Computing Bounds for Fault Tolerance using Formal Techniques

André Sülflow Görschwin Fey Stefan Frehse Ulrich Kühne Rolf Drechsler

Fachbereich 3 – Mathematik und Informatik
University of Bremen
28359 Bremen, Germany

{*suelflow,fey,sfrehse,ulrichk,drechsle*}@*informatik.uni-bremen.de*

***Abstract*— While facing continuously shrinking feature sizes, the demand for fault tolerance in digital circuits increases. Numerous approaches to achieve robustness on the design side have been presented. But ensuring that the fault tolerance is really achieved is a tough verification problem.**

Here, we propose a formal model and an effective algorithm to formally prove the robustness of a digital circuit. The proposed model uses a fixed bound in time to cope with the complexity of the 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 the experiments the method was evaluated on circuits with different fault detection mechanisms.

Keywords: Fault Tolerance, SAT, Formal Verification

I. Introduction

According to Moore's Law the number of components per area increases at an exponential rate in integrated chips. Continuously shrinking feature sizes drive this growth. As a consequence the influence of process variations threatens the correct operation of a circuit. Future circuits may have to cope with imperfect components [1]. Another consequence is an increase of externally induced transient faults [2].

Techniques to cope with imperfections and transient faults are available on the production level [3] or the design level [4], [5]. Even first tools to improve fault tolerance are available [6]. But especially at the design level proving fault tolerance is difficult. Simulation or emulation based methods [7] can only cover a small part of the state and input space of a circuit. Often a formal analysis determines the probability of a fault to propagate to a primary output (see e.g. [8]). But the computational effort is extremely high.

Methods commonly applied for formal verification can *prove* fault tolerance of an implementation. The approach of [9] proposes to use symbolic methods for the classical analysis of fault trees. But the faults have to be specified manually. Similarly, [10] and [11] rely on symbolic methods. These approaches analyze fault tolerance with respect to mutations of the implementation. As a result, [11] decides whether an implementation is fault tolerant or not, while [10] also provides data about the state space. Both techniques use the original circuit as a specification. The authors of [12] 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 [13] 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. Compared to [13] our approach is more efficient and fits practical requirements.

The contributions of our work are:

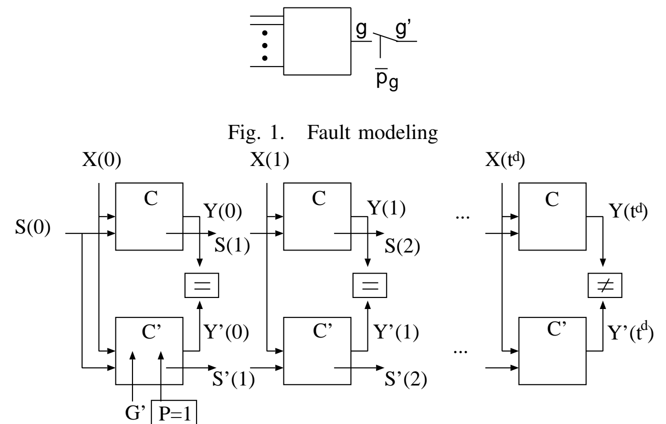

Fig. 1. Fault modeling

Fig. 2. Sequential comparison

- Determination of fault tolerance within a lower and an upper bound
- Trading accuracy for run time using a time bound
- Optimization techniques improve the efficacy

Experiments show the influence of the time bound on the bound for fault tolerance. 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 7.

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

II. 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 III why this is sufficient. A component g is *robust* iff the output behavior of $\mathbb{C}$ cannot change when g is faulty. Let T be the set of robust components in $\mathbb{C}$, then the robustness of $\mathbb{C}$ is given by $|T|/|\mathbb{C}|$.

As suggested in [13] 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 [14]): 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; this 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 faulty and 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 the constraints on the initial states $S(0)$ influence the result. If $S(0)$ is the set of reachable states, the exact value of the robustness is determined, when reaching the maximal sequential depth[1] of the correct circuit and a faulty circuit [13]. Otherwise typically a larger value for the robustness results, since not all faults may have been observed at POs when t^d is too small. Alternatively, if $S(0)$ denotes the reset states, the exact robustness is retrieved when t^d is larger than the maximal sequential depth plus an initialization sequence to reach the start state of the longest loop without repetition. Before reaching the maximal sequential depth, the components are not guaranteed to be robust. Finally, if $S(0)$ is not constrained at all, also non-reachable states are considered. Therefore, too many components might be classified non-robust, i.e. *false negatives* are introduced. As a result the robustness is typically smaller.

In the following we assume that $S(0)$ denotes the set of reachable states unless explicitly stated otherwise.

III. 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.

In practice some action has to be taken after detecting an internal malfunction. Otherwise the effects of multiple faults can accumulate and cause a disastrous failure. Therefore, we assume that a fault detection signal f^d exists. If a malfunction occurs, this is signaled by setting f^d within a given time bound of no more than t^d time steps. Moreover, it is safe to assume that at most one fault occurs within t^d time steps. This allows to retrieve exact bounds for the robustness while restricting the formal analysis to 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) $f^d = 1$ within t^d time frames before or when a faulty value at the POs occurs or
   b) $f^d = 0$, 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 $f^d = 1$.
3) Component g is not classified, if
   $f^d = 0$, a faulty value at the POs does not occur and the state differs from the fault free circuit after t^d time frames.

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

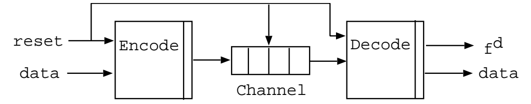

Fig. 3. Transmission system

To guarantee that a circuit is robust, neither non-robust nor non-classified components must remain. For a non-classified component, deviation from the expected output behavior has not been shown. But the deviation in the state may lead to faulty behavior in the future.

The same model also handles circuits that directly correct faults instead of flagging a fault. In this case f^d is assumed constant 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 [15]. 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^d. The timing is summarized like this:

- Encoding and transmission to the channel: 1 time step
- Transmission: 4 time steps (registers in the channel)
- Decoding, writing to the output, setting f^d: 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 injected in the decoding logic are recognized within 1 time frame by setting f^d. Therefore the robustness of this logic is classified.
  Faults injected in the channel change the state compared to the fault free model, but not all data has been decoded, yet. These faults are not recognized. Consequently, these components are not 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 T be the set of components classified as robust; S the set of components classified non-robust and U the set of components not classified, yet. Then, $\mathbb{C} = T \cup S \cup U$. Now, a lower bound R_{lb} and an upper bound R_{ub} for the robustness of the circuit $\mathbb{C}$ are given by:

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

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

Example 2: The bounds that are determined for the hamming model of the previous example are shown in Table I. The bounds are approaching each other, until all components are classified for $t = 6$.

TABLE I
HAMMING MODEL

t	$\|T\|$	$\|S\|$	$\|U\|$	R_{lb} %	R_{ub} %
0	5	2	275	1.77	99.29
1	48	27	207	17.02	90.43
2	73	40	169	25.89	85.82
3	98	52	132	34.75	81.56
4	123	64	95	43.62	77.30
5	148	78	56	52.48	72.34
6	191	91	0	67.73	67.73

IV. ALGORITHM

This section provides an incremental algorithm to transform the calculation of bounds for the robustness into a sequence of SAT instances [16]. A SAT solver [17] is used to determine the solutions. The algorithm is based on the approach introduced in Section II: The original circuit $\mathbb{C}$ and a copy $\mathbb{C}'$ are unrolled for an increasing number of $t \in [0 \ldots t^d]$ time frames. 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. This is valid, because all reachable states are considered as initial states $S(0)$. Also at most one fault predicate may take the value 1. The model supports faults in state elements as well as in combinational logic.

The algorithm in Figure 4 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) [18] (Line 6). The initial states of both copies are forced to be equal (Line 7). Depending on the formulation additional constraints may be inserted to limit the initial states $S(0)$ to reachable states. The number of fault predicates with value 1 is limited to one (Line 8).

Then, the sets of robust (T), non-robust (S), and non-classified (U) components are initialized (Lines 10–12). 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 13–43). As soon as all components are classified, i.e. $U = \emptyset$, the algorithm terminates. Fresh copies of $\mathbb{C}$ are appended to the unrolled circuits for $t > 0$ (Line 15–19). Additional logic is introduced to compare the POs in time frame t (Line 21), where $cmpPOs = 1$ indicates a different value for fault free and faulty copy. Similarly, $cmpFFs$ compares the states (Line 22).

Then the components S' that can be classified as non-robust in time frame t are determined (Lines 24–26). The POs are forced to different values and the fault detection signal f^d is forced to 0 (Line 24). Each satisfying solution provides a component that is non-robust. The newly classified non-robust components S' are returned by the subroutine *extractAllSolutions* shown in Figure 5. The subroutine extracts one non-robust component per satisfying solution (Line 4) and forces the fault predicate of this component to 0 afterward (Line 6).

The main routine in Figure 4 proceeds by removing the constraints on POs and f^d (Line 26). Next, the algorithm determines the remaining non-classified components U' in a similar way (Lines 28–30). In case of non-classified components the constraints $p_g = 0$ are removed (Line 31) before the next iteration for $t+1$ starts.

```
 1  function robustness (ℂ, t^d)
 2    create a copy ℂ'_0 of ℂ
 3    foreach component g ∈ ℂ'_0
 4      replace g by g'[g, p_g];
 5    done
 6    convert to SAT instance;
 7    force init states of ℂ'_0 and ℂ_0 to be equal;
 8    constrain ∑p_g == 1;
 9
10    T := ∅;
11    S := ∅;
12    U := all components g ∈ ℂ'_0;
13    t := 0;
14    while (t ≤ t^d && U ≠ ∅)
15      if (t > 0) then
16        create a copy ℂ'_t of ℂ;
17        create ℂ_t and connect to ℂ'_t;
18      fi
19      connect PIs of ℂ'_t and ℂ_t;
20
21      cmpPOs := at least one pair of POs is
            different;
22      cmpFFs := at least one pair of FFs is
            different;
23
24      add constraint UR := (cmpPOs & !f^d) = 1;
25      S' := extractAllSolutions();
26      remove constraint UR;
27
28      add constraint UC :=
            (!f^d & !cmpPOs & cmpFFs) = 1;
29      U' := extractAllSolutions();
30      remove constraint UC;
31      remove constraint ∀g ∈ U' : p_g = 0;
32
33      T' := U \ (S' ∪ U');
34      add constraint ∀g ∈ T' : p_g = 0;
35
36      T := T ∪ T';
37      U := U';
38      S := S ∪ S';
39
40      t := t + 1;
41      remove cmpPOs;
42      remove cmpFFs;
43    done;
44    return (T, S, U);
45  end function;
```

Fig. 4. Algorithm

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

Fig. 5. Retrieving all solutions

Now, the newly classified set of robust components is available (Line 33). These components do not have to be considered in further iterations and their fault predicates are fixed to 0 (Line 34).

Finally, the sets T, S and 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 36–42).

If non-classified components remain and t^d has not been reached, the next iteration starts. Otherwise the algorithm terminates and returns the three sets T, S and U.

V. OPTIMIZATION TECHNIQUES

This section extends the algorithm by some optimization techniques that reduce the computation time.

A. Partial Reachability

As discussed at the end of Section II, the initial states $S(0)$ have to be constraint to reachable states to retrieve exact results. Not adding these constraints may reduce the run time, but introduce false negatives, i.e. robust components may be classified as non-robust or cannot be classified. Therefore the resulting bounds for the robustness may be lower than the actual values. *False positives* cannot occur, as long as no reachable states are excluded from the set of initial states.

We evaluate this trade-off by applying a partial reachability analysis. This partial reachability analysis may be automated [19] or alternatively constraints for the initial states may be given manually as an invariant in some formal property language. For some designs providing the invariant manually seems to be reasonable, e.g. when the faulty state is identified easily in the state of the design. Consider *Triple Modular Redundancy* (TMR) – in the fault free state the three copies are in the same state. This can be formulated easily as an invariant. Both methods are studied in the experiments.

B. Fanout Free Regions

So far fault predicates have been associated to all components in a circuit. By reducing the number of fault predicates, the search space shrinks and consequently the run time decreases. Here we exploit *Fanout Free Regions* (FFRs) in a similar way as proposed for SAT based diagnosis [14].

Consider an FFR. Any fault occurring at an internal gate of the FFR must be propagated along the single output of the FFR to be observed at the POs or to manifest in the state bits. Therefore the algorithm proceeds in two steps. First, fault predicates are only introduced at outputs of FFRs. If such an output is classified as robust, all components within the corresponding FFR can be safely classified as robust as well. Otherwise, the internal components of the FFR are considered in a second step. Fault predicates are associated to all of the internal components of non-classified or non-robust FFRs.

The same argument applies to structural dominators in general. However, our current implementation only considers FFRs.

C. Clause Reuse

The reuse of learned information has been shown to improve the performance in computer aided design, for example in formal verification [20] or automatic test pattern generation [21], [22], [23]. In particular, SAT solvers store learned information in terms of conflict clauses that are easily accessible. Moreover, the conflict graph registers the *reason* for a conflict, i.e. the clauses that are necessary to imply learned information. This information can efficiently be exploited at a higher level [20], [24], [25]. If the clauses, which establish the *reason* of a conflict, remain in the problem instance, the corresponding conflict clauses can be reused as well.

The algorithm proposed in Section IV proceeds incrementally. During each iteration new copies of the circuits are added for time frame t. The copies corresponding to earlier time frames are not modified. Thus, a large portion of the problem instance remains unchanged, and learned information is kept. Only the logic to compare POs and states and the blocking clauses for non-classified components are removed after each iteration.

On an implementation level for example the interface of the SAT solver Chaff [26] provides groups of clauses. For each conflict clause the groups that were responsible for the conflict are recorded. This is less accurate than tracking individual clauses as the reason, but more efficient with respect to memory and run time. All clauses forming the circuits $\mathbb{C}$ and $\mathbb{C}'$ are contained in one group, while the logic for comparison and clauses to block non-classified components are summarized in another group.

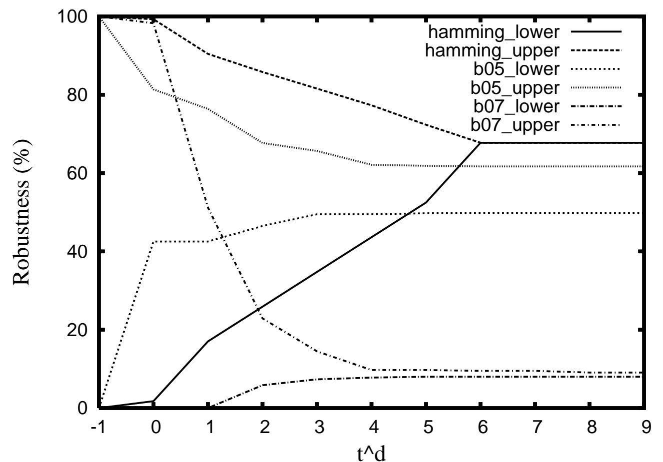

Fig. 6. Robustness vs. t^d

VI. Experimental Results

The experimental results are provided in the following. First, we analyze the influence of t^d on the robustness. Next we study the value of t^d necessary to classify all components and the influence of using information about reachable states. Finally, we consider the run time and evaluate the optimization techniques presented in Section V-B and Section V-C that do not bias the exactness but only improve run time.

The benchmark suite contains different types of sequential 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. As in the original benchmark suite they are denoted by b01–b10. Using the circuits b01–b07, fault tolerant TMR circuits were created. The circuit was replicated three times and a majority voter was added as combinational logic to drive the POs. As a result 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^d. While the states of the three instances are identical, no fault is detected, i.e. $f^d = 0$. Additionally, the Hamming model introduced in Example 1, provides fault detection.

In all cases gates were considered as components.

All experiments were carried out on an AMD Athlon 3700+ (2.2GHz, 1GB RAM, Linux). The SAT solver Chaff [26] was used for the implementation, since this solver provides a good interface to manage groups of clauses.

A. Influence of t^d

Figure 6 exemplary visualizes the bounds retrieved for three circuits. The initial states were limited to reachable states. Note, the initial bounds are marked with $t^d = -1$.

As already discussed in Section III, the exact value for the robustness of the Hamming model is retrieved at $t^d = 6$

TABLE II

MAXIMUM $t^d = 2$

circuit	$\lvert\mathbb{C}\rvert$	*#FF*	t^d	$\lvert U\rvert$	R_{lb} %	R_{ub} %
b01	62	5	2	0	1.61	1.61
b02	33	4	2	0	3.03	3.03
b03	195	30	2	134	0.00	68.72
b04	821	66	2	88	0.00	10.72
b05	1198	34	1	0	5.59	5.59
b06	71	9	1	0	0.00	0.00
b07	512	49	2	72	0.56	14.65
b08	223	21	2	35	0.00	15.70
b09	197	28	2	45	0.00	22.84
b10	260	17	2	32	0.00	12.31
b01-tmr	210	15	2	3	1.90	3.33
b02-tmr	111	12	2	3	1.80	4.50
b06-tmr	273	27	2	0	3.66	3.66
hamming	282	7	2	169	5.67	65.60

TABLE III

INFLUENCE OF CONSTRAINTS ON INITIAL STATES

	—all—			—reachable—		
circuit	t^d	R_{lb} %	R_{ub} %	t^d	R_{lb} %	R_{ub} %
b01	2	1.61	1.61	2	1.61	1.61
b02	2	3.03	3.03	2	3.03	3.03
b03	8	0.00	0.00	8	5.13	5.13
b05	1	5.59	5.59	[a]26	51.75	56.51
b06	1	0.00	0.00	1	0.00	0.00
b07	23	0.98	0.98	[b]36	7.23	7.42
b08	4	2.24	2.24	4	2.24	2.24
b09	8	0.00	0.00	8	0.51	0.51
b10	4	6.54	6.54	4	7.69	7.69
hamming	6	30.50	30.50	6	67.73	67.73

[a]Memory out: $|U| = 57$; [b]Memory out: $|U| = 1$

where lower and upper bound converge. In case of b05 and b07 the bounds approach each other quite rapidly at the beginning, but do not meet within 10 time frames. While 10% of the components cannot be classified for b05, only 1% remains 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.

Table II summarizes the results for $t^d \leq 2$ for some benchmark circuits. The table provides the name of the circuit (*circuit*), the number of components ($|\mathbb{C}|$), the number of state bits (*#FFs*), the number of iterations t before the algorithm terminated, the number of components not classified ($|U|$), lower bound R_{lb} and upper bound R_{ub}.

For two of the circuits only one iteration $t^d = 1$ was needed to classify all components. Lower and upper bound have the same value in these cases. Large differences for the bounds can be seen for the circuits b03, b09 and b12 – a large portion of components was not classified for $t^d = 2$ in these cases. By increasing the value of t^d, the resolution can be increased.

B. Reachability Analysis

The influence of using constraints on the initial states is studied. The reachable states are computed by two approaches: (1) building a *Binary Decision Diagram* (BDD)[27] and (2) manually providing an invariant for the TMR circuits (see Section V). The non-reachable states are avoided by extracting all BDD paths to zero, and blocking them in the SAT instance.

Table III presents the experimental results for circuits without TMR logic. No additional constraints are added on the left (*all*), whereas non-reachable states are computed and blocked using BDDs on the right (*reachable*).

TABLE IV

INFLUENCE OF REACHABLE STATES ($t^d = 0$)

	—b06-tmr—		—b05—	
%	R_{lb} %	R_{ub} %	R_{lb} %	R_{ub} %
100	30.77	97.80	41.32	81.22
80	21.98	89.01	20.38	59.85
60	17.58	84.62	6.01	47.75
40	17.58	84.62	5.93	47.75
20	14.65	81.68	5.93	47.75
0	3.66	70.70	5.59	47.41

For four circuits the results of both approaches are identical. For the rest of the circuits using stronger constraints leads to higher robustness, false negatives are avoided. In case of b05 and b07 a memory out occurred, thus the algorithm terminates leaving some non-classified components. However, a very accurate estimation of the robustness in lower and upper bounds results.

The trade-off between partial reachability analysis and exactness is studied for the circuits b06-tmr and b05. Table IV provides details. The percentage of randomly selected paths from the BDD is given in column %. Exact reachability analysis is performed if all paths are selected (100%). The results show that a partial reachability analysis may helps to improve the accuracy of the results when a full analysis is too expensive.

Results for TMR circuits are shown in Table V. Yet another approach is even more effective for this type of fault tolerance: A manually provided invariant matches the initial states of the three instances and forces them to be identical (*manual invariant*). The circuits with the suffix flt also have fault detection logic. With a non-exact analysis (*all*) the computed bounds are significant lower than the exact (*reachable*, *manual invariant*) values. Exactness is not biased using the manual invariant instead of a BDD approach. Both, i.e. full reachability and the manual invariant, yield the same bounds for the robustness.

Exact reachability analysis is known to be expensive. Within 1800 CPU seconds the BDD could be computed for the circuits b01, b02 and b06, only. The manual invariant overcomes this limitation. No extra computation has to be performed. Moreover, the manual invariant is more compact, e.g. for b06-tmr only 36 clauses are necessary compared to 62 BDD blocking clauses. The number of literals in each clause is greater or equal two in both approaches. As a result for circuits with a large number of states, less clauses may lead to further run time improvements. For b01, b02 and b06 no significant difference in the run time was observed.

Finally, for the practically very relevant case of circuits with fault detection the algorithm is very efficient. Each faulty component can be observed directly in the first time frame, therefore $t^d = 0$ is sufficient to classify all components and the robustness is in all cases over 97%. For other fault detection mechanisms that have a similarly low latency, similar efficiency is expected.

C. Run Time Improvements

The optimization techniques to improve the run time are studied in this Section. Table VI shows the results – only run times in CPU seconds are reported. Column *basic* refers to the plain algorithm that neither reuses clauses nor exploits FFRs. For column I learned information was reused, for column F structural knowledge about FFRs was applied. Finally, both techniques were combined for column $I+F$.

Most run time improvements are obtained by applying incremental SAT techniques, but also considering FFRs leads

TABLE V

INFLUENCE OF CONSTRAINTS ON INITIAL STATES ON TMR CIRCUITS

	—all—				—reachable—				—manual invariant—			
circuit	t^d	R_{lb} %	R_{ub} %	$\|U\|$	t^d	R_{lb} %	R_{ub} %	$\|U\|$	t^d	R_{lb} %	R_{ub} %	$\|U\|$
b01-tmr	2	3.33	3.33	-	[a]18	34.76	99.05	135	[a]18	34.76	99.05	135
b02-tmr	2	4.50	4.50	-	[a]49	23.42	99.10	84	[a]49	23.42	99.10	84
b03-tmr	8	1.26	1.26	-	[b] -	-	-	-	12	14.53	99.47	57
b04-tmr	2	0.62	10.90	264	[b] -	-	-	-	[a]2	62.19	99.69	963
b05-tmr	2	6.96	6.96	-	[b] -	-	-	-	[a]1	59.68	99.08	1545
b06-tmr	2	3.66	3.66	-	[a]22	60.44	97.80	102	[a]22	60.44	97.80	102
b07-tmr	2	1.55	14.96	216	[b] -	-	-	-	[a]4	5.46	99.50	1515
b01-tmrflt	0	4.69	4.69	-	0	99.06	99.06	-	0	99.06	99.06	-
b02-tmrflt	0	4.92	4.92	-	0	99.18	99.18	-	0	99.18	99.18	-
b03-tmrflt	0	9.31	9.31	-	[b] -	-	-	-	0	99.37	99.37	-
b04-tmrflt	0	2.40	2.40	-	[b] -	-	-	-	0	99.69	99.69	-
b05-tmrflt	0	7.04	7.04	-	[b] -	-	-	-	0	99.08	99.08	-
b06-tmrflt	0	46.13	46.13	-	0	97.89	97.89	-	0	97.89	97.89	-
b07-tmrflt	0	4.93	4.93	-	[b] -	-	-	-	0	99.50	99.50	-

[a]Memory out; [b]Time out, i.e. reachable states could not be computed within 1800 CPU seconds

TABLE VI

RUN TIME OPTIMIZATION

circuit	*basic*	*I*	*F*	$I+F$
b01	1.88	0.43	1.66	0.40
b02	0.65	0.13	0.64	0.12
b05	512.41	91.32	343.61	73.83
b01-tmr	17.98	5.04	15.89	4.12
b02-tmr	7.77	1.44	5.63	1.24
b06-tmr	19.06	5.31	16.01	4.11
hamming	73.88	14.79	70.46	13.86

to a faster analysis. The combination of both techniques speed-ups the analysis by a factor of up to 7.

VII. CONCLUSIONS

We presented an approach to formally prove the robustness of a circuit. The algorithm works on a bounded number of time steps and consequently determines a lower bound and an upper bound on the robustness. An incremental algorithm and additional optimization techniques are provided for an effective implementation. 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 mechanisms accurate values are determined efficiently.

REFERENCES

[1] S. Borkar, "Designing reliable systems from unreliable components: The challenges of transistor variability and degradation," *IEEE Micro*, vol. 25, no. 6, pp. 10–16, 2005.

[2] 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*, 2002, pp. 389–398.

[3] Q. Zhou and K. Mohanram, "Gate sizing to radiation harden combinational logic," *IEEE Trans. on CAD*, vol. 25, no. 1, pp. 155–166, 2006.

[4] 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. [Online]. Available: http://www.gigascale.org/pubs/426.html

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

[6] 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*, 2006, pp. 133–140.

[7] 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*, vol. 18, no. 3, pp. 261–271, 2002.

[8] M. Miskov-Zivanov and D. Marculescu, "Circuit reliability analysis using symbolic techniques," *IEEE Trans. on CAD*, vol. 25, no. 12, pp. 2638–2649, 2006.

[9] M. Bozzano, A. Cimatti, and F. Tapparo, "Symbolic fault tree analysis for reactive systems," in *Automated Technology for Verification and Analysis*, ser. LNCS, vol. 4762, 2007, pp. 162–176.

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

[11] 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*, 2006, pp. 176–181.

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

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

[14] A. Smith, A. Veneris, M. Fahim Ali, and A.Viglas, "Fault diagnosis and logic debugging using boolean satisfiability," *IEEE Trans. on CAD*, vol. 24, no. 10, pp. 1606–1621, 2005.

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

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

[17] N. Eén and N. Sörensson, "An extensible SAT solver," in *SAT 2003*, ser. LNCS, vol. 2919, 2004, pp. 502–518.

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

[19] A. Gupta, Z. Yang, P. Ashar, and A. Gupta, "SAT-based image computation with application in reachability analysis," in *Int'l Conf. on Formal Methods in CAD*, ser. LNCS, vol. 1954, 2000, pp. 354–371.

[20] O. Shtrichman, "Pruning techniques for the SAT-based bounded model checking problem," in *CHARME*, ser. LNCS, vol. 2144, 2001, pp. 58–70.

[21] W. Kunz and D. Pradhan, "Recursive learning: A new implication technique for efficient solutions of CAD problems: Test, verification and optimization," *IEEE Trans. on CAD*, vol. 13, no. 9, pp. 1143–1158, 1994.

[22] K. Chandrasekar and M. S. Hsiao, "Integration of learning techniques into incremental satisfiability for efficient path-delay fault test generation," in *Design, Automation and Test in Europe*, 2005, pp. 1002–1007.

[23] G. Fey, T. Warode, and R. Drechsler, "Using structural learning techniques in SAT-based ATPG," in *VLSI Design Conf.*, 2007, pp. 69–74.

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

[25] O. Shacham and K. Yorav, "On-the-fly resolve trace minimization," in *Design Automation Conf.*, 2007, pp. 594–599.

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

[27] R. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. on Comp.*, vol. 35, no. 8, pp. 677–691, 1986.