*EndWhile *EndIf
[1]// 1

Optimal ATPG

Jessica Davies and Sharad Malik

June 1, 2015

1 Introduction

When manufacturing digital circuits, it is important to check that the fabricated chip works properly. Chips that are detected to be faulty should be thrown away. It is important that no faulty chips are missed. However, this may be challenging since some faults can be difficult to detect. Furthermore, there are limits to the time and resources that can be invested to test each chip.

In order to test whether a given chip is faulty, the standard practise is to run the chip on a set of pre-determined inputs, called test patterns. If any of the test patterns cause the chip to produce output that is contrary to its design, then the chip can be classified as faulty and immediately thrown away.

For a given circuit design, and a set of possible faults to be detected, selecting a set of test patterns is called the Automatic Test Pattern Generation (ATPG) problem.

In this work, we assume that the circuit is combinational. We focus on detecting chips that are faulty due to exactly one of their lines behaving as if it were stuck at a value (0 or 1). This is commonly known as the stuck-at fault model. Let $\mathcal{F}$ be the set of all stuck-at-0 and stuck-at-1 faults for a circuit Φ.

We are interested in finding a set of test patterns, S, such that for each stuck-at fault $f \in \mathcal{F}$, there exists a test pattern in S that detects f. Such a set of test patterns detects all stuck-at faults, and is said to have complete coverage.

Definition 1. *A set of test patterns S has **complete coverage** of a set of faults $\mathcal{F}$ if for every fault $f \in \mathcal{F}$, there exists a test pattern $s \in S$ such that s detects f.*

Furthermore, we are interested in minimizing the number of test patterns, since this reduces the time required to test each chip. The size of a test set is also known as its compactness. We aim to find a test set that has complete coverage, and such that no other test set with complete coverage is more compact.

Definition 2. *A maximally compact, complete-coverage test set, or* **MCCC***, is a set of test patterns S_{min} that has complete coverage, and such that for any other set S' with complete coverage, $|S'| \geq |S_{min}|$.*

In the following section, we propose a new approach for calculating a MCCC test set.

2 MCCC Generation

We propose a new algorithm to find an MCCC test set for a given circuit Φ and set of faults $\mathcal{F}$. The algorithm is shown in Algorithm 2.

Find an MCCC test set for faults $\mathcal{F}$ of circuit Φ [1] MCCC$\Phi, \mathcal{F}$ $G =$ choose one fault $f \in \mathcal{F}$ $S = \emptyset$ $m = 1$ true $S =$ **testSet**$(\Phi,\, G,\, m)$ Returns a set of m test patterns that detects all faults in G, or $\emptyset$ if none exists.

$S == \emptyset$ $m = m + 1$ $S =$ **testSet**$(\Phi,\, G,\, m)$ $f =$ **undetected**$(\Phi,\, \mathcal{F} \setminus G,\, S)$ f is a fault in $\mathcal{F}$ that is $f ==$ null **break** S detects all faults in $\mathcal{F}$ $G = G \bigcup \{f\}$ not detected by S.
return S

Theorem 1. *Algorithm 2 terminates and returns an* MCCC *test set.*

Proof At each iteration, either the set of faults G is augmented with a new fault not already in G (lines 10 and 14), or, the loop terminates (line 12). Since the set of all possible faults $\mathcal{F}$ is finite, G can not be augmented an infinite number of times. Therefore the loop, and thus the algorithm, terminates. When the algorithm terminates, line 12 must have been executed because this is the only way for the loop to terminate. At this time, the current set of test patterns, S, detects all faults in $\mathcal{F}$ (by the definition of **undetected**). Therefore, the algorithm always returns a test set S with complete coverage. It remains to show that the returned test set is of minimum size. We prove this by induction on the number of loop iterations k, by proving that at line 10, there is no test set of size strictly smaller than m that detects all faults in G. In the first iteration of the loop, G contains only 1 fault, and therefore on line 6, S will contain exactly 1 test pattern. Since S is non-empty on line 7, the condition on line 7 fails and lines 8 and 9 will not be executed. Thus during the first iteration, on line 10 S is a min-size test set that covers the faults in G. Let $k \geq 1$ and assume that during loop iteration k, on line 10 S is a min-size test set that covers the faults in G. We prove that this is also the case for the $k + 1^{st}$ iteration. First, on the $k + 1^{st}$ iteration of the loop, G^{k+1} will be $G^k \bigcup \{f\}$. If the condition on line 7 fails, then $|S^{k+1}| = |S^k|$, so by the induction hypothesis, S^{k+1} is of minimum possible size to cover G^k, and therefore also to cover G^{k+1}. Otherwise, if the condition on line 7 succeeds, it means that there is no test set of size $|S^k|$ that detects all faults in G^{k+1}. Therefore, any test set that detects all faults in G^{k+1} must have size at least $|S^k| + 1$. However, this is the size of the test set returned on line 9 (**testSet** will not return $\emptyset$ because there *does* exist a test set of size $|S^k| + 1$ that detects all faults in G^{k+1}: just add a test pattern for the newest fault). So on line 10, in either case, S^{k+1} will be a min-size test set that detects all faults in G^{k+1}.

3 SAT-based ATPG

The algorithm proposed in Section 2 does not specify how the functions **testSet** and **undetected** are implemented. In this section, we describe a possible implementation of these two functions, based on the well-known SAT-based techniques for ATPG.

3.1 SAT-based Implementation of testSet

Here, we assume that the circuit is irredundant, so that all faults are detectable. We discuss the case where the circuit has undetectable faults in Section 3.2 below.

Figure 1 shows the circuit that we use to implement the **testSet** function. This circuit is translated to a CNF formula, and given to a SAT solver. If the formula is satisfiable, the truth assignment returned by the SAT solver will specify m test patterns that detect all faults in G. Otherwise, if the SAT solver returns UNSAT, it means there is no set of just m test patterns that can detect all faults in G.

The circuit in Figure 1 mentions sub-circuits C_i^f, for each fault i in G. These sub-circuits ensure that fault i is detected, using a standard (or improved) encoding of the test generation problem to SAT [**?**, **?**]. A simple implementation of C_i^f is shown in Figure 2, but in practise, we use the improved encoding of Chen and Marques Silva [**?**].

The size of the formula resulting from a CNF encoding of the circuit in Figure 1, is $O(k(n+m))$ where n is the size of the original circuit Φ, k is the current number of faults $|G|$, and m is the number of test patterns. During the execution of Algorithm 2, k begins at 1, and then grows by exactly 1 after each iteration of the loop (line 14). Note that $m \leq k$. We hope that the total number of loop iterations will be much less than $|F| = O(n)$. In this case, the largest formula passed to the SAT solver will be significantly smaller than $O(n^2)$.

3.2 SAT-based Implementation of undetected

Next, we describe the implementation of the second function of Algorithm 2, **undetected**. This function must find a fault in $\mathcal{F} \setminus G$ that is undetected by the current set of test patterns S. The general approach we propose is as follows.

We choose a fault $i \in \mathcal{F} \setminus G$ at random, and check whether it is detected by the current test pattern set S. We can check whether fault i is detected by test

$\bar{s_1}$

p_1 p_2 p_m C_1^f

[scale=0.5,unit=1mm]atpg$_circuit$

50p_1 p_2 C_2^f

30.

20p_1 p_m C_k^f

34.5T_k^m

Figure 1: A representation of the circuit used to implement **testSet**(Φ, G, m), where $k = |G|$. For each fault i in G, there is a corresponding sub-circuit C_i^f. The input lines $p_1, ..., p_m$ correspond to the m test patterns. A multiplexer with selector variables $\bar{s_i}$ is used to select which test pattern will be sent to C_i^f. There is a set of inputs $p_1, .., p_m$ and $\bar{s_1}, .., \bar{s_k}$ such that $T_k^m = 1$, if and only if there is a test set of size m that detects all faults in G.

[scale=0.5,unit=1mm]subcircuit p_k Φ Φ_i^f C_i^f

Figure 2: A representation of one sub-circuit C_i^f from Figure 1, that asserts that fault i of circuit Φ is detected. The output of the original circuit Φ is compared to the output of the faulty circuit Φ_i^f using an XOR.

set S by building a CNF formula like that in Figure 1, except with only a single sub-circuit C_i^f for the fault i, and a single multiplexer whose inputs are fixed to S. If the SAT solver returns a solution, then fault i is detected and we must choose another fault at random and try again. If all faults have been determined to be detected by S, then we return $\emptyset$ to indicate that S has complete coverage. Otherwise, we return the first fault we find that isn't detected by S.

If the circuit is not irredundant, then before a fault f is returned by **undetected**, we must first check whether f is detectable. This can be done by calling **testSet**$(\Phi, \{f\}, 1)$. If no test pattern is returned, the simulation must continue until an undetected, yet detectable, fault is found.

3.3 Incremental SAT Solving

The **testSet** function is called in each iteration of Algorithm 2, after either a new fault has been added to G, or the value of m has been increased. It is obvious that the CNF formula (described in Section 3.1) generated during a call to **testSet** will be very similar to the CNF formula generated in the previous iteration. Therefore, it may be beneficial to use an incremental SAT solver that can exploit this relationship, instead of solving each SAT instance independently.

In this section we specify which clauses must be added and removed from the CNF formula, when either G is augmented or m is increased. In our implementation, we use the incremental interface of Minisat to add clauses, as well as Minisat's support for solving under assumptions, to remove clauses. We hope that the clauses learned by Minisat in previous calls to the solve routine will improve the performance of subsequent calls.

3.3.1 Adding a Fault

When a fault i is added to G, we must add a new sub-circuit C_i^f to the formula shown in Figure 1. We must also add a new multiplexer that feeds into C_i^f. It is not hard to see that we only need to *add* clauses to the current formula, and we do not have to remove any clauses in this case.

3.3.2 Increasing the Test Set Size

When the number of test patterns is incremented by 1, we must add a new set of input variables p_{m+1} to the formula. Each existing multiplexer must be augmented with p_{m+1}, which requires adding a new selector variable to the multiplexer as well. The only clauses that need to be removed are the ones saying that at least one selector variable must be True. Now, each such clause must be replaced with a longer version of it, where the new selector variable has been added to the disjunction. That is, $(s_1 \vee, ..., \vee s_m)$ is removed, and $(s_1 \vee, ..., \vee s_m \vee s_{m+1})$ is added instead.

4 Related Work

The MCCC problem was proven to be NP-hard by Krishnamurthy and Akers in 1984 [**?**].

Almost all work on ATPG assumes that it is computationally infeasible to find an MCCC. Most existing approaches consider the faults one at a time, and use a SAT solver to find a test pattern for each fault.

Reddy et al. consider structural (rather than SAT-based) approaches for ATPG, and propose various heuristics for compact test set generation [**?**, **?**]. Some work on SAT-based ATPG proposes to exploit the internal data structures of the SAT solver [**?**]. Recently, there has been increased interest in SAT-based methods to produce compact test sets [**?**, **?**, **?**, **?**]. There is also work that attempts to throw out redundant test patterns, after a set of test patterns that covers all faults has been determined. Given a set of test patterns and the list of faults that each pattern detects, a minimum set covering problem can be set up to find the minimum size subset of the patterns that still detects all faults [**?**].

Note that this will not necessarily find an MCCC, since the size of the resulting minimized test set depends on the initial set of test patterns.

The only existing work on calculating an optimal MCCC is by Marques Silva[1], who proposed an encoding of the MCCC problem to an Integer Linear Programming (ILP) problem [**?**]. However, the ILP model is of size $O(n^3)$ where n is the size of the original circuit Φ, and the number of faults $|\mathcal{F}|$. Marques Silva does not report any experimental results concerning this translation to ILP. For large circuits with tens or hundreds of millions of gates and faults, the size of this ILP model, not to mention the additional memory required to solve it using Branch and Cut, is expected to be prohibitive.

As an example of a SAT-based approach that performs dynamic compaction, we describe the algorithm from [**?**]. The faults are first ordered according to a topological criteria. At each iteration of the algorithm, a group of faults is built up in a greedy manner. The group of faults must satisfy the criterion that a single test pattern can detect all faults in the group. This is done by checking whether or not the next fault in the list can be added to the current group while maintaining the criterion. The iteration ends after a certain number of faults have been rejected from the current group[2]. At the end of the iteration, a test pattern that detects all faults in the group is added to the final test set. This test pattern is then used for fault simulation, to remove other detected faults from the fault list. The next iteration begins with a new fault group, containing the next undetected fault in the list. The intuition for this approach is that it finds test patterns that can detect many faults, by finding groups of faults that can be detected using a single test pattern.

5 Experimental Results

TIGUAN is a recent SAT-based ATPG tool that performs dynamic compaction [**?**]. It supports a richer fault model, but can also handle stuck-at faults. Although TIGUAN is not publicly available, [**?**] reports the size of the test sets generated for all stuck-at faults[3] in a subset of ISCAS'85 and ISCAS'89 circuits.

Experiments were performed on ISCAS85, ISCAS89, and ITC99 circuits [**?**].

6 Alternative Approach Using Min-MUS

Let $F = F_0 \cup F_1$ be the set of detectable faults of circuit Φ, where F_0 and F_1 are the stuck-at-0 and stuck-at-1 faults respectively. Suppose we have already found

[1]Marques Silva calls the MCCC problem the *minimum test set* problem.
[2]Rejected faults are placed back on the list, to be retried in the next iteration.
[3]Alexander Czutro, private communication

a test set $S = \{p_1, ..., p_k\}$ that is complete for F (here, S can be of arbitrary size). We define the following propositional formula Θ_S[4]:

$$\Theta_S = \bigwedge_{p_i \in S} \left[(C_i(p_i) = o_i) \bigwedge_{j \in F_1} (\neg f_j \vee \ell^i_j) \bigwedge_{j \in F_0} (\neg f_j \vee \neg \ell^i_j) \right] \wedge [\Sigma_{j \in F} f_j = 1]$$

where $C_i(p_i)$ is the formula consisting of a copy of circuit Φ with its inputs set to p_i, and $o_i = \Phi(p_i)$ is a constant equal to the value of the output lines in the correct circuit Φ on input p_i. There is a fault variable f_j for each fault $j \in F$, and exactly one of these variables must be assigned to True. Finally, ℓ^i_j is the variable in $C_i(p_i)$ corresponding to the line affected by fault $j \in F$.

If Θ_S is satisfiable, then there exists a fault $j \in F$ such that the output of the faulty circuit is the same as the output of Φ for all $p_i \in S$. Therefore, S is not a complete test set.

Proposition 1. *Θ_S is unsatisfiable if and only if S is a complete test set.*

Proposition 2. *If Θ_S is unsatisfiable, then a minimal unsatisfiable core of Θ_S corresponds to a minimal complete test set.*

Proposition 3. *There exists a complete test set S such that an unsatisfiable core of Θ_S of minimum size corresponds to a* MCCC.

Proposition 3 suggests an alternative algorithm for finding a MCCC. We propose to find a complete test set S using existing heuristic methods, e.g. [**?**]. Given S, we build Θ_S and find a minimal or min-size core using existing methods such as [**?**, **?**].

[4]This formula is similar to the one defined by Fujita and Mishchenko [**?**], except here we constrain the number of simultaneous faults to be 1.