis, Optimization, Design Space Exploration, System-Level**

I. Introduction

Embedded systems are more and more part of our everyday life. Especially in mobile systems such as automobiles, airplanes, and trains, an embedded system is not a single isolated component, but a large and complex networked system where components heavily interact with each other. These so-called *networked embedded systems* can for example consist of more than 70 electronic control units (ECUs) in a modern automobile. An important factor that influences the reliability of the ECUs are their *places of activity*: ECUs move from dedicated and protected mounting spaces to installation spaces with *destructive agents*, i.e., mounting spaces near sensors, moving parts or even within the engine. These destructive agents in connection with manufacturing tolerances and aging processes increase the so-called *stress* for the system components that leads to a significant increase of their defect rate. While these new mounting spaces allow to save costs, the increased stress results in a significant decrease in the lifetime reliability of the system. With customers surveys revealing that *reliability* constitutes the most important purchase criteria, cf. [20], an appropriate trade-off between several objectives like monetary costs, energy consumption, volume consumption, performance, and lifetime reliability has to be found.

Common techniques to tolerate resource defects that result from the given stress at the components places of activity are based on component replication. However, these techniques can only be used in safety critical applications like airplanes where the available space and the monetary costs are not the limiting factors. Especially in the automotive area, monetary cost is one of the main objectives for the manufacturer that, thus, makes component replication prohibitive.

Contributions. Given the growing demand for reliable embedded systems and the increasing number of electronic components that are integrated at extreme places of activity, the work at hand proposes a system-level design methodology for a reliability-aware optimization of embedded systems.

For the reliability analysis, a formal analysis based on *Binary Decision Diagrams* (BDDs) [4] is proposed. The analysis technique is capable of automatically taking arbitrary system structures into account and derives the overall lifetime reliability of the system based on its *structure* and the *reliability functions* of the used components. To avoid costly resource replications, a methodology to reuse existing resources for the multiple instantiation of software tasks called *multiple binding* is proposed. The multiple binding is integrated in an automatic optimization approach of the hardware/software system that considers all kinds of given objectives like monetary costs, energy consumption, and especially lifetime reliability. To implement the capability to tolerate the occurring defects at runtime, a software architecture is proposed that observes the system and activates redundant software tasks if necessary. Therefore, system information collected at design time is used for an efficient controlling.

Outline. The rest of the paper is outlined as follows: Section II discusses related work. While the problem targeted in this paper is outlined in Sec. III, the proposed reliability analysis and optimization technique is presented in Sec. IV. Section V introduces the proposed software architecture for the online phase of the system. Section VI presents experimental results before the paper is concluded in Sec. VII.

II. Related Work

Up to now, several approaches have been presented for analyzing systems with respect to reliability and fault tolerance. An overview of these techniques can be found in [1]. In recent years, these techniques were integrated in the synthesis phase of a system in order to automatically design reliable hardware/software systems.

An early approach in the field of fault-tolerant ASIC design has been introduced in [12] where the hardware overhead caused by replication is minimized.

Several approaches target a reliable system design and, respectively, are focused on the tolerance of *soft errors*, i.e., transient and intermittent faults. An approach that unifies fault-tolerance via checkpointing and power management through *dynamic voltage scaling* is introduced in [22]. In [9], fault-tolerant schedules with respect to performance requirements are synthesized using task re-execution. The same authors introduce another approach for hard real-time systems in [10], using rollback recovery and active replication. Here, fault-tolerance policy assignment, checkpoint placement and task-to-processor mapping is performed automatically. Other approaches such as [19], [21] try to maximize reliability by selectively introducing redundancy. With this technique, only one objective (reliability) can be optimized while area and latency are treated as constraints. The work at hand expects the tolerance of soft errors at component level and targets

hardware defects of resources, e.g., ECUs, buses, or sensors at system level. A co-design framework which assesses reliability properties on system level has been presented in [3] that proposes a two step approach: In the first phase, a partitioning respecting constraints like area, power, costs, etc. is performed and in a second step, the reliability of critical parts is increased. A drawback of this approach is that this second step might worsen the solution obtained in the first step.

In [5], an approach is proposed which synthesizes reliable systems with the help of Genetic Algorithms (GA). The GA selects appropriate resources and determines the degree of redundancy such that the cost is minimal for a given reliability. For estimating the reliability which is an objective function for the GA, the authors apply a neural network. Similar to the last approach, the design space exploration used in the work at hand is performed using Evolutionary Algorithms. In contrast, the used approach allows for multiple objectives using state-of-the-art Multi-Objective Evolutionary Algorithms [2]. Furthermore, the reliability of the synthesized system is determined based on Binary Decision Diagrams (BDDs) that allow to analyze arbitrary system structures.

Lifetime reliability is introduced as an optimization objective into system-level design in [11]. The used fault-tolerance technique is resource replication that results in highly increased cost for reliability. The degree of redundancy is varied by the used *Evolutionary Algorithm*. However, the reliability analysis is not able to handle resource sharing and, thus, is not applicable to typical embedded systems where several software tasks share the components.

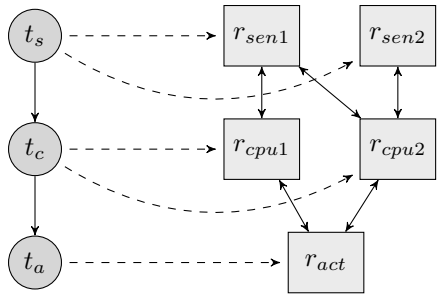
III. Problem Formulation

The work at hand aims to increase the reliability of embedded systems by a reliability-aware distribution of software tasks in the hardware/software design phase. An integral part of the hardware/software design phase is the so-called *design space exploration*. The task of design space exploration is to find the set of optimal *feasible implementations* for a given *specification*. Due to the various objectives of such systems, design space exploration can be formulated as a *multi-objective optimization problem*. Hence, there is generally not only one global optimal implementation, but a set of *Pareto-optimal implementations*. A Pareto-optimal implementation is better in at least one objective when compared to any other feasible implementation in the design space [23].

A. System Model

For the design space exploration, a model of the available hardware components as well as the software tasks that need to be distributed in the system is defined. The specification, thus, includes all possible design variants of the system to design. This graph-based specification consists of the *architecture*, the *application*, and a relation between these two views, the *mapping constraints* (see Fig. 1):

- The architecture is modeled by a graph $g_{\mathsf{a}}(R, E_{\mathsf{a}})$ and represents all available interconnected components, i.e., hardware *resources*. 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.



(a) application (b) architecture

Fig. 1. A simple system specification. The application (a) consist of three tasks with a sensor task t_s, a control task t_c evaluating the data from the sensor task, and an actuator task t_a that is responsible to executed the commands from the control task at the actuator. The architecture (b) consists of two different sensors suitable to carry out the sensor task, two CPUs 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.

- The application is modeled by a task graph $g_{\mathsf{t}}(T, E_{\mathsf{t}})$ that represents the behavior of the system. The vertices $t_1, ..., t_{|T|} \in T$ denote software *tasks* whereas the directed edges E_{t} are data dependencies between tasks.
- The relation between architecture and application is given by 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, with a mapping $m = (t, r) \in E_{\mathsf{m}}$ indicating that a specific task t can be executed on a hardware 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 hardware/software 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 hardware 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 software task is *bound* to a hardware resource that executes this task at runtime.

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:

$$\forall (t, \widetilde{t}) \in T : \exists\, m = (t, r), \widetilde{m} = (\widetilde{t}, \widetilde{r}) \in \beta : \\ (r = \widetilde{r}) \vee ((r, \widetilde{r}) \in E_{\mathsf{a}}) \quad (1)$$

Thus, an implementation x is defined as a pair (α, β), with a *feasible* implementation requiring a feasible binding for the given allocation, cf. Fig. 2.

B. Reliability Model

To allow the analysis of a given implementation, the tasks, resources, and components are annotated with properties in the specification. These properties are for example monetary costs, volume consumption, performance and many more. Based on the properties of each component, the objectives of the whole implementation are derived. This calculation is performed by so-called *evaluators* with each evaluator being an analysis methodology that is able to gather one or several objectives from the component properties of an implementation. Since the work at hand introduces lifetime reliability as an objective,

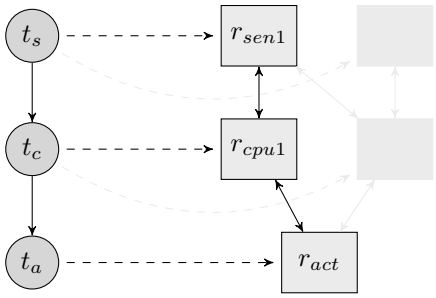


Fig. 2. A feasible implementation for the system specification given in Fig. 1. The allocated resources are r_{sen1}, r_{cpu1}, and r_{act} with t_a being bound to r_{sen1}, t_c being bound to r_{cpu1}, and t_a being bound to r_{act}.

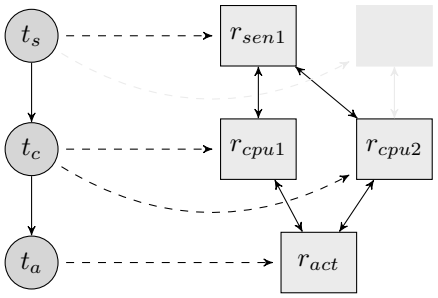


Fig. 3. A feasible implementation of the system specification given in Fig. 1 including a multiple binding of the control task t_c to both CPUs r_{cpu1} and r_{cpu2}. This implementation allows to tolerate the defect of one of the two allocated CPUs.

reliability properties of each component need to be provided for the reliability analysis presented in Sec. IV. The lifetime reliability of each hardware resource $r \in R$ is modeled as a *reliability function* $\mathcal{R}_r : \mathbb{R}^+ \rightarrow \mathbb{R}_{[0,1]}$ that returns the probability of the life time τ_{LT} of the resource being greater than a certain time τ:

$$\mathcal{R}_r(\tau) = \mathcal{P}[\tau_{LT} > \tau] \tag{2}$$

It holds that $\mathcal{R}(0) = 1$ and $\mathcal{R}(\infty) = 0$.

C. Design Space Exploration

The used design space exploration utilizes a meta-heuristic optimization approach, the *Multi-Objective Evolutionary Algorithm* (MOEA) [2]. MOEAs are heuristic optimization approaches that are inspired by natural evolution and are commonly applied in domains with large search spaces and multiple conflicting and especially non-linear objectives. These algorithms represent a single implementation of the problem by a *genotype* that is varied by genetic operations, i.e., *mutation* and *crossover*. Multiple of these genotypes, the so-called *population*, are optimized in parallel. In each *generation*, the genotypes are combined using the crossover operator follow by a mutation. Thus, the solutions of the given problem are improved iteratively. While the allocation $\alpha \subseteq R$ of resources is decoded from a bit vector where each bit encodes whether the associated resource is allocated or not, the binding $\beta \subseteq E_{\mathrm{m}}$ is decoded from priority lists. Thus, an implementation x can be fully represented as a genotype. Due to the increased search space that is a result of considering the multiple instantiation of software tasks, the simple genetic encoding introduced in this section has been replaced in [7] by a more sophisticated technique based on SAT-solvers known as *SAT-decoding* [14]. However, this encoding scheme shall not be discussed in the work at hand. In Sec. IV, an approach to include a reliability-increasing binding approach in the design space exploration is presented.

IV. Reliability Analysis and Optimization

As outlined in Sec. III-A, the steps of selecting a subset of resources, i.e., the allocation, and assigning each software task to an allocated resource, i.e., the binding, are the main steps to generate an implementation from a given specification. However, existing design space exploration approaches like [11], [15], [16], [18], [13] are limited in such a way that each software task is bound exactly once. Thus, the approaches cannot use software-redundancy to increase reliability, but rely on resource replication. As a remedy, an extension of the design space exploration by

1) generating implementations with multiple task instances to introduce software-redundancy and
2) providing a reliability evaluation method to trade-off lifetime reliability with the other given objectives

is proposed.

A. Multiple Binding

To use software-based redundancy, the activation of more than one mapping per task is allowed, cf. Fig. 3, thus creating several *instances* of one task, cf. [6]:

$$\forall t \in T : |\{m | m = (t, r) \in \beta\}| \geq 1 \tag{3}$$

The advantage of this multiple binding is the transparency for the designer, since no explicit modeling of a redundant layout of tasks is necessary. Moreover, it avoids the costly resource replication by allowing to use idle capacities of the allocated resources for software-redundancy.

Given the multiple binding approach, the binding β can be better represented by an *instance graph* $g_\beta(\beta, E_\beta)$. The vertices are the activated mappings β with each vertex representing an instance of a task in the system. An edge $e \in E_\beta$ represents a feasible communication, cf. Definition 1, between instances of two data dependent tasks.

B. Reliability Analysis

In this section, a reliability analysis approach is introduced in two steps: First, it is described how the so-called *structure function* φ [1] of the implementation can be generated from the system model and represented by *Binary Decision Diagrams* (BDDs) [4] which are an efficient representation of Boolean functions. A BDD is 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 is evaluated to determine the systems reliability in terms of the *Mean-Time-To-Failure* (MTTF).

1) 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 vector $\boldsymbol{\alpha} = (\boldsymbol{r_1}, \ldots, \boldsymbol{r_{|\alpha|}})$ is determined. This Boolean function indicates a proper working system by evaluating to $\varphi = 1$ and a system failure by evaluating to $\varphi = 0$, respectively. Here, for each allocated resource $r \in \alpha$, a binary variable $\boldsymbol{r}$ is introduced

with $\boldsymbol{r} = 1$ indicating a proper operation and $\boldsymbol{r} = 0$ a resource defect, respectively. For a given system specification $x = (\alpha, \beta)$, φ is determined as follows:

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

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

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

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

Here, $\boldsymbol{\beta} = (\boldsymbol{m_1}, \ldots, \boldsymbol{m_{|\beta|}})$ is a vector of Boolean variables encoding a task instance being active to ensure a proper working system. $\psi(\boldsymbol{\alpha}, \boldsymbol{\beta})$ encodes the following: At least one instance $m = (t, r) \in \beta$ of each task $t \in T$ is necessary for a working system. This is ensured by Term (5a). Term (5b) implies that any task instance relies on a proper working 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 function $C : \beta \times \beta \rightarrow \{0, 1\}$ shown in Equation (6) to indicate whether this communication is feasible or not.

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

For determining the reliability of the implementation with respect to resource defects, knowledge about the actual set of tasks that implements the proper working system is not necessary. In fact, to perform a correct reliability analysis, it is sufficient to know that there *exists* a set of proper working and correct communicating task instances that guarantee a proper working system for a given set of proper working resources. This is achieved by applying *existential quantification*[1] to $\psi(\boldsymbol{\alpha}, \boldsymbol{\beta})$ that results in the desired structure function $\varphi(\alpha)$, cf. Fig. 4.

2) 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 [17], the probability $\mathcal{P}$ of a proper working system 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 $\mathcal{R}_r : \mathbb{R}^+ \rightarrow \mathbb{R}_{[0,1]}$ is the reliability function of a single resource r and returns the probability of this component to work properly at time τ. To derive the reliability function $\mathcal{R}$ of the entire implementation, the structure function φ has to fulfill the following condition:

$$\varphi(\mathbf{x}) \geq \varphi(\widetilde{\mathbf{x}}) \text{ , if } \forall i \in \{0, \ldots, n\} : \mathbf{x_i} \geq \widetilde{\mathbf{x}_\mathbf{i}} \tag{8}$$

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

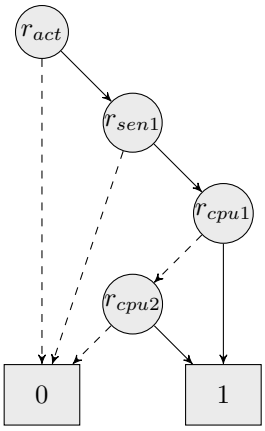


Fig. 4. The structure function φ of the implementation in Fig. 3, encoded as a Binary Decision Diagram (BDD). The regular edges correspond to the variable of the corresponding node being 1, while the dashed edges correspond to the variable being 0, respectively.

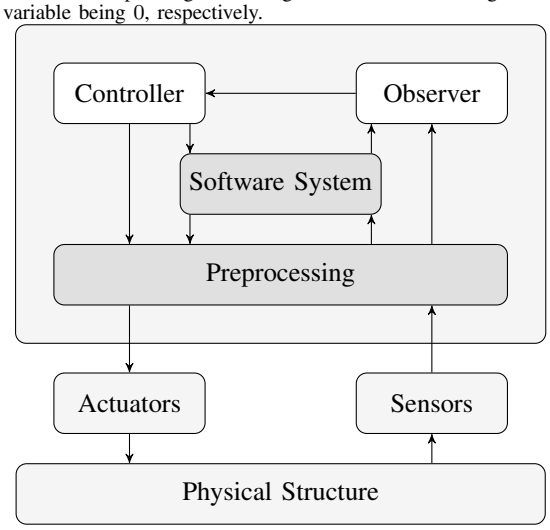


Fig. 5. The proposed architecture for the online phase of the implementation. A standard architecture that consist of actuators and sensors interacting with the underlying physical structure as well as a software system that communicates with the sensors and actuators including a preprocessing is extended by an observer and a controller. While the observer is responsible to detect resource defects, the controller relies on appropriate data structures determined at design time to activate task instances in order to tolerate present defects.

In other words, a properly working component 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 $\mathcal{R}_\varphi : \mathbb{R}^+ \rightarrow \mathbb{R}_{[0,1]}$ 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. In the work at hand, the MTTF is used as the measure for reliability which is calculated as follows:

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

V. ONLINE CONTROL

In the online phase of the system, an approach becomes necessary that allows to make use of the software-redundancy

introduced at design time. In Fig. 5, the proposed architecture is depicted. Given a standard architecture that consists of a software system which interacts with sensors and actuators by using a preprocessing, two software modules are added: (1) The so-called *observer* is responsible to observe the system and determine its current state, i.e., which resources have failed and which software tasks are currently activated and running. (2) In case the system state changes because of a defect, the *controller* is invoked by the observer. The controller uses knowledge derived at design time to decide which tasks need to be activated in order to ensure a proper working system. In the following, an algorithm that is guided by $\psi(\boldsymbol{\alpha}, \boldsymbol{\beta})$ as a data structure is presented: The proposed algorithm is based on information from the observer that that is extremely reliable and can detect[2] defects of all resources in the system. In particular, this observer is capable to obtain the binary vector $\boldsymbol{\alpha}$ that encodes working and defect resources. The algorithm itself is located at the controller which decides which task instances $\boldsymbol{\beta}$ are actually activated or deactivated, based on $\psi(\boldsymbol{\alpha}, \boldsymbol{\beta})$ and $\boldsymbol{\alpha}$. The overall algorithm is outlined in Algorithm 1.

Algorithm 1 Online control algorithm.

```
1:  α := (1, ..., 1) // initial state
2:  β with ψ(α, β) = 1 // initial activation
3:  while true do
4:      α := observe() // observe
5:      if ∃β : ψ(α, β) then
6:          β := min_dist{β|ψ(α, β) = 1} // new activation
7:          notify()
8:      else
9:          failure()
10:     end if
11: end while
```

The failure detection and, thus, the update of the $\boldsymbol{\alpha}$ vector is performed by the function *observe()*, carried out on the observer (line 4). In case a failure occurs, it is tested whether there exists a feasible task activation with respect to the currently proper working resources (line 5). If the test fails, the function *failure()* is used to signal that the system cannot work properly anymore to implement a safe behavior. If the test is successful, a new proper working implementation has to be achieved by a correct task activation $\boldsymbol{\beta}$ (line 6) and the tasks are activated using the function *notify()*. At this, the function $\min_{\text{dist}}$ aims to find a task activation that is similar to the former task activation to avoid the unnecessary activation and deactivation of tasks and, thus, keeps system reconfiguration at a minimum. This is achieved by a special reordering of ψ that is done at design time:

$$\boldsymbol{\alpha_0} < \ldots < \boldsymbol{\alpha}_{|\boldsymbol{\alpha}|} < \boldsymbol{\beta} \tag{11}$$

Given this variable order, the resource variables are setup first since they are fixed whenever the algorithm searches for a new task activation. The internal structure of a BDD is a tree with one root and a *true* (1) and *false* (0) terminal node. Each node of the tree represents a variable and has two outgoing edges that equal an assignment of *true* or *false*, respectively, to the corresponding variable. Thus, each traversal of the BDD from the root that reaches the *true* terminal node corresponds to a variable assignment that equals a proper working implementation. At this, the traversal for the $\boldsymbol{\beta}$ variables is guided by the previous task activation to reach the most similar task activation for the given BDD order. To prevent that an infeasible activation, i.e., the *false* terminal node is reached, a look-ahead is done for each node and its outgoing edges that results in an overall linear traversal complexity of $O(2 \cdot |\boldsymbol{\alpha}| \cdot |\boldsymbol{\beta}|)$.

[2]The aspect of failure detection is highly system dependent and will not be discussed in this work.

VI. Experimental Results

In the following, two testcases from the automotive area are used to give evidence of the effectiveness of the introduced methodology. For the reliability analysis and optimization at design time, a real-life specification for an adaptive light control is explored. For the performance of the proposed online algorithm, a specification that combines six automotive applications including an adaptive cruise control and brake-by-wire is used.

A. Reliability Analysis and Optimization

As a case-study for the reliability-aware design space exploration, an *Adaptive Light Control* (ALC) is used: This industrial example from the automotive area consists of 234 software tasks, 1103 available resources, and 1851 mapping edges. Thus, this large case study allows for $\approx 2^{375}$ binding possibilities, i.e., different possible implementations of the ALC. The experiments were carried out on an Intel Pentium 4 3.20GHz machine with 1GB RAM. The exploration time for 10 exploration runs for the ALC with the evolutionary algorithms being set to 500 generations and a population size of 100 individuals was 2h37m. The results are taken from [6].

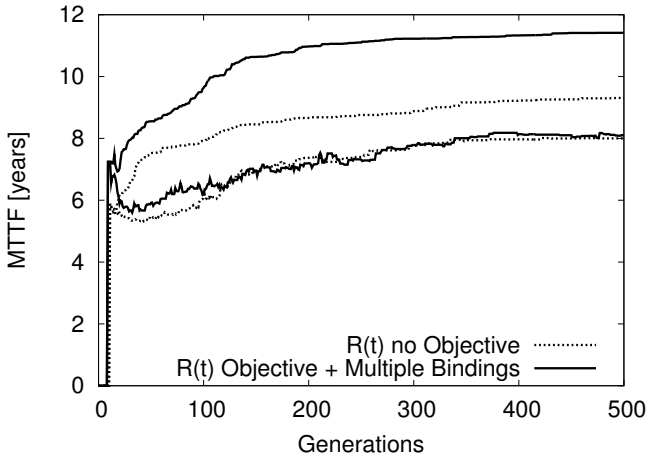


Fig. 6. Adaptive Light Control: Maximal and minimal MTTF throughout the exploration process.

Fig. 6 shows the average minimal and maximal MTTF-values over 10 exploration runs for the ALC. Due to the used optimization algorithm being an heuristic, the 10 exploration runs are necessary to determine significant average values. By introducing lifetime reliability as an optimization objective, implementations can be offered to the designer that are up to $\approx 20\%$ more reliable than the ones found by common design space exploration tools neglecting reliability. Note that

	BDD		time [μs]
	#nodes	size [kBytes]	
simple	60	1.44	8.8
complex	7198	172.75	21.1

TABLE I

PROPOSED ONLINE ALGORITHM EVALUATING THE SMALLEST AND THE MOST COMPLEX FEASIBLE IMPLEMENTATION.

all these implementations considered are optimal from a multi-objective point of view. In [7], it could be shown that by using the symbolic SAT-decoding optimization technique, the reliability is increased by about 38% at equal cost for the ALC.

B. Online Algorithm

In this section, a system specification that includes six applications from the automotive area is chosen to highlight the effectiveness of the proposed online algorithm. The overall number of software tasks is 51 with the given architecture consisting of 11 ECUs connected to one bus. Each task can be carried out by at least three ECUs, leading to a design space of $\approx 2^{80}$ implementations. From this specification, a large number of implementations were derived within a design space exploration. By evaluation the implementations, the simplest and most complex feasible implementation regarding the induced structural redundancy have been chosen to serve as testcases for the online algorithm. Thus, effectiveness as well as scalability can be evaluated. Note that for the most complex implementation, nearly all tasks are replicated two to three times in the system which results in a highly extensive automatic reliability analysis. For the evaluation of the online algorithm, a simulative approach is used that generated $10,000$ failure scenarios, each starting with a completely working system and simulating the time until complete system failure is reached. The simulation was carried out for an ARM 9 processor. The results are shown in Table I and taken from [8].

The memory needed to store the BDD data structure with $\approx 1.5kB$ to $\approx 175kB$ was acceptable in both cases. These memory requirements are appropriate for state-of-the-art ECUs. The time consumption of the algorithm to determine a new feasible binding in case a failure occurs ranges from $8.8\mu s$ to $21.1\mu s$ and is negligible small.

VII. CONCLUSION

The work at hand proposes a system-level design methodology for a reliability-aware optimization of embedded systems. A formal analysis based on *Binary Decision Diagrams* (BDDs) [4] is proposed to determine the reliability of a system implementation. This analysis technique is capable of automatically taking arbitrary system structures into account and deriving the overall lifetime reliability of the system based on the reliability functions of the system components. A technique to reuse existing resources for the multiple instantiation of software tasks called *multiple binding* is proposed that avoids costly resource replications. The multiple binding is integrated in an automatic design space exploration approach of the hardware/software system that considers all kinds of given objectives like monetary costs, energy consumption, and in particular lifetime reliability. The introduced techniques offer a set of high quality solutions to the designer where reliability is increased up to 20% for a real-world automotive case study by exploiting idle capacities trough multiple software task instantiation. Moreover, a software architecture for the runtime phase of the system has been proposed that observes the system and activates redundant software tasks if necessary. The experimental results show that using data structures derived already at design time allow for a reasonable fast decision which redundant software tasks need to be activated in order to keep the system operational.

REFERENCES

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

[2] S. Bleuler et al., "PISA - A Platform and Programming Language Independent Interface for Search Algorithms," in *Proc. of EMO '03*, 2003, pp. 494–508.

[3] C. Bolchini et al., "Reliability Properties Assessment at System Level: A Co-Design Framework," *Journal of Electronic Testing*, vol. 18, no. 3, pp. 351–356, 2002.

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

[5] D. W. Coit and A. E. Smith, "Reliability Optimization of Series-Parallel Systems Using a Genetic Algorithm," *IEEE Transactions on Reliability*, vol. 45, no. 1, pp. 254–260, 1996.

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

[7] ——, "Symbolic Reliability Analysis and Optimization of ECU Networks," in *Proc. of DATE '08*, 2008, pp. 158–163.

[8] ——, "Incorporating Graceful Degradation into Embedded System Design," in *Proc. of DATE '09*, 2009, pp. 320–323.

[9] V. Izosimov et al., "Synthesis of fault-tolerant schedules with transparency/performance trade-offs for distributed embedded systems," in *Proc. of DAC '04*, 2004, pp. 550–555.

[10] ——, "Synthesis of fault-tolerant embedded systems with checkpointing and replication," in *Proc. of DELTA '06*, 2006, pp. 440–447.

[11] A. Jhumka et al., "A dependability-driven system-level design approach for embedded systems," in *Proc. of DATE '05*, 2005, pp. 372–377.

[12] R. Karri and A. Orailoğlu, "Transformation-Based High-Level Synthesis of Fault-Tolerant ASICs," in *Proc. of DAC '92*, 1992, pp. 662–665.

[13] V. Kianzad and S. S. Bhattacharyya, "CHARMED: A Multi-Objective Co-Synthesis Framework for Multi-Mode Embedded Systems," in *Proc. of ASAP '04*, 2004, pp. 28–40.

[14] M. Lukasiewycz et al., "Efficient symbolic multi-objective design space exploration," in *Proc. of ASP-DAC '08*, 2008, pp. 691–696.

[15] S. Neema, "System Level Synthesis of Adaptive Computing Systems," Ph.D. dissertation, Vanderbilt University, Nashville, Tennessee, May 2001.

[16] R. Niemann and P. Marwedel, "An Algorithm for Hardware/Software Partitioning Using Mixed Integer Linear Programming," *Design Automation for Embedded Systems*, vol. 2, no. 2, pp. 165–193, 1997.

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

[18] L. Thiele et al., "Design Space Exploration of Network Processor Architectures," *Network Processor Design: Issues and Practices*, vol. 1, pp. 55–89, Oct. 2002.

[19] S. Tosun et al., "Reliability-Centric High-Level Synthesis," in *Proc. of DATE '05*, 2005, pp. 1258–1263.

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

[21] Y. Xie et al., "Reliability-Aware Cosynthesis for Embedded Systems," in *Proc. of ASAP '04*, 2004, pp. 41–50.

[22] Y. Zhang et al., "Energy-aware deterministic fault tolerance in distributed real-time embedded systems," in *Proc. of DATE '05*, 2005, pp. 372–377.

[23] E. Zitzler et al., "Performance assessment of multiobjective optimizers: an analysis and review." *IEEE Trans. Evol. Computation*, vol. 7, no. 2, pp. 117–132, 2003.