A Hybrid Algorithm to Conservatively Check the Robustness of Circuits

***Abstract*—With decreasing size of transistors, the impact of transient faults as well as the local and global variability of transistors increases, affecting system functions and performances. Formal verification may be used to prove that a circuit is robust against transient and parametric faults. However, a model including timing information combined with extracted electrical parameters is typically too large in practice. We propose a hybrid algorithm based on Boolean reasoning and simulation that decomposes the problem while still achieving completeness in verification. Our experiments show an average speed up of 748 compared to previous work on ISCAS-85 circuits and fault tolerant modifications.**

I. Introduction

Due to the continuing decrease of transistors' size systems become more powerful and require less energy. However, smaller transistors are more vulnerable to transient effects like *Single Event Transients* (SETs). To prevent an SET from causing a fault in the system, different methods can be used to provide fault tolerance. A circuit that is meant to be fault tolerant still needs to be checked if the adjustments that should provide fault tolerance really provide robustness against SETs.

An easy way to get a basic idea of the robustness of a circuit are simulation and testing. But these cannot prove the absence of possible errors except for very small circuits. On the other hand, a SAT formula that models the circuit in high detail can become very complex, especially if the signal that contains the SET reconverges.

We present a hybrid approach that combines simulation and formal verification to achieve scalability while keeping a detailed technology model. We describe a circuit in high detail dependent on the used technology. The resulting circuit is partitioned into a front and a back partition. Different partitionings are possible. For our work, we put all gates that are affected by reconvergence of the SET in the back partition. We can easily analyze the front partition by using a SAT solver. Afterwards, we simulate detected possible counterexamples on the whole circuit, generalize the counterexamples, and modify the SAT formula until a robustness can be decided.

Our algorithm considers logical, timing, and electrical masking as well as variability in the gates.

Different work about the analysis of SETs exists and focuses on different aspects. Several techniques focus only on logical masking [3, 4, 5, 10, 12] which leads to quick decisions as the abstraction levels like register or gate level can easily be decided. Other work emphasizes the effects of timing masking under delay faults [1, 9]. In [8] only electrical masking is considered. Logical, timing, and electrical masking as well as variability is considered in [11]. They presented the accuracy by comparing their model to Spice simulation. We use a similar but more detailed model that allows a preciser representation of delays. Miskov-Zivanov [6] uses a mathematical model with a high grade of detail that considers logical, timing, and electrical masking to compute the probability of errors. Finally, simulation-based approaches like Spice simulation or [7] can only check the effects of faults under a certain set of input assignments and specific parameters for the fault.

In summary, the contributions of this paper are an algorithm that

- is compositional as it partitions the circuit into two partitions to prevent reconvergence in the front partition,
- is hybrid and uses SAT solving to check the front partition and uses simulation to verify detected counterexamples on the complete circuit,
- considers logical, timing, and electrical masking,
- describes the gates in great detail, specific to the used technology and considering variability.

In the following Section II, we describe the models used for our algorithms. The algorithm itself is explained in Section III. Section IV discusses why our approach has a good performance. In Section V, we compare our algorithm to [11]. Finally, Section VI concludes the paper.

II. Preliminaries

Three-valued logic extends Boolean logic by a value X, meaning that it is not known if that variable is 1 or 0. Operations on variables are extended accordingly, e.g., $1 \wedge X = X$ and $0 \wedge X = 0$.

A *circuit* C consists of a set $C.G$ of gates, a set $C.I$ of input signals, and a set $C.O \subseteq C.G$ of outputs. To determine the connections between the gates, the functions $C.\mathsf{predecessors} : G \to 2^{I \cup G}$ and $C.\mathsf{successors} : (G \cup I) \to 2^G$ return the predecessors or successors of a gate or input signal.

The set $C.SO \subseteq C.O$ describes safe outputs and contains all outputs that are secured, i.e., can correct the effects of an SET in this output, e.g., using Razor [2].

An SET s that affects C is modeled by a number of parameters. The gate $s.g$ describes the location where the SET strikes. The SET begins at the time $s.b \in \mathbb{R}$ and ends at $s.e \in \mathbb{R}$. During an offset time at the beginning $s.o_b \in \mathbb{R}$ and at the end $s.o_e \in \mathbb{R}$, the signal becomes unknown. In between the offsets, the signal is inverted.

Each gate g is associated with an operation $g.\mathsf{op} : \{0,1,X\}^n \to \{0,1,X\}$ that describes the output of the gate under given input values. The function $g.\mathsf{delay} : \{0,1,X\}^n \times \{0,1,X\}^n \to \mathbb{R}$ returns the delay of the gate when the input changes, depending on the old and new input values. For our algorithm, we only need to consider changes in the inputs where a single input changes to X or was previously X due to the form of the SET. To remain conservative, we take the minimal possible delay with the given input values when a signal changes to X and the maximal delay when it previously was X. Due to our conservative handling of the delays, the time where a signal is X will become longer in any successor unless the SET is logically masked. In this case, we do not need delays as the signal is constant. The values $g.d_{min}$ and $g.d_{max}$ denote the minimal and maximal delay in $g.delay$, respectively. The output values of the gate over time are described by a vector $g.wave \in V^*$ where the set V contains

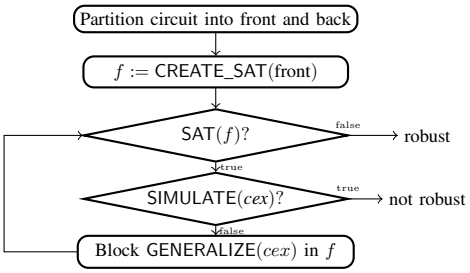

Fig. 1: Sketch of the algorithm

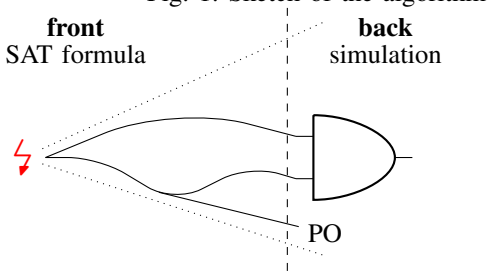

Fig. 2: Partition in front and back

three valued variables. The vector $g.switch \in \mathbb{R}^{g.wave.\,\mathsf{size}()}$ contains the times when the output of the gate changes to the next variable. The first variable in $g.wave$ describes the signal before the SET affects g and the last variable describes the signal after the SET has passed g. The variables in between describe value changes at the output of g due to the SET. The time $g.switch_i$ defines when the signal changes from $g.wave_i$ to $wave_{i+1}$. For better readability, we combine the vectors $g.wave = (v_1, v_2, \ldots, v_n)$ and $g.switch = (t_1, t_2, \ldots, t_{n-1})$ to a single vector $g.signal = (v_1, t_1, v_2, t_2, \ldots, v_{n-1}, t_{n-1}, v_n)$. For example, a constant signal would only require a single variable and no switch times and a gate g with $g.wave = (v_1, v_2)$ and $g.switch = (5)$ would change its output at time 5 from v_1 to v_2. The according vector $g.signal$ would be $(v_1, 5, v_2)$.

III. Checking for Robustness

Our compositional computation verifies if a circuit C is robust against an SET s. If C is not robust, a counterexample to the robustness is returned. A counterexample contains an input assignment that leads to faulty behavior of C under s. We assume the inputs to be constant and analyze the effects of the SET to a stabilized circuit.

The algorithm is sketched in Figure 1. In the first step, the circuit is partitioned into a front and back partition. The partitioning is sketched in Figure 2. Afterwards, a SAT formula f is created to describe the front partition. If f is not satisfiable, the circuit is guaranteed to be robust against s. Otherwise, the detected assignment is a counterexample *cex* against robustness of the front partition. The assignment *cex* is simulated on the whole circuit. If the primary outputs of C are affected by s under the assignment of *cex*, the circuit is not robust and we call *cex* a *real counterexample*. Otherwise, *cex* is a *spurious counterexample*, that is generalized and blocked in f. Afterwards, we continue to check if the modified f is satisfiable until we can make a decision if the circuit is robust.

In the following sections, we will explain the proposed algorithm in detail. We start with the top level algorithm in Section III-A decrease the level of abstraction with each section.

Algorithm 1: ROBUST_CHECK

input : a circuit C and an SET s
output: an assignment that leads to faulty behavior in C under s or "robust" if no such assignment exists

```
1  G_found := ∅
2  G_back := ∅
3  Q_search :=< s.g >
4  while Q_search ≠<> do
5  |  g := Q_search.pop()
6  |  if g ∈ G_found then
7  |  |  G_back := G_back ∪ {g}
8  |  else
9  |  |  G_found := G_found ∪ {g}
10 |  end
11 |  foreach g' ∈ C.successors(g) do  Q_search.push(g')
12 end
13 O_front := {g ∈ G \ G_back|
       (C.successors(g) ∩ G_back) ≠ ∅ ∨ g ∈ C.O}
14 f := CREATE_SAT(C, G \ G_back, O_front, s)
15 while SAT(f) do
16 |  a := getAssignment(f)
17 |  a_in := a_in : C.I → {0, 1, X} with a_in(i) = a(i)
18 |  (a_sim, real) := SIMULATE(a_in, C, s, C.G)
19 |  if real then
20 |  |  return a
21 |  else
22 |  |  foreach o ∈ {g ∈ C.O \ C.SO|f.po-faulty_g(a)} do
23 |  |  |  f.po-faulty_o.addClause(
   |  |  |      ¬ GENERALIZE(o, a_sim, a, C, s))
24 |  |  end
25 |  end
26 end
27 return "robust"
```

A. The algorithm ROBUST_CHECK

Algorithm 1 implements the sketch of Figure 1. In the beginning we partition the circuit into front and back partition. For the used partitioning, we want all gates in which the SET reconverges to be in the back partition. This partitioning leads to easy SAT formulas that can quickly be solved, To determine the gates in the back partition we use an approach similar to breadth search towards the outputs starting in the gate $s.g$ in lines 1 – 12. We also prepare the set O_{front} that contains all primary outputs as well as all gates in the front partition that have successors in G_{back} in line 13.

After the circuit is partitioned into the front and back partition, we create a SAT formula f to model the front partition. This SAT formula is satisfiable if the SET can reach the back partition and there is a possible fault in the circuit. Line 14 calls the respective algorithm CREATE_SAT.

While f is satisfiable, potential counterexamples exist which show the SET reaches the back partition. If f is satisfiable, we get an assignment a in line 16. The input assignment a_{in} of a is simulated on the complete circuit by calling the algorithm SIMULATE in line 18 to check if the potential counterexample is real. The simulation is very similar to the generation of the SAT formula, but simulates delays for the given input values accurately using the delay maps of gates. If the counterexample is real, it proves that the circuit is not

robust and the corresponding assignment is returned in line 20.

When the counterexample is spurious and the SAT formula assumes for a non-safe primary output $o \in C.O\ C.SO$ that o is affected by the SET, i.e., $f.po\text{-}faulty_o$ is true, we determine a minimal assignment that prevents the SET from reaching o by calling GENERALIZE. We add the generalized assignment to $f.po\text{-}faulty_o$ which is meant to be true if the SET could reach o. Since the assignment prevents o from being affected by the SET, we can modify $f.gate\text{-}faulty_o$ accordingly in line 23.

The loop from lines 15 – 26 further modifies f until either a real counterexample is found or f is not satisfiable any more. In the later case, the loop terminates and the algorithm returns that C is robust in line 27.

B. The algorithm CREATE_SAT

The algorithm to create the SAT formula that describes the front partition starts by initializing the SAT formula f with true in line 1 of Algorithm 2. We use a queue to iteratively compute the waveform and switch times for each gate. The queue Q is initialized with all successors of the primary inputs in lines 2 – 5. While Q is not empty, we pop the front element g of the queue. If g still has predecessors whose waveform is not defined yet, g is pushed to the back of Q as seen in lines 8 – 9.

Otherwise, we determine the initial signal $g.signal$ of g in line 11 by calling COMPUTE_WAVE. The signal is further modified by considering variable delays in line 12 by calling VARIABLE_DELAY, adding the SET in case $g = s.g$ in line 13 by calling ADD_SET, and finally considering electrical masking in line 14 by calling ELECTRICAL_MASKING.

After $g.signal$ computed, we add all successors of g to the queue that have not been added yet and are part of the front partition. This is done in lines 15 – 17.

After the loop is done, we require all inputs to be different from X in lines 20 – 22. Thus, we have only boolean inputs as all nominal behavior of the circuit is boolean as well and the value X can only be assigned due to the SET.

In a next step, we introduce a subformula $f.fo\text{-}faulty_o$ of f for each non-safe front output $o \in O_{front} \setminus C.SO$. This subformula evaluates to true or X if o is affected by the SET, i.e., faulty. The output o is faulty iff the signal of the o is not constant. These subformulas are generated for each output in lines 23 – 26.

We initialize further subformulas $f.po\text{-}faulty_o$ for each non-safe primary output $o \in C.O \setminus C.SO$. The formula $f.po\text{-}faulty_o$ estimates conservatively if o is affected by the SET. Initially, the formula is true iff at least one front output in the fanin of o is faulty. The according loop is in the lines 27 – 30.

The final subformula *overall-faulty* introduced in line 31 is true iff at least one front output is faulty. The variable *overall-faulty* describes that there is a potential error in the circuit. As we require counterexamples that describe such faults, we require *overall-faulty* to be different from 0 by adding the according clause to f in line 32.

The resulting SAT formula describes the behavior of the front partition and is only satisfiable if there is a fault in the front partition that could reach a primary output without considering the constraints induced by the reconvergencies only modeled in the back partition.

Algorithm 2: CREATE_SAT

input : a circuit C, a set G_{front} of gates without potential reconvergence of the SET, a set $O_{front} \subseteq G_{front}$ of output gates of the front part, and an SET s
output: a SAT formula that is satisfiable iff there can be a fault in the front output in our model of C under the SET s

```
1  f := true
2  Q := <>
3  foreach g ∈ ⋃_{i∈C.I} C.successors(i) do
4  |  Q.push(g)
5  end
6  while Q ≠ <> do
7  |  g := Q.pop()
8  |  if ∃p ∈ C.predecessors(g) : p.wave = ⊥ then
9  |  |  Q.push(g)
10 |  else
11 |  |  COMPUTE_WAVE(g, C, f)
12 |  |  VARIABLE_DELAY(g)
13 |  |  if g = s.g then ADD_SET(g, s, f)
14 |  |  ELECTRICAL_MASKING(g)
15 |  |  foreach suc ∈ C.successors(g) do
16 |  |  |  if suc.wave = ⊥ ∧ ¬Q.contains(suc)
   |  |  |        ∧ suc ∈ G_front then Q.push(suc)
17 |  |  end
18 |  end
19 end
20 foreach i ∈ C.I do
21 |  f.addClause(i ≠ X)
22 end
23 foreach fo ∈ O_front \ C.SO do
24 |  w = fo.wave
25 |  fo.fo-faulty_fo := new SAT subformula of f:
   |        ¬(w_0 = w_1 ∧ ··· ∧ w_{n-1} = w_n)
26 end
27 foreach po ∈ C.O \ C.SO do
28 |  w = po.wave
29 |  po.po-faulty_po := new SAT subformula of f:
   |        ⋁_{fo∈fanin(po)} fo.fo-faulty_fo
30 end
31 overall-faulty := ⋁_{o∈C.O\C.SO} o.po-faulty_o
32 f.addClause(overall-faulty ≠ 0)
33 return f
```

C. The algorithms COMPUTE_WAVE, VARIABLE_DELAY, ADD_SET, and ELECTRICAL_MASKING

The algorithm COMPUTE_WAVE determines the waveform and switch times of a gate g depending on the inputs and gates' operation and is shown in Algorithm 3. We define an index for each predecessor of g that refers to a position in the waveform of the predecessor. The current indices refer to the current inputs and will increase while the algorithm moves forward in time. We also introduce the current inputs *current-in* that depend on the current indexes. As a final preparation, we define the first variable of $g.wave$. The preparations are done in lines 1 – 7.

While there is still an index that refers to an existing switch time, we determine the minimal switch time m and

Algorithm 3: COMPUTE_WAVE

input : a gate $g \in C.G$, a circuit C, and a SAT formula f that is currently constructed

1 $n := |C.\,\mathsf{predecessors}(g)|$
2 $\{p_0, \ldots, p_n\} := C.\,\mathsf{predecessors}(g)$
3 $(i_0, \ldots, i_n) := (0, \ldots, 0)$
4 *current-in* $= (c_0, \ldots, c_n) := (p_0.\mathit{wave}_0, \ldots, p_n.\mathit{wave}_0)$
5 $w_g := (w_0)$
6 $s_g := ()$
7 $f.\,\mathsf{addClause}(w_0 = g.\,\mathsf{op}(\textit{current-in}))$
8 **while** $\exists j \in \{0, \ldots, n\} : i_j < p_j.\mathit{switch}.\,\mathsf{size}()$ **do**
9 $\quad m := \mathsf{min}(p_j.\mathit{switch}_{i_j} | j \in \{0, \ldots, n\})$
10 $\quad j := \mathsf{indexOf}(m)$
11 $\quad$ *current-in* $:= (c_0, \ldots c_{j-1}, p_j.\mathit{wave}_{i_j+1}, c_{j+1}, \ldots, c_n)$
12 $\quad i_j := i_j + 1$
13 $\quad s_g := s_g \circ (m + g.d_{\mathit{min}})$
14 $\quad v :=$ new Variable
15 $\quad w_g := w_g \circ (v)$
16 $\quad f.\,\mathsf{addClause}(v = g.\,\mathsf{op}(\textit{current-in}))$
17 **end**
18 $g.\mathit{wave} := w_g$
19 $g.\mathit{switch} := s_g$

Algorithm 4: VARIABLE_DELAY

input : a gate $g \in C.G$

1 $(t_1, \ldots, t_n) := g.\mathit{switch}$
2 **for** $j := 2, 4, \ldots, n$ **do**
3 $\quad t_j := t_j + (g.d_{\mathit{max}} - g.d_{\mathit{min}})$
4 **end**
5 $g.\mathit{switch} := (t_1, \ldots, t_n)$

the corresponding index j in the lines 9 and 10. We adjust the current inputs by using the next variable of the j-th input in line 11 and increase the index i_j by one in line 12. We add the next switch time which is the determined minimal switch time and the added minimal delay of g, i.e., $m + g.d_{\mathit{min}}$ in line 13 and a new variable v to the waveform which needs to be equal to the output of g with the changed inputs in the lines 14 – 16.

When considering the variable delay of g in the algorithm VARIABLE_DELAY shown in Algorithm 4, we exploit that there is no reconvergence in g as g is in the front partition. This leaves three cases for the waveform:

1) The output is constant
2) The output has the form of the SET: $vX\neg vXv$
3) The output has the form of the SET, but the middle part has been removed: vXv

As the variables do not have assigned values at this time, it is impossible to decide which case will hold, however we can do the following modification in all cases. Since we will only modify the switch times in this algorithm, the semantics of the output will not change if it is constant. Otherwise, we want to hold the output at X as long as possible within the limits of the delays to remain conservative. Since in a non-constant output every second variable is X, we set the switch times at those locations to the maximum delay instead of the minimum delay. Therefore we use the minimal delay when we change the output to X and use the maximal delay when we change back to another value.

Algorithm 5: ADD_SET

input : a gate $g \in C.G$, an SET s, and a SAT formula f

1 //Signal of g is constant before SET is induced
2 $v := g.\mathit{wave}_0$
3 $v_X, v_N :=$ new Variable
4 $g.\mathit{wave} := (v, v_X, v_N, v_X, v)$
5 $g.\mathit{switch} := (s.b, s.b + s.o_b, s.e - s.o_e, s.e)$
6 $f.\,\mathsf{addClause}(v_X = X \wedge v_N = \neg v)$

Algorithm 6: ELECTRICAL_MASKING

input : a gate $g \in C.G$

1 $t := g.d_{\mathit{min}}/2$
2 **for** $j := 0, \ldots, g.\mathit{switch}.\,\mathsf{size}() - 2$ **do**
3 $\quad$ **for** $k := j + 1, \ldots, g.\mathit{switch}.\,\mathsf{size}() - 1$ **do**
4 $\quad\quad$ //Equal checks between variables check if the variables are equal, not their values
5 $\quad\quad$ **if** $g.\mathit{switch}_k - g.\mathit{switch}_j \leq t \wedge$ $g.\mathit{wave}_j = g.\mathit{wave}_{k+1}$ **then**
6 $\quad\quad\quad g.\mathit{wave} := (g.\mathit{wave}_0, \ldots, g.\mathit{wave}_{j-1}, g.\mathit{wave}_{k+1}, \ldots, g.\mathit{wave}_{g.\mathit{wave}.\,\mathsf{size}()-1})$
7 $\quad\quad\quad g.\mathit{switch} := (g.\mathit{switch}_0, \ldots, g.\mathit{switch}_{j-1}, g.\mathit{switch}_{k+1}, \ldots, g.\mathit{switch}_{g.\mathit{switch}.\,\mathsf{size}()-1})$
8 $\quad\quad$ **end**
9 $\quad$ **end**
10 **end**

If we induce the SET s into a gate g, we use the algorithm ADD_SET shown in Algorithm 5. The waveform of g needs to be constant and only contain one variable as only the SET leads to a change in the output of a gate. In line 4 we generate a new waveform for g which corresponds to the SET and in line 5 we set the switch times according to the parameters of the SET.

The final modification to the waveform is done in the algorithm ELECTRICAL_MASKING in Algorithm 6. The electrical properties of a gate remove short glitches, i.e., changes of the value that last for a short time only. A common abbreviation for this time is half the delay of gate. To remain conservative, we use half the minimal delay and set the variable t for the threshold accordingly in line 1. In the entwined loops from line 2 – 10, we check if two equal variables have a distance of t or less between them. If so, the waveform and the according switch times between the two variables describe a glitch that is removed. So we adjust the waveform and switch times of g accordingly in lines 6 and 7.

A small example for these algorithms is shown in Figure 3.

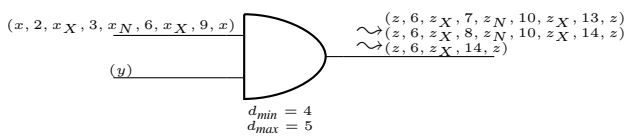

Fig. 3: Example for generation of waveform and switch times of a gate considering variable delays and electrical masking depending on gate

Algorithm 7: GENERALIZE

input : a gate o that is a primary output, an assignment $a_{sim} : (I \cup C.G) \to \{0, 1, X\}^*$ of inputs and gate signals to values, a partial assignment $a_{SAT}(I \cup C.G) \to \{0, 1, X\}^*$, a circuit C and an SET s

output: a SAT formula f, when f is true, an eventual fault in o cannot propagate to the outputs of C

1 $FI :=$ const-fanin(g, a_{SAT})
2 sortByDistance(FI, o)
3 $a_{gen} := a_{gen} : FI \to \{0, 1, X\}$ with $a_{gen}(i) = a_{sim}(i)$
4 **foreach** $i \in \{FI.\text{size}(), \ldots, 1\}$ **do**
5 $\quad a_{gen}(FI_i) := X$
6 $\quad (a_{sim}, real) :=$ SIMULATE$(a_{gen}, C, s,$ fanin$(o))$
7 $\quad$ **if** *real* **then** $a_{gen}(FI_i) := a_{sim}(FI_i)$
8 **end**
9 **return** $\bigwedge_{\{g \in FI | a_{gen}(g) \neq X\}} g = a_{gen}(g)$

D. The algorithm GENERALIZE

The algorithm GENERALIZE shown in Algorithm 7 gets the assignment of a spurious counterexample and a primary output o that is affected by the SET according to the SAT formula. We use a greedy approach and return a SAT formula that describes a generalized assignment that suffices to prevent the SET from propagating towards o.

In line 1, we get the vector *FI* that contains the deepest constant signals within the fanin of o. We stop the search for the fanin at the first constant signal in the front partition according to a_{SAT} as these are equal in the assignment of the counterexample as well as the simulation because the different considerations of delays do not matter for constant signals. By this, we can further generalize the assignment. For example, in an xor-gate both inputs are relevant as a change of any input changes the output. However, we do not necessarily care about the exact inputs of the gate but only the output which can have different possible input assignments.

Afterwards, in line 2, we sort the vector *FI* by the distance of the gates to o. In this order, we can start to check gates that have a higher distance earlier and eventually set their assignment to X before checking closer gates that often have a higher impact on o. For example, an or-gate where one constant input is 1 only needs that 1 for its output to remain 1 and can set all variables that affect the other input to X.

In the loop from line 4 – 8, we try for each gate FI_i, in order from high to low distance to o, to set the assignment $a_{gen}(FI_i)$ to X in line 5 and simulate the modified assignment in line 6. To avoid unnecessary overhead, we only simulate the gates within the fanin of o. If the modified counterexample is real, i.e., o evaluates to X or the SET propagates to o, the value of FI_i is relevant for the SET not propagating towards o and we need to reset $a_{gen}(i)$ to its original value $a(i)$ in line 7.

Finally, in line 9 we return a formula that blocks the generalized counterexample.

IV. Discussion

Our approach provides a good performance due to two main reasons. On the one hand the generated SAT formula for the front partition is very simple and quickly solved and on the other hand our generalization allows us to block a high number of counterexamples after a single solver call.

When generating the SAT formula for the front partition, we can exploit the absence of a reconverging SET. Thus, we can describe the output of each gate with at most three variables as explained in Section III-C. Additionally, no further variables are needed for the variable delays or electrical masking because it suffices to check for equal variables instead of equal values. The resulting SAT formula can usually solved within seconds or less and we can easily use the solver multiple times within a short time.

If we would block each spurious counterexample individually, the runtime would not be feasible for most circuits as there is usually a very high number of counterexamples. For this reason, we generalize counterexamples as shown in Section III-D. By generalizing detected counterexamples, we can block multiple similar counterexamples with one SAT solver call. The degree of generalization depends on the circuit but usually provides a significant speed up.

circuit	[11]	SAT-based	hybrid	solver calls
c432	123.1s	2.8s	0.5s	2
c432-TMR	432.5s	35.1s	2.3s	5
c432-TTMR	5943.4s	timeout	1534.1s	1342
c2670	67.8s	47s	0.4s	2
c2670-TMR	227.0s	588.9s	9.1s	3
c2670-TTMR	1758.2s	timeout	573.2s	1025
c7552	5982.3s	timeout	2.2s	1
c7552-TMR	timeout	timeout	70.7s	11
c7552-TTMR	timeout	timeout	timeout	2227+

TABLE I: Runtimes of all approaches and number of solver calls for hybrid approach on some circuits

V. Experiments

In our experiments, we compare our hybrid approach against [11]. In addition, we also consider a modified SAT-based version of our approach where all gates are considered as front gates and the effects of the SET are also included in the SAT formula, i.e., using the approach from [11] but utilizing our model. We use the ISCAS-85 benchmarks. Each circuit is analyzed in its regular version and in two fault tolerant variations. The first variation uses *Triple Modular Redundancy* (TMR) and the second uses *Timed TMR* (TTMR). Both variations are presented in [11].

For each circuit, we choose a random gate near the inputs as location for the SET. An SET close to the inputs usually affects more gates and leads to a larger number of gates where the SET overlaps to sufficiently compare the differences between the algorithms.

The experiments are run on a Dual-Core AMD Opteron Processor 2222 SE with 3 GHz and 64 GB main memory. The resulting runtimes are shown in Figure 4. In addition, a table for the runtimes of all approaches as well as the number of solver calls done by our hybrid approach is shown for some circuits in Table I.

We can see that our hybrid approach is usually faster than [11] unless both time out after six hours. Our SAT-based approach often has a runtime between the two other approaches, showing that our model also decreases the runtime.

Even on the bigger circuits our algorithm takes less than 2.5 seconds to find a counterexample that disproves robustness for the normal non-robust versions. Our algorithm generates a simple SAT formula for the front and finds counterexamples quickly. Since the circuits are not robust by themselves, only

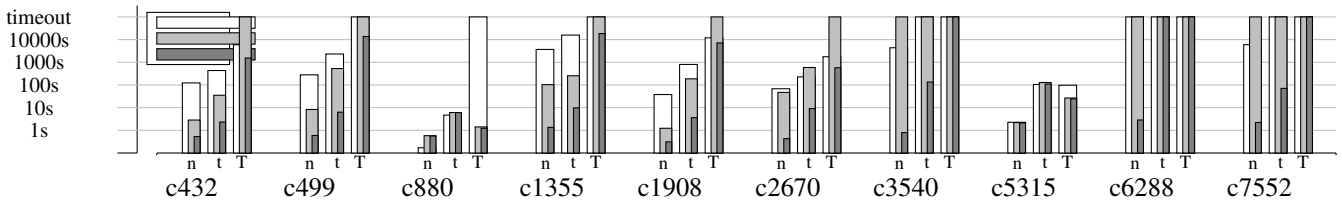

Fig. 4: Runtime experiments comparing [11] (white), our SAT-based approach (light gray), and our hybrid approach on ISCAS-85 circuits, using the normal (n), TMR (t), and TTMR (T) version

few counterexamples need to be simulated until a real counterexample is found. In comparison, [11] creates a complex SAT formula for each circuit which takes more time to solve.

For the robust circuits, our approach needs to generalize the detected counterexamples until robustness is proven. The generalization for the TMR circuits is quickly done as for each primary output the outputs of the two copies that are unaffected by the SET suffice to guarantee a correct value in the primary output. In both TMR and TTMR, we exploit that the fault correction is applied to each primary output individually and only need to analyze the relevant fanin.

In all TTMR circuits, our SAT-based approach times out. Due to the higher degree of detail for delays, the different switch times overlap and new variables need to be introduced to describe the value in between. With increasing depth of the circuit, this effect leads to an exponential growth of variables for each gate which especially affects the TTMR circuits. While this effect also becomes more relevant with increasing complexity, it only occurs when the SET reconverges in the front partition and therefore does not affect our hybrid approach.

Only the circuits c880 and c5315 in the normal and TMR version are decided faster by [11]. In these cases, the location of the SET leads to a very small back partition. Thus, the resulting SAT formula for the front is only slightly easier to solve than the one generated by [11]. As our implementation may need multiple counterexamples even in non-robust circuits, our runtime is slightly higher in these specific cases. However, over all experiments our new implementation provides an average speedup of 748.

We ran the experiments on c2670 again without using generalization. In the non-robust circuit, the number of detected spurious counterexamples only increased slightly as a real counterexample can easily be found in a small number of tries. In the robust cases, we generated 86 times more counterexamples and 340 times the previous runtime on average. These numbers show that the generalization is relevant for the runtime as discussed in Section IV.

As an alternative way to partition the circuit, we tried to put all gates within the back partition that require more than k variables. This alternate partitioning did not provide any significant speed up and took more time in most cases.

VI. Conclusion

We presented a hybrid algorithm to decide if a circuit is robust against a given SET. The algorithm partitions the circuit into a front and a back partition, uses SAT solving on the front partition and analyzes detected counterexamples by simulation. The experiments showed that dividing the problem this way can lead to a significant speed up.

References

[1] Mehdi Dehbashi and Görschwin Fey. SAT-based speed-path debugging using waveforms. In *IEEE European Test Symposium*, pages 1–6, 2014.

[2] Dan Ernst, Nam Sung Kim, Shidhartha Das, Sanjay Pant, Rajeev Rao, Toan Pham, Conrad Ziesler, David Blaauw, Todd Austin, Krisztian Flautner, and Trevor Mudge. Razor: a low-power pipeline based on circuit-level timing speculation. In *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, pages 7–18, 2003.

[3] Stefan Frehse, Görschwin Fey, Eli Arbel, Karen Yorav, and Rolf Drechsler. Complete and effective robustness checking by means of interpolation. In *Formal Methods in Computer-Aided Design*, pages 82–90, 2012.

[4] Jie Han, Hao Chen, Jinghang Liang, Peican Zhu, Zhixi Yang, and Fabrizio Lombardi. A stochastic computational approach for accurate and efficient reliability evaluation. *Computers, IEEE Transactions on*, pages 1336–1350, 2014.

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

[6] Natasa Miskov-Zivanov and Diana Marculescu. Multiple transient faults in combinational and sequential circuits: A systematic approach. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, pages 1614–1627, 2010.

[7] Kartik Mohanram. Simulation of transients caused by single-event upsets in combinational logic. In *Test Conference, 2005. Proceedings. ITC 2005. IEEE International*, pages 9 pp.–981, 2005.

[8] Martin Omaña, Giacinto Papasso, Daniele Rossi, and Cecilia Metra. A model for transient fault propagation in combinatorial logic. In *IEEE International On-Line Testing Symposium*, pages 111–115, 2003.

[9] Matthias Sauer, Alexander Czutro, Ilia Polian, and Bernd Becker. Small-delay-fault ATPG with waveform accuracy. In *Proceedings of the International Conference on Computer-Aided Design*, pages 30–36, 2012.

[10] Ashwin Seshia, Wenchao Li, and S Mitra. Verification-guided soft error resilience. In *Design, Automation Test in Europe Conference Exhibition*, pages 1–6, 2007.

[11] Niels Thole, Görschwin Fey, and Alberto Garcia-Ortiz. Conservatively analyzing transient faults. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, 2015.

[12] Michael Yoeli and Shlomo Rinon. Application of ternary algebra to the study of static hazards. *Journal of the ACM*, pages 84–97, 1964.