Improving CNF Representations in SAT-based ATPG for Industrial Circuits using BDDs

Daniel Tille[1] Stephan Eggersglüß[1] René Krenz-Bååth[2] Juergen Schloeffel[2] Rolf Drechsler[1]

[1]Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
{tille,segg,drechsle}@informatik.uni-bremen.de

[2]Mentor Graphics Development (Deutschland) GmbH, 21079 Hamburg, Germany
{rene_krenz-baath,juergen_schloeffel}@mentor.com

***Abstract*—It was shown in the past that ATPG based on the Boolean Satisfiability problem is a beneficial complement to traditional ATPG techniques. Its advantages can be observed especially on large industrial circuits. These circuits usually contain a lot of functional redundancy which, on the one hand, is often needed during operational mode, but on the other hand, causes dispensable overhead during ATPG. Using the traditional circuit-to-CNF transformation, this redundancy is also contained in the SAT instances.**

The contribution of this paper is a new technique to improve the SAT instance generation for SAT-based ATPG. The objective of the proposed method is to use *Binary Decision Diagrams* (BDDs) to optimize the resulting CNF representations. In order to apply the proposed technique to industrial circuits, we developed dedicated BDD operations using a multiple-valued logic. The experimental results, obtained on large industrial designs, show that the accomplished optimizations result in a considerable acceleration of the overall ATPG runtime as well as in a significant reduction of the unclassified faults.

I. Introduction

The continuous growth of today's circuit designs requires a constant improvement of state-of-the-art *Electronic Design Automation* (EDA) tools. Traditional ATPG techniques such as FAN [1], SOCRATES [2], and ATOM [3] reach their limits, i.e. the number of faults that cannot be classified by them has been increasing over the last years. During recent years, SAT-based ATPG algorithms became a promising complement [4], [5]. SAT-based methods proved to be highly advantageous in particular for hard-to-solve problem instances. Most modern SAT solvers, e.g. [6], [7], [8], require the modeling of a problem instance in *Conjunctive Normal Form* (CNF). Hence the circuit-based ATPG problem needs to be converted into a Boolean formula. This CNF representation is typically generated by processing each gate independently, i.e. without considering its adjacency [9].

Industrial circuits usually contain a large amount of functional redundancy. This has two main reasons. The redundancy could be caused by the limitation of the underlying library. In most cases, however, redundancy is intended. Additional gates are included in the circuit due to timing issues or robustness. In any case, this redundancy causes overhead for the ATPG process. A fault effect must be justified and propagated to compute a test pattern for some fault. Only the circuit's functionality is required during those steps; the redundancy increases the computational costs.

Since the traditional circuit-to-CNF transformation in SAT-based ATPG is performed for each gate independently [9], this redundancy is also included in the generated SAT instances. This yields a significant increase in CNF variables as well as in CNF clauses which interferes the subsequent solving process.

In this paper, we propose an optimized SAT instance generation method to overcome this drawback. We use *Binary Decision Diagrams* (BDDs) [10] to represent parts of the considered circuit. Those BDDs do not contain functional redundancies. Afterwards, CNF representations are derived from the BDD. Thus, they do not contain the functional redundancies either. As a result, both the number of CNF variables and CNF clauses can be reduced significantly.

While the BDD construction for Boolean circuits is quite straightforward – because there is a basic operation (*If-Then-Else* (ITE) operation) for each basic gate –, industrial circuits, which require a treatment in multiple-valued logic, cannot be handled directly. To overcome this drawback, we developed complex BDD operations which are also discussed in this work.

We apply SAT-based ATPG for the stuck-at fault model [11] and the transition fault model [12]. The experimental results conducted for large industrial designs show that the proposed approach is able to significantly accelerate the ATPG process. Additionally, the number of unclassified faults can be reduced considerably.

The paper is structured as follows. Previous work is discussed in Section II. Section III describes the construction of BDDs for industrial circuits. The presentation of our proposed approach is the objective of Section IV. Experimental results are reported in Section V. Finally, Section VI concludes the paper.

II. Previous Work

A. SAT-based ATPG

Test pattern generation with respect to some *Stuck-At Fault* (SAF) is the search for an input assignment, which conducts different values at some primary output between the faulty circuit and the correct circuit. In SAT-based ATPG, this search is transformed into a Boolean satisfiability problem. It was initially proposed in [9] and significantly improved in [13] and [14]. If a test for a particular SAF exists, then the corresponding problem instance is satisfiable and the resulting test pattern can be directly derived from the satisfying assignment. If the fault is undetectable, the SAT solver concludes unsatisfiability.

In the following, the circuit-to-CNF transformation for an SAF is reviewed. Figure 1 illustrates a combinational circuit. The *fault site* denotes a connection c where an SAF is assumed. The area denoted as *output cone*, contains all gates belonging to some path P from c to some primary output in

Daniel Tille is now with Verified Systems International GmbH, Bremen, Germany.

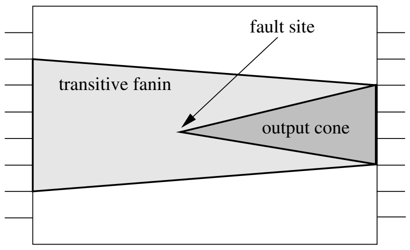


Fig. 1. Illustration of influenced circuit areas.

TABLE I
APPLICATION OF L_4

	0	1	U	Z
0	0	0	0	0
1	0	1	U	U
U	0	U	U	U
Z	0	U	U	U

(a) Truth table of an AND gate.

s	x_s	x_s^*
0	0	0
1	1	0
U	1	1
Z	0	1

(b) Boolean encoding of L_4.

the transitive fanout of c, $fanout^*(c)$. Let us denote the set of primary outputs reachable from c by O_c. Next the transitive fanin, $fanin^*(O_c)$, of all primary outputs in O_c is computed, see Figure 1.

As introduced in [13], two Boolean variables g_c and g_f are assigned to every gate g in the transitive fanout of c, $g \in fanout^*(c)$, to represent the fault-free circuit and the faulty circuit, respectively. Both circuits are generated by building the characteristic function for every gate [15]. Clearly all gates belonging to the transitive fanin of some primary output in the set O_c and are not contained in $fanout^*(c)$ only need to be modeled once, since the SAF at c does not influence their behavior. Additionally, a Boolean variable g_d is assigned to every gate $g \in fanout^*(c)$ to express a difference between the values g_c and g_f. If g_d is true, then the values g_c and g_f differ. A test pattern to detect the SAF at c is found if it is possible to compute an assignment such that there is a path P from c to some primary output, where the variable g_d for each gate $g \in P$ is true. This path is called *D-chain*.

The construction of SAT instances for the transition fault model was shown in [16].

B. Handling of Industrial Circuits

It is not sufficient to consider only the Boolean values 0 and 1 during test pattern generation as has been done in earlier approaches (e.g. [13]) for industrial applications. Industrial circuits further contain unknown values (denoted by U) and values at high impedance (denoted by Z). Of course, those values have to be considered during ATPG, too. This is accomplished using a 4-valued logic $L_4 := \{0, 1, U, Z\}$. The truth table of an AND gate in L_4 is given in Table II(a).

A problem formulated in L_4 has to be transformed into a Boolean problem in order to apply a Boolean SAT solver. Two Boolean variables are necessary to encode L_4. Table II(b) shows the encoding which has shown to be effective for SAT-based ATPG and which is used in the following. For instance, $1 \in L_4$ is encoded by $(1,0) \in \mathbb{B}^2$.

As a consequence, the Boolean variables g_c and g_f used to represent the gate g's values in the fault-free and the faulty circuit, respectively, are not sufficient anymore. If industrial circuits are handled by SAT-based ATPG, two variable g_c and g_c^* respectively g_f and g_f^* are used to represent both circuits.

A detailed overview on SAT-based ATPG for industrial circuits is given in [16].

C. Binary Decision Diagrams

A BDD is a rooted, directed, acyclic graph, where each node is either an inner node or one of the terminal nodes **0** and **1**. Each inner node has exactly two child nodes (the *low-child* and the *high-child*) and is labeled with a Boolean variable. A BDD is called *ordered* if there is a *variable order* π and on each path from the root node to a terminal node, the variables occur at most once and according to π. An ordered BDD is called *reduced ordered BDD* if it does not contain any two nodes representing the same Boolean function. Reduced ordered BDDs are a canonical representation of Boolean functions [10]. Canonicity allows significant performance improvements for operations such as equivalence checking or satisfiability checking. Additionally, reduced ordered BDDs are a highly effective representation for large combinational sets. In the following, we imply that all BDDs are reduced ordered BDDs. For more details about BDDs, we refer to e.g. [17].

There are multiple ways to derive a CNF from a BDD. The possibility we use is explained in the following. Each path from the root node to the zero-terminal node (***0**-path*) corresponds to one clause, which is constructed by the disjunction of the complemented outcome of each occurring variable.[1] Finally, all clauses generated this way are conjuncted.

Example 1. *An example is presented in Figure 2. Figure 2(a) shows a BDD B containing six nodes: the inner nodes $v_1, \ldots, v_4$, where v_1 is the root node, and the one-terminal and zero-terminal nodes v_5 and v_6, respectively. The BDD depends on three variables, $x_1, \ldots, x_3$. The dashed lines denote low-edges and the solid lines denote high-edges. B is ordered (the variable order is $\pi = (x_1, x_2, x_3)$) and reduced. It contains three **0**-paths, so the CNF ϕ_B consists of three clauses.*

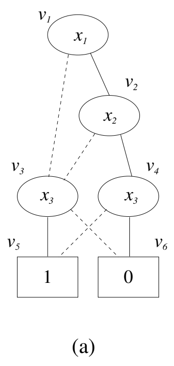
*The first **0**-path is given by $p_1 = v_1 \rightarrow v_2 \rightarrow v_4 \rightarrow v_6$. All variables occurring on p_1 have a positive outcome, because the path is traversed only via high-edges. The variable assignment $(x_1, x_2, x_3) = (1,1,1)$ referring to the path evaluates the function to false. Therefore, it has to be excluded from the solution space. This is achieved by the clause $\omega_1 = (\overline{x}_1 + \overline{x}_2 + \overline{x}_3)$.*

The corresponding clauses for paths $p_2 = v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_6$ and $p_3 = v_1 \rightarrow v_3 \rightarrow v_6$ are $\omega_2 = (\overline{x}_1 + x_2 + x_3)$ and $\omega_3 = (x_1 + x_3)$, respectively. Finally, the CNF is given by $\phi_B = \omega_1 \cdot \omega_2 \cdot \omega_3$ (see Figure 2(b)).

Note, that ϕ_B in the example above is not minimal. The clause ω_2 can be replaced by $\omega_2' = (x_2 + x_3)$. State-of-the-art BDD packages as used in this approach are able to efficiently calculate such prime implicants during their path enumeration.

As mentioned above, the number of clauses in the CNF is equal to the number of **0**-paths in the BDD. Unfortunately, the number of **0**-paths is exponential with respect to the number of nodes in the BDD in the worst case. A sophisticated approach which introduces auxiliary variables in order to reduce the number of paths in the BDD is proposed in [18].

[1]A variable's outcome depends on the path used from the respective node. If the child node is reached via a low edge, the variable's outcome is negative. Otherwise, it is positive.



$$(\overline{x}_1 + \overline{x}_2 + \overline{x}_3)$$
$$(\overline{x}_1 + x_2 + x_3)$$
$$(x_1 + x_3)$$

(b)

Fig. 2. (a) BDD B, (b) CNF representation ϕ_B of B.

The minimization of the number of paths in a BDD using different sifting strategies is described in [19].

D. Related Work

An approach to reduce the size of CNF representations of Boolean formulas is proposed in [20]. Gates with fanout count of 1 are identified and merged with their fanout gate during their translation into CNF. This method works only on Boolean formulas and is not efficiently applicable to industrial circuits. Another SAT instance simplification technique is proposed in [21]. It describes a method where a CNF is optimized with respect to its size by applying recursive self-subsumption within the CNF. The method is not able to identify redundant circuit parts.

A structural simplification technique is presented in [22]. *And-Inverter Graphs* (AIGs) – used as representation of Boolean functions – are structurally and locally optimized. A CNF that can be derived from the AIG is thus also optimized. Since our approach addresses industrial circuits, AIG-based optimization is not directly applicable. An adaption to process AIGs over L_4 (similar to the adaption for BDDs described in the next Section) is necessary.

A special class of BDDs, *Structurally Synthesized BDDs* (SSBDDs), is employed in [23] to improve ATPG. The authors propose to use SSBDDs as a representations of *Fanout-Free Regions* (FFRs) in order to apply fault collapsing and circuit compaction. Afterwards, they use an equivalent of the classical ATPG algorithm PODEM [24] to generate test patterns. SSBDDs are not applicable to our approach because they preserve structural information.

III. BDD Representation of Industrial Circuits

The generation of BDD representations of Boolean circuits is straightforward. There exists a basic BDD operation for each basic gate, i.e. transforming a primitive into a BDD requires only one basic BDD operation, i.e. one *ITE operation*.

Figure 3(a) shows the BDD representing the *characteristic function* for a Boolean AND gate with inputs i_1 and i_2 and output o. All paths from the root-node to the **1**-node correspond to a legal assignment for the gate, i.e. evaluate the characteristic function to true.

Generating BDDs for industrial circuits is more difficult. The contained gates implement functions over L_4 which requires a Boolean encoding as described in Section II-B. Moreover, there is no basic BDD operation available to generate a gate's BDD directly.

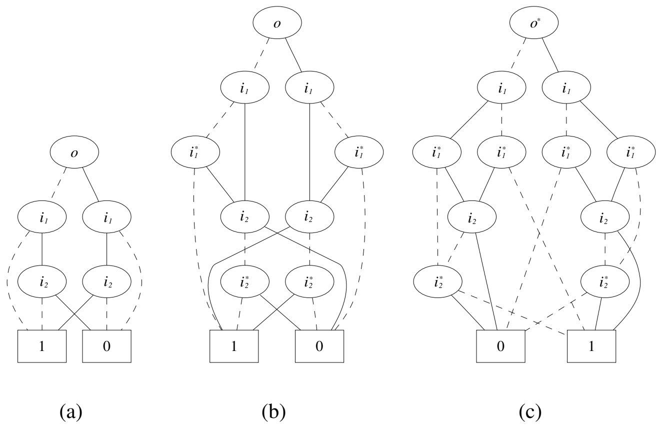


Fig. 3. Example for BDDs: (a) Boolean AND gate, (b) 4-valued AND gate – output o, (c) 4-valued AND gate – output o^*.

In order to be able to process industrial circuits, we implemented an approach to handle gates defined over L_4. Since the representation of such gates is very complex (remind the truth table of an 2-input AND gate depicted in Table II(a)), several ITE operations are necessary to generate the BDD of only one gate. We derived this chain of basic BDD operations for each occurring gate primitive automatically from the respective truth tables and our chosen Boolean encoding of L_4. Each minterm of the function, i.e. each one-column in the truth table, is transformed into a **1**-path of the resulting BDD. So, this pre-computed ITE-chain is processed whenever a gate is transformed into a BDD representation.

Example 2. *Figures 3(b)-3(c) show the BDDs for an AND gate in L_4 where each BDD represents one of both Boolean output functions. They were constructed using the approach described above. Our implemented 4-valued BDD operation to generate the BDDs needs 66 basic BDD operations.*

The assignment $a := (i_1,i_1^*,i_2,i_2^*,o,o^*) = (1,0,1,0,1,0)$ *(which corresponds in L_4 to the correct expression $1 \wedge 1 = 1$) results in **1**-paths in both BDDs. However, the assignment* $a = (1,1,0,0,1,0)$ *(which corresponds in L_4 to the incorrect expression $U \wedge 0 = 1$) results in a **0**-path at the BDD in Figure 3(b). This indicates the illegal assignment.*

Note that both BDDs in the above example actually can be shared in one BDD. However, the distinction was made for the sake of clarity.

IV. Improved Circuit-to-CNF Transformation

A. Motivation

As already mentioned, industrial circuits contain a lot of functional redundancy [25]. This redundancy might be intended to guarantee the correctness of the chip, e.g. with respect to timing, during operational mode. During ATPG, however, it causes dispensable overhead. Figure 4 gives an illustration of our argumentation. Assume, the fault effect can be injected and propagated along the solid lines. Redundancy that might be contained within the clouds can interfere those steps: additional implications have to be performed or branching heuristics are distorted. If the redundant parts are removed, both steps can be improved without influencing the correctness of the ATPG process.

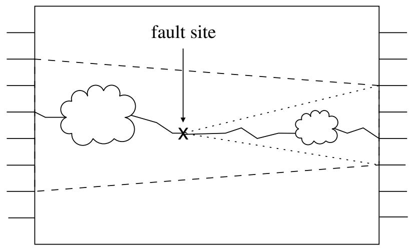


Fig. 4. Redundancy in the circuit is overhead during ATPG.

As discussed in Section II-A, the instance generation in SAT-based ATPG usually does not consider the adjacent circuit structure. Therefore, the redundancy in the circuit is also contained in the CNF – in form of "unnecessary" variables and clauses. Since a SAT instance's size has strong influence on the SAT solver's performance, a compact CNF without redundant information can speed up the solving process.

Examples for actual redundancy occurring in industrial circuits are shown in Figure 5. Figure 5(a) presents a small part of the industrial circuit *p99k*, provided by NXP Semiconductors GmbH. It can be seen that the entire sub-circuit consisting of four gates actually implements a three-input NAND gate. Figure 5(b) illustrates another form of redundancy. The depicted sub-circuit of circuit *p44k* contains two inverter chains. Inverting a signal twice results in the signal itself. So it does not have any influence on the circuit's logical function. However, it may be important to assure the signal's integrity. Therefore, the circuit structure itself must not be changed. The used SAT algorithms do not work on the circuit structure. Therefore, optimizing the CNF representation allows for improvement without altering the circuit itself.

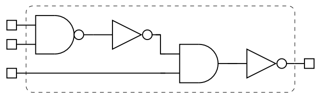
B. Method

The basic idea of the method is described in the following. First, information about *Fanout-Free Regions* (FFRs) is retrieved from the circuit structure. This can be done easily and many ATPG tools already perform this during a preprocess. Then, a single BDD representation is generated for each FFR. The FFRs are treated as individual, independent sub-circuits which is beneficial due to two reasons:

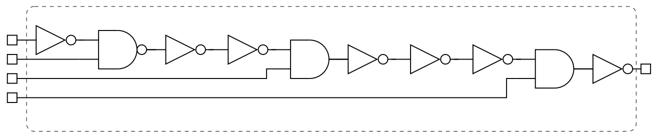
- FFRs are tree-like structures; only FFR-inputs can be shared. It was proven in [26] that the BDD representation of tree-like structures is compact. Therefore, the BDDs can be generated very efficiently.
- Many FFRs are functionally equivalent in industrial circuits. The BDD generation can profit since in state-of-the-art BDD packages only one BDD representing all equivalent FFRs has to be build.

Afterwards, the method to derive a CNF representation from a BDD described in Section II-C is used to transform each BDD into an intermediate data structure. It implements the CNF representation of an FFR as a template and depends on the Boolean variables $\{x_1, \ldots, x_n\}$ where n is the number of variables in the BDD. Thus, the time-consuming prime implicant extraction from the BDD has to be performed only once. Since functionally equivalent FFRs share the same BDD, they also share the same intermediate data structure. Each FFR only stores a pointer to it as an attribute. A memory explosion is highly unlikely due to the large extent of sharing.

Note, that all steps described so far are done prior to the actual test pattern generation. The following describes the steps performed for each fault.



(a)



(b)

Fig. 5. Example for redundancy in a circuit.

The SAT instance generation is generally accomplished as described in Section II-A (see Figure 1). The fault site c is marked, the set of influenced outputs O_c is determined, and the transitive fanin of O_c is traversed. Unlike in the traditional approach, the Boolean variables g_c, g_c^*, g_f, g_f^*, and g_d are only assigned to FFR-inputs and FFR-outputs.

Then, the CNF for each FFR is derived from the intermediate data structure instead of using the circuit structure to build a SAT instance. The actually assigned Boolean variables are mapped onto the variables of the intermediate data structure. Obviously, for FFRs contained in $fanout^*(c)$, the CNF generation has to be applied twice, for the fault-free and for the faulty circuit. It is easy to see that D-chains are modeled along FFRs in this approach.

Example 3. *Assume the two FFRs F_1 and F_2 implement the Boolean AND function. Both FFRs are described by the BDD presented in Figure 3(a) and thus represented by the intermediate CNF representation.*

$$\phi_g = (o + \bar{i}_1 + \bar{i}_2)(\bar{o} + i_1)(\bar{o} + i_2),$$

where the Boolean variable o is associated with the FFR-output and i_1 and i_2 are associated with the two FFR-inputs, respectively.

Whenever an FFR is visited during the circuit traversal within the actual ATPG, Boolean variables are assigned to the FFR-inputs and FFR-outputs, e.g. x_5, x_6 (inputs), and x_7 (output) for F_1 and x_{12}, x_{28} (inputs), and x_{64} (output) for F_2 Then, the recently allocated variables replace the variables in ϕ_g resulting in

$$\begin{aligned}\phi_{F_1} &= (x_7 + \overline{x}_5 + \overline{x}_6)(\overline{x}_7 + x_5)(\overline{x}_7 + x_6) \text{ and}\\ \phi_{F_2} &= (x_{64} + \overline{x}_{12} + \overline{x}_{28})(\overline{x}_{64} + x_{12})(\overline{x}_{64} + x_{28}).\end{aligned}$$

The CNFs ϕ_{F_1} and ϕ_{F_2} are added to the SAT instance in order to represent F_1 and F_2, respectively. Processing the next fault can lead to other variables x_i, x_j, x_k associated with the FFRs. Then, the ϕ_{F_i} have the same shape but depend on x_i, x_j, x_k. It can also happen that an FFR F is not considered for a fault. Obviously, ϕ_F is then not added to the SAT instance.

Note that the fault site could occur on a connection inside an FFR. A BDD-based optimization is not feasible in that case, because then this particular connection will not have a Boolean variable where the fault can be modeled. Therefore, this single FFR containing the fault is treated the traditional way, i.e. its CNF is build using the usual circuit-to-CNF transformation.

TABLE II
OVERVIEW IN THE AVERAGE SAT INSTANCE SIZES FOR THE STUCK-AT FAULT MODEL AND THE TRANSITION FAULT MODEL

	Stuck-at Faults Model				Transition Fault Model			
	Traditional		BDD-based		Traditional		BDD-based	
Circuit	Vars.	Cls.	Vars.	Cls.	Vars.	Cls.	Vars.	Cls.
b14	3931	10802	2041	8167	11225	31639	4860	21468
b15	5071	14320	2635	11778	10618	30527	4785	23183
b17	4515	12403	2364	10407	11069	31005	4737	22494
b18	4376	11875	2369	9808	12935	35978	5884	25075
b20	5301	14601	2517	10555	15445	43208	6427	28586
b21	5374	14825	2469	10399	15573	43650	6291	28430
b22	5154	14180	2434	10325	15579	43435	6478	28638
p44k	24498	61535	19256	52639	35502	90622	20089	58965
p49k	67709	235737	28640	130813	117958	407267	51740	238434
p77k	397	1044	164	626	2353	6302	804	3458
p80k	2684	6314	1905	4882	2667	6906	1885	5612
p88k	1677	4021	1068	2949	6626	16344	3558	10722
p99k	1900	4342	1441	3448	4243	11151	2114	7015
p141k	26096	78564	20429	76325	71554	227845	46232	188710
p177k	30075	87477	20340	74795	84842	259636	48865	196263
p456k	5124	14709	3029	10797	18131	56148	10297	39318
p462k	3397	9497	2505	8879	17965	50543	12039	44026
p565k	1190	3146	762	2545	4735	14411	3548	12428
p1330k	12928	40243	8497	35403	29111	88025	18493	71920

TABLE III
EXPERIMENTAL RESULTS FOR THE STUCK-AT FAULT MODEL

			Traditional		BDD-based		
Circuit	Targets	Untest.	Ab.	Time	Ab.	Time	%Time
b14	22,700	156	0	1:27m	0	1:10m	80%
b15	21,850	727	0	2:48m	0	2:48m	100%
b17	76,493	1,958	0	6:46m	0	6:52m	101%
b18	264,043	2,844	0	14:06m	0	14:08m	100%
b20	45,461	319	0	3:40m	0	2:22m	65%
b21	46,156	378	0	3:25m	0	2:37m	77%
b22	67,540	344	0	4:28m	0	3:12m	72%
p44k	64,105	2,385	0	1:33h	0	1:13h	78%
p49k	142,461	774	4,842	6:43h	790	2:59h	44%
p77k	163,310	9,181	0	0:18m	0	0:17m	94%
p80k	197,834	124	1	8:04m	0	5:49m	72%
p88k	147,742	2,640	0	2:56m	0	2:23m	81%
p99k	162,019	2,141	0	2:02m	0	1:37m	80%
p141k	267,948	13,815	1,273	2:17h	200	1:40h	73%
p177k	268,176	13,840	1,002	2:06h	273	1:38h	78%
p456k	740,660	35,396	182	59:23m	42	33:43m	57%
p462k	673,465	132,249	491	1:37h	250	1:18h	80%
p565k	1,025,273	28,287	0	7:37m	0	6:53m	90%
p1330k	1,510,574	44,299	59	1:13h	56	1:01h	84%

C. *Implementation Details*

In the following, we discuss issues regarding the implementation of the proposed technique. We chose to apply the well-known CUDD package [27] version 2.4.1 as BDD package.

Although BDDs can usually represent FFRs in a compact manner, the number of paths might grow exponentially. For instance, the BDD representation of the EXOR function depending on n variables consists of only $2 \cdot n + 1$ BDD nodes. However, there are 2^{n-1} **0**-paths. During first experiments, we observed that the number of prime implicants and hence the number of clauses derived from a BDD often exceeds the number of clauses needed by the traditional approach. This generally reduces the efficiency of the SAT solver.

Therefore, we decided to use a limitation that is able to prevent an increase of the CNF size. During the BDD construction for an FFR, the number of clauses that would be required to generate the corresponding CNF in the traditional way is calculated. This is accomplished using a look-up table, since the size depends only on the gate type and the number of gate inputs. If the number of **0**-paths in the BDD representing the FFR, i.e. the number of clauses in the CNF, is greater than the number of clauses needed by the traditional circuit-to-CNF transformation approach, the BDD is dismissed. In that case, the CNF for the FFR is *always* generated the traditional way.

Especially large FFRs exceed that bound frequently. Therefore, we try to reduce the number of paths during the BDD construction by using a decomposition strategy similar to this described in [18]. Whenever the path number of the current sub-BDD exceeds the limit calculated for the current sub-FFR, the most recent BDD operation is undone, a new BDD variable is created for each gate input to represent its sub-BDD, and the BDD operation is repeated. This increases the number of variables only slightly, but the reduction with respect to the number of paths is significant. On average, 80%-90% of all generated BDDs meet the limitation discussed above and thus provide a more compact CNF than the traditional approach.

V. EXPERIMENTAL RESULTS

This section contains an experimental evaluation of our proposed methodology. The new technique was applied to a set of benchmarks consisting of the publicly available ITC'99 circuits [28] and twelve large industrial designs provided by NXP Semiconductors GmbH. The name of the benchmarks reflects the approximate number of elements contained in the circuit. For example benchmark *p1330k* contains roughly 1.3 million elements. The SAT solver used is MiniSat version 2.0 [8]. The experiments were performed on an AMD Athlon 64 system (3.0 GHz, 4 GByte RAM, GNU/Linux).

The BDD generation and transformation into the intermediate data structure for each considered circuit requires less than one minute of CPU time and the additional memory consumption does not exceed 50 MByte.

Table II gives an overview on the CNF sizes for SAT-based ATPG. The circuits' names are shown in the first column. Column *Traditional* presents the results of the traditional approach where each CNF is generated without considering the adjacent circuit structure of the gates. Our proposed technique is presented in column *BDD-based*. The average SAT instance sizes with respect to the number of variables (columns *Vars.*) and the number of clauses (columns *Cls.*) are given for the stuck-at fault model as well as for the transition fault model in the respective columns. As described earlier, our proposed method aims for optimizing the SAT instances such as redundancy removal which leads to more compact CNFs. This can be confirmed by the shown results. The average sizes of the conducted SAT instances are reduced significantly for all circuits.

Table III provides the experimental results for the stuck-at fault model. The first three columns contain information about the circuits, such as circuit name, number of targets, and number of untestable targets, respectively. The set of targets contains the number of faults to be considered after fault collapsing. The number of unclassified faults (column *Ab.*) and the required runtime (column *Time*) are reported for both approaches. The test pattern generation for a fault is aborted after 5 million propagations in the SAT solving process. Furthermore, column *%Time* shows the percentage of the runtime needed by the BDD-based approach with respect to the runtime required by the traditional approach. The experimental results show a considerable speed-up of the entire ATPG process for almost all circuits. The number of unclassified faults is also reduced significantly; see circuit *p49k* for example.

TABLE IV
EXPERIMENTAL RESULTS FOR THE TRANSITION FAULT MODEL

			Traditional		BDD-based		
Circuit	Targets	Untest.	Ab.	Time	Ab.	Time	%Time
b14	40,068	2,831	273	16:40m	7	10:33m	63%
b15	38,094	5,392	32	16:13m	0	12:37m	78%
b17	133,804	20,272	50	49:23m	4	37:38m	76%
b18	459,360	97,321	386	3:59h	2	2:43h	68%
b20	80,606	5,212	558	36:39m	2	22:27m	61%
b21	82,060	5,410	576	38:47m	2	24:03m	62%
b22	119,810	7,539	831	58:33m	4	35:19m	60%
p44k	109,806	45,139	4,861	8:40h	232	4:55h	57%
p49k	255,326	6,302	—		—		
p77k	282,728	199,008	0	45:31m	0	21:55m	48%
p80k	311,416	14,293	15	13:21m	4	9:13m	69%
p88k	256,050	22,263	0	33:12m	0	21:47m	66%
p99k	274,376	28,026	0	20:43m	0	14:23m	69%
p141k	361,502	95,275	48,737	39:14h	12,210	18:55h	48%
p177k	410,240	97,867	58,072	45:17h	15,904	22:37h	50%
p456k	1,177,260	165,496	12,333	20:17h	1,032	7:42h	38%
p462k	1,134,924	485,788	1,469	13:24h	1,030	10:41h	80%
p565k	1,524,044	87,213	65	2:59h	0	2:24h	80%
p1330k	2,464,440	237,352	496	11:34h	207	8:48h	76%

Please note that the achieved speed-ups using our proposed technique base besides the acceleration of the SAT solving process also on a faster SAT instance generation. We observed that on average 10% of the runtime reduction are thanks to an accelerated CNF generation process.

Table IV presents an overview on the experimental results for the transition fault model. Note that neither of both approaches was able to finish ATPG for circuit *p49k* within the given limit of 72 hours. Again, a significant reduction of the number of unclassified faults can be observed. Using our method speeds up the ATPG process for all considered circuit considerably. The runtime could be decreased for circuit *p456k* to only 38% of the traditional approach's runtime and the number of unclassified faults was reduced by more than a factor of eleven. For the circuit *p44k*, the number of unclassified fautls has been reduced by more than 95%. As a result, the application of the proposed technique clearly enhances the robustness of the ATPG process for large industrial circuits.

VI. CONCLUSIONS

The contribution of this work is a method to accelerate SAT-based ATPG for industrial designs and strengthen its robustness. It employs BDDs as a canonical data structure for Boolean functions in order to remove redundancy. CNF representations that are derived from BDDs do not contain this redundancy either. This reduces the resulting problem instances with respect to the number of clauses and the amount of CNF variables. Experiments confirm that using our technique considerably speeds up the ATPG process and reduces the number of unclassified faults significantly.

ACKNOWLEDGEMENTS

Parts of this research work were supported by the German Federal Ministry of Education and Research (BMBF) in the Project MAYA under the contract number 01M3172B and by the German Research Foundation (DFG) under contract number DR 287/15-1.

REFERENCES

[1] H. Fujiwara and T. Shimono, "On the acceleration of test generation algorithms", *IEEE Trans. on Computers*, vol. 32, no. 12, pp. 1137–1144, 1983.

[2] M. H. Schulz, E. Trischler, and T. M. Sarfert, "SOCRATES: A highly efficient automatic test pattern generation system", *IEEE Trans. on Computer-Aided Design for Circuits and Systems*, vol. 7, no. 1, pp. 126–137, 1988.

[3] I. Hamzaoglu and J. H. Patel, "New techniques for deterministic test pattern generation", *Jour. of Electronic Testing: Theory and Applications*, vol. 15, no. 1–2, pp. 63–73, 1999.

[4] R. Drechsler, S. Eggersglüß, G. Fey, A. Glowatz, F. Hapke, J. Schloeffel, and D. Tille, "On acceleration of SAT-based ATPG for industrial designs", *IEEE Trans. on Computer-Aided Design for Circuits and Systems*, vol. 27, no. 7, pp. 1329–1333, 2008.

[5] A. Czutro, I. Polian, M. Lewis, P. Engelke, S. M. Reddy, and B. Becker, "TIGUAN: Thread-parallel integrated test pattern generator utilizing satisfiability analysis", in *Proc. of the Int'l Conf. on VLSI Design*, 2009, pp. 227–232.

[6] J. P. Marques-Silva and K. A. Sakallah, "GRASP: A search algorithm for propositional satisfiability", *IEEE Trans. on Computers*, vol. 48, no. 5, pp. 506–521, 1999.

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

[8] N. Eén and N. Sörensson, "An extensible SAT solver", in *Proc. of the Int'l Conf. on Theory and Applications of Satisfiability Testing*, 2004, vol. 2919 of *Lecture Notes in Computer Science*, pp. 502–518.

[9] T. Larrabee, "Test pattern generation using Boolean satisfiability", *IEEE Trans. on Computer-Aided Design for Circuits and Systems*, vol. 11, no. 1, pp. 4–15, 1992.

[10] R. E. Bryant, "Graph-based algorithms for boolean function manipulation", *IEEE Trans. on Computers*, vol. 35, no. 8, pp. 677–691, 1986.

[11] M. A. Breuer and A. D. Friedman, *Diagnosis & reliable design of digital systems*, Computer Science Press, 1976.

[12] J. A. Waicukauski, E. Lindbloom, B. K. Rosen, and V. S. Iyengar, "Transition fault simulation", *Proc. of the IEEE Design & Test of Computers*, vol. 4, no. 2, pp. 32–38, 1987.

[13] P. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Combinational test generation using satisfiability", *IEEE Trans. on Computer-Aided Design for Circuits and Systems*, vol. 15, no. 9, pp. 1167–1176, 1996.

[14] J. P. Marques-Silva and K. A. Sakallah, "Robust search algorithms for test pattern generation", in *Proc. of the Int'l Symp. on Fault-Tolerant Computing*, 1997, pp. 152–157.

[15] G. S. Tseitin, "On the complexity of derivation in the propositional calculus", *Studies in Constructive Mathematics and Mathematical Logic*, vol. Part II, pp. 115–125, 1968.

[16] R. Drechsler, S. Eggersglüß, G. Fey, and D. Tille, *Test Pattern Generation using Boolean Proof Engines*, Springer, 2009.

[17] R. Drechsler and D. Sieling, "Binary decision diagrams in theory and practice", *Software Tools for Technology Transfer*, vol. 3, no. 2, pp. 112–136, 2001.

[18] G. Cabodi, S. Nocco, and S. Quer, "Improving SAT-based bounded model checking by means of BDD-based approximate traversals", *Jour. of Universal Computer Science*, vol. 10, no. 12, pp. 1693–1730, 2004.

[19] G. Fey and R. Drechsler, "Minimizing the number of paths in BDDs: Theory and algorithm", *IEEE Trans. on Computer-Aided Design for Circuits and Systems*, vol. 25, no. 1, pp. 4–11, 2006.

[20] M. N. Velev, "Efficient translation of boolean formulas to CNF in formal verification of microprocessors", in *Proc. of the ASP Design Automation Conf.*, 2004, pp. 310–315.

[21] N. Eén and A. Biere, "Effective preprocessing in SAT through variable and clause elimination", in *Proc. of the Int'l Conf. on Theory and Applications of Satisfiability Testing*, 2005, vol. 3569 of *Lecture Notes in Computer Science*, pp. 61–75.

[22] N. Eén, A. Mishchenko, and N. Sörensson, "Applying logic synthesis for speeding up SAT", in *Proc. of the Int'l Conf. on Theory and Applications of Satisfiability Testing*, 2007, pp. 272–286.

[23] J. Raik and R. Ubar, "Feasibility of structurally synthesized BDD models for test generation", in *Proc. of the IEEE European Test Works.*, 1998, pp. 145–146.

[24] P. Goel, "An implicit enumeration algorithm to generate tests for combinational logic", *IEEE Trans. on Computers*, vol. 30, no. 3, pp. 215–222, 1981.

[25] A. Kuehlmann and F. Krohm, "Equivalence checking using cuts and heaps", in *Proc. of the Design Automation Conf.*, 1997, pp. 263–268.

[26] K. M. Butler and M. R. Mercer, *Assessing Fault Model and Test Quality*, Kluwer Academic Publishers, 1992.

[27] F. Somenzi, "Efficient manipulation of decision diagrams", *Software Tools for Technology Transfer*, vol. 3, no. 2, pp. 171–181, 2001.

[28] F. Corno, M. Sonza-Reorda, and G. Squillero, "RT-level ITC 99 benchmarks and first ATPG results", in *Proc. of the IEEE Design & Test of Computers*, 2000, pp. 44–53.