Computing Bounds for Fault Tolerance using Formal Techniques[*]

Görschwin Fey André Sülflow Rolf Drechsler

Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
{fey,suelflow,drechsle}@informatik.uni-bremen.de

ABSTRACT

Continuously shrinking feature sizes result in an increasing susceptibility of circuits to transient faults, e.g. due to environmental radiation. Approaches to implement fault tolerance are known. But assessing the fault tolerance of a given circuit is a tough problem.

Here, we propose the use of formal methods to assess the robustness of a digital circuit with respect to transient faults. Our formal model uses a fixed bound in time to cope with the complexity of the underlying sequential equivalence check. The result is a lower and an upper bound on the robustness. The underlying algorithm and techniques to improve the efficiency are presented. In experiments the method is evaluated on circuits with different fault detection mechanisms.

Categories and Subject Descriptors

B.8.1 [**Performance and Reliability**]: Reliability, Testing, and Fault-Tolerance; B.6.3 [**Logic Design**]: Design AidsVerification

General Terms

Verification, fault tolerance, robustness

Keywords

Fault Tolerance, SAT, Formal Verification

1. INTRODUCTION

According to Moore's Law the number of components per area increases at an exponential rate in integrated circuits. One consequence is an increase of externally induced transient faults [20].

Techniques to cope with transient faults are available on the production level [26] or the design level [1, 10]. Even first tools to improve fault tolerance are available [25].

Proving robustness with respect to transient faults is difficult. Simulation or emulation based methods [6, 18] can only cover a small portion of the states and the input space of a circuit. A formal analysis determines the probability of a fault to propagate to a primary output (see e.g. [16]). But the computational effort is extremely high.

Methods commonly applied for formal verification can *prove* fault tolerance of an implementation. The approach of [3] proposes to use symbolic methods for the classical analysis of fault trees. But the faults have to be specified manually. Similarly, [13] and [15] rely on symbolic methods. These approaches analyze fault tolerance with respect to mutations of the implementation. As a result, [13] decides whether an implementation is fault tolerant or not, while [15] also provides data about the state space. Both techniques use the original circuit as a specification. The authors of [19] determine fault tolerance with respect to given formal properties. Only faults in state bits are considered. None of the techniques mentioned so far provides insight about circuit structures that are not fault tolerant.

The technique proposed here is similar to [11] and uses the same fault model. A circuit is classified as robust if no fault tampers the output behavior. Detailed feedback about components that are not fault tolerant is returned. Thus, the approach in [11] provides a good basis, but has several limitations when considering practical problems. For example the formal analysis may have to consider a large number of time steps before providing a result and reachability analysis is required to get accurate information.

In contrast, our approach is more efficient and fits practical requirements. The main contributions of our work are:

- Practical model for robustness checking

  We explain why a full formal analysis is an overkill in practice and how a fault detection mechanism helps to prove fault tolerance.

- Fault tolerance within a lower and an upper bound

  While running, our technique delivers bounds on the fault tolerance by determining robust, non-robust and non-classified components. Non-classified components point to potential *Silent Data Corruption* (SDC).

- Restricted observation window for formal analysis

  Restricting the observation time significantly improves the performance.

- Avoiding full reachability analysis

  Full reachability analysis is often too expensive. We show how to use a light-weight reachability analysis when assessing fault tolerance.

Experimental results evaluate the approach and the performance of the algorithm. The formal approach expectedly requires long run times, but proves fault tolerance with respect to any possible input stimulus. Even circuits where full reachability analysis is not feasible are effectively handled by our algorithm. The optimization techniques, i.e. the consideration of structural information and the reuse of learned information, improve the performance by a factor of up to 11.

[*]This work has been funded in part by DFG grants DR 287/19-1 and FE 797/5-1.

DAC 2009, July 26 - 31, 2009, San Francisco, California, USA.
Copyright 2009 ACM 978-1-60558-497-3 -6/08/0006 ...$10.00.

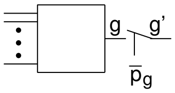

Figure 1: Fault modeling

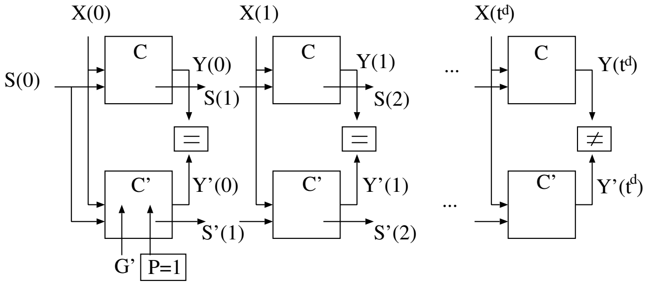

Figure 2: Sequential comparison

This paper is structured as follows: The underlying fault model and a basic approach to determine the robustness are discussed in the next section. Section 3 introduces a model bounded in time that also yields bounds on the robustness of a circuit. The incremental algorithm is explained in Section 4, while Section 5 presents optimization techniques to improve efficiency. Experimental results are given in Section 6. Section 7 concludes the paper.

2. FAULT MODEL AND BASIC APPROACH

We consider a synchronous sequential circuit $\mathbb{C}$ with *Primary Inputs* (PIs) X, *Primary Outputs* (POs) Y and state bits S. The number of components in $\mathbb{C}$ is denoted by $|\mathbb{C}|$. Here, a component may be a gate, a module or a source level expression in the hardware description language. Our fault model assumes that a *faulty* component behaves non-deterministically in one time frame, i.e. the value of the output of the component does not depend on the values of the inputs. We consider single faults only and justify in Section 3 why this is sufficient. A component g is *robust* iff the output behavior of $\mathbb{C}$ cannot change when g is faulty. Let $\mathbb{T}$ be the set of robust components in $\mathbb{C}$, then the robustness of $\mathbb{C}$ is given by $|\mathbb{T}|/|\mathbb{C}|$.

As suggested in [11] we use an instance of *Boolean satisfiability* (SAT) to measure robustness. In the following the output signal of component g is associated to variable g as well. A fault is modeled as follows (the formulation is similar to SAT-based diagnosis [21]): For a component g, a fault predicate p_g and a new variable g' are introduced; then g is replaced by $\overline{p}_g \rightarrow g' = g$ as shown in Figure 1. Consequently, the value of g' is specified by the circuit structure if $p_g = 0$. But if a fault at g is asserted by $p_g = 1$, g' may take any value. Given a circuit $\mathbb{C}$, the circuit $\mathbb{C}'$ is created by replacing each component as explained above. Then, P denotes the set of fault predicates and G' denotes the set of newly introduced variables to replace the outputs of components.

Now, the SAT instance is created as shown in Figure 2: The circuit $\mathbb{C}$ is unrolled for t^d time frames as in bounded model checking [2]; the unrolled circuit is compared to the copy $\mathbb{C}'$ connected to $t^d - 1$ instances of $\mathbb{C}$; the POs in the final time frame t^d are forced to be different; only one variable in P may take the value 1. This SAT instance is satisfied iff the output behavior of the faulty and the original circuit differ in time frame t^d. This may only happen, when a faulty value is injected at component g with $p_g = 1$, i.e. component g is not robust. By finding all satisfying assignments, the non-robust components are retrieved. Once a component g has been found non-robust, further solutions for this component are blocked by inserting the constraint $p_g = 0$ into the SAT instance. All non-robust components are calculated by iteratively incrementing t^d.

Note, that this model already provides an improvement over [11]. Fault injection is only required in the first time step instead of all time steps. If the set of initial states $S(0)$ is equal to the set of reachable states S^*, the exact value of the robustness is determined when reaching the maximal sequential depth[1] of the correct circuit and a faulty circuit [11]. In the following we assume that $S(0)$ is equal to S^* until relaxing this constraint in Section 3.2.

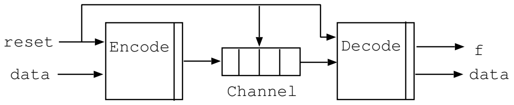

Figure 3: Transmission system

3. BOUNDS FOR ROBUSTNESS

In this section we adjust the notion of robustness and the model to practical requirements. As a "side effect" the computational effort decreases. First we justify an observation window to restrict the number of time steps considered by the formal analysis. Then we explain how to relax reachability analysis in the context of our approach.

3.1 Observation Window

After detecting an internal malfunction some action has to be taken at the system level in practice. Otherwise the effects of multiple faults may accumulate and may cause a disastrous failure. Therefore, we assume that a fault detection signal f exists. If a malfunction occurs, this is signaled by setting f within a given time bound of no more than t^d time steps. Assuming a very low probability for more than one fault within t^d time steps, this is safe. By this we retrieve exact bounds for the robustness while restricting the formal analysis to an *observation window* of t^d time steps.

We apply a case split to determine the robustness of a component g. Assume component g behaves faulty, then the robustness of g is assessed as follows:

1. Component g is robust, if
   (a) $f = 1$ within t^d time frames before or when a faulty value at the POs occurs or
   (b) $f = 0$, a faulty value at the POs does not occur and after t^d the same state is reached as in the fault free circuit.
2. Component g is non-robust, if
   a faulty value at the POs occurs within t^d time frames and before $f = 1$.
3. Component g is not classified, if
   $f = 0$, a faulty value at the POs does not occur and the state differs from the fault free circuit after t^d time frames.

To guarantee that a circuit is robust, neither non-robust nor non-classified components must remain. A non-robust component is a threat – a fault in this component may cause wrong output values. Knowledge about non-classified components is also essential. A fault in such a component cannot directly influence the output values, but changes the internal state of the circuit, i.e. causes SDC. If this is not detected within the required observation time t^d, an undetected error is immanent in the circuit. Effects of multiple faults may accumulate and eventually cause erroneous output.

The same model also handles circuits that directly correct faults instead of flagging a fault. In this case f is assumed to be constantly 0. As a result case 1(a) of the case split given above does not occur.

EXAMPLE 1. *A (7,4)-Hamming-Code recognizes and repairs single faults [12]. Figure 3 shows a transmission using an encoder for 4 bit* `data`*, a bit-wise serial channel and a decoder. A failure in the transmitted code word is flagged by setting f. The timing is summarized like this:*

[1] The sequential depth is the longest trace without repeating a state.

Table 1: Hamming model

t	$\|\mathbb{T}\|$	$\|\mathbb{S}\|$	$\|\mathbb{U}\|$	R_{lb} %	R_{ub} %
0	5	2	277	1.76	99.30
1	54	28	208	18.72	90.34
2	83	41	170	28.23	86.05
3	112	53	133	37.58	82.21
4	141	65	96	46.69	78.48
5	170	79	57	55.56	74.18
6	217	93	0	70.00	70.00

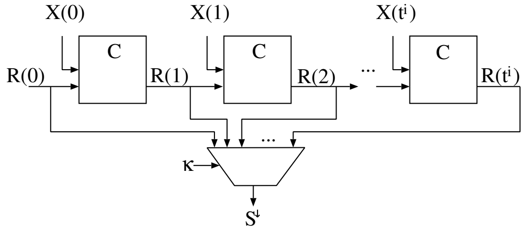

Figure 4: Constraining initial state

- *Encoding and transmission to the channel: 1 time step*
- *Transmission: 4 time steps (registers in the channel)*
- *Decoding, writing to the output, setting f: 1 time step*

The determined robustness depends on the value of t^d:

- $t^d < 6$*: The data from $t = 0$ did not arrive at the POs, yet. Faults in the decoding logic are detected within 1 time frame by setting f. Therefore these components are classified.*

  Faults in the channel change the state, but not all data has been decoded, yet. These faults are undetected, yet, and the components cannot be classified. While incrementing t^d, more and more components are classified.
- $t^d = 6$*: The input data reaches the POs. Faults that can be detected are flagged. All components are classified.*
- $t^d > 6$*: Faults injected at $t = 0$ do not influence the state of the model after more than 6 time frames.*

Let $\mathbb{T}$ be the set of components classified as robust; $\mathbb{S}$ the set of components classified non-robust and $\mathbb{U}$ the set of components not classified, yet. Then, $\mathbb{C} = \mathbb{T} \cup \mathbb{S} \cup \mathbb{U}$. Now, a lower bound R_{lb} and an upper bound R_{ub} for the robustness of the circuit $\mathbb{C}$ are given by:

$$R_{lb} = \frac{|\mathbb{T}|}{|\mathbb{C}|} = 1 - \frac{|\mathbb{S} \cup \mathbb{U}|}{|\mathbb{C}|} \quad \text{and} \quad R_{ub} = \frac{|\mathbb{T} \cup \mathbb{U}|}{|\mathbb{C}|} = 1 - \frac{|\mathbb{S}|}{|\mathbb{C}|}$$

EXAMPLE 2. *The bounds determined for the hamming model of the previous example are shown in Table 1. The bounds approach each other, until all components are classified for $t = 6$.*

3.2 Reachability

Up to now we assumed, that $S(0)$ was the set of reachable states S^*. Consequently, the approach determined exact bounds for the robustness. In practice, reachability analysis is often not feasible due to the computational complexity. For example the computation of S^* with a *Binary Decision Diagram* (BDD) [4] often requires a large amount of main memory or exceeds run time limitations. Therefore we relax this requirement in the following.

In practice the bounds derived above for the robustness depend on the set of states $S(0)$ applied in the initial time frame of the formal analysis. Assume that an overapproximation $S^{\uparrow}$ and an underapproximation $S^{\downarrow}$ of the reachable states are available, i.e. $S^{\downarrow} \subseteq S^* \subseteq S^{\uparrow}$.

When considering the subset $S^{\downarrow}$ some states reachable during normal operation are excluded. Therefore some components may not be activated even though relevant during normal operation, e.g. when the ADD-operation in a CPU is never activated. These components are classified as robust instead of non-robust. The robustness determined when using $S^{\downarrow}$ is larger than the real value. Therefore, when calculating an upper bound of the robustness using $S^{\downarrow}$ is safe, i.e. $R_{ub}(S^{\downarrow}) \geq R_{ub}(S^*)$.

Similarly, considering unreachable states using $S^{\uparrow}$ decreases the calculated robustness. Consider a circuit with *Triple Modular Redundancy* (TMR). In the fault free case the three redundant modules are in the same state. Consequently, any internal fault of one module is masked and all component are robust. But when also unreachable states are considered where the state of the three modules deviates, a fault may change the output behavior of the overall circuit. Additional components may be classified as non-robust and it is safe to use the overapproximation to determine a lower bound on the robustness, i.e. $R_{lb}(S^{\uparrow}) \leq R_{lb}(S^*)$.

No full reachability analysis is required. Instead we apply light-weight approximations.

For $S^{\downarrow}$ we integrate partial reachability analysis into the formal analysis using the structure shown in Figure 4. The original circuit $\mathbb{C}$ is unrolled for t^i time frames and starts from a set of states $R(0)$ known to be reachable, e.g. the reset state. Then, $S^{\downarrow}$ contains any state reachable from $R(0)$ within t^i time steps as k is left unconstrained. The unrolling depth t^i controls the accuracy: $S^{\downarrow}$ remains an underapproximation of the reachable states when t^i is smaller than the state space diameter[2].

When leaving $R(0)$ unconstrained, $R(t^i)$ provides an overapproximation $S^{\uparrow}$ of the reachable states. But for our experiments we use a faster approach that assumes all states are reachable. Alternatively, in case of TMR circuits an invariant can force the states of the three redundant modules to be equal.

Of course, these light-weight approximations may be replaced by more elaborate approaches like the SAT-based procedure in [5]. In this case compactly representing $S(0)$ is crucial.

4. ALGORITHM

This section provides an incremental algorithm to transform the calculation of bounds for the robustness into a sequence of SAT instances [7]. A SAT solver [9] is used to determine the solutions. Parameters for the algorithm are the circuit $\mathbb{C}$, the set of states to be considered S^{use} and the size of the observation window t^d. The algorithm is based on the approach introduced in Section 2: The original circuit $\mathbb{C}$ and a copy $\mathbb{C}'$ are unrolled for an increasing number of time frames $t \in [0 \ldots t^d]$. The initial states of both copies are identical. The POs of time frame t are forced to different values. Circuit $\mathbb{C}'$ contains fault injection logic in time frame 0. If all fault predicates p_g are set to 0, both copies behave identically. The problem is unsatisfiable.

To analyze single faults, fault injection logic in time frame 0 is sufficient which reduces the search space compared to [11]. This is valid, because all states in S^{use} are considered as initial states of the formal analysis, i.e. $S(0) = S^{\text{use}}$. Also at most one fault predicate may take the value 1. The model supports faults in state elements as well as in combinational logic or at primary inputs.

The algorithm in Figure 5 shows the incremental algorithm that determines the lower and upper bound for the robustness. Once a component is classified, this information is used in the following iterations to reduce the run time. Given a circuit $\mathbb{C}$, a copy $\mathbb{C}'$ with fault injection logic is created (Lines 2–5). Both copies are converted into *Conjunctive Normal Form* (CNF) [23] (Line 6). The initial states of both copies are forced to be equal (Line 7). The initial states $S(0)$ for the formal analysis are restricted to S^{use} (Line 8). Whether the algorithm computes exact bounds, a lower bound or an upper bound for the robustness depends on S^{use} as explained in

[2] The state space diameter is the largest length of the shortest trace between any two states.

```
1  function robustness ($\mathbb{C}$, $S^{\text{use}}$, $t^d$)
2    create a copy $\mathbb{C}_0'$ of $\mathbb{C}$
3    foreach component $g \in \mathbb{C}_0'$
4      replace $g$ by $g'[g,p_g]$;
5    done
6    convert to SAT instance;
7    force init states of $\mathbb{C}_0'$ and $\mathbb{C}_0$ to be equal;
8    force $S(0) = S^{\text{use}}$
9    constrain $\sum p_g == 1$;
10
11   $\mathbb{T} := \emptyset$;
12   $\mathbb{S} := \emptyset$;
13   $\mathbb{U} :=$ all components $g \in \mathbb{C}_0'$;
14   $t := 0$;
15   while ($t \le t^d$ && $\mathbb{U} \neq \emptyset$)
16     if (t > 0) then
17       create a copy $\mathbb{C}_t'$ of $\mathbb{C}$;
18       create $\mathbb{C}_t$ and connect to $\mathbb{C}_t'$;
19     fi
20     connect PIs of $\mathbb{C}_t'$ and $\mathbb{C}_t$;
21     constrain $f$ = 0 in $\mathbb{C}_t$;
22     $cmpPOs$:= at least one pair of POs differs;
23     $cmpFFs$:= at least one pair of FFs differs;
24
25     add constraint UR := $(cmpPOs \,\&\, !f) = 1$;
26     $\mathbb{S}'$:= extractAllSolutions();
27     remove constraint UR;
28
29     add constraint UC := $(!f \,\&\, !cmpPOs \,\&\, cmpFFs) = 1$;
30     $\mathbb{U}'$:= extractAllSolutions();
31     remove constraint UC;
32     $\forall g \in \mathbb{U}'$: remove constraint $p_g = 0$;
33
34     $\mathbb{T}' := \mathbb{U} \setminus (\mathbb{S}' \cup \mathbb{U}')$;
35     $\forall g \in \mathbb{T}'$: add constraint $p_g = 0$;
36
37     $\mathbb{T} := \mathbb{T} \cup \mathbb{T}'$;
38     $\mathbb{U} := \mathbb{U}'$;
39     $\mathbb{S} := \mathbb{S} \cup \mathbb{S}'$;
40
41     $t := t+1$;
42     remove $cmpPOs$;
43     remove $cmpFFs$;
44   done;
45   return ($\mathbb{T}$,$\mathbb{S}$,$\mathbb{U}$);
46 end function;
```

Figure 5: Algorithm

```
1  function extractAllSolutions ()
2      $M := \emptyset$;
3      while (satisfiable) do
4          $G = \{g | p_g == 1\}$;
5          $M := M \cup G$;
6          add constraint $p_g = 0$;
7      done;
8      return M;
9  end function;
```

Figure 6: Retrieving all solutions

Section 3.2. The number of fault predicates with value 1 is limited to one (Line 9).

Then, the sets of robust ($\mathbb{T}$), non-robust ($\mathbb{S}$), and non-classified ($\mathbb{U}$) components are initialized (Lines 11–13). In the beginning all components are non-classified. Next, the sets are incrementally updated for time frame t, starting at $t = 0$ up to $t = t^d$ (Lines 14–44). As soon as all components are classified, i.e. $\mathbb{U} = \emptyset$, the algorithm terminates. Fresh copies of $\mathbb{C}$ are appended to the unrolled circuits for $t > 0$ (Line 16–20). A constraint forces $\mathbb{C}$ to behave fault free (Line 21). Additional logic compares the POs in time frame t (Line 22), where $cmpPOs = 1$ indicates a different value for fault free and faulty copy. Similarly, $cmpFFs$ compares the states (Line 23).

Then the components $\mathbb{S}'$ that can be classified as non-robust in time frame t are determined (Lines 25–27). The POs are forced to different values and the fault detection signal f is forced to 0 (Line 25). Each satisfying solution provides a component that is non-robust. The newly classified non-robust components $\mathbb{S}'$ are returned by the subroutine *extractAllSolutions* shown in Figure 6. The subroutine extracts one non-robust component per satisfying solution (Line 4) and forces the fault predicate of this component to 0 afterwards (Line 6).

The main routine in Figure 5 proceeds by removing the constraints on POs and f (Line 27). Next, the algorithm determines the remaining non-classified components $\mathbb{U}'$ in a similar way (Lines 29–31). In case of non-classified components the constraints $p_g = 0$ are removed (Line 32) before the next iteration for $t+1$ starts.

Now, the newly classified set of robust components is available (Line 34). These components do not have to be considered in further iterations and their fault predicates are fixed to 0 (Line 35).

Finally, the sets $\mathbb{T}$, $\mathbb{S}$ and $\mathbb{U}$ are updated by adding or assigning the newly classified components, t is increased, and the additional logic to compare POs and states is removed (Lines 37–43).

If non-classified components remain and t^d has not been reached, the next iteration starts (Line 15). Otherwise the algorithm terminates and returns the three sets $\mathbb{T}$, $\mathbb{S}$ and $\mathbb{U}$. As explained above the parameter S^{use} determines whether exact values or approximations are returned.

5. OPTIMIZATION TECHNIQUES

The algorithm presented so far solves multiple sequential equivalence checking problems and sequential equivalence checking is a hard problem itself. Therefore, optimization is required to improve the performance.

Knowledge about structural dominators is known to be often helpful in CAD algorithms. For example, the output of a fanout free region is a dominator for all nodes within the region. The notion of dominators is more general. A component g is dominated by a component e, if any path from g to a primary output or state bit passes along e. Thus fault effects from g must propagate along e. If component e is robust, component g is robust as well. We determine dominators using the algorithm from [14]. Then, the algorithm to determine robustness runs in two steps. First, faults are only injected into components that dominate others. Second, the dominated components of non-robust and non-classified dominators are considered for a detailed classification. This speeds up the overall run time because the search space is pruned.

Instead of sequential equivalence checking also sequential *Automatic Test Pattern Generation* (ATPG) may be used as the underlying engine. In this case one problem instance is created per fault that has to be considered. As an advantage, the size of the problem instance shrinks by only including those parts of the circuit that may be influenced by the particular fault. Moreover, similar to combinational ATPG, propagation constraints can be used to improve the performance of the engine [22, 8]. For an evaluation we used a SAT-based sequential ATPG engine. However, on the circuits considered the algorithm of Section 4 using equivalence checking was faster than the ATPG approach. This can be explained by the structural similarity of the problem instances. The algorithm of Section 4 creates one problem instance for all faults. This problem instance is kept and extended by further copies of the circuit until t^d is reached. Using the concept of incremental SAT [24], the proof engine keeps learned information for reuse in subsequent calls. Using ATPG, independent problem instances are created for all faults and similar information has to be learned from scratch. Therefore, we choose the algorithm based on equivalence checking for further experiments.

6. EXPERIMENTAL RESULTS

Experimental results are provided in the following. Our benchmark suite contains different types of sequential circuits that allow to explain the results by considering the structure of the circuits:

- without fault tolerance,
- with *Triple Modular Redundancy* (TMR) and
- with fault detection.

The circuits without fault tolerance are taken from the ITC’99 benchmark suite named by their original names b01–b13. Using these circuits, fault tolerant TMR circuits were created. The circuit was

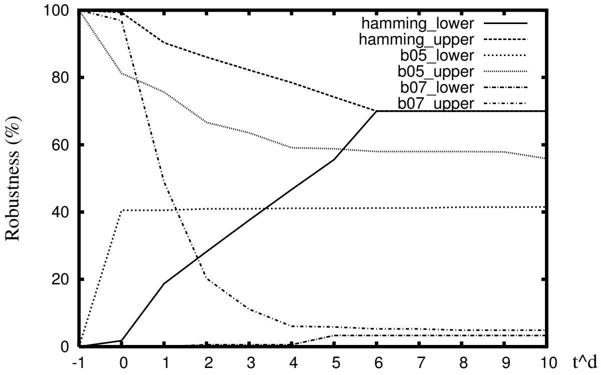

Figure 7: Robustness vs. t^d

replicated three times and a combinational majority voter drives the POs. These TMR circuits have the same sequential depth as the original circuit.

To create circuits with fault detection, the TMR circuits were extended with a signal f. While the states of the three instances are identical, no fault is detected, i.e. $f = 0$. Otherwise a fault is signaled, i.e. $f = 1$. Faults at the PIs do not activate f, because the TMR instances behave equivalently. Additionally, the Hamming model introduced in Example 1, provides fault detection. In all cases PIs, FFs and gates were considered as components.

All experiments were carried out on an AMD Athlon(tm) 64 X2 Dual Core CPU (3.0GHz, 4GB RAM, Linux). The SAT solver Chaff [17] with incremental SAT extension [24] has been used. The consideration of structural dominators improved the performance by a factor of up to 11 (see Section 5).

First, we analyze the influence of the observation window t^d on the robustness. Next the influence of constraints on $S(0)$ and the quality of approximate bounds are evaluated and finally discussed. Note, that a comparison to [11] is not given due to differing models, i.e. the bounds, the fault signal f and handling reachability.

6.1 Influence of t^d

Figure 7 exemplarily visualizes the exact bounds retrieved for three circuits using S^* while extending the observation window. Note, the initial bounds are marked with $t^d = -1$.

As already discussed in Section 3, the exact value for the robustness of the Hamming model is retrieved at $t^d = 6$ where lower and upper bound converge. In case of b05 and b07 the bounds approach each other quite rapidly in the beginning, but do not meet within 10 time frames. While 15% of the components cannot be classified for b05, only 2% remain non-classified for b07. This shows that the convergence behavior of the bounds significantly depends on the design. The incremental algorithm may be stopped as soon as the bounds are close enough or no progress can be observed.

6.2 Influence of t^i

Table 2 summarizes further results. The overapproximation $S\uparrow$ does not constrain $S(0)$. For $S\downarrow$ the approach shown in Figure 4 was used, $R(0)$ denotes the reset state and t^i was set to 0, 1, or 10, respectively. BDD-based exact reachability analysis provides S^*.

Columns $|\mathbb{C}|$ and $|FF|$ give the number of components and state holding elements in the circuit, respectively. Column t^d denotes the length of the observation window required to classify all components. At $t^d = 10$ the algorithm has been stopped. Columns $|U|$ and *time* give the number of non-classified components and the run time in seconds for computation, respectively.

While increasing t^i more reachable states become observable at $S(0)$ and thus the accuracy of the approximate upper bound increases. For example, the results in Table 2 show a significant improvement for b08 and b10 between $t^i = 1$ and $t^i = 10$. But the overhead for computing reachable states "on-the-fly" increases when increasing t^i – more run time is required.

Additionally, for TMR circuits the number of components known to be non-classified increases. Some components classified as robust for small values of t^i become non-classified, e.g. when a fault in one of the three TMR instances does not affect the output behavior but changes the state.

The TMR circuits with fault detection mechanism immediately detect the error in one of the instances and thus an early classification as robust is possible. Already for $t^i = 0$ the upper bound is nearly correct and requires far less computation time in comparison to TMR circuits without fault detection. Only components close to PIs remain non-classified. Therefore t^d has to be increased while the faulty values are not observable at the POs.

6.3 Quality of approximate bounds

For non-TMR circuits, the bounds $R_{lb}(S\uparrow)$ and $R_{ub}(S\downarrow)$, $t^i = 10$ are close to the exact ones $R_{lb}(S^*)$ and $R_{ub}(S^*)$. Only for b05 and b13 the exact bounds differ significantly. Here, increasing t^i or t^d may help to improve the accuracy of the approximate bounds.

For TMR circuits the distance between lower and upper bound is large. The lower bound $R_{lb}(S\uparrow)$ is not tight enough, because the initial states of the TMR instances are allowed to differ. Either an exact analysis or a manual invariant is required to improve the accuracy. Experiments using an invariant to force identical initial states, yielded an almost exact lower bound $R_{lb}(S\uparrow)$. Due to page limitation no details are reported here.

The approximate bounds for circuits with fault detection are close to the exact bounds. Constraining the specification to fault free behavior by setting $f = 0$ in C_t forces the submodules to start in identical states. Consequently, the lower bound becomes more accurate and an exact analysis is not required.

6.4 Discussion

As shown in Table 2, BDD-based reachability analysis often exceeds the run time limitation. Thus, no exact information about fault tolerance can be computed. Our approximation algorithm still provides a partial classification of components giving insight about fault tolerance of the circuits.

For both, the approximate as well as the exact bounds, non-classified components are left for some circuits. Here, a large difference between upper and lower bound due to non-classified components always points to potential immanent undetected errors that do not yet materialize in the output response. Such SDC may lead to faulty output responses in combination with other faults. Thus, the knowledge about the non-classified components is mandatory.

In summary, the main focus of the experiments is on the formal model and the formal algorithm to determine robustness. The proposed model fits practical needs by restricting the observation window. The approximate bounds provide information about fault tolerance even when an exact analysis cannot be applied. For some circuits run time is a bottleneck. Here, providing a set of manual invariants decreases the run time and increases the accuracy.

7. CONCLUSIONS

We presented an approach to formally prove the robustness of a circuit. The algorithm works on a bounded number of time steps and determines a lower bound and an upper bound on the robustness. Approximate sets of reachable states are sufficient to determine these bounds. An incremental algorithm and additional optimization techniques are provided. The results show that even if only a small number of time steps is considered, an exact value of the robustness can often be obtained. Otherwise a subset of non-robust and robust components is provided, that can be used for further design modifications. Especially, for circuits with fault detection mechanism accurate values are determined efficiently.

Future work focuses on improving the effectiveness for very large circuits. That is, using multiple engines like simulation, sequential ATPG and the proposed algorithm within a single framework or by using abstraction and hierarchical information. Moreover, assessing robustness in presence of multiple faults remains an open issue.

Table 2: Influence of constraints on initial states

			$S^{\uparrow}$				$S^{\downarrow}, t^i = 0$				$S^{\downarrow}, t^i = 1$				$S^{\downarrow}, t^i = 10$				S^*				
circuit	$\lvert\mathbb{C}\rvert$	$\lvert FF\rvert$	t^d	R_{lb} %	$\lvert U\rvert$	$time(s)$	t^d	R_{ub} %	$\lvert U\rvert$	$time(s)$	t^d	R_{ub} %	$\lvert U\rvert$	$time(s)$	t^d	R_{ub} %	$\lvert U\rvert$	$time(s)$	t^d	R_{lb} %	R_{ub} %	$\lvert U\rvert$	$time(s)$
Without fault tolerance																							
b01	64	5	4	**0.00**	-	0.19	5	**10.94**	-	0.26	4	**1.56**	-	0.25	4	**0.00**	-	0.50	4	**0.00**	**0.00**	-	0.24
b02	34	4	3	**0.00**	-	0.05	4	**44.12**	-	0.06	4	**20.59**	-	0.06	3	**0.00**	-	0.13	3	**0.00**	**0.00**	-	0.09
b03	199	30	9	**0.00**	-	6.41	10	**22.61**	12	9.11	10	**5.53**	3	7.66	9	**0.50**	-	13.10	9	**0.50**	**0.50**	-	6.23
b04	832	66	5	**0.00**	-	60.57	3	**86.78**	-	8.32	5	**64.90**	-	29.65	5	**0.00**	-	131.34					[b]
b05	1199	34	2	**5.59**	-	57.42	10	**91.33**	124	98.60	10	**90.91**	126	107.26	10	**90.33**	354	477.72	10	**41.54**	**55.88**	172	225.09
b06	73	9	2	**0.00**	-	0.18	2	**16.44**	-	0.15	2	**2.74**	-	0.22	2	**0.00**	-	0.50	2	**0.00**	**0.00**	-	0.24
b07	513	49	10	**0.00**	3	22.71	10	**87.52**	190	61.06	10	**87.52**	198	68.44	10	**87.33**	268	188.22	10	**3.31**	**4.87**	8	44.21
b08	232	21	10	**0.00**	9	5.71	10	**91.38**	81	11.06	10	**90.08**	88	13.29	10	**5.17**	11	35.77	10	**0.00**	**3.88**	9	5.80
b09	198	28	9	**0.00**	-	4.25	10	**85.86**	79	10.38	10	**75.76**	88	14.48	10	**47.47**	42	16.34	10	**0.00**	**0.51**	1	5.30
b10	271	17	10	**0.00**	1	6.65	10	**62.36**	40	8.88	10	**32.84**	8	7.80	10	**1.85**	4	20.83	10	**0.37**	**1.85**	4	8.73
b11	874	31	10	**0.11**	14	123.98	10	**88.44**	133	83.09	10	**83.07**	142	99.76	10	**8.92**	14	216.44	10	**6.18**	**7.78**	14	256.67
b12	1302	121	10	**0.00**	2	239.98	10	**81.64**	449	470.24	10	**75.88**	386	460.31	10	**52.07**	457	960.20					[b]
b13	410	53	10	**0.73**	8	15.60	10	**66.10**	112	36.31	10	**62.44**	108	38.86	10	**52.68**	105	62.04	10	**2.68**	**7.56**	20	1793.98
Tripple modular redundancy																							
b01-tmr	212	15	4	**1.89**	-	2.63	10	**98.11**	114	15.82	10	**98.11**	132	22.33	10	**98.11**	135	43.08	10	**34.43**	**98.11**	135	26.97
b02-tmr	112	12	3	**1.79**	-	0.69	10	**99.11**	42	2.47	10	**98.21**	66	4.39	10	**98.21**	84	9.40	10	**23.21**	**98.21**	84	5.02
b03-tmr	637	90	9	**1.26**	-	76.56	10	**98.74**	438	208.95	10	**98.74**	513	286.89	10	**98.74**	534	563.92	10	**14.91**	**98.74**	534	4336.39
b04-tmr	2579	198	5	**0.62**	-	697.26	[a]9	**99.69**	21	19493.90	[a]6	**99.38**	276	9730.20	[a]0	**99.69**	2483	31280.70					[b]
b05-tmr	3922	102	2	**6.96**	-	1137.48	10	**99.08**	450	1256.48	10	**99.06**	477	1419.27	10	**99.06**	1179	5638.39					[b]
b06-tmr	275	27	2	**3.64**	-	3.00	10	**97.45**	81	16.30	10	**97.09**	99	24.63	10	**97.09**	102	58.70	10	**60.00**	**97.09**	102	26.22
b07-tmr	1612	147	10	**0.99**	-	2046.31	10	**99.50**	738	834.42	10	**99.50**	763	925.15	10	**99.50**	976	2373.76					[b]
b08-tmr	741	63	10	**1.08**	35	88.05	10	**99.46**	292	142.78	10	**99.46**	330	174.77	10	**99.46**	690	634.43	10	**5.94**	**99.33**	692	2180.33
b09-tmr	604	84	9	**3.31**	-	51.67	10	**99.83**	315	122.03	10	**99.67**	405	172.53	10	**99.67**	423	302.18	10	**24.17**	**99.67**	456	1572.46
b10-tmr	878	51	10	**1.37**	3	1464.80	10	**99.20**	402	273.23	10	**98.52**	552	484.63	[a]9	**98.06**	792	17176.30	10	**7.86**	**98.06**	792	2574.79
b11-tmr	2683	93	10	**0.56**	42	2721.79	10	**99.78**	666	1261.08	10	**99.52**	852	1738.78	10	**99.52**	2412	15212.20					[b]
b12-tmr	3965	363	[a]6	**0.30**	9	2933.94	10	**99.82**	2031	6657.70	10	**99.82**	2067	7311.42	10	**99.82**	3210	19253.10					[b]
b13-tmr	1330	159	10	**2.18**	24	432.77	10	**99.25**	693	653.31	10	**99.25**	729	766.78	10	**99.10**	863	1526.89					[b]
With fault detection																							
b01-tmrflt	215	15	1	**88.84**	-	0.17	1	**98.14**	-	0.06	1	**98.14**	-	0.10	1	**98.14**	-	0.32	1	**98.14**	**98.14**	-	0.26
b02-tmrflt	123	12	2	**88.62**	-	0.07	0	**99.19**	-	0.02	4	**98.37**	-	0.05	2	**98.37**	-	0.11	2	**98.37**	**98.37**	-	0.12
b03-tmrflt	648	90	4	**91.36**	-	2.11	4	**98.77**	-	0.77	4	**98.77**	-	0.98	4	**98.77**	-	5.85	4	**98.77**	**98.77**	-	4138.53
b04-tmrflt	2590	198	1	**95.56**	-	45.29	0	**99.69**	-	0.96	3	**99.38**	-	4.91	1	**99.27**	-	5377.38					[b]
b05-tmrflt	3933	102	1	**45.51**	-	352.61	0	**99.08**	-	4.23	1	**99.06**	-	6.23	1	**99.06**	-	51.10					[b]
b06-tmrflt	286	27	1	**71.33**	-	0.56	1	**97.55**	-	0.09	1	**97.20**	-	0.13	1	**97.20**	-	2.38	1	**97.20**	**97.20**	-	0.50
b07-tmrflt	1623	147	1	**93.53**	-	17.70	0	**99.51**	-	0.58	10	**99.51**	1	3.31	10	**99.51**	1	25.91					[b]
b08-tmrflt	752	63	10	**91.89**	8	5.97	10	**99.47**	1	1.15	10	**99.47**	9	5.33	10	**99.47**	9	12.22	10	**98.27**	**99.34**	8	1747.29
b09-tmrflt	615	84	2	**97.72**	-	1.33	0	**99.84**	-	0.10	10	**99.67**	-	0.96	10	**99.67**	-	2.36	2	**99.67**	**99.67**	-	1283.33
b10-tmrflt	889	51	4	**89.99**	-	5.46	2	**99.21**	-	0.42	3	**98.54**	-	1.25	4	**98.09**	-	29.66	4	**98.09**	**98.09**	-	1172.55
b11-tmrflt	2694	93	3	**96.85**	-	23.31	1	**99.55**	-	1.65	3	**99.52**	-	4.58	3	**99.52**	-	86.68					[b]
b12-tmrflt	3976	363	1	**97.91**	-	184.09	5	**99.82**	-	4.59	4	**99.82**	-	6.16	1	**99.82**	-	216.73					[b]
b13-tmrflt	1341	159	2	**89.56**	-	7.70	0	**99.25**	-	0.65	0	**99.25**	-	0.90	10	**99.11**	8	25.31					[b]
hamming	284	7	6	**70.00**	-	20.86	6	**70.98**	-	19.57	6	**70.65**	-	21.88	6	**70.00**	-	38.01	6	**70.00**	**70.00**	-	21.74

[a] Abnormal termination of SAT solver; [b] BDD could not be computed within 5h

8. REFERENCES

[1] T. Austin and V. Bertacco. Deployment of better than worst-case design: Solutions and needs. In *Int'l Conf. on Comp. Design*, pages 550–558, 2005.

[2] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1579 of *LNCS*, pages 193–207. Springer Verlag, 1999.

[3] M. Bozzano, A. Cimatti, and F. Tapparo. Symbolic fault tree analysis for reactive systems. In *Automated Technology for Verification and Analysis*, volume 4762 of *LNCS*, pages 162–176, 2007.

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

[5] P. Chauhan, E. Clarke, and D. Kroening. Using SAT based image computation for reachability analysis. Technical Report CMU-CS-03-151, School of Computer Science, Carnegie Mellon University, 2003.

[6] P. Civera, L. Macchiarulo, M. Rebaudengo, M. S. Reorda, and M. Violante. An FPGA-based approach for speeding-up fault injection campaigns on safety-critical circuits. *Jour. of Electronic Testing: Theory and Applications*, 18(3):261–271, 2002.

[7] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:506–521, 1960.

[8] R. Drechsler, S. Eggersglüß, G. Fey, A. Glowatz, F. Hapke, J. Schlöffel, and D. Tille. On acceleration of SAT-based ATPG for industrial designs. *IEEE Trans. on CAD*, 27(7):1329–1333, 2008.

[9] N. Eén and N. Sörensson. An extensible SAT solver. In *SAT 2003*, volume 2919 of *LNCS*, pages 502–518, 2004.

[10] D. Ernst, N. S. Kim, S. Das, S. Pant, T. Pham, R. Rao, C. Ziesler, D. Blaauw, T. Austin, and T. Mudge. Razor: A low-power pipeline based on circuit-level timing speculation. In *Micro Conference*, 2003.

[11] G. Fey and R. Drechsler. A basis for formal robustness checking. In *Int'l Symp. on Quality Electronic Design*, pages 784–789, 2008.

[12] R. W. Hamming. Error detecting and error correcting codes. *Bell System Technical Jour.*, 9:147–160, April 1950.

[13] U. Krautz, M. Pflanz, C. Jacobi, H. W. Tast, K. Weber, and H. T. Vierhaus. Evaluating coverage of error detection logic for soft errors using formal methods. In *Design, Automation and Test in Europe*, pages 176–181, 2006.

[14] T. Lengauer and R. E. Tarjan. A fast algorithm for finding dominators in a flowgraph. *ACM Trans. Program. Lang. Syst.*, 1(1):121–141, 1979.

[15] R. Leveugle. A new approach for early dependability evaluation based on formal property checking and controlled mutations. In *IEEE International On-Line Testing Symposium*, pages 260–265, 2005.

[16] M. Miskov-Zivanov and D. Marculescu. Circuit reliability analysis using symbolic techniques. *IEEE Trans. on CAD*, 25(12):2638–2649, 2006.

[17] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Design Automation Conf.*, pages 530–535, 2001.

[18] A. Pellegrini, K. Constantinides, D. Zhang, S. Sudhakar, V. Bertacco, and T. Austin. CrashTest: A fast high-fidelity FPGA-based resiliency analysis framework. In *Int'l Conf. on Comp. Design*, 2008.

[19] S. A. Seshia, W. Li, and S. Mitra. Verification-guided soft error resilience. In *Design, Automation and Test in Europe*, pages 1442–1447, 2007.

[20] P. Shivakumar, M. Kistler, S. W. Keckler, D. Burger, and L. Alvis. Modeling the effect of technology trends on the soft error rate of combinational logic. In *Int'l Conf on Dependable Systems and Networks*, pages 389–398, 2002.

[21] A. Smith, A. Veneris, M. Fahim Ali, and A.Viglas. Fault diagnosis and logic debugging using boolean satisfiability. *IEEE Trans. on CAD*, 24(10):1606–1621, 2005.

[22] P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli. Combinational test generation using satisfiability. *IEEE Trans. on CAD*, 15:1167–1176, 1996.

[23] G. Tseitin. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logic, Part 2*, pages 115–125, 1968. (Reprinted in: J. Siekmann, G. Wrightson (Ed.), Automation of Reasoning, Vol. 2, Springer, Berlin, 1983, pp. 466-483.).

[24] J. Whittemore, J. Kim, and K. Sakallah. SATIRE: A new incremental satisfiability engine. In *Design Automation Conf.*, pages 542–545, 2001.

[25] C. Zhao and S. Dey. Improving transient error tolerance of digital VLSI circuits using RObustness COmpiler (ROCO). In *Int'l Symp. on Quality Electronic Design*, pages 133–140, 2006.

[26] Q. Zhou and K. Mohanram. Gate sizing to radiation harden combinational logic. *IEEE Trans. on CAD*, 25(1):155–166, 2006.