This is the author's version of the work. The definitive work was published in *Proceedings of Design, Automation and Test in Europe (DATE 2008), pp. 158-163, 2008.* The work is supported in part by the German Science Foundation (DFG), SFB 694 and SPP 1148.

Symbolic Reliability Analysis and Optimization of ECU Networks

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

Increasing reliability at a minimum amount of extra cost is a major challenge in todays ECU network design. Considering reliability as an objective already in early design phases has the potential to avoid expensive modifications in later design phases. Hence, there is a need for an appropriate optimization process and efficient analysis techniques to evaluate the found implementations. In this paper, we will show how symbolic techniques can be used to efficiently analyze and optimize such reliable systems. The contribution of this paper is (1) a symbolic reliability analysis that makes use of a partitioned structure function and (2) a symbolic optimization process based on binary ILP solvers. Our case study from the automotive area will show a significant speed-up using our analysis technique. Moreover, our optimization approach is able to offer implementations with considerably improved reliability at no additional costs as well as implementations with reduced costs without decreasing their reliability.

1. Introduction

Nowadays, embedded systems applied for example in automotive or avionics applications consist of many *electronic control units* (ECUs) that control up to hundreds of sensors and actuators. The ECUs themselves are connected via shared or private buses leading to complex networked embedded systems hosting often hundreds of tasks. Different constraints such as monetary costs, energy consumption, bus load, and reliability are imposed on the implementation of these systems.

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. These spaces can be found near sensors or within, e.g., an engine or moving parts. This development increases the amount of permanent hardware defects, and, hence, should be considered already at early phases of the ECU network design.

Although common methods such as resource replication are widely used to increase the systems reliability, they can only be used in safety critical applications 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 in system design. Furthermore, the available space for integrating electronics and ECUs is heavily constrained and, depending on the actual position in the vehicle, expensive. This makes resource replication prohibitive.

As a remedy, we will propose a novel symbolic system level design methodology that performs an automated optimization of the system design and integrates the ability to reuse resources in the system model. Thus, we are able to avoid unreasonable resource replication. Moreover, we will introduce an efficient symbolic technique for the reliability analysis of such systems. This analysis technique makes use of a partitioning technique that efficiently utilizes *Binary Decision Diagrams* (BDDs) [3] to determine the reliability of large ECU networks. The symbolic optimization technique is based on specialized binary *ILP solvers* [13] and accelerates the sifting of the design space while simultaneously considering all kinds of objectives, especially reliability.

The remainder of the paper is outlined as follows: Section 2 discusses prior work on the field of reliable system design. In Section 3 we will give a short formulation of the problem we target in this work. Section 4 introduces our symbolic analysis technique while our optimization approach is presented in Section 5. Section 6 shows the results of the introduced techniques applied to an automotive ECU network before we conclude the paper in Section 7.

2. Related Work

For the analysis of embedded systems, several approaches have been presented that focus on reliability and fault-tolerance. An overview of these techniques can be found in [2].

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 [16]. In [6], fault-tolerant schedules with respect to performance requirements are synthesized using task re-execution. The

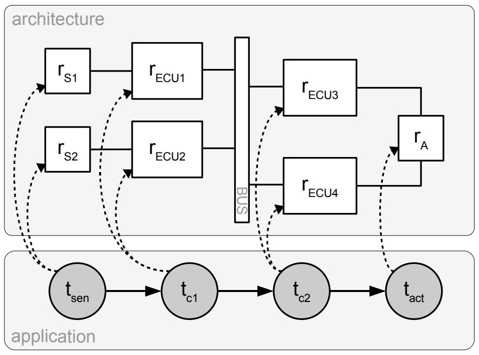

Figure 1. A system specification. The dashed edges correspond to a possible mapping of tasks to resources.

same authors introduce another approach for hard real-time systems in [7], using rollback recovery and active replication. Hereby, fault-tolerance policy assignment, checkpoint placement and task-to-processor mapping is performed automatically. Other approaches such as [14, 15] 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. In our approach, we expect the tolerance of soft errors to be considered at ECU level and target hardware defects of resources, e.g., ECUs, buses, or sensors at system level.

[8] introduces reliability as an optimization objective into system-level design. The used fault-tolerance technique is resource replication that leads to highly increasing cost for reliability. The degree of redundancy is varied by the used *Evolutionary Algorithm*. Anyway, the reliability analysis is not able to handle resource reuse for multiple tasks.

In [4], a reliability analysis and optimization approach is presented, tailored for tolerating hardware defects. The introduced analysis technique uses a BDD representation for the system structure as well, but suffers for complex examples as they are common in ECU network design. Moreover, a *Multi-Objective Evolutionary Algorithm* (MOEA) is used for the optimization process. We will compare our improved symbolic analysis as well as our symbolic optimization to this approach in Section 6, showing a significant speed-up in the analysis and solutions of higher quality in the optimization approach.

Symbolic optimization of embedded systems without a redundant layout of tasks is presented in [10, 11, 9]. We will extend the approach presented in [9] by introducing a redundant task layout into the used ILP.

3. Problem Formulation

In this paper, we target the problem of *design space exploration* of ECU networks at system level with a focus on reliability. Hereby, the assumed failure model are permanent failures due to resource defects. 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 implementations*. A Pareto-optimal implementation is better in at least one objective when compared to any other feasible implementation in the design space [17].

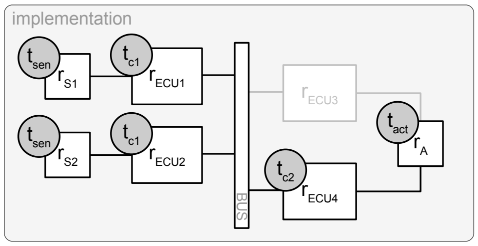

Figure 2. A feasible implementation.

3.1. Specification

Our formal specification consists of the *architecture*, the *application*, and the relation between these two views:

- The architecture is modeled by a graph $g_{\mathsf{a}}(V_{\mathsf{a}}, E_{\mathsf{a}})$ and represents possible interconnected hardware resources. The vertices $r_1, ..., r_{|V_{\mathsf{a}}|} \in V_{\mathsf{a}}$ represent the individual resources, e.g., ECUs, buses, or sensors. The edges E_{a} model available communication links.
- The application is modeled by a task graph $g_{\mathsf{t}}(V_{\mathsf{t}}, E_{\mathsf{t}})$ that describes the behavior of the system. The vertices $t_1, ..., t_{|V_{\mathsf{t}}|} \in V_{\mathsf{t}}$ denote tasks whereas the directed edges E_{t} are data dependencies. Furthermore, we assume the tasks to execute periodically as is most often the case in automotive ECU networks.
- The set of *mapping edges* E_{m} are indicating whether a specific task can be executed on a hardware resource. Each mapping edge $m_1, ..., m_{|E_{\mathsf{m}}|} \in E_{\mathsf{m}}$ is a directed edge from a task to a resource.

An example for a system specification is shown in Figure 1. An *implementation* is deduced from the specification and consists of two main parts:

- The *allocation* $\alpha \subseteq V_{\mathsf{a}}$ represents the ECU network resources that will actually be used and implemented.
- The *binding* $\beta \subseteq E_{\mathsf{m}}$ determines on which allocated resource each task is executed.

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.*

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

With this knowledge, we define an implementation x as a pair (α, β), with a feasible implementation requiring a feasible binding for the given allocation. A feasible implementation for the example in Figure 1 is shown in Figure 2.

3.2. Multiple Binding and the Instance Graph

In common design space exploration approaches, the number of activated mappings of each task is constrained to be one [8, 10, 11]. Hence, this leads to a system without a redundant layout of tasks. In our approach, the activation of more than one mapping per task is allowed, thus creating several *instances* of one task:

$$\forall t \in V_\mathrm{t} : |\{m | m = (t, r) \in \beta\}| \geq 1 \tag{2}$$

The positive effect of this multiple binding is the transparency for the designer, since no explicit modeling of a redundant layout of tasks is needed.

A binding β can also be represented by an *instance graph* $g_\beta(V_\beta, E_\beta)$. The vertices correspond 1:1 to activated mappings $m \in \beta$ and each vertex represents 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.

The feasible implementation in Figure 2 shows a multiple binding, e.g., task t_{C1} is mapped to r_{ECU1} as well as r_{ECU2}. Figure 3 contains the corresponding instance graph. Note that there is no communication link between $S1$ and $ECU2$ as well as $S2$ and $ECU1$. Therefore, no edge exists between the instances (t_{sen}, r_{S1}) and (t_{C1}, r_{ECU2}) as well as (t_{sen}, r_{S2}) and (t_{C1}, r_{ECU1}).

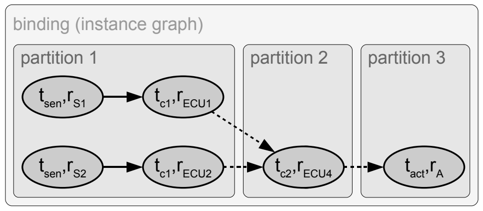

Figure 3. A partitioned instance graph. Dashed edges can be neglected during the reliability analysis.

4. Symbolic Reliability Analysis

In this section, our improved symbolic reliability analysis is introduced in three steps: First, it is described how the so called *structure function* φ [2] can be generated from the system model and represented by *Binary Decision Diagrams* (BDDs) [3] which are an efficient representation of Boolean functions. Afterwards, a partitioning algorithm is introduced that solves the existing problem of oversized BDDs by splitting φ, thus creating smaller BDDs that can be more efficiently put together. In the final step, φ is evaluated to determine the systems reliability.

4.1. The Structure Function φ

To model the systems behavior 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 calculated, cf. [4]. Hereby, 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. This Boolean function indicates a working system by evaluating to $\varphi = 1$ and a system failure by evaluating to $\varphi = 0$, respectively. For a given system specification, this function can be calculated as follows:

$$\varphi_{g_\beta}(\boldsymbol{\alpha}) = \exists \boldsymbol{\beta} : \psi_{g_\beta}(\boldsymbol{\alpha}, \boldsymbol{\beta}) \tag{3}$$

$$\psi_{g_\beta}(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \bigwedge_{t \in V_\mathrm{t}} \left[\bigvee_{m=(t,r) \in V_\beta} \boldsymbol{m} \right] \wedge \tag{4a}$$

$$\bigwedge_{m=(t,r) \in V_\beta} \boldsymbol{m} \rightarrow \boldsymbol{r} \wedge \tag{4b}$$

$$\bigwedge_{(t,\widetilde{t}) \in E_\mathrm{t}} \bigwedge_{\substack{m=(t,r), \\ \widetilde{m}=(\widetilde{t},\widetilde{r}) \in V_\beta}} \boldsymbol{m} \wedge \widetilde{\boldsymbol{m}} \rightarrow C(m, \widetilde{m}) \tag{4c}$$

Hereby, $\boldsymbol{\beta} = (\boldsymbol{m_1}, \ldots, \boldsymbol{m_{|\beta|}})$ is a vector of Boolean variables encoding a task instance being active to ensure a working system. At least one instance of each task $t \in V_\mathrm{t}$ is needed to define a working system. This is ensured by Term (4a). Term (4b) implies that any task instance relies on a correctly working resource. Furthermore, if two instances of data dependent tasks are assumed to contribute to a working system, they must be able to communicate. Term (4c) uses Equation (5) to indicate whether this communication is feasible or not.

$$C(m, \widetilde{m}) = \begin{cases} 1, & \text{if } (m = (t, r), \widetilde{m} = (\widetilde{t}, \widetilde{r})) \in E_\beta \\ 0, & \text{else} \end{cases} \tag{5}$$

φ returns 1 if and only if for a given subset of proper working resources of a given implementation, there exists a set of working and correct communicating task instances that guarantee a proper working system.

4.2. The Partitioning Approach

The worst-case complexity of building a BDD is exponential in the number of variables, i.e., $\mathcal{O}(2^{|\alpha|+|\beta|})$ for φ in Equation (3). Building the BDD for φ has two main phases: First, the BDD for ψ is constructed including all temporal variables $\boldsymbol{\beta}$. Afterwards, the temporal variables are eliminated using the $\exists$-quantification. Hence, this may result in very large BDDs after the first phase, while the BDDs strongly decrease in size during the second phase. Moreover, if the BDDs of the first phase grow too large to fit the used computers memory, an analysis becomes impossible. As a remedy, we show how the construction of the φ-BDD can be split such that an *early quantification* reduces the size of the temporal ψ-BDDs. The quantified temporal BDDs can be connected with only the $\boldsymbol{\alpha}$ variables being left, thus strongly decreasing the complexity of creating the φ-BDD.

Lemma 1. *Let g_{t} be a task graph and g_β be a corresponding instance graph. Given a data dependency $e = (t, \widetilde{t}) \in E_{\mathsf{t}}$, it holds:*

$$\psi_{g_\beta = (V_\beta, E_\beta)} = \psi_{\widetilde{g_\beta} = (V_\beta, \widetilde{E_\beta})}$$
$$\textit{with } \widetilde{E_\beta} = E_\beta \setminus \{e' | e' = (m, \widetilde{m})\} \quad (6)$$

if
$$\forall m = (t, r), \widetilde{m} = (\widetilde{t}, \widetilde{r}) \in \beta : (m, \widetilde{m}) \in E_\beta. \quad (7)$$

In other words, if all instances of two data dependent tasks t and $\widetilde{t}$ can communicate with each other, their data dependency e can be neglected without falsifying ψ.

Proof. If Equation (7) is fulfilled by a data dependency e, Equation (5) will always evaluate to 1. Hence, e has no effect on ψ and thus can be neglected. □

Definition 2. *An instance graph g_β can be partitioned into two subgraphs g_{β_1}, g_{β_2} with $\beta_1 \cap \beta_2 = \emptyset$ and $\beta_1 \cup \beta_2 = \beta$ if Lemma 1 holds for all $(t_1, t_2) \in E_{\mathsf{t}}$ with $m_1 = (t_1, r_1) \in \beta_1$ and $m_2 = (t_2, r_2) \in \beta_2$.*

An algorithm that determines these partitions has a quadratic worst-case complexity $\mathcal{O}(|\beta|^2)$ and is dominated by the complexity of building the ψ-BDDs. A partitioned instance graph is shown in Figure 3. With Definition 2, Equation (3) can be split as follows:

Lemma 2. *Given an instance graph g_β and two partitions g_{β_1}, g_{β_2}, it holds:*

$$\varphi_{g_\beta}(\boldsymbol{\alpha}) = \varphi_{g_{\beta_1}}(\boldsymbol{\alpha}) \wedge \varphi_{g_{\beta_2}}(\boldsymbol{\alpha}) \quad (8)$$

Proof. See Appendix. □

With this Lemma, the aspired *early quantification* is performed since the ψ-BDDs of the subgraphs can be quantified using the $\exists$-quantifier before they are connected. Hence, this can create a major speed-up and can prevent a failure of the analysis due to outsized BDDs.

4.3. Evaluating φ

In the following, we describe how to quantify the reliability of the ECU network based on the determined structure function φ. Since the reliability R_r of each resource $r \in V_{\mathsf{a}}$ is given in the system model, e.g., by distribution functions like an *exponential distribution* or a *Weibull distribution*, the reliability of the system at time t can can be calculated through using a modified *Shannon-decomposition* [12] on the BDD representing φ:

$$R(t) = \varphi = R_r(t) \cdot \varphi|_{\boldsymbol{r}=1} + (1 - R_r(t)) \cdot \varphi|_{\boldsymbol{r}=0} \quad (9)$$

If a *Mission Time* (MT) of the system is given, this decomposition directly quantifies the reliability $R(MT)$ of the system. In our experimental results, the *Mean Time To Failure* (MTTF) $\int_0^\infty R(t)dt$ is used as the measure of reliability and is determined by a numerical integration of Equation (9). MTTF denotes the expected value for the failure-free time of the system.

5. Symbolic Optimization

With the introduced efficient automatic reliability analysis, reliability can be introduced as one out of many objectives in the optimization of ECU networks. Of course, redundant task layout through multiple binding leads to many additional implementations that all have to be considered during the optimization. Moreover, already finding a single feasible implementation is known to be an NP-complete problem [5]. Since we have to deal with large and complex ECU networks, this becomes the main challenge for state-of-the-art population-based optimization approaches [4, 8] that are usually more focused on the search for feasible implementations than on the optimization of the found implementations. As a remedy, we extend the symbolic optimization strategy presented in [9] that formalizes the problem of finding a feasible implementation as an *Integer Linear Program* with binary variables (*0-1 ILP*) [1]. This enables the optimization process to focus on the optimization of the found feasible implementations. In this section, the formalization of our problem as a 0-1 ILP including multiple-binding of tasks is introduced and it is described how a symbolic search can be used for an efficient optimization approach.

5.1. 0-1 ILP Model

The goal of *Integer Linear Programming* is the optimization of a linear objective function, subject to linear equality and inequality constraints. If the objective function is omitted, an ILP solver will find a single solution that fulfills all constraints. As the first step towards a 0-1 ILP model, an implementation has to be encoded into a binary vector $\mathbf{x} = (\boldsymbol{r_1}, ..., \boldsymbol{r_{|V_{\mathsf{a}}|}}, \boldsymbol{m_1}, ..., \boldsymbol{m_{|E_{\mathsf{m}}|}})$. Hereby, for each resource r and mapping edge m a binary variable $\boldsymbol{r}$ and $\boldsymbol{m}$ is introduced, respectively. For a variable $\boldsymbol{r}$, a 1 indicates that this resource is part of the allocation α and for a variable $\boldsymbol{m}$, the value 1 means that this mapping edge is in use and part of the binding β. Correspondingly, the value 0 indicates that the resource or mapping edge, respectively, are not allocated or used.

One vector $\mathbf{x} \in X = \{0, 1\}^{|V_{\mathsf{a}}| + |E_{\mathsf{m}}|}$ represents one implementation x. Therefore, the constraints have to formulated such that they are satisfied if and only if the implementation $\mathbf{x}$ is feasible, cf. Definition 1. These constraints can be build by the following rules:

Constraint Rule 1. *Each used mapping edge has to end at an allocated resource.*

$$\forall m = (t, r) \in E_{\mathsf{m}} : \quad \boldsymbol{r} - \boldsymbol{m} \geq 0 \quad (10)$$

Constraint Rule 2. *Data-dependent tasks have to be mapped to the same or to an adjacent resource.*

$$\forall m = (t, r) \in E_{\mathsf{m}} \wedge (t, \tilde{t}) \in E_{\mathsf{t}} :$$
$$-\boldsymbol{m} + \sum_{\substack{\tilde{m} = (\tilde{t}, \tilde{r}) \in E_{\mathsf{m}}: \\ r = \tilde{r} \vee (r, \tilde{r}) \in E_{\mathsf{a}}}} \tilde{\boldsymbol{m}} \geq 0 \quad (11)$$

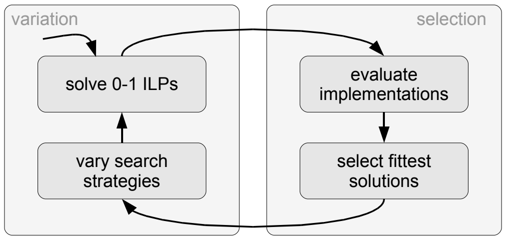

Figure 4. The symbolic optimization process, driven by an Evolutionary Algorithm.

Constraint Rule 3. *For each task $t \in V_\mathsf{t}$ at least one mapping edge has to be activated.*

$$\forall t \in V_\mathsf{t} : \sum_{m=(t,r)\in E_\mathsf{m}} \boldsymbol{m} \geq 1 \tag{12}$$

This last constraint rule is crucial for the creation of redundancy, since it allows multiple instances of one task in the system. Note that this constraint can also be varied within the search process, e.g., by incrementing the lower bound to force the solver to multiple bind a certain instance or by introducing an upper bound for the number of activated mappings, restricting the maximum number of instances of a certain task, respectively.

5.2. Overall Optimization Process

With the constraints defined in Section 5.1, an ILP solver will find one specific feasible implementation for a given ECU network. Specialized 0-1 ILP solvers, the so called *PB solvers*, are known to be superior to common ILP solvers on problems with binary variables [1]. These PB solvers are implementing a binary search backtracking strategy. The search strategy is guided by two vectors: $\rho \in \mathbb{R}^n$ and $\sigma \in \{0,1\}^n$. Hereby, ρ denotes the priority of a certain variable within the search while σ determines the prioritized value.

Different implementations can be found by varying the *search process* or vectors ρ and σ, respectively. Therefore, our optimization algorithm varies the vectors ρ and σ and searches for optimal implementations. This search towards optimal implementations is performed by an *Evolutionary Algorithm* (EA), a heuristic that is based on the principles of biological evolution. An EA is performing in two steps: (1) A variation of the implementations by *mutation* and *crossover*, and (2), a selection of good implementations based on their fitness. In our case, the variation of the EA is performed on the vectors ρ and σ. With these vectors, a feasible implementation is determined. The selection of the EA is based on a fitness value of each feasible implementation that is calculated with respect to the determined objectives such as, e.g., costs, area consumption, and reliability. The iterative overall optimization approach is illustrated in Figure 4.

method	analysis time [ms]	deviation
standard [4]	474	2103
partitioned	12.1	8.27

Table 1. ALC: Average time consumption for the reliability analysis of one implementation.

6. Experimental Results

This section presents the results of the introduced techniques applied to a complex specification of an ECU network from the automotive area. In particular, this specification of an *Adaptive Light Control* (ALC), cf. [4], consists of 234 tasks. There are 1103 resources available including several ECUs, buses, and sensors. With 1851 mapping edges, this case study has $\approx 2^{375}$ binding possibilities. Additionally, bus load and end-to-end timing constraints have to be fulfilled. The constraint handling is part of the optimization process whereas violated constraints are penalized by the Evolutionary Algorithm.

We will compare our results to the results presented in [4]. Therefore, we used the same settings for the Evolutionary Algorithm, i.e., a population size of 100 implementations for 500 generations. All experiments were carried out on an Intel Pentium 4 3.20 GHz machine with 1 GB RAM.

First, the runtimes of the analysis methods are compared. This study is based on 125,750 different implementations of our real-world example. The results in Table 1 show that our introduced partitioning approach creates a 39-time speed-up on average. The high standard deviation is due to the poor scaling of the BDDs on implementations with a high degree of redundancy. This behavior is particularly apparent with the standard technique where even some implementations could not be evaluated due to the limited memory of the used machine. In contrast, these complex implementations have been analyzed by our partitioning approach even without a significantly higher time consumption.

The results of our symbolic optimization approach compared to the EA-based algorithm presented in [4] are shown in Figure 5. In this case, the results are compared with respect to the following objectives: (1) Reliability given in terms of mean time to failure (MTTF) in years is to be maximized and (2) abstract costs that arise from the usage of mounting space and monetary resource costs are to be minimized. Comparing both approaches, our symbolic optimization offers implementations with a ≈ 38 % greater MTTF at equal cost. Comparing the approaches by reliability, the implementations found by the symbolic approach are about ≈ 32 % cheaper at equal reliability what is a remarkably value at system level. Low cost implementations are of special interest in automotive applications. A comparison of these implementations shows that the symbolic approach offers an 18 % cheaper implementation that even has an 8 % greater MTTF. These results show that our symbolic approach clearly outperforms the standard EA-based approach and offers a variety of implementations of remarkably high quality to the ECU network designer.

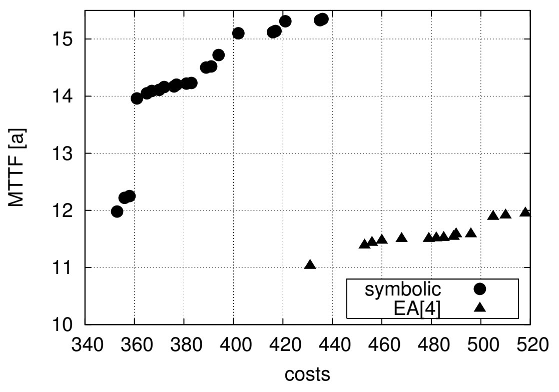

Figure 5. Best implementations found via our symbolic and EA-based [4] optimization. Shown is the cost - reliability trade-off for a real automotive light control application.

In [4], the average runtime of an optimization run is stated as about 13,000 s. The runtime of one symbolic optimization run is 380 s on average which equals a more than 30-time speed-up, gained by our new analysis approach. Thus, the introduced case study showed that our optimized symbolic reliability analysis and optimization considerably outperforms former approaches in both, speed and quality of the found implementations.

7. Conclusion

In this paper, we introduced symbolic techniques for the automatic reliability analysis and optimization of ECU networks at system level. The presented analysis technique makes use of a partitioning algorithm to overcome the problems arising from outsized BDDs. Our extended symbolic optimization approach prunes infeasible implementations from the optimization process and allows for a faster and better convergence to global optimal implementations. Both introduced techniques are tailored to consider the reuse of resources as a redundancy technique to efficiently handle the amount of extra cost. The case study of a complex automotive adaptive light control showed that our partitioning algorithm could achieve a significant speed-up compared to a common analysis approach. Furthermore, the introduced symbolic optimization methodology strongly outperforms former methods that are based on Evolutionary Algorithms only. In the future, we will extend our reliability analysis technique to combine the consideration of permanent defects as well as temporary soft-errors for ECU networks.

References

[1] F. A. Aloul, A. Ramani, I. L. Markov, and K. A. Sakallah. Generic ilp versus specialized 0-1 ilp: an update. In *Proc. of ICCAD '02*, pages 450–457, 2002.

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

[3] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

[4] M. Glaß, M. Lukasiewycz, T. Streichert, C. Haubelt, and J. Teich. Reliability-Aware System Synthesis. In *Proc. of DATE '07*, pages 409–414, 2007.

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

[6] V. Izosimov, P. Pop, P. Eles, and Z. Peng. Synthesis of fault-tolerant schedules with transparency/performance trade-offs for distributed embedded systems. In *Proc. of DAC '04*, pages 550–555, 2004.

[7] V. Izosimov, P. Pop, P. Eles, and Z. Peng. Synthesis of fault-tolerant embedded systems with checkpointing and replication. In *Proc. of DELTA '06*, pages 440–447, 2006.

[8] A. Jhumka, S. Klaus, and S. A. Huss. A dependability-driven system-level design approach for embedded systems. In *Proc. of DATE '05*, pages 372–377, 2005.

[9] M. Lukasiewycz, M. Glaß, C. Haubelt, and J. Teich. Sat-decoding in evolutionary algorithms for discrete constrained optimization problems. In *Proc. of CEC '07*, pages 935–942, 2007.

[10] S. Neema. *System Level Synthesis of Adaptive Computing Systems*. PhD thesis, Vanderbilt University, Nashville, Tennessee, May 2001.

[11] R. Niemann and P. Marwedel. An Algorithm for Hardware/Software Partitioning Using Mixed Integer Linear Programming. *Design Automation for Embedded Systems*, 2(2):165–193, 1997.

[12] A. Rauzy. New Algorithms for Fault Tree Analysis. *Reliability Eng. and System Safety*, 40:202–211, 1993.

[13] H. M. Sheini and K. A. Sakallah. Pueblo: A modern pseudo-boolean sat solver. In *Proc. of DATE '05*, pages 684–685, 2005.

[14] S. Tosun, N. Mansouri, E. Arvas, M. Kandemir, and Y. Xie. Reliability-Centric High-Level Synthesis. In *Proc. of DATE '05*, pages 1258–1263, 2005.

[15] Y. Xie, L. Li, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin. Reliability-Aware Cosynthesis for Embedded Systems. In *Proc. of ASAP '04*, pages 41–50, 2004.

[16] Y. Zhang, R. Dick, and K. Chakrabarty. Energy-aware deterministic fault tolerance in distributed real-time embedded systems. In *Proc. of DATE '05*, pages 372–377, 2005.

[17] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca. Performance assessment of multiobjective optimizers: an analysis and review. *IEEE Trans. Evol. Computation*, 7(2):117–132, 2003.

A. Appendix

Proof. Given Equation (6), $\psi_{g_\beta}(\boldsymbol{\alpha}, \boldsymbol{\beta})$ can be calculated as:

$$\psi_{g_\beta}(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \psi_{g_{\beta_1}}(\boldsymbol{\alpha}, \boldsymbol{\beta_1}) \wedge \psi_{g_{\beta_2}}(\boldsymbol{\alpha}, \boldsymbol{\beta_2}) \tag{A.1}$$

Following Equation (3) leads to:

$$\varphi_{g_\beta}(\boldsymbol{\alpha}) = \exists \boldsymbol{\beta} : \left(\psi_{g_{\beta_1}}(\boldsymbol{\alpha}, \boldsymbol{\beta_1}) \wedge \psi_{g_{\beta_2}}(\boldsymbol{\alpha}, \boldsymbol{\beta_2})\right) \tag{A.2}$$

By definition[1], this can also be written as:

$$\varphi_{g_\beta}(\boldsymbol{\alpha}) = \exists \beta_1 : \psi_{g_{\beta_1}}(\boldsymbol{\alpha}, \boldsymbol{\beta_1}) \wedge \exists \beta_2 : \psi_{g_{\beta_2}}(\boldsymbol{\alpha}, \boldsymbol{\beta_2}) \tag{A.3}$$

This is equivalent to Equation (8). □

[1] $\exists x, y : f(x,z) \wedge g(y,z) = \exists x : f(x,z) \wedge \exists y : g(y,z)$