Exploiting Data-Redundancy in Reliability-Aware Networked Embedded System Design *

Martin Lukasiewycz, Michael Glaß, and Jürgen Teich
University of Erlangen-Nuremberg, Germany
{martin.lukasiewycz,glass,teich}@cs.fau.de

ABSTRACT

This paper presents a system-level design methodology for networked embedded systems that exploits existing data-redundancy to increase their reliability. The presented approach not only supports a reliability-aware embedded system design from scratch, but also enables the redesign of existing systems to increase the reliability with a minimal communication overhead. The proposed approach contributes (a) algorithms to automatically identify inherent data-redundancy and (b) an automatic design space exploration that is capable of exploiting the revealed data-redundancy. A symbolic analysis is presented that quantifies the reliability of a system, enabling the usage of reliability as one of multiple conflicting optimization objectives. The proposed approach is applied to a real-world case study from the automotive area, showing a significantly increased reliability with a negligible communication overhead.

Categories and Subject Descriptors

C.4 [**PERFORMANCE OF SYSTEMS**]:
Modeling techniques, Fault tolerance

General Terms

Reliability

1. INTRODUCTION

Modern networked embedded systems as found in the automotive or avionics area consist of up to a hundred computational units, interconnected via arbitrary communication resources like buses and gateways. A large number of distributed functions are executed on the computational units, inducing a high data volume on the communication resources. In general, these distributed functions are designed and implemented separately, resulting in a certain degree of redundancy in terms of the communication data. This existing *data-redundancy* can be used to increase the reliability of a system. Consider some functions from the automotive area like, e.g., *X-by-wire* [19] or advanced driver assistance functions: These functions require several input data that are obtained from sensors. For instance, the *vehicle speed* is delivered by either wheel sensors, radar sensors, or via the analysis of GPS data. If a function receives the current vehicle speed from all these sensors, the functionality is not affected in case one of the sensors is still working properly. However, due to the separate development of the functions, information about this redundant presence of the vehicle speed in the overall integrated system is commonly neglected. On the other hand, the increase of reliability has to be traded against the additional communication overhead. Due to the complexity of this task, an automatic design flow, as illustrated in Fig. 1 and proposed in the work at hand, is a promising approach for obtaining high quality implementations in terms of the reliability and a set of remaining objectives.

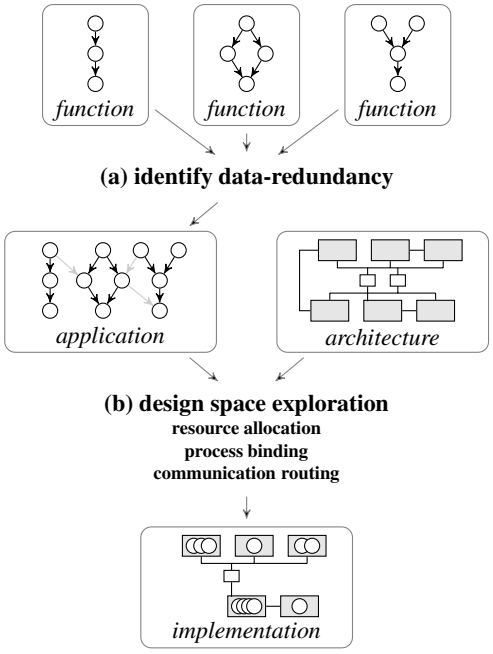


Figure 1: Proposed design flow: Several functions are combined to an application that includes the inherent data-redundancy. Given the application and a target architecture, the design space exploration that includes the resource allocation, process binding, and communication routing is used to find a high quality implementation, utilizing the data-redundancy.

Contributions of the paper: Increasing the reliability with a

*Supported in part by the German Science Foundation (DFG), SFB 694

CODES+ISSS'09, October 11–16, 2009, Grenoble, France.

minimal amount of additional cost is one of the most challenging tasks in the design of complex networked embedded systems, cf. [1]. To tackle this problem, the work at hand proposes a novel embedded system-level design methodology that aims to increase the reliability of the networked embedded system by exploiting its inherent data-redundancy. It contributes (a) algorithms that are capable of identifying existing data-redundancy among functions, resulting in a novel application model that includes existent data-redundancy, and (b) a state-of-the-art design space exploration that utilizes an analytical approach to quantify the reliability of a system.

As illustrated in Fig. 1 (a), an identification of the data-reduncancy of the distributed functions is carried out to obtain a redundancy-aware application. For this purpose, it becomes necessary to gather detailed information about the distributed functions, i.e., the produced and consumed *signals*. Here, a signal is the basic data unit. One process consumes and produces multiple different signals, given in a set-based representation. To add the additional data-redundancy information directly to the application, this work presents both a graph-based and a functional approach. The functional approach extends the graph-based approach and is capable of modeling also more complex data-dependencies as common in, e.g., data-fusion scenarios.

The design space exploration as illustrated in Fig. 1 (b) follows the Y-chart approach: For a given application and architecture, a high quality implementation is determined within a multi-objective optimization. The presented design space exploration performs a resource allocation, process binding, and communication routing concurrently. For this purpose, the design space exploration problem is encoded in linear constraints with multiple also non-linear conflicting objectives such that a novel hybrid optimization approach is utilized. This hybrid optimization approach uses an Integer Linear Program with binary variables (0-1 ILP) and an Evolutionary Algorithm (EA). While the 0-1 ILP ensures that obtained implementations are feasible, the EA iteratively improves the objectives of these implementations. In order to utilize the data-redundancy and to increase the reliability of the implementations, a quantification of the reliability becomes necessary. In this paper, a reliability analysis is proposed based on the usage of *Boolean functions*, efficiently encoded in *Binary Decision Diagrams* (BDDs) [2]. With a proposed construction scheme, a BDD representation of the system structure can be derived and evaluated to obtain appropriate measures like, e.g., the *Mean-Time-To-Failure* (MTTF).

Organization of the paper: The remainder of the paper is outlined as follows: Section 2 discusses related work. Section 3 introduces the system model and the proposed extension for the modeling of the data-redundancy in the application. The algorithms for identification of the inherent data-redundancy in the application are presented in Section 4. The proposed design space exploration approach as well as the developed reliability analysis are introduced in Section 5. Experimental results are given in Section 6 before the paper is concluded in Section 7.

2. RELATED WORK

In the past, a lot of effort has been put in the improvement of reliability of automotive electronic components, cf. [21]. Recently, several approaches target a robust and reliable system design at various levels of hardware and software as a whole: At lower levels of abstraction, reliability is increased by introducing techniques that are tailored to special bus architectures like, e.g., TTP/C [5, 13] and their communication layers. Other approaches like [6] deal with the verification of reliability only, neglecting a possible reliability increase by utilizing given redundancy. At system level, approaches to increase reliability are embedded in the tasks of resource allocation, task binding, message routing, and scheduling. Fault-tolerance via checkpointing and power management based on *dynamic voltage scaling* is introduced in [22]. In [11], fault-tolerant quasi-static schedules with respect to given deadlines are proposed. The same authors present an approach for hard real-time systems in [4], using rollback recovery and active replication. Approaches such as [18, 20] try to maximize reliability by selective redundancy-introduction strategies with reliability being the only objective while other objectives are treated as constraints. The system-level design approach presented in [12] introduces reliability as an optimization objective, while using resource replication to increase reliability. In [7], a reliability analysis and optimization approach is presented, that is tailored to the toleration of hardware defects in ECU networks by performing a multiple binding of tasks to ECUs to create redundancy at task level. All approaches actively introduce redundancy using, e.g., task re-execution, multiple task-binding, or resource multiplication without utilizing the existing data-redundancy. The result is additional overhead ranging from relatively low cost solutions like task re-execution increasing runtimes, over task multiplication increasing the necessary computational capacities, up to resource multiplication that increases monetary costs and the system size.

To the best of the authors knowledge, the work at hand is the first embedded system-level design methodology that exploits existing data-redundancy in networked embedded systems. The additional overhead resulting from the utilization of this type of redundancy is restricted to sending multi-cast messages, increasing the function delay and busload marginally.

3. MODEL

This section presents the basic exploration model and proposes an extension of the application model to allow the handling of redundancy.

3.1 Basic Model

The exploration model is defined by a *specification* that consists of an *application* and an *architecture*. From this specification, various *implementations* can be derived by defining the *allocation* of the architecture, the *mapping* of the application, and the *routing* of communication data. This Y-chart approach is illustrated in Fig. 1.

The specification consists of an *architecture graph* G_R and an *application graph* G_T :

- The architecture is given by a directed graph $G_R(R, E_R)$. The vertices R represent resources such as processors, memories, or buses. The directed edges E_R indicate available communication connections between two resources.
- The application is given by a bipartite directed graph $G_T(T, E_T)$ with $T = P \cup C$. The vertices T are either process tasks $p \in P$ or communication tasks $c \in C$. Each edge $e \in E_T$ connects a vertex in P to one in C, or vice versa. Each process task can have multiple incoming edges that indicate the data-dependencies to communication information of the predecessor communication tasks. On the other hand, each communication task has exactly one predecessor process task as the sender, but a process task can of course have multiple successor communication tasks. To allow multicasts, each communication task can have multiple successor process tasks.

Each process task $p \in P$ can be implemented on a resource from R_p with $R_p \subseteq R$. Each communication task $c \in C$ can be routed on a subset of resources from R_c with $R_c \subseteq R$. An example of an application graph is illustrated in Fig. 2. An architecture and the corresponding graph representation is given in Fig. 3.

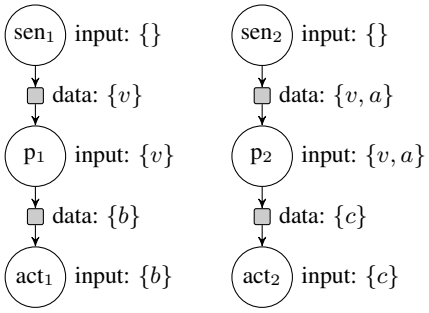


Figure 2: Sample application including two distributed functions, each consisting of one sensor task, one processing task, and one actuator task. The communication data passed between these tasks fulfills the input requirements.

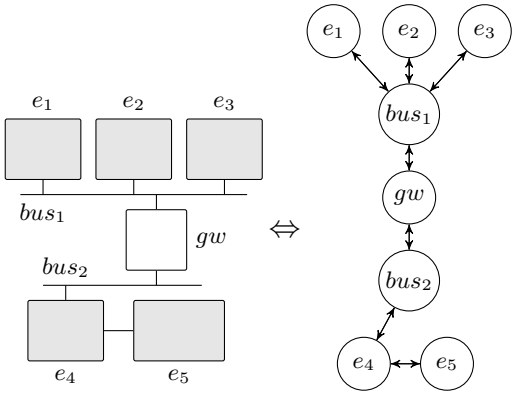


Figure 3: Sample architecture consisting of five ECUs $\{e_1, e_2, e_3, e_4, e_5\}$, two buses $\{bus_1, bus_2\}$, and a gateway gw as well as the corresponding graph representation.

One implementation consists of the allocation graph G_A that is deduced from the architecture graph and a function i that maps the application onto the allocation graph.

- The allocation is a directed graph $G_A(A, E_A)$ that is an induced subgraph of the architecture graph G_R. The allocation contains all resources that are available in the current implementation and the edges are induced from the graph G_R such that G_A is aware of the communication connections.
- Each process task $p \in P$ is bound to exactly one allocated resource $i(p)$ such that $i(p) \in (A \cap R_p)$. Each communication task $c \in C$ is routed on a tree that is a subgraph of the allocation such that $i(c) \subseteq G_A$ with all vertices in R_c. These bindings and routings have to be performed such that all data-dependencies given by the following two conditions are satisfied:

  1. For each communication task $c \in C$, the root of the routing has to equal the binding of the predecessor sender process task $p \in P$. It holds:

  $$\forall (p, c) \in E_T : root(i(c)) = i(p) \quad (1)$$

  2. For each process task $p \in P$, the routings of the predecessor communication tasks $c \in C$ have to be routed on the same resource as the binding of process p. It holds:

  $$\forall (c, p) \in E_T : i(p) \in i(c) \quad (2)$$

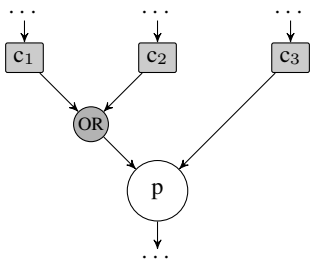


Figure 4: Example of a process task p with preceding communication tasks $\{c_1, c_2, c_3\}$. The *OR* node indicates that the process p requires c_1 or c_2 as well as c_3 to work properly.

An implementation is *feasible* if all requirements regarding the process and communication mapping as well as the data-dependencies are fulfilled.

With the definition of a feasible implementation, the task of the *design space exploration* can be formulated as the following multi-objective optimization problem:

DEFINITION 1 (DESIGN SPACE EXPLORATION).

minimize $f(x)$
subject to:
x *is a feasible* implementation

In real-world problems, the objective function f consists of multiple functions including also non-linear calculations. In single-objective optimization, the feasible set of networks is totally ordered, whereas in multi–objective optimization problems, the feasible set is only partially ordered and, thus, there is generally not only one global optimum, but a set of *Pareto solutions*. A Pareto-optimal solution is better in at least one objective when compared to any other feasible solution.

3.2 Model Extension

In order to allow the basic model to handle redundancy, the specification as well as the implementation have to be extended.

Regarding the specification, the architecture remains the same as in the basic model. However, the application has to be extended by an additional *OR* node type. Thus, the application graph is $G_T(T, E_T)$ with $T = P \cup C \cup O$ where O is a set of *OR* nodes. Each *OR* node has multiple communication task predecessors from the set C and exactly one process task successor from the set P. Thus, the *OR* node indicates that the succeeding process task requires at least one preceding communication task. In Fig. 4, an example of an application with an *OR* node is given: The process task $p \in P$ requires the data either from c_1 or c_2 as well as the data from c_3. This extension of the application is downward compatible to the application in the basic model.

Regarding the implementation, the definition of an allocation remains the same while the mapping function i has to be extended to consider the *OR* nodes. The *OR* nodes are not mapped. The binding of process tasks and routing of communication tasks remains the same. Instead, the two requirements for the satisfaction of data-dependencies, i.e., Equation (1) and (2), have to be extended by a third requirement:

- 3. For each process task $p \in P$ and preceding *OR* node $o \in O$, at least one preceding communication task $c \in C$ of the *OR* node o has to be routed on the same resource as the binding of process p. It holds:

$$\forall (o, p) \in E_T \ \exists (c, o) \in E_T : i(p) \in i(c) \quad (3)$$

This additional requirement ensures the correctness of the data-dependencies for redundant communication data.

4. IDENTIFYING DATA-REDUNDANCY

To exploit information on data-redundancy methodically, the basic system model needs to be converted to the extended model with *OR* nodes as presented in the previous section. In general, distributed functions are developed and specified separately by designer teams. Thus, knowledge about data-redundancy within each function can be exploited already in the specification phase. However, in the integration phase of the design process, the various distributed functions are combined to the system application such that the data-redundancy between the distributed functions is mostly neglected. Due to the immense size and complexity of the overall application, methodical approaches become necessary to reveal existent data-redundancy. In this section, a graph-based and a functional approach are presented that allow for an automatic identification of existing data-redundancy. All proposed algorithms in this section have a polynomial complexity. Thus, the runtime of these algorithms is negligible small compared the design space exploration and the reliability analysis.

Data-dependencies: In the used system model, the basic communication data unit is a *signal* $s \in S$. In general, each communication task, e.g., a *message*, can contain multiple signals. For each communication task $c \in C$, the function $data : C \rightarrow 2^S$ defines the set of signals contained in c. On the other hand, each process task performs some calculations that are based on the input signals. Thus, for each process task $p \in P$, the function $input : P \rightarrow 2^S$ returns the set of required signals for a proper functionality of p.

4.1 Graph-based Approach

Given the $data$ and $input$ functions for each communication or process task, respectively, a fully graph-based approach is applied to extend the basic application graph (which is a collection of the distributed functions) to a redundancy-aware application graph. This procedure is achieved through the following three algorithms: The first algorithm adds the *OR* nodes, while the subsequent two algorithms simplify the application graph.

Algorithm 1 interconnects the communication and process tasks with *OR* nodes to a redundancy-aware application graph.

Algorithm 1 Add redundancy (*OR* node) to application G_T.

Require: $G_T(T, E_T)$ with $T = P \cup C$
1: remove each $(c, p) \in E_T$ with $p \in P, c \in C$
2: **for** $p \in P$ **do**
3: **for** $s \in input(p)$ **do**
4: add new *OR* node o to T
5: add edge (o, p) to E_T
6: **for** $c \in C$ **do**
7: **if** $s \in data(c)$ **then**
8: add edge (c, o) to E_T
9: **end if**
10: **end for**
11: **end for**
12: **end for**

The algorithm requires a common application graph and it is assumed that the $data$ and $input$ functions for each communication or process task, respectively, are known, i.e., user-specified. First, the edges from the communication tasks to the process task are removed (line 1). The algorithm iterates over each process $p \in P$ and each corresponding input signal $s \in input(p)$ (line 2, 3). For each pair of p and s, exactly one *OR* node o is added to the graph and connected to p (line 4, 5). Finally, the communication tasks

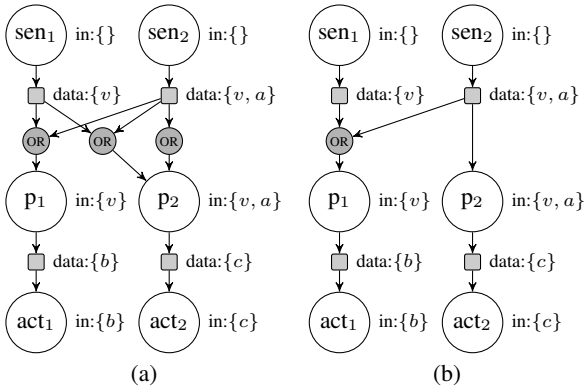


Figure 5: The application graph from Fig. 2 extended by *OR* nodes to a redundancy-aware application graph. This figure illustrates the application graph after applying Algorithm 1 (a) and the same graph after the simplification (b).

$c \in C$ are interconnected with the *OR* nodes that contain the corresponding signal s (line 6-10). Given the application graph in Fig. 2, this algorithm results in the redundancy-aware application graph in Fig. 5 (a).

Given a graph generated by Algorithm 1, two simplifications are applied to remove redundant nodes and edges. Algorithm 2 removes obviously redundant *OR* nodes from the graph. This is applied if an *OR* node has exactly one preceding communication task.

Algorithm 2 Remove redundant *OR* nodes from application graph G_T.

Require: $G_T(T, E_T)$ with $T = P \cup C \cup O$
1: **for** $o \in O$ **do**
2: **if** $|(c, o) \in E_T| = 1 \wedge |(o, p) \in E_T| = 1$ **then**
3: remove *OR* node o from T and all incident edges
4: add edge (c, p) to E_T
5: **end if**
6: **end for**

The algorithm is applied to each *OR* node $o \in O$ in the graph (line 1). If an *OR* node has exactly one predecessor $c \in C$ and successor $p \in P$ (line 2), it is redundant and can be removed (line 3). Instead, an edge between c and p is added (line 4). For the example in Fig. 5 (a), this algorithm removes the redundant *OR* node between the communication task created by sen_2 and p_2.

Algorithm 3 applies the absorption law to the redundancy-aware application graph and removes redundant edges and *OR* nodes.

Algorithm 3 Simplify application G_T by applying the absorption law.

Require: $G_T(T, E_T)$ with $T = P \cup C \cup O$
1: **for** $p \in P$ **do**
2: **for** $t, t' \in C \cup O$ with $t \neq t'$ and $(t, p), (t', p) \in E_T$ **do**
3: $C_t = (\{\tilde{t} | (\tilde{t}, t) \in E_T\} \cup \{t\}) \cap C$
4: $C_{t'} = (\{\tilde{t} | (\tilde{t}, t') \in E_T\} \cup \{t'\}) \cap C$
5: **if** $C_t \subseteq C_{t'}$ **then**
6: remove *OR* node t' from T and all incident edges
7: **end if**
8: **end for**
9: **end for**

The algorithm is applied for each process task $p \in P$ in the graph (line 1). An iteration over all pairs $t, t' \in C \cup O$ of preceding tasks or *OR* nodes, respectively, of p is performed (line 2). Two sets C_t and $C_{t'}$ are filled with the preceding communication tasks $c \in C$ (line 3, 4): If t is a communication task, it is added to C_t. Otherwise, t is an *OR* node and the predecessors of t are communication tasks and added to C_t. Correspondingly, this procedure is carried out for t' and $C_{t'}$. If C_t is a subset of $C_{t'}$, the node t' is absorbed and, thus, the edge (t', p) becomes redundant and is removed (line 5-7).

Applying the previous and this simplification algorithm to the application graph in Fig. 5 (a) results in the graph in Fig. 5 (b). Here, Algorithm 3 removes the dependency of the process p_2 to the communication task created by sen_1: The process task p_2 requires the input data $\{v, a\}$. In case the communication task created by sen_2 fails to be sent to the process p_2, the process p_2 also fails since it requires the signal a which is unique in the application. On the other hand, the communication task created by sen_1 cannot increase the reliability of the process p_2 since the signal v is already contained in the necessary communication task created by sen_2.

4.2 Functional Approach

In the following, an approach based on Boolean functions is presented that is capable of extending an application by *OR* nodes in order to consider the inherent data-redundancy. This approach replaces the graph-based Algorithm 1 and is more flexible due to its capability to model more complex data-dependencies.

The functional approach requires a specific function $r_p : 2^C \rightarrow \{0, 1\}$ for each process $p \in P$. This function r_p returns 1 if a subset of received communications C with their contained signals allows a proper functionality of p, and 0 otherwise. In the following, the function r_p is represented as a Boolean function $r_p : \{0, 1\}^{|C|} \rightarrow \{0, 1\}$ where the input $\mathbf{C} = (\mathbf{c_1}, \ldots, \mathbf{c_{|C|}})$ is a binary vector that contains a 0 or 1, respectively, for each $c \in C$. A value of 1 indicates that the process p receives the corresponding communication c correctly, and 0 otherwise. The functional approach further requires that the Boolean function r_p is given in *conjunctive normal form* (CNF). A CNF is a conjunction of *clauses*, where a clause is a disjunction of *literals*. A literal is a variable (*positive*) or a negated variable (*negative*). It is further required that the literals of the CNF are always positive, inducing a monotonicity that states that additional input data is never detrimental for a process.

Given the Boolean functions r_p as CNF for each $p \in P$, a transformation from a common application graph to a redundancy-aware application is given in Algorithm 4.

Algorithm 4 Add redundancy (*OR* node) to application G_T with the corresponding Boolean functions r_p for each process $p \in P$.

Require: $G_T(T, E_T)$ with $T = P \cup C$
1: remove each $(c, p) \in E_T$ with $p \in P, c \in C$
2: **for** $p \in P$ **do**
3: r_p is the Boolean function for p as CNF
4: **for** clause a in r_p **do**
5: add new *OR* node o to T
6: add edge (o, p) to E_T
7: **for** literal l in clause a **do**
8: $c \in C$ is the positive variable of literal l
9: add edge (c, o) to E_T
10: **end for**
11: **end for**
12: **end for**

Corresponding to Algorithm 1, the edges between the communication tasks and process tasks are removed (line 1). The algorithm iterates over each process $p \in P$ (line 2). Each specific Boolean function r_p constructs the redundancy-aware application graph as follows: For each clause, a new *OR* node is added and connected to the corresponding process p (line 4-6). For each literal in the clause, the communication c that corresponds to the literal is connect to the *OR* node (line 7-9). If *unit clauses*, i.e., clauses with only a single literal, are directly connected to the corresponding communication task and the absorption law is applied to each Boolean function r_p, the simplification as provided by Algorithm 2 and 3 is performed implicitly.

The Boolean function r_p in CNF that corresponds to Algorithm 1 is

$$r_p(\mathbf{C}) = \bigwedge_{i \in input(p)} \bigvee_{\substack{c \in C \wedge \\ i \in data(c)}} \mathbf{c}. \tag{4}$$

Given the functional requirements for each process, an individual Boolean function can be defined that represents the input requirements. For instance, a process p that performs a *data-fusion* from at least two sensors can be stated as the following Boolean function:

$$r_p(\mathbf{C}) = \bigwedge_{i \in input(p)} \Big(\sum_{\substack{c \in C \wedge \\ i \in data(c)}} \mathbf{c} \geq 2 \Big). \tag{5}$$

This Boolean function has to be converted into a CNF by a conjunction of all the disjunctions over all pairs of $c, c' \in C$.

Application Duality: The application graph can be dual such that there exists (1) an application graph G_T^s for the minimal system requirements and (2) an application graph G_T^f for the system functionality. Here, the graph G_T^s is used for the system design and allows to model additional necessary redundancy by default. The graph G_T^f models the requirement of a correct functionality of the application and is used for the reliability analysis. Correspondingly to the function r_p, the functions r_p^s and r_p^f, respectively, are used for the construction of G_T^s and G_T^f, respectively. Here, it holds

$$\forall p \in P : r_p^s(\mathbf{C}) \leq r_p^f(\mathbf{C}) \tag{6}$$

indicating that if the system requirement r_p^s is fulfilled, also the functional requirement r_p^f has to be fulfilled.

For example, if a process $p \in P$ requires at least two redundant inputs to work properly and the designer wants a redundancy with one additional input, the requirement functions with four valid input communication tasks $c_0, c_1, c_2, c_3 \in C$ have to be defined as follows:

$$r_p^f(\mathbf{C}) = ((\mathbf{c_0} + \mathbf{c_1} + \mathbf{c_2} + \mathbf{c_3}) \geq 2) \tag{7}$$

$$r_p^s(\mathbf{C}) = ((\mathbf{c_0} + \mathbf{c_1} + \mathbf{c_2} + \mathbf{c_3}) \geq 3) \tag{8}$$

Converting these functions into a CNF is straightforward. In the following, for the sake of simplicity, it is assumed that $G_T = G_T^s = G_T^f$.

5. DESIGN SPACE EXPLORATION

Recently, an efficient approach to solve discrete optimization problems with stringent constraints has been proposed in [14]. This hybrid optimization approach synergizes *Integer Linear Programs* (ILPs) and *Evolutionary Algorithms* (EAs). Since the design space exploration given in Def. 1 also has to cope with stringent constraints arising from the requirements for a feasible implementation, this section presents a transformation of the introduced problem to a *constrained combinatorial problem*. In order to use reliability as an additional objective of the design space exploration, a reliability analysis for an implementation exploiting the data-redundancy as introduced is proposed.

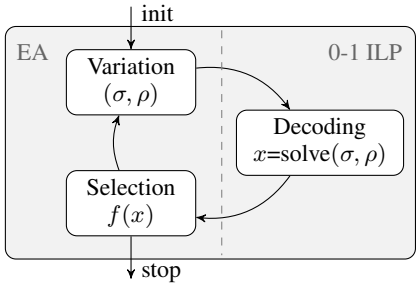


Figure 6: Hybrid optimization approach for constrained combinatorial problems.

5.1 Efficient Optimization of Constrained Combinatorial Problems

A *constrained combinatorial problem* is defined as:

DEFINITION 2 (CONSTRAINED COMBINATORIAL PROBLEM).

$$\begin{aligned} &\text{minimize } f(\mathbf{x}) \\ &\text{subject to:} \\ &A\mathbf{x} \leq b \textit{ with } \mathbf{x} \in \{0,1\}^n, A \in \mathbb{Z}^{m,n}, b \in \mathbb{Z}^m \end{aligned}$$

The objective function f allows multi-dimensional and non-linear calculations. $Ax \leq b$ represents a set of linear constraints which restrict the binary search space. Constraints that are not linearizable have to be handled as an additional objective.

The general constrained combinatorial problem cannot be solved by *Integer Linear Program* (ILP) solvers which are restricted to a single linear objective function. On the other hand, meta-heuristic optimization methods like *Evolutionary Algorithms* (EA) which rely on the iterative *variation* and *selection* do not perform well on optimization problems with many constraints and only a few feasible solutions.

Recently, an efficient optimization methodology for constrained combinatorial problems also known as the *SAT decoding* optimization approach [14] has been proposed. This hybrid optimization approach as illustrated in Fig. 6 is based on an EA and a backtracking based ILP with binary variables (0-1 ILP). The *variation* of the backtracking search strategy (σ, ρ) consisting of a binary vector for the decision phases and a real valued vector for the priorities as well as the *selection* based on the objective vector $f(x)$ is performed by the EA. The 0-1 ILP *decodes* the backtracking search strategy (σ, ρ) to a solution x that fulfills all linear constraints. This optimization approach iteratively improves found solutions such that with a higher number of iterations and more evaluated solutions, respectively, the quality of the results increases.

5.2 Model Encoding

In the following, the design space exploration problem formulated in Definition 1 is transformed to a constrained combinatorial problem as stated in Definition 2. A binary encoding and linear constraints have to be defined such that each solution $\mathbf{x}$ of this problem equals a feasible implementation x. The binary encoding consists of the following variables:

- $\mathbf{r}$ one variable for each resource $r \in R$ indicating whether this resource is in the allocation (1) or not (0).
- $\mathbf{p_r}$ one variable for each process task $p \in P$ and the available resources $r \in R_p$ indicating whether the process task is bound on the resource (1) or not (0).
- $\mathbf{c_r}$ one variable for each communication task $c \in C$ and the available resources $r \in R_c$ indicating whether the communication task is routed over the resource (1) or not (0).
- $\mathbf{c_{r,n}}$ one variable for each communication and resource pair indicating on which communication step $n \in \mathbb{N}$ (communication tasks are propagated in steps) a communication is routed over the resource.

The linear constraints are formulated as follows:

$\forall p \in P$:

$$\sum_{r \in R_p} \mathbf{p_r} = 1 \tag{9a}$$

$\forall c \in C$:

$$\sum_{r \in R_c} \mathbf{c_{r,0}} = 1 \tag{9b}$$

$\forall c \in C, p \in \{\tilde{p} | (\tilde{p}, c) \in E_T\}, r \in R_p \cap R_c$:

$$\mathbf{p_r} - \mathbf{c_{r,0}} = 0 \tag{9c}$$

$\forall p \in P, c \in \{\tilde{c} | (\tilde{c}, p) \in E_T\} \cap C, r \in R_p \cap R_c$:

$$\mathbf{c_r} - \mathbf{p_r} \geq 0 \tag{9d}$$

$\forall p \in P, o \in \{\tilde{o} | (\tilde{o}, p) \in E_T\} \cap O, r \in R_p \cap \{R_c | (c, o) \in E_T\}$:

$$\sum_{(c,o) \in E_T} \mathbf{c_r} - \mathbf{p_r} \geq 0 \tag{9e}$$

$\forall c \in C, r \in R_c$:

$$\mathbf{c_{r,1}} + \mathbf{c_{r,2}} + ... + \mathbf{c_{r,n}} \leq 1 \tag{9f}$$

$$\mathbf{c_{r,1}} + \mathbf{c_{r,2}} + ... + \mathbf{c_{r,n}} - \mathbf{c_r} \geq 0 \tag{9g}$$

$\forall c \in C, r \in R_c, i = \{1, ..., n\}$:

$$\mathbf{c_r} - \mathbf{c_{r,i}} \geq 0 \tag{9h}$$

$\forall c \in C, r \in R_c, i = \{1, .., n-1\}$:

$$-\mathbf{c_{r,i+1}} + \sum_{\tilde{r} \in R_c \wedge e=(\tilde{r},r) \in E_R} \mathbf{c_{\tilde{r},i}} \geq 0 \tag{9i}$$

$\forall p \in P, r \in R_p$:

$$\mathbf{r} - \mathbf{p_r} \geq 0 \tag{9j}$$

$\forall c \in C, r \in R_c$:

$$\mathbf{r} - \mathbf{c_r} \geq 0 \tag{9k}$$

$\forall r \in R$:

$$-\mathbf{r} + \sum_{c \in C \wedge r \in R_c} \mathbf{c_r} + \sum_{p \in P \wedge r \in R_p} \mathbf{p_r} \geq 0 \tag{9l}$$

Equation (9a) ensures that each process task is bound exactly once. The Equations (9b) and (9c) imply that each communication task has exactly one root that equals the used resource of the predecessor process task. Analogously, for each process task the predecessor communication tasks have to be routed on the corresponding resources as stated in Equation (9d). Equation (9e) states that if the predecessor is not a communication task but a an *OR* node, at least one predecessor communication task has to be routed on the corresponding resources. Equation (9f) ensures that a communication task can pass a resource at most once such that no loops occur. A communication task has to be existent in one communication step on a resource in order to be correctly routed on this resource as implied by the Equations (9g) and (9h). Equation (9i) states that a communication is only possible between adjacent resources. The Equations (9j) and (9k) imply that a process or communication task, respectively, is bound or routed on an allocated resource only. On the other hand, Equation (9l) states that a resource is only allocated if at least one process is bound or a communication is routed on this resource. Moreover, this representation allows to specify

additional linear or linearizable constraints like, e.g., on maximal computational load, maximal bus load, or maximal memory size for each resource.

Given a single solution $\mathbf{x}$ of this linear search problem, the corresponding implementation x is deduced by constructing the allocation from the $\mathbf{r}$ variables, the binding for each process task from the $\mathbf{p_r}$ variables, and the routing of the communication task from the $\mathbf{c_r}$ and $\mathbf{c_{r,n}}$ variables. Thus, the objective function in Definition 2 is calculated such that a solution $\mathbf{x}$ is converted into a feasible implementation x and the objective function from Definition 1 is applied directly.

5.3 Reliability as an Optimization Objective

In order to quantify the reliability in terms of an implementation, an automatic analysis is presented. In the work at hand, the basic failure model are resource defects. Other errors like soft errors within a process execution or incidental transmission errors on the buses are assumed to be treated at lower levels of abstraction with techniques discussed in the related work section like, e.g., process re-execution or error-correcting codes. Thus, reliability is assumed to be the capability of the system to work properly under given resource defects. Here, the well-known *Mean Time To Failure* (MTTF) [10] is used as an appropriate measure of reliability. The MTTF value is calculated based on the reliability specifications of each single resource. Due to resource sharing, it is not possible to simply consider combinations of series/parallel structures as proposed in several former approaches, cf. [3, 12]. Instead, the presented methodology is based on *Boolean functions*, using an evaluation technique of reliability-related measures as introduced in [8].

Structure Function Generation: In contrast to previous approaches, the presented methodology performs the analysis of each function. This allows the design space exploration to focus on specific, safety-relevant functions instead of a potentially expensive reliability increase for the whole application. A function is typically build of complex *sensor-controller-actuator chains*, including cycles as well as multiple input, output, and controller processes. It is sufficient to ensure a proper working of the corresponding *actuator processes* for each function.

A *structure function* for any actuator process $p^* \in P$ is generated from an implementation. This structure function is a Boolean function $\varphi_{p^*} : \{0,1\}^{|R|} \rightarrow \{0,1\}$ indicating if the process p^* is properly working (1) or not (0) under a given set of working resources encoded in the binary vector $\mathbf{R} = (\mathbf{r_1}, \ldots, \mathbf{r}_{|\mathbf{R}|})$. This Boolean function is defined as follows:

$$\varphi_{p^*}(\mathbf{R}) = \tag{10a}$$

$$\exists_{p \in P}\, \mathbf{p} : \tag{10b}$$

$$\mathbf{p}^* \tag{10c}$$

$$\wedge \bigwedge_{p \in P} (\mathbf{p} \rightarrow \mathbf{i}(\mathbf{p})) \tag{10d}$$

$$\wedge \bigwedge_{\substack{p \in P, (\tilde{p},c),(c,p) \in E_T \\ \wedge c \in C}} (\mathbf{p} \rightarrow (\tilde{\mathbf{p}} \wedge \varphi_c(i(p)))) \tag{10e}$$

$$\wedge \bigwedge_{\substack{p \in P, (o,p) \in E_T \\ \wedge o \in O}} (\mathbf{p} \rightarrow \bigvee_{(\tilde{p},c),(c,o) \in E_T} (\tilde{\mathbf{p}} \wedge \varphi_c(i(p))) \;) \tag{10f}$$

$$\varphi_c(r) = \begin{cases} \mathbf{r} \wedge \varphi_c(\tilde{r}), & \text{if } r \neq root(i(c)) \text{ with } (\tilde{r}, r) \in i(c) \\ \mathbf{r}, & \text{else} \end{cases} \tag{10g}$$

Equation (10a) defines the Boolean function: The Boolean function depends on a binary vector $\mathbf{R}$ defining whether each corresponding resource is working properly (1) or not (0). Internally, the Boolean function also depends on the process variables $\mathbf{P}$ indicating whether the process is active to ensure a proper functionality of p^* (1) or not (0). These process variables are eliminated with the existence quantification in Term (10b) that states that there must exist at least one activation of the process. Term (10c) states that the process p^* is active. Term (10d) ensures that a process can only be active if the target resource is working properly. Term (10e) handles the data-dependency of process tasks without redundancy. The term states that for each process p, each preceding process $\tilde{p}$ must be active and the all resources on the path between p and $\tilde{p}$ for the communication task must be working properly. The communication path of task c from a resources r to its root is determined by the recursive function in Equation (10g), resulting in a Boolean function that is a conjunction of all resources along this path. Correspondingly to Term (10e), the data-dependency of process tasks with redundancy is handled in Term (10f). Here, for each *OR* node, at least one preceding process task has to be active and the resources on the path between these tasks must be properly working.

Equation (10a) relies on P as a subset of all processes that are reachable by a backward-traversal from p^* only, since those processes that are not backward-reachable do not contribute to the reliability of p^*. Note that a straightforward enumeration and disjunction of all backward-paths from the process p^* that could be applied instead of Equation (10a) results in an explosion of the number of paths for a growing number of *OR* nodes in the application. Instead, the symbolic approach in the work at hand shows a reasonable scalability due to the efficient encoding. Moreover, the presented approach can even be applied on application graphs with cycles.

Extending this approach to analyze the reliability of multiple processes $P^* \subseteq P$ is done straightforward by the conjunction

$$\varphi_{P^*}(\mathbf{R}) = \bigwedge_{p^* \in P^*} \varphi_{p^*}(\mathbf{R}). \tag{11}$$

Evaluation: In order to derive the desired MTTF value, the *reliability function* $\mathcal{R}$ of the system needs to be evaluated. For a compact representation of the Boolean function φ, a *Binary Decision Diagram* (BDD) is used [2]. 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.

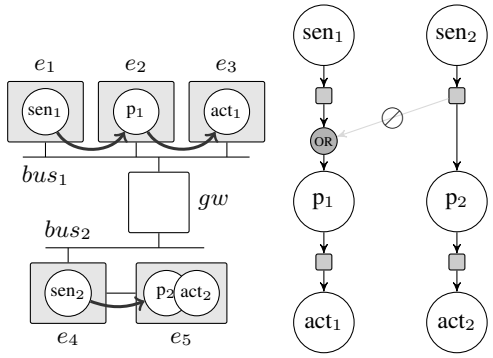
Using a specific SHANNON-decomposition as proposed in [16], the probability $\mathcal{P}$ of a proper working system at time t is determined. This decomposition scheme can be applied to the BDD directly and is defined as follows:

$$\mathcal{P}(t, \varphi) = \mathcal{R}_r(t) \cdot \mathcal{P}(t, \varphi_{|\mathbf{r}=1}) + (1 - \mathcal{R}_r(t)) \cdot \mathcal{P}(t, \varphi_{|\mathbf{r}=0}) \tag{12}$$

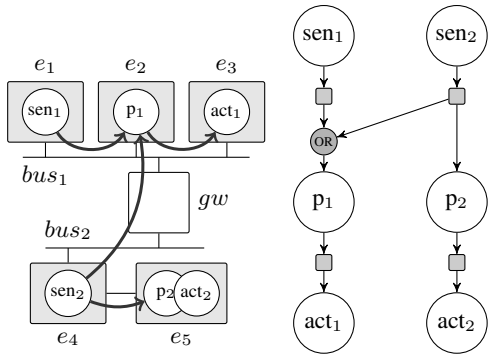
This function determines the probability of a structure function φ to evaluate to 1 at a given time t. The function $\mathcal{R}_r : \mathbb{R}^+ \rightarrow \mathbb{R}_{[0,1]}$ is the reliability function of a single system component r and returns the probability of this component to work properly at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

$$\varphi(\mathbf{x}) \geq \varphi(\widetilde{\mathbf{x}}) \text{ , if } \forall i \in \{0, \ldots, n\} : \mathbf{x_i} \geq \widetilde{\mathbf{x}}_\mathbf{i} \tag{13}$$

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 $\mathcal{R}_\varphi : \mathbb{R}^+ \rightarrow \mathbb{R}_{[0,1]}$



(a) Implementation that ignores existing data-redundancy.



(b) Implementation that exploits the data-redundancy.

Figure 7: Two possible implementations of the same redundancy-aware application from Fig. 5 (b) on the architecture shown in Fig. 3.

of the overall system is given by:

$$\mathcal{R}_\varphi(t) = \mathcal{P}(t, \varphi) \tag{14}$$

Based on the reliability function of the system, the desired MTTF is calculated as follows:

$$\text{MTTF}(\varphi) = \int_0^\infty \mathcal{R}_\varphi(t)dt \tag{15}$$

Other reliability and reliability-related measures like, e.g., the *Mission-Time* (MT), can be derived given the reliability function as well. Using BDDs for a representation of the structure function, this approach is efficient and capable of evaluating complex functions.

Exemplarily, two implementations of the same redundancy-aware application on the same target architecture are given in Fig. 7. Here, the implementation (a) ignores the data-redundancy of process p_1 while implementation (b) has an additional routing of the message from sen_2 to p_1. Applying the presented approach for the quantification of the reliability of the process act_1 results in the backward-reachable application graphs illustrated in Fig. 8 and the following structure functions:
Implementation (a):

$$\varphi^a_{act_1}(\mathbf{R}) = \mathbf{e_1} \wedge \mathbf{e_2} \wedge \mathbf{e_3} \wedge \mathbf{bus_1} \tag{16}$$

Implementation (b):

$$\varphi^b_{act_1}(\mathbf{R}) = (\mathbf{e_1} \vee (\mathbf{e_4} \wedge \mathbf{bus_2} \wedge \mathbf{gw})) \wedge \mathbf{e_2} \wedge \mathbf{e_3} \wedge \mathbf{bus_1} \tag{17}$$

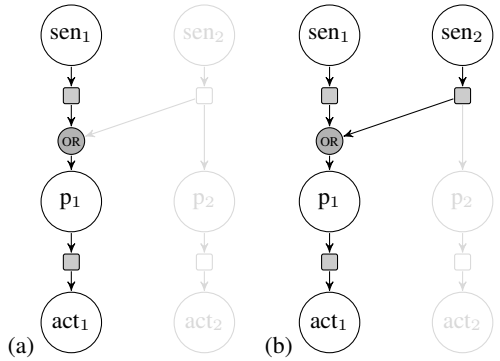


Figure 8: Backward reachability from the act_1 function for the implementation variants from Fig. 7.

Function	#processes	#messages	max. Delay [ms]
ACC	18	17	100
BW	8	7	50
C1	9	8	250
C2	10	9	150

Table 1: Detailed information about the application of the used case study.

For any reliability function of the single components it holds

$$\mathcal{R}_{\varphi^a_{act_1}}(t) < \mathcal{R}_{\varphi^b_{act_1}}(t), \tag{18}$$

i.e., the reliability of implementation (b) is higher than the reliability of implementation (a). Here, the increase of the reliability is solely reached by the multi-cast routing of the message generated by sen_1 from e_4, over bus_2, gw, and bus_1 to e_2 resulting in an additional communication overhead.

6. EXPERIMENTAL RESULTS

The design space exploration approach is implemented using the publicly-available tool OPT4J [15] that supports the optimization of constraint combinatorial problems as presented in [14]. All following experimental results were carried out on an Intel Core 2 Quad 2.66 GHz with 3GB RAM.

A case study modeling a typical automotive subnetwork is used to give evidence of the applicability of the proposed methodology. The network architecture consists of 15 ECUs, connected via two CAN buses, one FlexRay bus, and a central gateway. The 9 sensors, and 5 actuators are connected via LIN buses to the ECUs. The application consist of four functions, an *adaptive cruise control* (ACC), a *brake-by-wire* (BW), an *air conditioning function* (C1), and a *multimedia control* (C2). In Table 1, the number of processes and messages per functions is given as well as a maximal end-to-end delay for each function. Thus, the application contains 46 processes and 42 messages in total.

6.1 Routing Exploration

First, a given reference implementation derived by hand is improved by solely optimizing the routing of the messages as well as some system parameters, i.e., the priorities of the processes and messages. The reference implementation, listed in Table 2, has the monetary cost of 216.80€ (Euro) and an energy consumption of 11745 mA (Milliamperes) and respects all real-time constraints of the functions. The functions are distributed according to today's

	Costs [€]	Energy [mA]	ACC & BW Delay [ms]	ACC & BW MTTF [a]
reference	216.80	11,745	99.8 & 30.2	33.2
optimization:				
common	216.80	11,745	**46.1 & 21.6**	33.2
red.-aware	216.80	11,745	55.3 & 32.6	**42.2**

Table 2: Reference implementation, and two optimized implementations with and without using a redundancy-aware exploration. Here, only the routing and system parameters were optimized.

real-world networks with the ACC being implemented in the FlexRay subnetwork, BW and C1 each being implemented in one of the CAN subnetworks, and C2 being implemented over both CAN subnetworks.

Here, the routing and some parameters of the reference implementation are optimized, i.e., the mapping of the functions and the resource allocation including the topology of the network is not changed. Therefore, the monetary cost and the energy consumption are not affected. Instead, the reliability of the ACC and BW function measured in the *Mean-time-to-Failure* (MTTF) in years (a) as well as the end-to-end delay are optimized. The end-to-end delay is calculated by a non-linear timing model based on recurrence functions and a fix point search, cf. [9, 17]. The reliability or MTTF of the ACC and BW function, respectively, is maximized with the presented methodology. A higher utilization of an existing data-redundancy leads to an increased reliability but also to a higher end-to-end latency due to the additional communication and busload. Thus, these two objectives are conflicting.

The optimization is performed using 1350 objective evaluations. Using a common optimization approach that optimizes the parameters of the system such as the priorities of the processes and the tasks only improves the end-to-end latency. The result of this *common* optimization approach is given in Table 2. The runtime of this optimization was 6.7 seconds. On the other hand, the optimization that is able to utilize the data-redundancy improves the end-to-end latency as well as the reliability of the system. Compared to the common approach the runtime was slightly higher with 7.9 seconds. The result of the *redundancy-aware* optimization, as given in Table 2, shows an improvement of the MTTF of the ACC and BW function by more than 25%. The end-to-end delay of the ACC and BW function is improved compared to the reference solution but, as expected, still worse than the best solution without the usage of the data-redundancy. On the other hand, the end-to-end delay of the BW function is worse than the reference solution but still meeting the latency constraint. In automotive networks as well as in general, meeting the given deadlines is typically sufficient. Therefore, from the perspective of a designer, the implementation obtained by the *redundancy-aware* optimization is better than the implementation obtained by the *common* optimization.

6.2 Design Space Exploration

Finally, the presented methodology is applied in a complete design space exploration, i.e., the optimization of the allocation of the architecture, mapping of the processes, routing of the messages, and parameter determination. Here, the automotive network is optimized in terms of the monetary costs in Euro(€) and energy consumption in Milliamperes (mA), and the reliability of the ACC and BW function in years (a). The monetary costs are approximated by a linear function based on the cost per resource with additional cost like, e.g., wiring being neglected. The energy consumption is approximated by a non-linear energy model based on the average utilization of the ECUs.

Common automotive constraints such as the bus load (maximal load 40% for the CAN bus) and ECU (maximal utilization 95%) constraints have to be respected. Since these constraints are linear, they are directly integrated in the ILP formulation introduced in Section 5.2 and, thus, implicitly respected. The real-time constraints regarding the end-to-end delay from the sensors to the corresponding actuators as given in Table 1 are determined by a non-linear calculation. These non-linear constraints cannot be integrated in the ILP formulation and are, therefore, calculated separately for each implementation. If one end-to-end delay deadline is not satisfied, the implementation is discarded.

The design space exploration of the subnetwork was performed using a *common* optimization approach and the presented *redundancy-aware* optimization approach. The optimization is constrained to 5100 objective evaluations. Here, the runtime of the *common* optimization was 29.2 seconds and of the *redundancy-aware* optimization 45.4 seconds. This difference in runtime is explained by the larger search space of the *redundancy-aware* optimization as well as the larger BDDs that are necessary for the reliability analysis due to the data-redundancy. The results of the optimization are given in Figure 9.

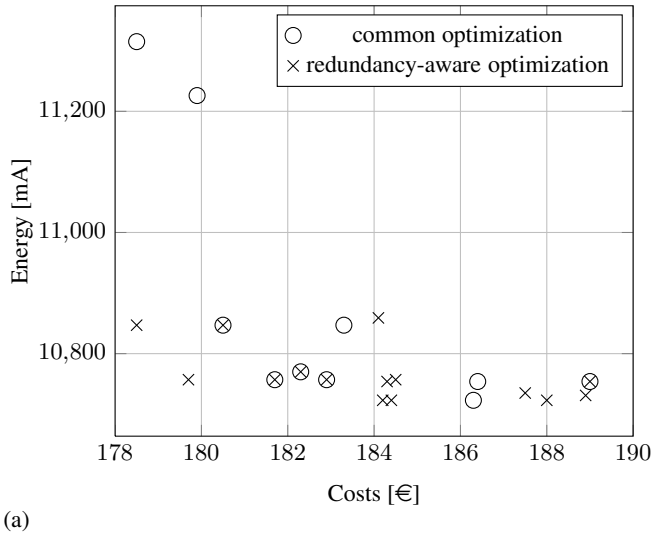
Compared to the reference implementation, the optimization improves the reference implementation in all three objectives, monetary cost, energy consumption, and reliability. The found implementations allow to decrease the monetary cost by 13% to 18% while decreasing the energy consumption by about 3.8% to 9.5% at the same time, see Fig. 9 (a). These results are obtained by binding the processes such that a higher utilization is achieved and some ECUs become redundant and can be removed from the implementation. At the same time, the priorities of the messages and processes are varied such that the real-time constraints are still respected. Regarding the reliability, the MTTF of the ACC and BW function can be increased by up to 40% compared to the reference implementation, see Fig. 9 (b). Here, the implementations that are obtained by the *redundancy-aware* optimization have a better reliability than all implementations obtained by the common optimization. At the same time, the monetary cost and energy consumption of the *redundancy-aware* optimization are not worse. Due to the larger search space and the higher number of feasible implementation, the *redundancy-aware* optimization even finds cost efficient implementations with a clearly better energy consumption.

The short exploration time and the scalability analysis of the used optimization methodology presented in [14] permit a projection that the presented approach is also applicable on markedly more complex systems.

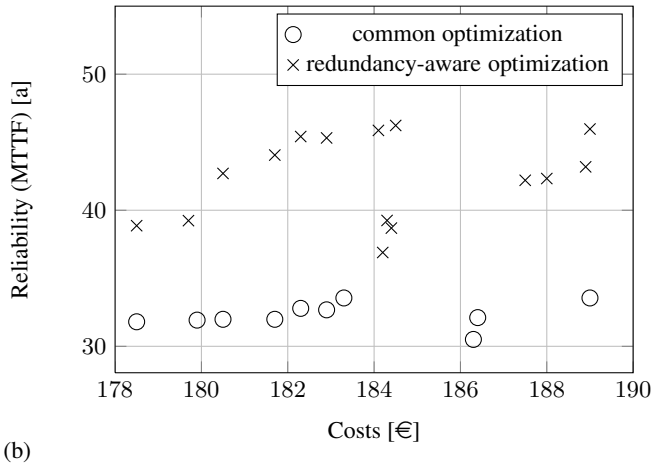
7. CONCLUSION

This paper presents a design flow for the design of reliable networked embedded systems. The paper covers the specification of such networks by algorithms that are capable of identifying data-redundancy of distributed functions. Here, a graph-based approach is presented that utilizes detailed information of the signal transmissions to add the data-redundancy to the application. The functional approach is a further extension that allows a generic definition for the required signals of a process by Boolean functions. Given a redundancy-aware application and a target architecture, the presented design space exploration exploits the data-redundancy to increase the reliability while optimizing the remaining objectives. For the efficient determination of the reliability of an implementation, a symbolic approach based on Boolean functions, represented by Binary Decision Diagrams (BDDs), is proposed.

The experimental results show a real-world case study from the automotive area consisting of four complex functions implemented on a multi-cluster system. First, an exploration of the routing and



(a)



(b)

Figure 9: Results of the design space exploration of the case study in two two-dimensional plots. The objectives are monetary costs, energy consumption, and reliability, while the end-to-end delay of the functions is handled as an additional non-linear constraint.

bus parameters shows that the reliability is significantly improved compared to the reference implementation, i.e., in the presented case study approximately 25%. Here, the additional function delay caused by the communication overhead does not affect the functionality of the application. Finally, a full design space exploration including resource allocation, process binding, communication routing, and parameter optimization is applied to the case study. The results show that the additional data-redundancy leads to more reliable implementations within an optimization, i.e., in the presented case study up to 40%. On the other hand, the redundancy-aware optimization is also capable of improving the remaining objectives like the energy consumption or monetary costs compared to a common design space exploration without the capability of handling data-redundancy.

8. REFERENCES

[1] M. Broy. Challenges in automotive software engineering. In *Proceedings of the 28th international conference on Software engineering*, pages 33–42, 2006.

[2] R. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys (CSUR)*, 24(3):293–318, 1992.

[3] D. Coit and A. Smith. Reliability optimization of series-parallel systems using a genetic algorithm. *IEEE Transactions on Reliability*, 45(2):254–260, 1996.

[4] P. Eles, V. Izosimov, P. Pop, and Z. Peng. Synthesis of fault-tolerant embedded systems. In *Proc. of DATE '08*, pages 1117–1122, 2008.

[5] B. Gaujal and N. Navet. Optimal replica allocation for TTP/C based systems. In *Proc. of FeT '03*, 2003.

[6] T. Gerke and D. Bollati. An Automated Model Based Design Flow for the Design of Robust FlexRay Networks. In *SAE International, 2008-01-1031*, 2008.

[7] M. Glaß, M. Lukasiewycz, F. Reimann, C. Haubelt, and J. Teich. Symbolic Reliability Analysis and Optimization of ECU Networks. In *Proc. of DATE '08*, pages 158–163, 2008.

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

[9] M. G. Harbour, M. H. Klein, and J. P. Lehoczky. Timing analysis for fixed-priority scheduling of hard real-time systems. *IEEE Trans. Softw. Eng.*, 20(1):13–28, 1994.

[10] IEC 60050-191 (1990). Int. Electrotechnical Vocabulary, Chapter 191 - Dependability and Quality of Service. (Amend. 1, 1999, Amend, 2, 2002).

[11] V. Izosimov, P. Pop, P. Eles, and Z. Peng. Scheduling of fault-tolerant embedded systems with soft and hard timing constraints. In *Proc. of DATE '08*, pages 915–920, 2008.

[12] 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.

[13] H. Kopetz, G. Bauer, and S. Poledna. Tolerating Arbitrary Node Failures in the Time-Triggered Architecture. SAE 2001 World Congress, Detroit, Mich. In *SAE International*, 2001.

[14] 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.

[15] Opt4J. Meta-heuristic Optimization Framework for Java. http://www.opt4j.org/.

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

[17] K. Tindell, A. Burns, and A. Wellings. Calculating controller area network (can) message response times. *Control Engineering Practice*, 3:1163–1169, 1995.

[18] 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.

[19] C. Wilwert, F. Clement, T. LORIA, and F. Nancy. Evaluating quality of service and behavioral reliability of steer-by-wire systems. In *Proceedings of ETFA'03*, volume 1, 2003.

[20] 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.

[21] E. Zanoni and P. Pavan. Improving the reliability and safety of automotive electronics. *IEEE Micro*, 13(1):30–48, 1993.

[22] 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.