Automated Redesign with the General Redesign Engine

Alexander Feldman [*,***] **Gregory Provan** [**] **Johan de Kleer** [***]
Lukas Kuhn [***] **Arjan van Gemund** [*]

[*] *Delft University of Technology, Delft, The Netherlands*
(email: {a.b.feldman,a.j.c.vangemund}@tudelft.nl)
[**] *University College Cork, Cork, Ireland*
(email: g.provan@cs.ucc.ie)
[***] *Palo Alto Research Center, Palo Alto, CA, USA*
(email: {dekleer,lukas.kuhn}@parc.com)

Abstract: Given a system design (SD), a key task is to optimize this design to reduce the probability of catastrophic failures. We consider the task of redesigning an SD to minimize the probability of particular faults by introducing components selected from a component library. We have implemented a General Redesign Engine (GRE), which uses model-based reasoning techniques and Boolean functional synthesis from component libraries, to automate redesign for combinational circuits. For a significant subset of observations leading to catastrophic (forbidden) modes we demonstrate that GRE trades off redesign cost for increased fault tolerance, and shows a significant advantage compared to the Triple-Modular Redundancy (TMR) method. Our algorithm has a wide application in AI, including automated software and hardware design, error detection, reconfiguration and recovery, and modular robotics.

Keywords: model-based design; model-based redesign; fault-tolerance; modular redesign; functional synthesis; component libraries.

1. INTRODUCTION

Technological systems are designed to trade off a variety of characteristics; for example, a computer is designed to trade off cost, processing speed, size, reliability, etc. Engineering design has adopted the principles of modularity, regularity and hierarchy as keys to cost-effective and reliable design, both in theory and practice (Suh, 1990). A key part of this design process is the use of component libraries, which enable reuse of well-designed components/sub-systems. As the complexity of technological systems increases, module re-use increases, based on (a) symmetrical and regular structures and (b) developing standards for components and dimensions, since this regularity and component-based methodology translates into reduced design, fabrication and operation costs.

Given an SD, one key step is to verify that the SD meets its primary objectives. For example, for a control system one must verify that the system cannot enter a forbidden state through control actions. Verification can provide guarantees about controllability, observability, etc. However, verification of this type is a computationally intensive process, and may not cover the possibility that forbidden states may occur due to faults.

Achieving tolerance to faults is addressed during the design process. Fault/defect-tolerant design for hardware is based on adding redundancy to tolerate known faults or manufacturing defects, using reconfigurable blocks/components. Standard fault-tolerance techniques, which include methods such as dual-modular redundancy, triple-modular redundancy (TMR), triple interwoven redundant logic, and quadruple logic (Han et al., 2005), introduce pre-defined redundant circuit topologies. For example, a triple-modular redundancy (TMR) topology triplicates each gate and then collates the output signals using a voter (arbitration component), which computes the correct value based on output majority. A TMR circuit can be further triplicated to obtain nine copies of the original module and two layers of majority-voter gates; this design process can be repeated to achieve increasing levels of fault tolerance, resulting in designs called cascaded triple modular redundancy (CTMR) or recursive triple modular redundancy (RTMR).

Such traditional fault/defect-tolerant design approaches have several drawbacks. Among them are the increase in the number of components, system cost, system complexity, and potentially latency. Another drawback is that it optimizes overall system reliability rather than reliability to the most significant faults. In some cases redundancy for critical components is manually introduced. However, this approach usually covers just a subset of the single points of failure and not multiple-fault catastrophic states.

Rather than adopt standard redundancy-based methods for increasing fault tolerance, which employ fixed topology-replacement (e.g., replacing a single component with a TMR sub-system), we propose a model-based redesign method that uses an optimization algorithm to generate a cost-optimal design that is tolerant to a set A of catastrophic faults. Given an observation corresponding to a catastrophic fault α_i, this approach selects potential components to add to the circuit from a component library and searches over all possible circuit topologies to find a cost-minimal redesign that reduces the likelihood of α_i.

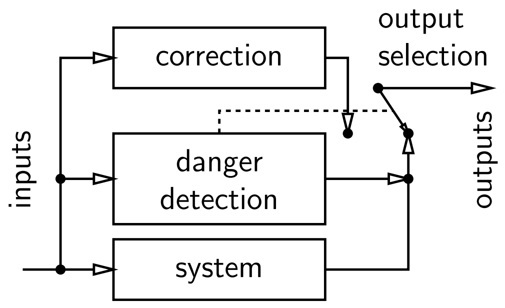


Fig. 1. Danger detection and correction

The approach we propose is conceptually shown in Fig. 1. Given a system, we automatically construct a danger detection system that, when presented with an observation corresponding to a catastrophic failure, switches an output selector from the original system to an (also automatically designed) correction subsystem. To automatically design the danger detection and correction subsystems, we compose a cost-minimal Boolean function from a component library.

Our contributions are as follows: (1) we formally define the problem of model-based redundancy redesign; (2) we propose an algorithm, called GRE, for automated redesign based on synthesis from component libraries; and (3) we empirically show that GRE creates cheaper redesigns for a standard benchmark circuit, as compared to the standard TMR fault-tolerance approach.

2. RELATED WORK

A wide range of different design approaches have appeared in the literature. Our approach is closest to the area of Reliability-Based Design Optimization (RBDO), a technique that attempts to optimize a design with respect to the reliability of the system (Du et al., 2008). The standard RBDO algorithm consists of a double-loop, in which the outer loop performs design optimization, with calls to the inner loop, which is a reliability oracle that computes the system reliability function for each proposed design. There are several variants of the double-loop RBDO approach, such as a single-loop method (Liang et al., 2008).

Our approach can be viewed as consisting of a double-loop algorithm, in which the outer loop performs design optimization over an observation O corresponding to a catastrophic fault, with calls to the inner loop, which is a diagnosis oracle that computes the diagnosis function, given O, for each proposed design. Our approach is different in that it computes diagnoses rather than a reliability function, and optimizes tolerance to a specific set of faults rather than optimizing overall system reliability (and hence tolerance to a distribution of faults).

The closest RBDO approach to our is described in (Monga and Zuo, 1998), where the notion of life-cycle warranty is addressed; here, the cost function, which is minimized via a Genetic Algorithm (GA), includes costs of manufacture, installation/setup, and repair (both during and beyond the warranty period). This approach is extended in (Liu et al., 2007), who describe the objective function for diagnostics-optimal design of a product under warranty from a manufacturer's point of view, focusing on the robustness of the diagnostics-oriented design to the key model parameters and decision variables. In comparison to these approaches, we use a component library to add new components, rather than assuming a fixed system model and using GA to optimize the model structure.

The literature contains a wide variety of methods for component design, but few methods for redesigning a system with a given structure. Allison et al. (Allison et al., 2007) analyze methods for design based on optimal system partitioning (to reduce computational complexity of the design process). There are methods for finding the backbone or funnel variables which are critical for every possible design (Menzies and Singh, 2003). Other design or redesign approaches employ hierarchical (López-Arévalo et al., 2007) or qualitative abstractions (Ollinger and Stahovich, 2004) of the system being designed. In our approach we could adopt any of these methods, but our methodology is fundamentally different from all of these techniques in that we aim to modify an existing design to improve its tolerance to catastrophic faults, and not create a design from scratch without any aim for improving fault tolerance.

Several authors have proposed design methodologies based on system partitioning, e.g., (Allison et al., 2007; Chen et al., 2007). For example, (Chen et al., 2007) proposes a redesign methodology based on pattern-based decomposition to rapidly locate and isolate the portions of the design model that must be recomputed to satisfy redesign requirements. This approach transforms the system equations into an incidence matrix mapping equations by variable, in which variables participating in equation i as assigned a 1 in row i, and a 0 otherwise. The matrix is then transformed into a block-angular matrix in which the blocks represent the subproblems formed by decomposition, and the interaction part represents the coordination imposed on the subproblems. Because our approach assumes components from a component library, as well as the possibility of hierarchically organizing the components, such a problem decomposition is already incorporated in our methodology.

Design of circuits is quite different than circuit optimization (McCluskey, 1956). Circuit optimization aims to optimize the design of a circuit with respect to the function f that the circuit computes. Our design problem is much more sophisticated, in that we aim to take an existing (possibly optimized) circuit and redesign it to optimize fault tolerance given a cost function. As a consequence, the algorithms used in the two problems are very different.

This work also bears some relationship to redesign to compensate for defects which occur during the manufacturing process (Tahoori, 2005), or for nano-structures (Simsir et al., 2008).[1] In defect-based redesign, defective components, e.g., on a manufactured chip, are isolated using test and diagnosis methods, and the resulting data stored on a defect map, which identifies the usability of the (programmable) elements of the manufactured chip.

[1] A thorough survey of this approach, as applied to FPGAs, is contained in (Cheatham et al., 2006).

Defect tolerance is achieved by reconfiguring key processes to avoid defective resources, resulting in modifications to the logic and architecture design. In our approach, we assume that the system is defect-free, and we aim to increase the tolerance of the design to *possible* catastrophic faults.

Our redesign approach also bears some resemblance to diagnosis-based reconfiguration (Chen and Provan, 2001; Stumptner and Wotawa, 1998). In contrast to this work, which aims to restore system functionality given a diagnosed fault, our approach aims to increase the system's tolerance to *anticipated* catastrophic faults, and not to faults that have already occurred.

3. PRELIMINARIES

We first give a brief overview of the process of creating models from a component library within a Model-Based Reasoning framework. We assume that we can create a system-level model by composing components from a component library (Gössler and Sifakis, 2005; Keppens and Shen, 2001).

We call a well-defined model fragment a *component*. We assume that each component can operate in a set of behavior-modes, which we formalize using an assumable M. Further, there are two classes of components: primitive and composite. A primitive component is the simplest model fragment to be defined.

Definition 1. (Primitive Component). A primitive component C, $\langle$SD, M, c, p, IN, OUT$\rangle$ is specified using a set of propositional **Wff** SD over a set of variables V, assumable $M \in V$, a cost function $c : M \mapsto (0; \infty)$, a mode probability function $p : M \mapsto [0; 1]$, and input/output variables, IN, OUT $\in V$.

3.1 A Running Example

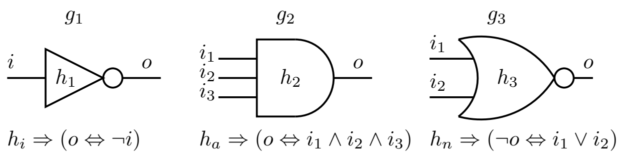
We illustrate the notions in this paper with a model of a Boolean circuit. Fig 2 shows three primitive components from a circuit component library; for example, component g_1 an inverter, has mode-variable h_1, input i, output o, and system description $h_i \Rightarrow (o \Leftrightarrow \neg i)$. The cost function c denotes the cost of the component (e.g., manufacturing cost, power consumption, chip area, etc.), and the mode probability function p denotes the probability distribution function over the component modes.

A composite component consists of a collection of primitive components which are merged according to a set χ of composition rules (Gössler and Sifakis, 2005). In this paper we assume the standard composition rules of discrete circuits; specifying the semantics of composition is beyond the scope of this paper, and we refer the reader to (Gössler and Sifakis, 2005; Keppens and Shen, 2001) for details.

A set of (primitive/composite) components defines a component library.

Definition 2. (Component Library). A component library $\mathcal{L}$ is defined as a set of (primitive/composite) components.

The example component library $\mathcal{L}$, shown in Fig 2, contains fault models for three gates (an inverter, a 3-input AND-gate, and a NOR-gate). The assumable variable is h, and the cost and fault probability functions are the same for all components, $c = 1$, and $p(h = \textbf{True}) = 0.95$.



g_1
g_2
g_3
i
h_1
o
i_1
i_2
i_3
h_2
o
i_1
i_2
h_3
o
$h_i \Rightarrow (o \Leftrightarrow \neg i)$
$h_a \Rightarrow (o \Leftrightarrow i_1 \wedge i_2 \wedge i_3)$
$h_n \Rightarrow (\neg o \Leftrightarrow i_1 \vee i_2)$

Fig. 2. An example component library

3.2 Models and Systems

Given a library of components, we can build system models, which we specify as follows.

Definition 3. (Model-Based System). Given a component library $\mathcal{L}$, a diagnostic system DS, $\langle$SD, COMPS, OBS, c, $p\rangle$, contains SD, COMPS, c, and p, constructed from $\mathcal{L}$ according to the rules χ, and a set of observable variables OBS corresponding to IN $\cup$ OUT, the inputs and outputs for SD.

In order to construct a model we have to (1) choose a multiset of components from the universe (component library) $\mathcal{L}$, (2) to create a system topology by interconnecting the selected components, (3) to disambiguate the variable names in the model and (4) to compose the failure probability and cost functions p and c. The 2-to-4 line demultiplexer shown in Fig. 3 can be built from the Fig. 2 components.

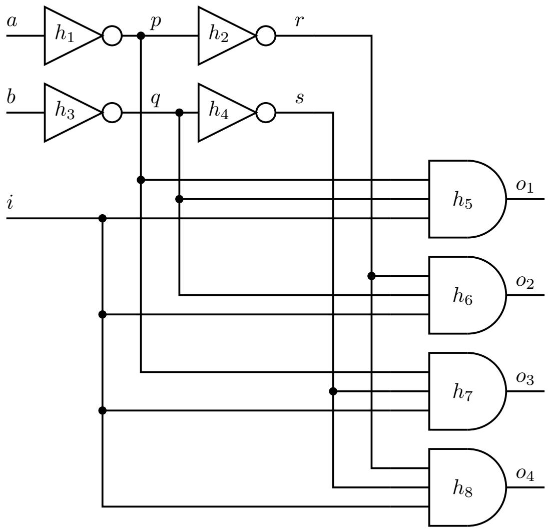


Fig. 3. A demultiplexer circuit

After mapping the variables in $\mathcal{L}$ we get the following propositional system description:

$$\text{SD} = \begin{cases} h_1 \Rightarrow (a \Leftrightarrow \neg p) \\ h_2 \Rightarrow (p \Leftrightarrow \neg r) \\ h_3 \Rightarrow (b \Leftrightarrow \neg q) \\ h_4 \Rightarrow (q \Leftrightarrow \neg s) \\ h_5 \Rightarrow (o_1 \Leftrightarrow i \wedge p \wedge q) \\ h_6 \Rightarrow (o_2 \Leftrightarrow i \wedge r \wedge q) \\ h_7 \Rightarrow (o_3 \Leftrightarrow i \wedge p \wedge s) \\ h_8 \Rightarrow (o_4 \Leftrightarrow i \wedge r \wedge s) \end{cases}$$

The assumables are COMPS $= \{h_1, h_2, \ldots, h_8\}$, the observables are OBS $= \{a, b, i, o_1, o_2, o_3, o_4\}$.

3.3 Diagnostic Modeling

Model-based systems can be used for a variety of purposes, such as simulation and diagnosis. If we specify a model with modes denoting failure-states, then we call this a *diagnosis model.* Further, given an observation, we can compute a *diagnosis* for such a model in terms of an assignment to the system's mode-variables.

Definition 4. (Diagnosis). Given an DS, an observation α over some variables in OBS, and an assignment ω to all variables in COMPS, ω is a diagnosis iff $\mathrm{SD} \wedge \alpha \wedge \omega \not\models \perp$.

Continuing our running example, consider an observation vector $\alpha_1 = i \wedge \neg a \wedge \neg b \wedge o_4$. Throughout this paper we specify a diagnosis ω as the set of its negative literals. There are a total of 256 possible assignments to all variables in COMPS. Example diagnoses are $\omega_1 = \{\neg h_8\}$ and $\omega_2 = \{\neg h_1, \neg h_4\}$.

Definition 5. (Probability of a Diagnosis). The probability of a diagnosis ω, $\Pr(\omega)$, is defined as:

$$\Pr(\omega) = \prod_{x \notin \omega} g(x) \prod_{x \in \omega} 1 - g(x)$$

Note that $\Pr(\omega)$ gives a prior (non-normalized) probability of ω, i.e., the health pdf is conditioned on the model topology, but not on the observation. Computing *a posteriori* probabilities require computating all diagnoses and applying Bayes rule (de Kleer and Williams, 1987), but our task needs *a priori* probabilities only. According to Def. 5, $\Pr(\omega_1) \approx 0.035$ and $\Pr(\omega_2) \approx 0.0018$.

Definition 6. (Probability-Minimal Diagnosis). A diagnosis ω^* is defined as probability-minimal if no diagnosis $\tilde{\omega}^*$ exists such that $\Pr(\tilde{\omega}^*) < \Pr(\omega^*)$.

Given an DS and an observation α, the probability of the probability-minimal diagnoses is denoted as $\mathrm{PrMin}(\mathrm{DS}, \alpha)$. Continuing our example, ω_1 is probability-minimal, while ω_2 is not, as $\Pr(\omega_2) < \Pr(\omega_1)$ and $\mathrm{PrMin}(\mathrm{DS}, \alpha_1) \approx 0.035$.

Other authors use different minimality criteria such as subset-minimality diagnoses, minimal-cardinality diagnoses, kernel diagnoses (in a slightly different diagnostic framework), etc. (de Kleer et al., 1992).

4. MODEL-BASED REDESIGN

This section describes the general redesign problem, and our redesign algorithm, which makes use of the model-based relationship between system mode assignment and observable assignment. In a model-based system, there is a functional relationship $\phi : h \rightarrow \mathrm{OBS}$, such that any $h^* \in h$ induces a unique $\mathrm{OBS}^* \in \mathrm{OBS}$. [2]

Consider DS and DS$'$ implemented with components from the same library $\mathcal{L}$ and having the same set of observable variables OBS. Consider also the assignment ν to all variables in COMPS such that SD is functioning correctly (in our example ν would be $h_1 \wedge h_2 \wedge \cdots \wedge h_8$). In general, a system may have a set $\mathcal{H}_{nom}$ of nominal states. We denote the Boolean function implemented by our system description and conditioned on $\nu \in \mathcal{H}_{nom}$ as SD_ν.

[2] In general, some $\mathrm{OBS}^* \in \mathrm{OBS}$ corresponds to multiple possible $h^* \in h$.

Definition 7. (Nominal Equivalence). Given DS and DS$'$ we will say that nominally functioning DS is equivalent to nominally functioning DS$'$ (denoted as $\mathrm{DS} \equiv_{nom} \mathrm{DS}'$) iff $\mathrm{SD}_\nu \equiv \mathrm{SD}'_\nu$.

4.1 Problem Statement

We now formally define our redesign problem. Our objective is to redesign DS such that we make the system more robust to a set of (catastrophic) faults. Given the relationship between mode assignments and observable assignments, we can define our redesign objective as making a system more robust to a set of (catastrophic) observations which correspond to faults. We call this set of dangerous observations A.

Problem 1. (Cost-Optimal Redundancy Redesign). Given a component library $\mathcal{L}$, DS $= \langle \mathrm{SD}, \mathrm{COMPS}, \mathrm{OBS}, c, p\rangle$, and a set of (catastrophic) observations $A = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$, compute a redesign $\mathcal{R} = \langle \mathrm{SD}^+, \mathrm{COMPS}^+\rangle$, such that for DS$' = \langle \mathrm{SD} \cup \mathrm{SD}', \mathrm{COMPS} \cup \mathrm{COMPS}', \mathrm{OBS}, c, p\rangle$ it holds that:

1. $\mathrm{DS} \equiv_{nom} \mathrm{DS}'$,
2. $\forall \alpha : \alpha \in A, \mathrm{PrMin}(\mathrm{DS}', \alpha) < \mathrm{PrMin}(\mathrm{DS}, \alpha)$,
3. $\sum_{x \in \mathrm{COMPS}^+} c(x)$ is minimized.

4.2 Model-Based TMR

Consider SD and a set of assumable variables $H = \{h_1, h_2, h_3, h_4, h_8\}$ (e.g., these are components identified as critical). A TMR redesign adds two layers of redundancy and a voting mechanism which chooses a consensus of its inputs. Applying TMR to H in DS gives us the TMR diagnostic system DS$''$, the system description of which is shown in Fig. 4 and Fig. 5 (the outputs o_4', o_4'', and o_4''' of the circuit shown in Fig. 4 are connected to the identically named inputs of the circuit shown in Fig. 5).

Let us denote the set of assumable variables in the voting mechanism as COMPS_v (for the TMR design in Fig. 4 and Fig. 5 $\mathrm{COMPS}_v = \{h_9, h_{10}, h_{11}, h_{12}\}$). Consider the following fault probability function g for the TMR circuit in Fig. 4 and Fig. 5:

$$g(x) = \begin{cases} \epsilon_1, \text{ for } x \in \mathrm{COMPS}_v \\ \epsilon_2, \text{ for } x \in \mathrm{COMPS} \setminus \mathrm{COMPS}_v \end{cases}$$

Let $A' = \{\alpha_i\}$ be the set of all observations of DS such that all diagnoses of SD and α_i contain variables in H only. Next, suppose that $\epsilon_1 < \epsilon_2$. It can be seen that for any $\alpha_i \in A'$, $\mathrm{PrMin}(\mathrm{DS}'', \alpha_i) < \mathrm{PrMin}(\mathrm{DS}, \alpha_i)$ and $\mathrm{DS} \equiv_{nom} \mathrm{DS}''$. In the experimentation section that follows, we will see that if we consider a subset of A', it is often possible to reduce the number of components and the cost of the redesigned circuit.

4.3 Danger Detection and Correction

Consider the running example from Fig. 3 and a set A containing two (dangerous) observations $A = \{\alpha_1, \alpha_2\}$, $\alpha_1 = i \wedge \neg a \wedge \neg b \wedge o_4$, $\alpha_2 = i \wedge a \wedge b \wedge \neg o_4$. Figure 6 shows

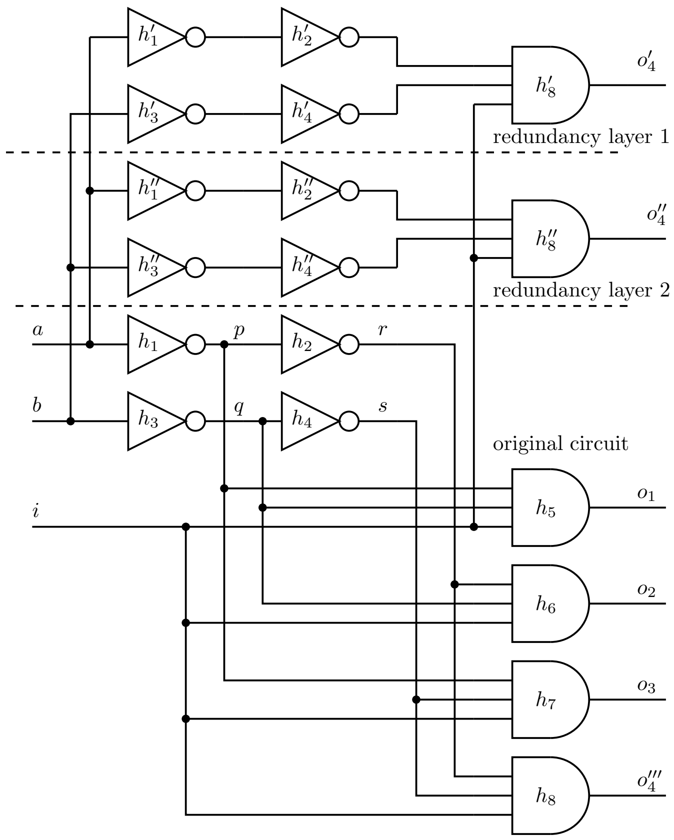


Fig. 4. A triple-redundant demultiplexer circuit

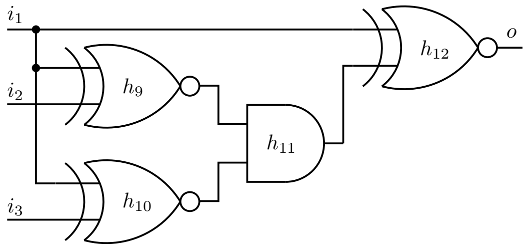


Fig. 5. Majority voting circuit

a circuit D, which produces a true output d iff any of the observations in A is detected. Figure 7 shows a two-input multiplexer S, which, depending on the control signal d, routes the original signal o_4, or the corrected o_4^*, to the output $\tilde{o_4}$.

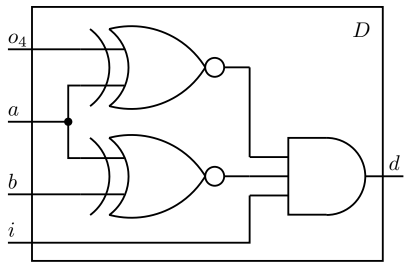


Fig. 6. Example of a danger detection circuit

To complete the running example we construct a correction circuit C which, given an input of interest $i \wedge a \wedge b$ or $i \wedge \neg a \wedge \neg b$, produces a correct output (o_4 and $\neg o_4$, respectively). The resulting circuit is shown in Fig. 8. Clearly the cost of this circuit (17 gates) is smaller than the one of the TMR circuit (20 gates).

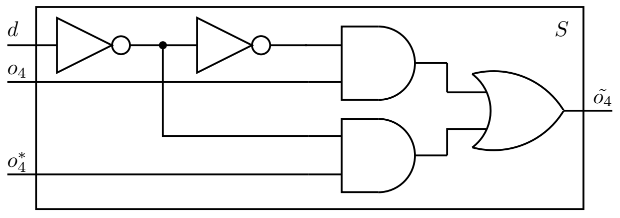


Fig. 7. Example of an output selection circuit

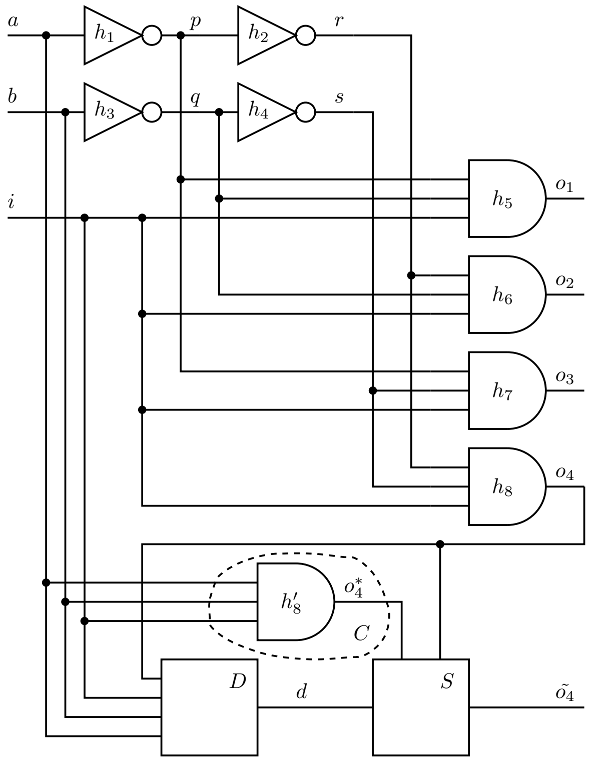


Fig. 8. A fault-tolerant demultiplexer circuit

In this paper we assume that the probability of a failure in the danger detection and output selection subcircuits is less than or equal to the probability of any failure in the original system description.

5. AUTOMATED FAULT-TOLERANT REDESIGN

This section presents an algorithm that solves Problem 1 (finding a cost-optimal circuit which is logically equivalent to the original when healthy, but which decrease the a priori probability of some predefined set of observations). Our algorithm consists of (1) (nearly) cost-minimal synthesis of models from component libraries and (2) mapping (catastrophic) observations into component sets (subcircuits) which are used for the synthesis of detection and correction subcircuits.

5.1 Synthesis from Component Libraries

Algorithm 1 composes a cost-minimal function from a component library $\mathcal{L}$, the resulting function evaluating to true for a set of measurement-points A. Furthermore, Alg. 1 is provided with a target function SD^* which is true in A but not necessarily cost-minimal.

Lustig et al. (Lustig and Vardi, 2009) prove that, when formulated in terms of Linear Temporal Logic, the problem of system synthesis from component libraries is undecidable. Algorithm 1 is guaranteed to terminate because it considers problems with (non-zero) cost smaller than r, where r is the cost of a target circuit SD^* that implements the function Alg. 1 attempts to synthesize.

Algorithm 1 searches the space of all possible Boolean circuits of cost smaller than r and checks the equivalence of each candidate to SD^* by making a call to the CheckEquivalence function in line 6. Although the general problem of equivalence checking of two Boolean functions is known to be NP-hard, our implementation solves a simpler problem by comparing the candidate function in A only. In particular, we make $|A|$ queries to a Logic-Based Truth Maintenance System (Forbus and de Kleer, 1993) which is of polynomial time complexity at the expense of incompleteness.

Algorithm 1 Boolean functional synthesis

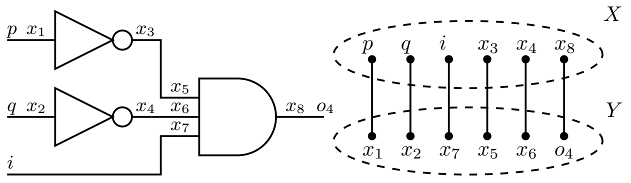
```
 1: function Synthesize($\mathcal{L}$, $\mathrm{SD}^*$, COMPS, $A$)
        inputs: $\mathcal{L}$, component library
                $\mathrm{SD}^*$, target system description
                COMPS, target component variables
                $A$, set of terms
        returns: system description
        locals variables: $\mathrm{SD}^*$, system description
                          $r$, real, cost bound
                          $m$, variable multiset, components
                          $g$, bigraph, connections
 2:     $r \leftarrow \sum_{x \in \mathrm{COMPS}} c(x)$
 3:     while $m \leftarrow$ NextMultiset(COMPS, $c$, $r$) do
 4:         while $g \leftarrow$ RandConnections($\mathrm{SD}^*$, $m$) do
 5:             $\mathrm{SD}' \leftarrow$ MakeCircuit($m$, $g$)
 6:             if CheckEquivalence($\mathrm{SD}'$, $A$) then
 7:                 return $\mathrm{SD}'$
 8:             end if
 9:         end while
10:     end while
11:     return $\mathrm{SD}^*$
12: end function
```

Algorithm 1 uses a double-loop to iterate over all Boolean circuit compositions. First, in the outer loop (line 3), Alg 1 considers all possible component multisets. The inner loop (line 4) generates all possible interconnections between the chosen components (and between the systems' inputs and outputs).

It is possible to construct $M = \left(\!\!\binom{n}{k+1}\!\!\right)$ multisets of cardinality up to n from a library containing k components. Although M grows exponentially, for small n, we can consider circuits of non-trivial size. For example, $\left(\!\!\binom{32}{7}\!\!\right) = 2\,760\,681$, which is a modest number of iterations.

Our implementation of NextComponents is more complex, as it considers components of variable cost. Although there are less memory-intensive approaches, NextComponents constructs all multisets of components with the sum of their costs smaller than r, sorts the result in order of increasing cost, and returns the cheapest multiset of components. On subsequent calls NextComponents simply iterates over the remaining elements in the list.

A system interconnection can be represented as a bigraph of $|X|$ and $|Y|$ nodes (cf. Fig. 9 for an illustration). In this case X contains a node for each of the target system's inputs and all candidate components' outputs while Y contains a node for each of the system's outputs and all components' inputs. We have a total of $2^{|X|+|Y|}$ different bigraphs, hence Alg. 1 considers a total search space $O((m+1)^n \cdot 2^{|X|+|Y|})$, which renders the brute-forcing of this search space infeasible even for the smallest systems.



p x_1
x_3
x_5
q x_2
x_4 x_6
x_7
i
x_8 o_4
X
p q i x_3 x_4 x_8
Y
x_1 x_2 x_7 x_5 x_6 o_4

Fig. 9. A circuit and its interconnection bigraph

It is possible to prune a significant portion of the search space by skipping over ill-formed circuits. Such circuits are ones connecting multiple-outputs, having unconnected inputs or outputs or disconnected components, having feedback, etc. Furthermore there are many symmetrical interconnections, e.g., the ordering of the inputs of an and-gate is irrelevant (this may not be the case if we consider arbitrary logic functions, for example an adder or a multiplier).

One way to sample from the space of all possible interconnections is to consider a subset of all possible biadjacent matrices of a certain size, with constraints for producing well-formed circuit and bypassing symmetries. This is done by the RandConnections subroutine, making use of a specialized Constraint Satisfaction (CS) solver. Note that the use of random sampling for generating possible interconnections turns Alg 1 into an incomplete algorithm, i.e., it may fail to find an equivalent circuit. Improving the completeness and performance of Alg 1 is a separate topic which we will not discuss in details in this paper.

5.2 The General Redesign Engine

Alg. 2 shows the pseudo-code for the General Design Engine (GRE). Given an initial set of components, COMPS, GRE first computes a component subset $\mathrm{COMPS}^* \subseteq \mathrm{COMPS}$ such that no malfunctioning of a component in $\mathrm{COMPS} \setminus \mathrm{COMPS}^*$ leads to a danger $\alpha \in A$. This is done in lines 2 – 8 of Alg. 2. The most complex computation there is performed by the diagnostic engine (MinDiagnosis).

Returning to our running example, we have two sets of probability-minimal diagnoses: $\Omega_1 = \{\{\neg h_8\}\}$ for α_1, and $\Omega_2 = \{\{\neg h_1\}, \{\neg h_2\}, \{\neg h_3\}, \{\neg h_4\}, \{\neg h_8\}\}$ for α_2. Taking the union of all components in Ω_1 and Ω_2 we have $\mathrm{COMPS}^* = \{h_1, h_2, h_3, h_4, h_8\}$.

Once GRE has computed COMPS^*, it calls the auxiliary subroutine Subcircuit which removes from the original system description **Wff** modeling components not in

Algorithm 2 General redesign engine

```
1:  function GRE($\mathcal{L}$, DS, $S$, $A$)
        inputs: $\mathcal{L}$, component library
                DS, diagnostic system
                $S$, set of Wff, output selection
                $A$, set of terms, dangers
        returns: set of Wff, redesigned model
        local variables:
            $C$, set of Wff, correction circuit
            $D$, set of Wff, danger detection circuit
            COMPS$^*$, set of assumable variables
            SD$^*$, set of Wff, redesign target
            $B$, set of terms, propagated dangers
            $\bar{B}$, set of terms, corrected dangers
            $\Omega$, set of diagnoses
            $\alpha$, term, danger
            $\omega$, variable set, diagnosis
2:      COMPS$^* \leftarrow \emptyset$
3:      for all $\alpha \in A$ do
4:          $\Omega \leftarrow$ MinDiagnoses(DS, $\alpha$)
5:          for all $\omega \in \Omega$ do
6:              COMPS$^* \leftarrow$ COMPS$^* \cup \omega$
7:          end for
8:      end for
9:      SD$^* \leftarrow$ Subcircuit(SD, COMPS$^*$)
10:     $B \leftarrow$ PropagateValues(SD, SD$^*$, $A$)
11:     $\bar{B} \leftarrow$ CorrectOutputs(SD$^*$, COMPS$^*$, $B$)
12:     $C \leftarrow$ Synthesize($\mathcal{L}$, SD$^*$, COMPS$^*$, $B$)
13:     $D \leftarrow$ Synthesize($\mathcal{L}$, SD$^*$, COMPS$^*$, $\bar{B}$)
14:     return $D \cup S \cup C$
15: end function
```

COMPS* and their interconnections. Doing this for our running examples results in the following SD*:

$$\text{SD}^* = \begin{cases} h_1 \Rightarrow (a \Leftrightarrow \neg p) \\ h_2 \Rightarrow (p \Leftrightarrow \neg r) \\ h_3 \Rightarrow (b \Leftrightarrow \neg q) \\ h_4 \Rightarrow (q \Leftrightarrow \neg s) \\ h_8 \Rightarrow (o_4 \Leftrightarrow i \wedge r \wedge s) \end{cases}$$

It may be the case that the inputs and outputs to SD* are not necessarily inputs and outputs to SD. For example, if we consider the single dangerous observation $A' = \{\{i \wedge a \wedge b \wedge \neg o_1 \wedge \neg o_2 \wedge \neg o_3 \wedge \neg o_4\}\}$, the detection/correction target subcircuit is formed by components with assumable variables h_2, h_4, and h_8. In the latter case we need the dangerous inputs/outputs propagated to the inputs/outputs of the detection/correction target subcircuit SD*. This is done by the PropagateValues function, which uses, for example, Boolean constraint propagation (Forbus and de Kleer, 1993). Evaluating PropagateValues with A' as input gives us $\{\{\neg p, \neg q, \neg o_4\}\}$ and evaluating PropagateValues with the danger set A (cf. the running example) as input results in the unmodified A (in the latter case SD* and SD share the same inputs and outputs).

In order to generate the correction circuit, GRE needs to (1) take the inputs assignment from each danger observation and (2) to compute the outputs for these inputs given a nominal functioning of the model. This is performed by the CorrectOutputs function in line 11 of Alg. 2.

Once we have computed the target subcircuit SD*, the function Synthesize generates the correction circuit C (line 12) and the danger detection circuit D (line 13). Due to the randomized nature of Alg 1 C and D are nearly cost optimal with respect to the component library $\mathcal{L}$.

6. EXPERIMENTAL RESULTS

In this section we experimentally study the trade off between the redesign cost and the number of suppressed dangers. The complexity of Alg. 1 constrains the size of the models we can automatically redesign with GRE. As improving the performance of Alg. 1 is outside the scope of this paper, we have studied the GRE trade offs on Boolean circuits having less 10 gates (cf. Table 1 for an overview).

Name	Description	|IN|	|OUT|	|COMPS|
poly	Boolean polycell	5	2	5
add	2-bit adder	3	2	5
sub	2-bit subtractor	3	2	7
demux	2-4 demultiplexer	3	4	8

Table 1. GRE models.

The component library we have used for testing GRE is standard, consisting of an inverter, and 2-input XOR, AND, NAND, OR, and NOR gates. We have assumed that all components have the same cost ($c = 1$).

An advantage of GRE is that it allows the suppression of, for example, all observations leading to single-faults. Such redesign strategies are very applicable in practice, as decreasing the likelihood of double-faults and faults of higher cardinality necessitates the use of a voting mechanism with lower failure probability than a multiple-cardinality fault.

Name	TMR Cost	GRE Cost	Savings
poly	23	17	26%
add	23	16	30%
sub	29	20	30%
demux	40	29	28%

Table 2. GRE cost savings for single-faults.

Table 2 shows the cost of TMR and the cost of a GRE redesign suppressing all observations leading to single-faults. We gain an almost constant cost decrease between 26% and 28%. Note that the cost saving can be arbitrarily larger with non-uniform component costs.

The trade off between the redesign cost and the number of suppressed symptoms for the four benchmark circuits is shown in Fig. 10. For each benchmark circuit we have computed the set of all observations A leading to faults of cardinality k. For each cardinality k, we have repeatedly sampled (20 times) A and computed the redesign cost of suppressing all dangers in A.

Figure 10 shows that for many sets of dangers, GRE computes a correction circuit of a cost smaller than the one of the original circuit (the latter is shown with black dashed line in Fig. 10). Interestingly, it is cheaper to suppress smaller sets of higher-cardinality faults (e.g.,

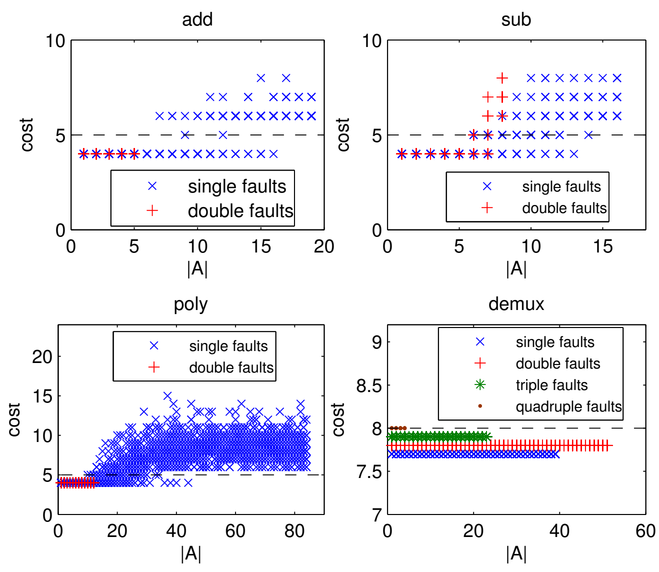


Fig. 10. GRE costs vs. number of suppressed symptoms (correction subcircuits).

double-faults), than, for example all single-faults. This can be useful in circumstances, where faults with smaller a priori probability have higher impact (this necessitates the introduction of a loss function which is a subject of future work).

From the experiments of redesigning small circuits, it is clear that GRE can be used to trade off reliability for cost and after the performance of the circuit synthesis algorithm is improved to handle larger models, GRE will be useful for a range of model-based redesign problems.

7. CONCLUSION

This paper presents a novel model-based algorithm, GRE, for redesigning systems to display fault-tolerance to specified faults. GRE uses a cost-optimal Boolean functional synthesis algorithm to generate a cost-minimal detection and correction mechanism. We have empirically shown that GRE creates cheaper redesigns compared to the standard TMR fault-tolerance approach.

GRE can be applied to several tasks beyond fault-tolerant design, since it is based on model-based redesign. For example, it can redesign systems to avoid "forbidden" modes, as is done in safety-verification.

REFERENCES

Allison, J., Kokkolaras, M., and Papalambros, P. (2007). Optimal partitioning and coordination decisions in decomposition-based design optimization. In *Proc. DETC'07*, 4–7.

Cheatham, J., Emmert, J., and Baumgart, S. (2006). A survey of fault tolerant methodologies for FPGAs. *Design Automation of Electronic Systems*, 11(2), 501–533.

Chen, L., Macwan, A., and Li, S. (2007). Model-based rapid redesign using decomposition patterns. *Journal of Mechanical Design*, 129, 283.

Chen, Y.L. and Provan, G. (2001). Model-based control reconfiguration: A shipboard system example. In *Proc. ISIC'01*, 320–325.

de Kleer, J., Mackworth, A., and Reiter, R. (1992). Characterizing diagnoses and systems. *Artificial Intelligence*, 56(2-3), 197–222.

de Kleer, J. and Williams, B. (1987). Diagnosing multiple faults. *Artificial Intelligence*, 32(1), 97–130.

Du, X., Guo, J., and Beeram, H. (2008). Sequential optimization and reliability assessment for multidisciplinary systems design. *Structural and Multidisciplinary Optimization*, 35(2), 117–130.

Forbus, K. and de Kleer, J. (1993). *Building Problem Solvers*. MIT Press.

Gössler, G. and Sifakis, J. (2005). Composition for component-based modeling. *Science of Computer Programming*, 55(1-3), 161–183.

Han, J., Gao, J., Qi, Y., Jonker, P., and Fortes, J. (2005). Toward hardware-redundant, fault-tolerant logic for nanoelectronics. *IEEE DESIGN & TEST*, 328–339.

Keppens, J. and Shen, Q. (2001). On compositional modelling. *The Knowledge Engineering Review*, 16(2), 157–200.

Liang, J., Mourelatos, Z., and Tu, J. (2008). A single-loop method for reliability-based design optimisation. *International Journal of Product Development*, 5(1-2), 76–92.

Liu, Z.J., Chen, W., Huang, H.Z., and Yang, B. (2007). A diagnostics design decision model for products under warranty. *International Journal of Production Economics*, 109(1-2), 230–240.

López-Arévalo, I., Bañares-Alcántara, R., Aldea, A., and Rodríguez-Martínez, A. (2007). A hierarchical approach for the redesign of chemical processes. *Knowledge and Information Systems*, 12(2), 169–201.

Lustig, Y. and Vardi, M. (2009). Synthesis from component libraries. In *Proc. FOSSACS'09*, 395–409.

McCluskey, E.J. (1956). Minimization of Boolean functions. *The Bell System Technical Journal*, 35(5), 1417–1444.

Menzies, T. and Singh, H. (2003). Many maybes mean mostly the same thing. In *Soft Computing in Software Engineering*, 125–150.

Monga, A. and Zuo, M. (1998). Optimal system design considering maintenance and warranty. *Computers and Operations Research*, 25(9), 691–705.

Ollinger, G. and Stahovich, T. (2004). RedesignIT – A model-based tool for managing design changes. *Journal of Mechanical Design*, 126, 208–216.

Simsir, M., Cadambi, S., Ivancic, F., Roetteler, M., and Jha, N.K. (2008). Fault-tolerant computing using a hybrid nano-CMOS architecture. In *Proc. VLSID'08*, 435–440.

Stumptner, M. and Wotawa, F. (1998). Model-based reconfiguration. In *Proc. AID'98*, 45–64.

Suh, N.P. (1990). *The Principles of Design*.

Tahoori, M. (2005). A mapping algorithm for defect-tolerance of reconfigurable nano-architectures. In *Proc. CAD'05*, 668–672.