Symbolic Voter Placement for Dependability-Aware System Synthesis

Felix Reimann, Michael Glaß, Martin Lukasiewycz, Joachim Keinert,
Christian Haubelt, and Jürgen Teich
Hardware-Software-Co-Design, Department of Computer Science
University of Erlangen-Nuremberg, Germany
{felix.reimann, glass, martin.lukasiewycz, keinert, haubelt, teich}@cs.fau.de

ABSTRACT

This paper presents a system synthesis approach for dependable embedded systems. The proposed approach significantly extends previous work by automatically inserting fault detection and fault toleration mechanisms into an implementation. The main contributions of this paper are 1) a dependability-aware system synthesis approach that automatically performs a redundant task binding and placement of voting structures to increase both, reliability and safety, respectively, 2) an efficient dependability analysis approach to evaluate lifetime reliability and safety, and 3) results from synthesizing a Motion-JPEG decoder for an FPGA platform using the proposed system synthesis approach. As a result, a set of high-quality solutions of the decoder with maximized reliability, safety, performance, and simultaneously minimized resource requirements is achieved.

Categories and Subject Descriptors

J.6 [**Computer-Aided Engineering**]: Computer-aided design (CAD); C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems

General Terms

Reliability, Security

1. INTRODUCTION

Embedded systems typically consist of many interconnected processing units, e.g., homogeneous or heterogeneous MPSoCs, automotive and avionics ECU networks, etc. Since they are operating under different radiation and varying temperature conditions, a high dependability of such system is needed. For example, high radiation can lead to a charging of single transistors resulting in so called *single event upsets* (SEU) and a fault of the circuit. Whether this fault affects the circuit's functionality permanently or temporarily, depends on the device as well as on the location of the fault, cf. [12]. If the SEU occurs in the configuration memory cell of an FPGA, the configuration and, hence, the functionality will be corrupted permanently until the next reconfiguration is performed. On the other hand, transient faults occur if the SEU only affects flip-flops.

CODES+ISSS'08, October 19–24, 2008, Atlanta, Georgia, USA.

In this paper, we propose a new dependability optimizing system synthesis approach that automatically performs a redundant binding of tasks to increase reliability and inserts fault detection mechanisms to improve the safety. The fault detection and, if possible, fault toleration is done by *voters* which are hardware resources or software tasks that aim to find a majority of k identical values delivered by n redundant task instances. Such voters are known as k-out-of-n-majority voting structures and typically implemented as so called *duplex voter* (2-out-of-2-majority) or the well known *Triple Modular Redundancy* (TMR, 2-out-of-3-majority).

The entire proposed system synthesis approach is transparent for the designer, i.e., the system specification given by the designer should be as abstract as possible, while implementation details are hidden and derived automatically by an optimization process. The optimization process itself is based on a symbolic multi-objective design space exploration. In order to guide the exploration, an efficient *dependability* analysis approach that evaluates *reliability* and *safety* is proposed. At this, the *Mean Time To Failure* (MTTF) to quantify lifetime reliability and the so called *Mean Time To Unsafe Failure* (MTTUF) known from literature to quantify the safety of a given implementation are used. As a result, the proposed approach significantly extends previous work by means of 1) automatically integrating fault detection and correction mechanisms and 2) automatically evaluating not only the reliability but also the safety of an implementation, thus, leading to high-quality solutions. The effectiveness of our proposed approach is shown by providing results from automatically exploring the design space of a Motion-JPEG decoder application and automatically synthesizing the solutions for an FPGA platform.

The remainder of the paper is organized as follows: After discussing related work in Sec. 2, Sec. 3 gives a motivating example and a formal problem definition. The design space exploration performing the redundant task binding and the voter placement is presented in Sec. 4. Section 5 discusses the dependability evaluation while results of the Motion-JPEG decoder case study are presented in Sec. 6.

2. RELATED WORK

For the analysis and design of dependable embedded systems, several approaches have been presented.

Many dependability-aware design methodologies focus on *permanent faults*. Approaches like [19, 21] aim to design dependable embedded systems by adding redundancy. However, other important objectives, e.g., monetary costs, are treated as constraints, leading in general to suboptimal solutions. In [11], reliability is treated as an objective during optimization in a system-level design using simple resource multiplication. This leads to an enormous deterioration of

other objectives, i.e., costs. An approach that overcomes these drawbacks by using resource reusage via multiple binding of tasks is presented in [6]. In [24], an approach also considering the influence of faults on subsequent fault rates due to the impact of run-time rebinding on temperature profiles is proposed.

On the other hand, several approaches target dependable systems design by focusing 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 [23]. In [10], fault-tolerant schedules with respect to hard and soft timing constraints are synthesized using task re-execution. The same authors propose another approach in [9], using rollback recovery and active replication. Finally, in [15] an approach trading energy for reliability is presented. Again, task re-execution is used to recover from transient faults.

For all these approaches, fault detection is mandatory, but not introduced automatically. Although [8] place TMR structures, the approach is restricted to simplex or fully-meshed TMR solutions, neglecting in safety in general and, thus, neglecting the positive aspect of other voting structures on the dependability of the system. Hence, our approach differs from the above mentioned approaches by considering an automatic integration of combined fault detection and fault toleration mechanisms using arbitrary voter structures into an embedded system. Note that the fault detection introduced by our proposed approach can be incorporated by the above mentioned approaches relying, e.g., on task re-execution, in order to further improve system reliability. Moreover, the tradeoffs between the achieved *reliability*, *safety* and other objectives typical for embedded systems, e.g., area and power consumption, throughput, etc., can be evaluated by our proposed approach automatically. To the best of our knowledge, no other publications on automatically integrating fault detection and fault toleration mechanisms with respect to many conflicting objectives into a dependable system design exist.

3. PROBLEM FORMULATION

In this paper, the problem of including *dependability-awareness* into system synthesis is targeted. This synthesis determines a set of possible high-quality implementations for a given system specification by a *design space exploration* approach. Afterwards, the designer chooses its preferred implementation that is synthesized automatically for the desired platform.

In dependable system design, structural redundancy is an important technique to improve the system characteristics with respect to reliability and safety. The usage of voting structures allows to detect and, if possible, tolerate transient and permanent errors in case a fail-silent behavior cannot be assumed. This is typically the case when dealing with SoCs and MPSoCs. Consider Fig. 1 denoting different voting structures and their characteristics. With the simplex structure having lowest cost, the duplex at increased cost and the TMR at highest cost, the diagram in Fig. 1 shows the need to consider reliability, safety, and cost together in the optimization of dependable embedded systems.

For an automated flow, the introduced redundancy and voting structures should be transparent for the designer but part of the optimization. Thus, the dependability-awareness is introduced at the highest design level where transparency for the designer is still given and, therefore, is embedded in the phase of design space exploration.

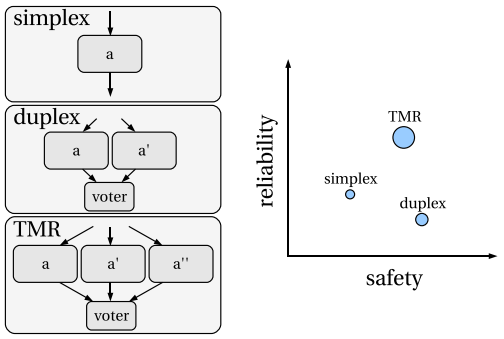


Figure 1: Voting structures and their corresponding characteristics regarding reliability and safety. The size of the points in the diagram corresponds to the cost of the voting structure.

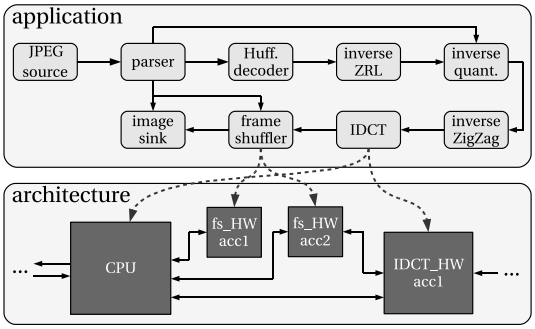


Figure 2: System specification of a Motion-JPEG decoder consisting of the application, an excerpt of the available architecture, and mapping edges from tasks to resources.

In embedded system design, different objectives like, e.g., monetary cost, area and power consumption, or dependability have to be considered. Thus, the carried out design space exploration is a multi-objective optimization problem, that aims to find a set of so called *Pareto-optimal implementations*. An implementation is Pareto-optimal, if it is better in at least one objective when compared to any other feasible implementation.

3.1 Specification

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

- The architecture is modeled by an architecture 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 resources, e.g., CPUs, hardware accelerators, memories, or buses. 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.
- The set of *mapping edges* E_{m} indicates that 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 of a system specification is shown in Fig. 2.

3.2 Implementation

An *implementation* is deduced from the specification and consists of three main parts:

- The *allocation* $\alpha \subseteq V_\mathsf{a}$ represents the resources that will actually be used in the implementation.
- The *binding* $\beta \subseteq E_\mathsf{m}$ determines on which allocated resource each task is executed. To enable redundancy, multiple mapping edges of each task can be activated, thus, creating several *instances* per task.
- The *voter placement* $V \subseteq \{v_1, \ldots, v_{|\alpha|}\}$ determines whether a voter is allocated.

With this knowledge, we define an implementation x as a triple (α, β, V), with α being a feasible allocation, β being a feasible binding and V being a feasible voter placement. An allocation is feasible if there exists a feasible binding for this allocation.

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.*

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

The problem of finding a feasible binding itself is known to be *NP*-complete [13]. A voter placement is called *feasible* if every task instance that needs a voting of its input data is able to carry out the voting on an available voter.

4. OPTIMIZATION

The used design space optimization approach performs the redundant task binding and voter placement. It is based on the combination of a *Pseudo-Boolean* (PB) solver [2] and a modern *Multi-Objective Evolutionary Algorithm* (MOEA) [25]. A PB solver is based on a backtracking strategy and efficiently solves *Integer Linear Programs* (ILPs) with an empty objective function and binary variables. The task of a PB solver is to find an $\mathbf{x} \in \{0,1\}^n$ that satisfies a set of linear constraints with binary variables. A single linear constraint is formulated as

$$a^T \mathbf{x} \circ b \tag{2}$$

with $a \in \mathbb{Z}^n$, $b \in \mathbb{Z}$ and $\circ \in \{<, \leq, =, \geq, >\}$. Commonly, the backtracking strategy in PB solvers is guided by two vectors $\rho \in \mathbb{R}^n$ and $\sigma \in \{0,1\}^n$ defining the priority and desired phase of a binary variable.

MOEAs are a *population*-based optimization approach taking advantage of the principles of biological evolution. The optimization is done iteratively in two alternating steps, the *variation* and *selection*. In the variation, new solutions, i.e., a new *generation*, is created from a set of existing solutions in the population. This is done by *crossover* and *mutation* operators. Correspondingly, the selection sorts out the worst solutions to ensure a convergence to the optimal solutions. The initial population is typically generated randomly.

Combining the PB solver with an MOEA allows the optimization of the system considering multiple, also conflicting and non-linear objectives. The main optimization flow is illustrated in Fig. 3. This approach is known as *SAT decoding* and has been presented in [13]. It is known to be superior to common methods that are based on ILPs or MOEAs only.

To utilize this optimization approach, it is necessary to encode the problem of finding a feasible allocation and binding, as well as the voter placement into an ILP by defining a set of linear constraints.

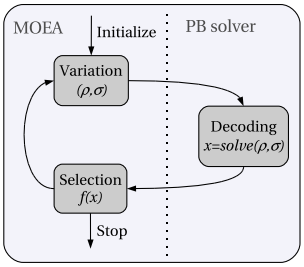


Figure 3: SAT decoding: A combined MOEA and PB solver optimization flow.

4.1 The ILP Model

To encode the constraints for a feasible implementation as an ILP, binary variables for each resource and mapping are introduced. The binary variable $\boldsymbol{r}$ of the corresponding resource r is 1, if the resource is allocated ($r \in \alpha$), and 0, otherwise. The binary variable $\boldsymbol{m}$ of the corresponding mapping m is 1, if the mapping is active ($m \in \beta$), and 0, otherwise. With the binary representation of the implementation, the constraints have to be formulated as follows[1]:

$\forall t \in V_\mathsf{t}$:

$$3 \geq \sum_{m=(t,r)\in E_\mathsf{m}} \boldsymbol{m} \quad \geq 1 \tag{3a}$$

$\forall m = (t, r) \in E_\mathsf{m}$:

$$\boldsymbol{r} - \boldsymbol{m} \quad \geq 0 \tag{3b}$$

$\forall m = (t, r) \in E_\mathsf{m} \wedge (t, \widetilde{t}) \in E_\mathsf{t}$:

$$-\boldsymbol{m} + \sum_{\substack{\widetilde{m}=(\widetilde{t},\widetilde{r})\in E_\mathsf{m}: \\ r=\widetilde{r}\vee(r,\widetilde{r})\in E_\mathsf{a}}} \widetilde{\boldsymbol{m}} \quad \geq 0 \tag{3c}$$

$\forall m = (t, r) \in E_\mathsf{m} \wedge (\widetilde{t}, t) \in E_\mathsf{t}$:

$$-\boldsymbol{m} + \sum_{\substack{\widetilde{m}=(\widetilde{t},\widetilde{r})\in E_\mathsf{m}: \\ r=\widetilde{r}\vee(\widetilde{r},r)\in E_\mathsf{a}}} \widetilde{\boldsymbol{m}} \quad \geq 0 \tag{3d}$$

Fulfilling these constraints leads to a feasible implementation according to Def. 1. Equation (3a) forces every task to be bound at least once and at most three times. At this step, the multiple binding of tasks, cf. [6], is performed. Equation (3b) ensures that whenever a task instance is bound to a resource, the resource must be allocated. Equation (3c) assures that for a task with a data-dependent successor, at least one instance of the successor is bound to the same or to an adjacent resource. Equation (3d) ensures the same for data-dependent predecessors.

A voter v_r is modeled using the binary variable $\boldsymbol{v}_r$ and is placed in the implementation as, e.g., dedicated voting unit, extended hardware accelerator or logical operation in processor such that it is usable by resource r. The voter is allocated ($v_r \in V$), if $\boldsymbol{v}_r = 1$. The $\boldsymbol{v}_m$ variables are used as indicators whether a task instance m needs a voting operation of its input data. In this case, a logical voting operation is carried out by the voter assigned to the resource the task is bound to. Of course, other assignments to globally shared voters or improved voter strategies (cf. [14]) etc. are encodable as well. The voter placement is performed by

[1]For a better representation, the generic constraints allowing a k-out-of-n redundancy were constraint to a practical relevant maximum of a 2-out-of-3 redundancy.

the following constraints:
$\forall m = (t, r) \in E_{\mathsf{m}} \wedge (\widetilde{t}, t) \in E_{\mathsf{t}}$:

$$\sum_{\substack{\widetilde{m}=(\widetilde{t},\widetilde{r})\in E_{\mathsf{m}}:\\ r=\widetilde{r}\vee(\widetilde{r},r)\in E_{\mathsf{a}}}} \widetilde{\boldsymbol{m}} - 3\cdot\boldsymbol{v}_m + 2\cdot\boldsymbol{m} \quad \leq 3 \tag{3e}$$

$$\sum_{\substack{\widetilde{m}=(\widetilde{t},\widetilde{r})\in E_{\mathsf{m}}:\\ r=\widetilde{r}\vee(\widetilde{r},r)\in E_{\mathsf{a}}}} \widetilde{\boldsymbol{m}} - 2\cdot\boldsymbol{v}_m \quad \geq 0 \tag{3f}$$

$$\boldsymbol{m} - \boldsymbol{v}_m \quad \geq 0 \tag{3g}$$

$\forall r \in E_{\mathsf{a}}$:

$$\sum_{m=(t,r)\in E_{\mathsf{m}}} \boldsymbol{v}_m - |E_{\mathsf{m}}|\cdot\boldsymbol{v}_r \quad \leq 0 \tag{3h}$$

Equation (3e) states that a voter is necessary if the task instance m is active and at least two predecessor instances exist. Equation (3f) and Eq. (3g) imply that a voter is not necessary if less than two predecessors exist or the instance is not active, respectively. Thus, these constraints ensure that a logical voter becomes necessary if and only if the task instance is used and has more than one predecessor instance. The placement of the voter itself is ensured by Eq. (3h) that allocates the voter for the resource if a logical voter is used by a task instance on this resource.

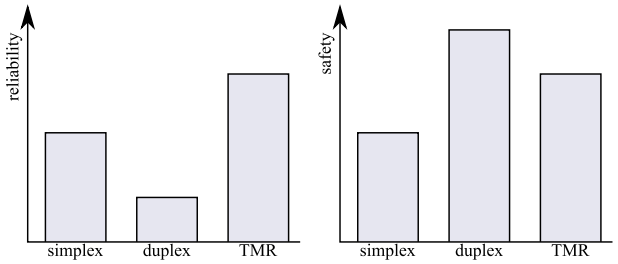
5. DEPENDABILITY EVALUATION

This section shows that a reasonable dependability optimization requires the evaluation of both, lifetime reliability and safety. An algorithm for an efficient calculation of the strongly-related measures *Mean Time To Failure* (MTTF) to quantify lifetime reliability and *Mean Time To Unsafe Failure* (MTTUF) [3] to quantify safety with respect to the used voting structures is presented.

5.1 Reliability and Safety

The need to evaluate both, reliability and safety of a dependable system, can be explained by reconsidering Fig. 1. In case reliability is used as the only measure, the probability that at least one resource fails is greater in a duplex structure than in a simplex structure. With a lower reliability and higher costs, the duplex structure is, thus, a sub-optimal implementation. Hence, the duplex structure forms a barrier for common heuristic optimization methods such that either the single bindings or the TMR structures are local minima. This is illustrated in Fig. 4(a). Starting the heuristic with single bindings only to keep the costs of the implementation low, hinders the optimization to find solutions with a high reliability. As a remedy, considering also the safety, the duplex structure is no longer sub-optimal and the mentioned drawbacks are cleared out, cf. Fig. 4(b). Note that, in case only safety is used, the TMR structure is a sub-optimal implementation. Thus, only a consideration of both aspects of a dependable system can handle the characteristics of different voting structures.

With safety, an additional objective is added to the design space exploration. However, adding another conflicting objective is known to deteriorate the optimization process, cf. [17]. On the other hand, additional objectives that harmonize with each other have little impact on the optimization complexity. Thus, we propose the usage of two closely related measures to quantify reliability and safety. As shown in Fig. 4, these two measures only diverge in some cases of arbitrary voting structures like, e.g. the duplex voting strategy. While the MTTF denotes the expected time of the system to operate correctly, the MTTUF denotes the expected



(a) reliability (b) safety

Figure 4: Analysis of different voting structures.

time that the system operates until a hazardous failure happens that the system cannot detect. That hazardous failure may result in a safety critical situation, thus, the MTTUF, although strongly related to the MTTF, is a safety measure.

As stated, reliability and safety only diverge in some cases of k-out-of-n-majority[2] voting structures. Given the same reliability $R_r(t)$ for all components and $R_v(t)$ for the voter, the reliability of an k-out-of-n-system calculates as

$$R_{k,n}(t) = R_v(t)\sum_{i=k}^{n}\binom{n}{i}R_r(t)^i(1-R_r(t))^{n-i}. \tag{4}$$

Under the same conditions, the safety calculates as

$$S_{k,n}(t) = R_v(t)\sum_{i=(n-k)+1}^{n}\binom{n}{i}R_r(t)^i(1-R_r(t))^{n-i}. \tag{5}$$

Thus, in case of single binding and TMR voting structures, reliability and safety are equal, while they differ in case of a duplex structure.

5.2 Calculating MTTF and MTTUF

Given the reliability of a system $R(t)$ and the safety $S(t)$, the MTTF calculates as $\int_0^\infty R(t)dt$ and the MTTUF as $\int_0^\infty S(t)dt$. To evaluate the reliability $R(t)$ and safety $S(t)$ of a found feasible implementation (α, β), the reliability analysis approach presented in [6] is used. Since a *fail-silent* [16] behavior is assumed in [6], an explicit introduction of the voting structures is needed. To evaluate a system including its implemented voting strategies, the structure function $\varphi : \{0,1\}^{|\alpha|+|V|} \to \{0,1\}$ with the Boolean vectors $\boldsymbol{\alpha} = (\boldsymbol{r}_1, \ldots, \boldsymbol{r}_{|\alpha|})$ and $\boldsymbol{V} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{|V|})$ has to be calculated and represented as a *Binary Decision Diagram* (BDD) [1]. φ evaluates to 1 if and only if the system is operating properly under the given set of properly working resources and voters. Otherwise, φ evaluates to 0. Hereby, for each allocated resource $r \in \alpha$ and voter $v \in V$, the corresponding binary variables $\boldsymbol{r} = 1$ and $\boldsymbol{v} = 1$ indicate a proper operation while $\boldsymbol{r} = 0$ and $\boldsymbol{v} = 0$ indicate a defect, respectively.

Using the techniques presented in [6, 18], φ can be used to derive the reliability and safety, respectively, by the following recursive definition with $R_r(t)$ being the specific reliability for a single component r:

$$\varphi(t) = R_r(t)\cdot\varphi|_{r=1}(t) + (1-R_r(t))\cdot\varphi|_{r=0}(t) \tag{6}$$

In the following, the formula to calculate φ to evaluate the system reliability $R(t) = \varphi(t)$ as well as an formula to construct φ to evaluate the safety of the system $S(t) = \varphi(t)$ is presented.

[2] $k \geq \lceil \frac{n}{2} \rceil$

5.2.1 Calculating φ for the Reliability Analysis

To calculate the structure function φ for the system reliability $R(t)$, the same binary variable encoding for tasks, mappings, resources, and voters as introduced in Sec. 4.1 are used. By definition, the structure function φ evaluates to 1, if and only if the system is working properly:

$$\varphi_{G_\beta}(\boldsymbol{\alpha}, \boldsymbol{V}) = \exists \boldsymbol{\beta} : \psi_{G_\beta}(\boldsymbol{\alpha}, \boldsymbol{V}, \boldsymbol{\beta}) \tag{7}$$

Equation (7) states that a system is working properly if there still exists a feasible binding of tasks, encoded by $\boldsymbol{\beta} = \{0,1\}^{|\beta|}$, under a given set of properly working or defect resources and voters.

$$\psi_{G_\beta}(\boldsymbol{\alpha}, \boldsymbol{V}, \boldsymbol{\beta}) = \bigwedge_{t \in V_\mathrm{t}} \left[\bigvee_{m=(t,r) \in V_\beta} \boldsymbol{m} \right] \wedge \tag{8a}$$

$$\bigwedge_{m=(t,r) \in V_\beta} \boldsymbol{m} \rightarrow \boldsymbol{r} \quad \wedge \tag{8b}$$

$$\bigwedge_{(\widetilde{t},t) \in E_\mathrm{t}} \bigwedge_{m=(t,r) \in V_\beta} \boldsymbol{m} \rightarrow C(m, \widetilde{t}) \tag{8c}$$

Equation (8a) and Eq. (8b) ensure the demands for a feasible system known from Sec. 4.1: There is at least one active mapping of each task and a mapping can only be active if its resource works properly with respect to $\boldsymbol{\alpha}$. A correct handling of data dependencies and voting structures is ensured by Eq. (8c). In particular, the function C used in Eq. (8c) deals with the different voting structures that may or may not be used:

$$C(m, \widetilde{t}) = \begin{cases} \widetilde{\boldsymbol{m}}, & \text{if } \neg \boldsymbol{v_m}; \\ \boldsymbol{v}_r \wedge \bigwedge\limits_{\substack{\forall \widetilde{m}=(\widetilde{t},\widetilde{r}), m'=(\widetilde{t},r') \in V_\beta: \\ \widetilde{m} \neq m' \\ (\widetilde{m},m),(m',m) \in E_\beta}} \widetilde{\boldsymbol{m}} \wedge \boldsymbol{m}', & \text{if } \boldsymbol{v_m}. \end{cases} \tag{9}$$

In case no logical voting is needed, $\boldsymbol{v_m} = 0$, a task instance needs an active task instance of each precedent task to work properly. In case a logical voting is needed, $\boldsymbol{v_m} = 1$, a majority or a consistent result has to be reached by properly working instances of the predecessor tasks and properly working voter. Note, as introduced in Sec. 4.1, the presented formulas are restricted to simplex, duplex and TMR structures, but can be extended to the general k-out-of-n case easily.

5.2.2 Calculating φ for the Safety Analysis

For the calculation of φ for the system safety $S(t)$, Eq. (7) and Eq. (8) can be reused. The distinction between φ for reliability and safety calculation takes part by modifying Eq. (9) to

$$C(m, \widetilde{t}) = \begin{cases} \widetilde{\boldsymbol{m}}, & \text{if } \neg \boldsymbol{v_m}; \\ \boldsymbol{v}_r \wedge \bigwedge\limits_{\substack{\forall \widetilde{m}=(\widetilde{t},\widetilde{r}), m'=(\widetilde{t},r') \in V_\beta: \\ \widetilde{m} \neq m' \\ (\widetilde{m},m),(m',m) \in E_\beta}} \widetilde{\boldsymbol{m}} \vee \boldsymbol{m}', & \text{if } \boldsymbol{v_m}. \end{cases} \tag{10}$$

In case a voting is needed, $\boldsymbol{v_m} = 1$ ensures that as long as one task instance is working correctly, the system will come to a majority of correct results or will recognize a dissent and, thus, can handle the failure to keep its safety property.

6. EXPERIMENTAL RESULTS

As a case-study, a system specification of a Motion-JPEG decoder, cf. Fig. 2, is used.

The overall design-flow can be found in [7] and is outlined as follows: The system specification is given as a SystemC behavioral model. Each SystemC module corresponds to a task in the task graph and can be transformed into software modules by code transformation or can be transformed into hardware accelerators (resources) by using a behavioral synthesis tool. In the latter case, Forte's Cynthesizer [4] is used and allows a quick extraction of important characteristics of the hardware accelerator like, e.g., throughput or required area. For the desired Xilinx FPGA target platform, resource needs in form of flip-flops, look-up tables, and block RAMs are estimated. Using the specified behavioral model and the estimation of the characteristics, a design space exploration model can be derived automatically. The design space exploration offers high quality solutions to the designer, including reliability and safety optimization using redundancy and voting techniques, automatically and fully transparent. The designer chosen solution can be compiled automatically using the behavioral model, the synthesized hardware accelerators, and basic components from a library using the Xilinx Embedded Development Kit (EDK) [22].

An important characteristic for this approach is the reliability of the used components. For FPGA platforms, various studies have been carried out to estimate failure rates, cf. [5, 20]. The synthesis of the SystemC models allows to estimate the number of slices for each used hardware resource. With a platform specific estimation, the number of configuration bits l_r for each hardware resource can be derived using the number of slices and BRAMs needed. In [20], experimental data has been presented that shows a relation between the size of the design and reliability. Given this, the following failure model can be used to derive the failure rate of each resource r:

$$\lambda_r = \frac{0.13 \cdot l_\mathrm{r}}{l_\mathrm{platform}} \tag{11}$$

The failure rate λ_r is calculated using a constant factor depending on the radiation environment, the number of configuration bits of the resource and the configuration bits of the entire platform l_platform. At this, the lowest radiation factor was used, cf. [20]. Thus, all important characteristics are available for the automatic design space exploration of the Motion-JPEG example.

The Motion-JPEG decoder specification consists of $8,000$ lines of SystemC code. For the exploration of the Motion-JPEG example, an architecture template was created with one MicroBlaze softcore processors, 56 FIFO communication links and 57 modules generated by behavioral synthesis. The complete specification includes 396 mapping edges for the actors resulting in about $2.7 \cdot 10^{57}$ possible implementation alternatives.

The experiment was carried out on an Intel Pentium 4 3.20 GHz machine with 2 GB RAM. For the set of high-quality implementations given in Fig. 5, about $7,800$ implementations were evaluated. The evaluation of one solution took 30 s, leading to an overall runtime of about 60 hours. Note that the SystemC simulation is very runtime extensive compared to the safety and reliability analysis, that takes about 24 ms per implementation.

A reference solution x_0 without multiple task binding and, thus, without voters was selected to quantify the amount of additional costs as well as reliability and safety increase using the proposed approach. By estimation x_0 uses $38,922$ LUTs, $14,239$ flip-flops and 99 BRAMs. Its MTTF as well as the MTTUF is 4.81 hours. The expected delay is 12.61 ms and the throughput is 81.07 fps. In the following table different solutions are compared to x_0.

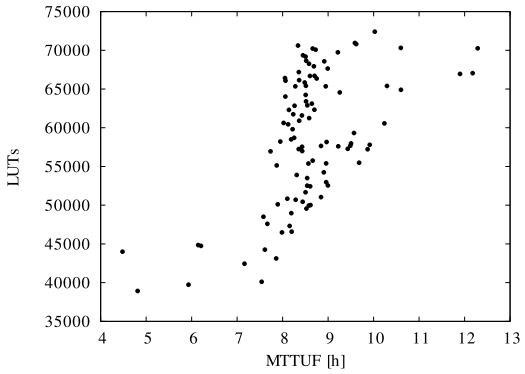


Figure 5: The high-quality solutions found by the exploration. This projection of the optimal solutions compares the safety of the implementations and their costs on the FPGA using the needed LUTs.

ID	LUTs	flip-flops	BRAM	MTTF	MTTUF
x_1	+80 %	+88 %	+3 %	+81 %	+81 %
x_2	+26 %	+51 %	+24 %	+69 %	+69 %
x_3	+28 %	+61 %	-3 %	+18 %	+83 %

The most reliable solution x_1 with an MTTF and MTTUF of 8.72 h comes with three TMR voting structures with one of them voting the largest hardware accelerator. Thus, the size of the implementation strongly increases. Since the voters are implemented in hardware, the delay increase of 3.1 μs (0.025 %) is negligible small. The solution x_2 represents a tradeoff between reliability and size. Instead of the redundant binding of the biggest module, two smaller modules are triplicated. Solution x_3 represents a solution with a high safety at relatively low costs. This is due to the usage of five duplex voters. Note that even by inserting five voters in the data path, the delay only increases by 58.9 μs (0.47 %).

In order to evaluate the quality of our estimations used in the design space exploration, several high-quality solutions were selected and automatically implemented on a Xilinx Virtex II FPGA platform. The actual FPGA resources in form of LUTs and flip-flops are about 5 % less than estimated. The needed BRAMs are even about 43 % less than estimated. This is due to post synthesis optimizations. The actual delay is about 16 % higher and, correspondingly, the throughput is about 20 % smaller. This is due to the used performance analysis that does not consider the inherent scheduling overhead. None the less, this values show that the estimations made via the optimization including multiple bound tasks and voting structures are acceptable.

7. CONCLUSION

In this paper, a system synthesis approach for dependable embedded systems was presented. By a symbolic placement of arbitrary voting structures, fault detection and fault toleration mechanisms are automatically integrated into the system implementations. A saving of additional costs is forced by a multiple binding of tasks that allows for resource reusage. To evaluate the introduced dependability and to allow an optimization together with other objectives like, e.g., monetary costs, power consumption, or latency, an efficient symbolic analysis approach was proposed that quantifies lifetime reliability and safety. The effectiveness and applicability of the proposed approach has been shown by automatically optimizing and by synthesizing a Motion-JPEG decoder.

8. REFERENCES

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

[2] D. Chai and A. Kuehlmann. A fast pseudo-Boolean constraint solver. In *Proc. of DAC '03*, pages 830–835, New York, NY, USA, 2003. ACM Press.

[3] T. DeLong, D. Smith, and B. Johnson. Dependability metrics to assess safety-critical systems. *IEEE Trans. on Reliability*, 54(3):498–505, 2005.

[4] Forte Design Systems. *Cynthesizer*. http://www.forteds.com/.

[5] E. Fuller, M. Caffrey, A. Salazar, C. Carmichael, and J. Fabula. Radiation characterization, and SEU mitigation, of the Virtex FPGA for space-based reconfigurable computing. In *Proc. of NSREC '00*, pages 129–134, 2000.

[6] M. Glaß, M. Lukasiewycz, T. Streichert, C. Haubelt, and J. Teich. Reliability-aware system synthesis. In *Proc. of DATE '07*, pages 409–414, 2007.

[7] C. Haubelt, J. Falk, J. Keinert, T. Schlichter, M. Streubühr, A. Deyhle, A. Hadert, and Jürgen Teich. A SystemC-based design methodology for digital signal processing systems. *EURASIP Journal on Embedded Systems*, 2007:1–22, Jan. 2007.

[8] A. Israr and S. A. Huss. Specification and design considerations for reliable embedded systems. In *Proc. of DATE '08*, pages 1111–1116, 2008.

[9] 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.

[10] 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.

[11] 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.

[12] F. L. Kastensmidt, G. Neuberger, L. Carro, and R. Reis. Designing and testing fault-tolerant techniques for SRAM-based FPGAs. In *Proc. of Computing Frontiers*, 2004.

[13] M. Lukasiewycz, M. Glaß, C. Haubelt, and J. Teich. Efficient symbolic multi-objective design space exploration. In *Proc. of ASP-DAC '08*, pages 691–696, 2008.

[14] A. Patooghy, S. G. Miremadi, A. Javadtalab, M. Fazeli, and N. Farazmand. A solution to single point of failure using voter replication and disagreement detection. In *Proc. of DASC '06*, pages 171–176, 2006.

[15] P. Pop, K. H. Poulsen, V. Izosimov, and P. Eles. Scheduling and voltage scaling for energy/reliability trade-offs in fault-tolerant time-triggered embedded systems. In *Proc. of CODES '07*, pages 233–238, 2007.

[16] D. Powell, G. Bonn, D. Seaton, P. Verissimo, and F. Waeselynck. The Delta-4 approach to dependability in open distributed computing systems. *Symposium on Fault-Tolerant Computing*, pages 56–61, 1995.

[17] R. Purshouse and P. Fleming. On the evolutionary optimization of many conflicting objectives. *IEEE Trans. on Evolutionary Computation*, 11(6):770–784, 2007.

[18] A. Rauzy. New algorithms for fault tree analysis. *Reliability Eng. and System Safety*, 40:202–211, 1993.

[19] 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.

[20] M. Wirthlin, E. Johnson, N. Rollins, M. Caffrey, and P. Graham. The reliability of FPGA circuit designs in the presence of radiation induced configuration upsets. *Proc. of FCCM '03*, pages 133–142, 2003.

[21] 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.

[22] Xilinx. *Embedded System Tools Reference Manual – Embedded Development Kit EDK 8.1ia*, Oct. 2005.

[23] 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.

[24] C. Zhu, Z. P. Gu, R. P. Dick, and L. Shang. Reliable multiprocessor system-on-chip synthesis. In *Proc. of CODES '07*, pages 239–244, 2007.

[25] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm for multiobjective optimization. In *Proc. of EUROGEN '02*, pages 19–26, 2002.