Increasing Robustness of SAT-based Delay Test Generation using Efficient Dynamic Learning Techniques

Stephan Eggersglüß Rolf Drechsler
Institute of Computer Science, University of Bremen
Bibliothekstr. 1, 28359 Bremen, Germany
{segg, drechsle}@informatik.uni-bremen.de

Abstract

Due to the increased speed in modern designs, testing for delay faults has become an important issue in the post-production test of manufactured chips. A high fault coverage is needed to guarantee the correct temporal behavior. Today's ATPG algorithms have difficulties to reach the desired fault coverage due to the high complexity of modern designs. In this paper, we describe how to efficiently integrate the reuse of learned information into state-of-the-art SAT-based ATPG algorithms and, by this, reduce the number of unclassified faults significantly. For further reduction, a post-classification phase is presented. Experimental results for ATPG for delay faults on large industrial circuits show the robustness and feasibility of the approach.

1. Introduction

An important task in the design flow is the post-production test of each manufactured chip. Test patterns generated by *Automatic Test Pattern Generation* (ATPG) algorithms are applied to ensure that no erroneous chip is delivered to customers. However, classical ATPG algorithms like FAN [1] or SOCRATES [2] have difficulties to cope with the increased complexity of modern circuits. A high fault coverage is needed – in particular for designs in safety-critical application, e.g. in the field of automotive electronics.

Moreover, due to the increasing speed in modern chip design, testing of delay faults becomes more and more important. For delay testing, at least two time frames have to be considered. ATPG for delay fault models is computationally more complex than e.g. ATPG for stuck-at faults. Therefore, it needs generally more run time and results in a larger number of unclassified faults, i.e. the fault coverage is typically lower than e.g. the fault coverage for stuck-at faults.

Recently, ATPG algorithms based on *Boolean Satisfiability* (SAT) [3]–[6] turned out to be an efficient complement to existing classical ATPG algorithms. SAT-based algorithms do not work on a netlist but on a Boolean formula represented in *Conjunctive Normal Form* (CNF). Due to the homogeneity of the CNF, efficient techniques such as *Boolean Constraint Propagation* (BCP) [7], non-chronological backtracking and conflict-based learning [8] can be applied to speed up the search. Especially due to the use of conflict clauses, SAT-based ATPG algorithms are very robust for hard-to-test faults and can classify many faults for which classical ATPG algorithms cannot find a solution in reasonable time [6]. However, many faults still remain unclassified during delay test generation.

1.1. Previous Work

A promising concept to reduce the number of unclassified faults is the reuse of *pervasive conflict clauses* which was introduced for ATPG for stuck-at faults in [4]. Pervasive conflict clauses depend only on the circuit's function and are fault independent. In [4], only the plain concept was introduced and no results concerning the benefit were given.

An incremental SAT engine was proposed in [9]. In this work, the SAT solver is never released and conflict clauses are shared between subsequent similar SAT instances. This concept was applied for stuck-at test generation in [10]. Furthermore, a circuit-based learning strategy based on stuck-at fault partitioning was proposed. However, the use of fault dropping in industrial practice decreases the benefit significantly. Moreover, the computational overhead caused by the use of an external database restricts the amount of learned information.

The reuse of learned information for SAT-based ATPG for *Path Delay Faults* (PDF) was presented in [11]. However, the reuse is based on the time consuming task of unsatisfiable core extraction to identify path segments which cannot sensitized together.

1.2. Contribution

In this paper, we consider the problem of delay test generation and present efficient techniques to prune the search space by information learned during ATPG for previous faults. Due to a tight integration into a SAT-based ATPG algorithm, dynamically learned conflict clauses can be efficiently passed from one target fault to another and, by this, reduce the number of unclassified faults significantly. This approach works in an incremental manner but differs from incremental SAT, because subsequent SAT instances do not have to be similar to exploit the advantages.

Keeping all learned information causes a large memory overhead. Therefore, different strategies to identify unimportant clauses are presented to achieve a good trade-off between the memory consumption and the number of unclassified faults. Furthermore, we propose the use of a *Post-Classification Phase* (PCP). In particular, in this phase, faults are classified which were aborted in the beginning of the ATPG run. Experimental results for transition delay and path delay test generation on large industrial circuits show the robustness and feasibility of the proposed approach.

The rest of the paper is structured as follows: Section 2 briefly presents the basics of SAT-based delay test generation and introduces pervasive conflict clauses. The efficient integration of dynamic learning is presented in Section 3.

The concept of a PCP is proposed in Section 4. Section 5 introduces different dynamic learning strategies for experimental evaluation and Section 6 presents the experimental results. Conclusions are drawn in the last section.

2. Preliminaries

In Section 2.1, the most common delay fault models are introduced. Section 2.2 deals with the basics of SAT-based ATPG. The concept of pervasive conflict clauses is presented in Section 2.3.

2.1. Delay Fault Models

The most accurate delay fault model is the *Path Delay Fault Model* (PDFM) [12]. It captures small as well as large delay defects on a path from an input to an output.

The target of ATPG is to generate a test that initializes the desired transition and sensitizes the off-path inputs of p according to a sensitization criterion. By this, the transition is propagated to an output (observation point). An off-path input is not located on p but drives a gate g which is located on p. Two time frames t_1, t_2 have to be considered during ATPG in order to initialize and launch a transition.

The sensitization criterion defines the quality of the generated test pattern. Test patterns generated with the *non-robust* sensitization criterion guarantee the detection of the PDF if no other delay fault is present. The *robust* sensitization criterion is superior to the non-robust sensitization criterion because it guarantees the detection of a delay fault if other delay faults occur in the circuit. Robust test patterns are harder to obtain than non-robust test patterns. For more details about sensitization criteria, we refer to [13].

The major drawback of the PDFM is the exponential number of paths in modern circuits. Testing all paths is therefore not applicable in practice. Usually, a small number of critical paths is extracted for ATPG.

The *Transition Delay Fault Model* (TDFM) [14] is less accurate than the PDFM. Here, the delay on a connection is assumed to be large enough that it can be observed along any sensitized sub-path from the fault site to an output. The number of faults is linear in the number of connections. Consequently, it provides a better fault coverage in practice than the PDFM and is therefore widely used in industry.

2.2. SAT-based ATPG

To apply a SAT solver to a circuit-oriented problem, e.g. ATPG, the problem is formulated as a Boolean formula in CNF. A CNF Φ in m Boolean variables is a conjunction of n clauses. Each clause is a disjunction of literals. A literal is a Boolean variable (x) or its complement ($\overline{x}$). The CNF Φ is *satisfied* if all clauses are satisfied. A clause is satisfied if at least one literal of the clause is satisfied. The CNF Φ is said to be *unsatisfiable* iff no solution can be found that satisfies Φ. The task of a SAT solver for a given Φ is to find a satisfying assignment or to prove that no such assignment exists.

In the following, the circuit-to-CNF conversion is briefly described. More information can be found in [15]. A Boolean variable is assigned to each connection in circuit C. The CNF Φ_g for each gate g in C is derived from the characteristic function which can be constructed using the truth table. The CNF Φ_C representing the circuit's function is then constructed by the conjunction of the CNFs of all gates $g_1, \ldots, g_n \in C$:

$$\Phi_C = \prod_{i=1}^{n} \Phi_{g_i}$$

If two time frames have to be considered, two Boolean variables – one for each time frame – have to be assigned to each connection and the CNF for each gate has to be extracted for both time frames. Here, the variables of the corresponding time frame have to be used. This procedure and the correct modeling of flipflops is described in detail in [16]. If additional values, e.g. unknown or static values, are needed, a multiple-valued logic and a corresponding Boolean encoding have to be used; for further details see e.g. [6], [17].

The Φ_C has to be extended by the fault-specific constraints Φ_F for generating a test for fault F. Only a small necessary part of the circuit has to be included in the CNF (Φ_C^F) for reasons of efficiency. For example, for the PDFM, only the support of the output of the path is contained in Φ_C^F. More formally, a test for F is generated by evaluating the following formula:

$$\Phi_{test}^F = \Phi_C^F \cdot \Phi_F$$

2.3. Pervasive Conflict Clauses

The generation of additional implications, i.e. learning, to speed up ATPG was first proposed in [2] and have been improved continuously thereafter, e.g. in [18]. State-of-the-art SAT solvers, e.g. [7], [8], [19], [20], employ an inherit learning feature. They benefit significantly from the use of conflict clauses. If a conflict occurs during the search, the conflict is analyzed and a conflict clause is generated, i.e. learning is performed. In a circuit-oriented problem, a conflict clause corresponds to a conflicting value assignment of connections. The recorded conflicts can be used to derive additional implications efficiently.

Conflict clause generation is done by resolution of those clauses which are responsible for the conflict. Such a clause avoids that the SAT solver reenters this non-solution search space again. By this, large parts of the search space are pruned and the search is accelerated. A procedure to generate powerful conflict clauses is presented in detail in [21].

As stated in Section 2.2, the CNF is composed of two parts: the circuit part and the fault-specific part. Conflict clauses which have their source only in clauses from the circuit part are *fault independent* and are called pervasive conflict clauses [4] (or pervasive clauses in the following). They can be reused in subsequent SAT instances to prune search space without running in the conflict first. On the other hand, conflict clauses which are generated using at least one clause from the fault-specific part are *fault dependent*. Those have to be discarded after solving the CNF.

3. Integration of Dynamic Learning

In this section, it is shown how dynamic learning can be efficiently integrated in SAT-based ATPG algorithms. An external database is used in [10]. The pervasive clauses are

extracted after each run and transformed into a gate level representation. Each time a new SAT instance is built, each clause in the database is checked whether is can be reused for the current SAT instance. If it can be reused, the clause is transformed from the gate level representation to CNF. The transformation steps and the check procedure took much run time and limited the amount of learned information. In contrast, we propose the use of an internal database in combination with an efficient watch-list strategy to overcome this disadvantage.

Crucial for an efficient transfer of pervasive clauses is a permanent variable assignment to connections[1]. Due to a permanent variable assignment, all pervasive clauses can be stored in the internal database "as is". Thus, expensive transformation steps are avoided. Newly learned clauses can be stored efficiently inside the SAT solver by using pointers.

3.1. Pervasive Conflict Clause Identification

In this section, it is described how pervasive clauses can be identified. When building the SAT instance for fault F, it is easily possible to distinguish between fault independent clauses (Φ_C^F) and fault dependent clauses (Φ_F). All learned clauses that have their source in at least one fault dependent clause have to be discarded and cannot generally be reused for subsequent SAT instances. Two methods are used:

- *ID identification* – The variable with the highest ID is defined as a fixed upper limit u after the permanent variable assignment has been done. If new variables have to be defined for Φ_F, e.g. describing faulty circuitry, only variables with an ID larger than u are used. Thus, if a learned clause contains at least one variable with an ID larger than u, the clause is discarded.
- *Literal identification* – The literal λ is added to each fault dependent clause c that do not use only new – fault dependent – variables. The literal λ is set to false by an *incremental assumption* [20] before each run. By this, c keeps the original meaning – λ is redundant. Because λ is only used in one phase, each conflict clause derived from c contains also λ and can easily be identified.

In summary, each learned clause $c = (l_1 + \ldots + l_n)$ is checked after each run and discarded if the following holds:

$$\bigvee_{i=1}^{n}((l_i = \lambda) \vee (\mathrm{ID}(l_i) > u))$$

Otherwise, c is a pervasive clause.

3.2. Variable-based Activation

Not all stored clauses should be reused in each SAT instance. Subsequent SAT instances may target faults from other regions of the circuit. A pervasive clause learnt from one part of the circuit is useless if that part is not included in the current SAT instance. Due to the large number of pervasive clauses in industrial designs, checking each single clause is not feasible and may outweigh the benefit. Therefore, the concept of *variable-based activation* is introduced. Here, the internal database is modeled as a watch-list (similar to the watch-list used for fast BCP presented in [7]).

1. However, this causes some overhead in the search algorithm of the SAT solver which can be adjusted by a slight modification.

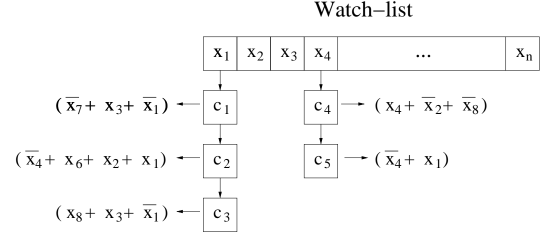

Figure 1. Illustration of a watch-list example

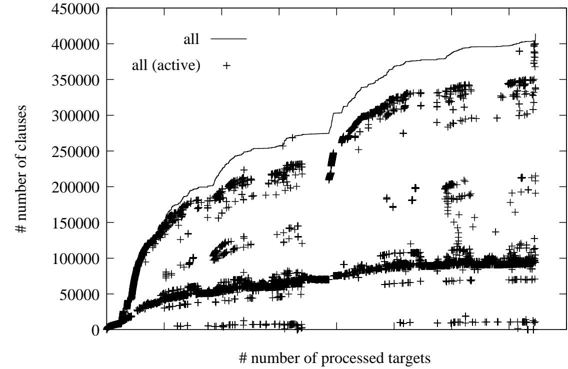

Figure 2. Progress of (active) learned clauses - p57k transition faults

A list of clauses is assigned to each variable x_i of the circuit. These clauses are said to be "watched" by x_i. When a new pervasive clause p is learned, one variable x_q contained in p is chosen. Then, p is added to the list of x_q, i.e. p is "watched" by x_q. In the following, the set of clauses which is watched by variable x_q is denoted by $\omega(x_q)$. An example watch-list is illustrated in Figure 1. The variables are denoted by $x_1, \ldots, x_n$, and $c_1, \ldots, c_5$ are pervasive clauses.

By this, the set of pervasive clauses for Φ_{test}^F can be efficiently determined. All learned clauses which are "watched" by variables contained in Φ_{test}^F can be added directly to Φ_{test}^F. In other words, when the CNF Φ_g for gate g with output variable v_g is added to Φ_{test}^F, the "watched" clause set $\omega(v_g)$ is added, too. The clauses of $\omega(v_g)$ are said to be "active" for Φ_{test}^F. Those clauses which come from other parts of the circuits do not have to be considered and, therefore, cause no run time overhead. These clauses are "inactive" for Φ_{test}^F.

However, a learned clause $p = (l_1 + \ldots + l_n)$ is only useful for Φ_{test}^F if all variables are contained in Φ_{test}^F. Otherwise, at least one variable remains unconstrained and can satisfy p. Nonetheless, it is sufficient to let a learned clause be "watched" by one variable. This is because the overhead of determining whether all variables are contained in Φ_{test}^F is higher than adding some "useless" clauses to Φ_{test}^F.

4. Post-Classification Phase

In this section, the use of a post-classification phase is motivated and described. Consider the diagram shown in Figure 2. In this diagram, the progress of the number of learned clauses of ATPG for TFs for an industrial design is shown. On the abscissa, the progress in terms of targeted faults is denoted. The number of learned clauses is presented on the ordinate. The upper line shows the total number

Table 1. Number of Learned Clauses

	Path Delay Fault Model						Transition Delay Fault Model					
Circ	paths	all	Act 1/4	Act 1/2	DL 3	DL 20	targets	all	Act 1/4	Act 1/2	DL 3	DL 20
b14	1,666	2,855	1,222	1,746	1,097	2,314	20,163	22,335	11,985	16,978	9,143	21,324
b15	3,696	1,862	1,193	1,777	890	1,803	25,087	25,349	10,831	17,553	4,730	18,555
b17	10,000	13,903	6,955	9,343	4,910	12,456	80,996	90,580	39,050	61,327	21,340	63,530
b18	10,000	18,217	10,187	14,465	6,921	17,786	301,012	670,081	315,590	536,267	103,238	367,080
p44k	10,000	7,576	8,274	8,770	7,925	7,214	65,484	93,077	56,478	73,486	31,475	92,567
p57k	10,000	816,963	505,244	663,183	83,921	485,227	14,921	414,227	180,443	287,797	51,456	232,409
p77k	10,000	0	0	0	0	0	180,679	620,815	434,153	488,908	98,757	560,056
p80k	10,000	587,137	252,765	400,175	73,913	369,813	16,729	120,645	47,636	71,298	24,550	98,252
p88k	10,000	61,267	33,013	46,976	22,469	60,838	56,024	63,340	35,504	48,397	22,861	55,609
p99k	10,000	119,056	50,751	77,790	17,267	86,254	41,668	292,777	97,156	173,223	34,603	171,128
p177k	10,000	129,261	66,103	91,973	28,251	120,386	99319	2,084,080	904,505	1,355,935	193,429	912,423
p456k	10,000	112,475	53,527	77,899	23,864	74,926	158,595	4,511,460	2,548,415	3,122,967	436,570	1,623,043
p462k	10,000	13,734	7,093	9,930	4,626	13,363	319,483	234,315	140,584	187,110	108,247	215,139
p565k	10,000	169,865	74,965	122,033	14,454	100,210	107,787	930,337	568,341	741,698	185,040	583,180
p1330k	10,000	8,007	4,274	6,261	965	6,369	269,425	233,479	106,462	158,285	72,438	187,938
total		100%	52.2%	74.3%	14.1%	65.9%		100%	53%	70.5%	13.4%	50.0%

of learned clauses and the crosses presents the number of "active" clauses for each target fault.

The diagram clearly shows that faults targeted at the beginning of the ATPG run have access to a significant smaller number of learned clauses than faults targeted at the end of the ATPG run. Furthermore, our experiments have shown that – when using dynamic learning – most of the remaining unclassified faults appear at the beginning of the ATPG run. The ordering of the faults clearly has an influence on the classification. The effect can be mitigated by using a post-classification phase. The post-classification phase starts after all faults were targeted. In this phase, all faults that were aborted in the ATPG run, are targeted again. The benefit is that – in most cases – the search algorithm has access to significantly more learned clauses than in the previous ATPG call for these faults. Therefore, it is likely that many faults that were aborted in the previous phase can now be classified. Optionally, in this phase, the time limit can be decreased to save run time.

5. Learning Strategies

SAT solvers generate a large number of conflict clauses during their search. Keeping all pervasive clauses results in a large memory overhead. Furthermore, the efficiency of the BCP routine of the SAT solver can be decreased by too many clauses. The clause size was proposed as a selection criterion in [4]. Any clause that contains more literals than a user-defined maximum is discarded. In [10], the maximum size was set to only three literals. In this section, an additional selection criterion is introduced and the selection criteria chosen for experimental evaluation are presented.

State-of-the-art SAT solvers have an inherit feature to judge the importance of a learned conflict clause [19]. The importance of a learned clause is defined over the frequency of being involved in conflicts. Therefore, all learned conflict clauses have an *activity value* [19], which is increased each time a conflict clause is considered in the conflict analysis. Conflict clauses of lesser importance, i.e. with a low activity value, are discarded in intervals. This feature can be exploited as an additional selection criterion when using dynamic learning. The following strategies are chosen for experimental evaluation:

- *all* – After each run, all learned pervasive clauses are stored in the internal database and kept for subsequent SAT instances.
- *Act 1/4* – The learned pervasive clauses are ordered according to their activity values and the quarter having the highest activity value is stored in the internal database.
- *Act 1/2* – The learned pervasive clauses are ordered according to their activity values and the half part having the highest activity value is stored.
- *DL 3* – All learned pervasive clauses with at most three literals are kept for subsequent SAT instances (as done in [10]).
- *DL 20* – All learned pervasive clauses with at most 20 literals[2] are kept for subsequent SAT instances.

6. Experimental Results

In this section, the experimental results for publicly available benchmarks (ITC'99) as well as for industrial circuits provided by NXP Semiconductors are presented. All experiments were carried out on a AMD 64-Bit Opteron (32768 MB, 2,8 GHz, GNU/Linux). A modified version of MiniSat v1.14 [20] was used as SAT solver.

6.1. Dynamic Learning Strategies

Experiments were performed for the PDFM (robust sensitization) and for the TDFM. The reader is referred to [16] and [17], respectively, for the concrete SAT modeling. A set of 10,000 paths with a length of at least 50 gates are chosen for PDF testing. At most 15 paths start from the same input to target different parts of the circuit. For the TDFM, all possible faults are targeted. Here, fault dropping is enabled during test generation. Table 1 shows the number of learned clauses for each strategy. The name of the circuit is given in column *Circ*. For the industrial circuits, the name denotes roughly their size. For example, p1330k contains over 1.3 million elements. Furthermore, the number of targeted paths (column *paths*) and targeted faults of strategy *all* (column *targets*) are presented. The average relative number of all learned clauses is given in the last row. It is notable that strategy *DL 3* keeps only a small number of clauses compared to the other strategies. The reason why no learned clauses are kept for p77k (PDFM) is that the untestability of each path was trivial to detect.

2. The number 20 was chosen due to the distribution of learned clause sizes in experimental evaluations. Preliminary studies have shown that this size is more beneficial than a size of 10 as used in [4].

Table 2. Experimental Results - Robust Path Delay Test Generation

	w/o DL		all			Act 1/4			Act 1/2			DL 3			DL 20		
Circ	Ab.	Time	Mem	Ab.	Time	Mem	Ab.	Time	Mem	Ab.	Time	Mem	Ab.	Time	Mem	Ab.	Time
b14	0	0:50m	+3M	0	0:56m	+3M	0	0:57m	+3M	0	0:57m	+3M	0	0:58m	+3M	0	0:55m
b15	0	1:52m	+0M	0	1:47m	+0M	0	1:55m	+0M	0	1:50m	+0M	0	1:49m	+3M	0	1:47m
b17	0	4:47m	+2M	0	5:15m	+1M	0	5:23m	+1M	0	5:16m	+1M	0	5:21m	+1M	0	5:13m
b18	6	5:39m	+4M	0	5:54m	+3M	0	6:03m	+3M	0	6:01m	+2M	0	5:59m	+3M	0	5:57m
p44k	0	1:20h	+0M	0	20:03m	+1M	0	21:30m	+1M	0	20:46m	+1M	0	24:56m	+0M	0	19:54m
p57k	1539	49:56m	+159M	93	1:22h	+96M	382	1:14h	+131M	191	1:19h	+9M	927	56:41m	+62M	108	52:08m
p77k	0	1:00m	+0M	0	1:03m	+0M	0	1:03m	+0M	0	1:02m	+0M	0	1:03m	+0M	0	1:03m
p80k	547	17:11m	+117M	140	20:08m	+40M	248	17:10m	+74M	207	18:23m	+8M	335	16:28m	+45M	155	17:36m
p88k	0	7:58m	+7M	0	6:56m	+4M	0	7:17m	+5M	0	7:02m	+2M	0	7:05m	+7M	0	6:52m
p99k	34	4:46m	+16M	3	4:26m	+5M	5	4:22m	+9M	3	4:23m	+0M	4	4:25m	+8M	5	4:21m
p177k	223	32:28m	+23M	16	31:25m	+16M	79	37:31m	+19M	41	34:22m	+11M	177	35:07m	+20M	21	31:29m
p456k	171	30:19m	+14M	16	27:16m	+6M	27	27:12m	+9M	18	27:17m	+3M	35	28:20m	+6M	7	26:34m
p462k	0	8:53m	+3M	0	9:10m	+2M	0	9:12m	+2M	0	9:11m	+2M	0	9:13m	+3M	0	9:11m
p565k	115	9:56m	+32M	31	9:37m	+10M	47	9:26m	+20M	41	9:32m	+2M	82	9:31m	+10M	32	9:20m
p1330k	0	19:47m	2M	0	20:39m	+2M	0	20:39m	+2M	0	21:34m	+0M	0	20:47m	+2M	0	20:40m
total	2635	4:36h	+382M	299	4:06h	+189M	788	4:03h	+279M	501	4:06h	+44M	1560	3:46h	+173M	328	3:33h

Table 2 reports the results for the PDF test generation. The results for the TDFM are shown in Table 3. The number of faults that could not be classified within the time interval of seven MiniSat restarts[3] are presented in columns entitled *Ab*. Columns named *Time* present the overall run time for ATPG. The memory overhead compared to the version without dynamic learning (column *w/o DL*) is shown in columns *Mem*. Time is given either in CPU minutes (m) or CPU hours (h). Memory overhead is presented in Megabyte (M). The results are summed up for all circuits in the last row.

The results show that the number of unclassified faults can be significantly reduced by the proposed use of dynamic learning for both fault models. Keeping all learned clauses reduced the number of aborted faults for all circuits to only 11% for the PDFM and to only 0.8% for the TDFM.

Especially the reduction for the hard-to-test circuits p177k and p456k (TDFM) is remarkable. However, the disadvantage of strategy *all* is the high memory consumption. The circuit p456k (TDFM) needs for example over 1GB of additional memory. In contrast, the amount of additional memory used by strategy *DL 3* is negligible. However, the number of unclassified faults is increased by a factor of five (PDFM) and by a factor of 20 (TDFM) compared to strategy *all*. The activity-based strategies *Act 1/4, Act 1/2* have less aborts than strategy *DL 3*, but they suffer also from their high memory consumption, especially for the TDFM. See for explanation the overall number of learned clauses presented in Table 1.

The best trade-off between the number of unclassified faults and the memory consumption can be achieved by strategy *DL 20*. Here, only 1.5% of the number of unclassified faults of *w/o DL* remain unclassified for the TDFM. The number is – compared to strategy *all* – also only slightly increased for the PDFM (12%). At the same time, the memory consumption of *DL 20* can be significantly reduced. Although the number of unclassified faults is higher than for strategy *Act 1/2* for the TDFM, strategy *DL 20* should be preferred due to the smaller memory consumption. The run time can also be decreased as a side effect of this strategy.

In summary, the strategy *DL 20* is the first choice for both delay fault models. If only few memory resources are available, a fair reduction can be achieved by using *DL 3*.

3. A restart is defined as a certain number of conflicts (100 at the beginning). After each restart, this number is increased (by 50%).

6.2. Post-Classification Phase

The experimental results of the proposed post-classification phase are presented in Table 4. Due to page limitation, results are only reported for the strategies *all* and *DL 20* and only the industrial circuits are considered. The post-classification phase is not applied for the ITC'99 benchmarks, because no aborts were produced. For comparison, column *w/o DL* shows the number of unclassified faults without dynamic learning. Column entitled *Time* presents the additional time needed for the post-classification phase. The number of faults that were aborted in the previous phase but could be classified during the post-classification phase is given in column *Diff*. Column *Ab.* shows the overall number of unclassified faults after the post-classification phase.

Roughly 4% of the ATPG run time is needed for the post-classification phase for all circuits for the PDFM. For the TDFM, the percentage is only 2% of the ATPG run time. About 80% of the remaining unclassified PDFs can be classified in this phase in total. For the TDFM, still 43% of the previously unclassified faults can additionally be classified. As a result, the number of unclassified PDFs. decreases to only 2.2% (*DL 20*: 2.8%) of those of *w/o DL*. The number of unclassified TDFs further decreases to only 0.5% (*DL 20*: 1.2%).In summary, the use of the post-classification phase can further reduce the number of aborted faults significantly. The run time overhead for the post-classification phase is negligible.

7. Conclusions

A high fault coverage for delay faults is needed to ensure that manufactured chips meet their timing specification. This requires robust ATPG algorithms producing only very few aborts. In this paper, we presented how dynamic learning techniques can be efficiently integrated into SAT-based ATPG algorithms. Dynamically learned information can easily be passed to subsequent SAT instances by the proposed techniques. This prunes the search space for other target faults as well.

Different learning strategies were presented to reduce the high memory overhead. As a result, a good trade-off between run time, memory consumption and robustness of the ATPG algorithm is achieved. Experiments on publicly available benchmarks and on industrial circuits show that

Table 3. Experimental Results - Transition Delay Test Generation

	w/o DL		all			Act 1/4			Act 1/2			DL 3			DL 20		
Circ	Ab.	Time	Mem	Ab.	Time	Mem	Ab.	Time	Mem	Ab.	Time	Mem	Ab.	Time	Mem	Ab.	Time
b14	0	3:03m	+2M	0	2:57m	+1M	0	2:46m	+2M	0	2:53m	+1M	0	2:52m	+2M	0	2:55m
b15	0	3:56m	+4M	0	3:59m	+1M	0	3:43m	+2M	0	3:48m	+0M	0	3:38m	+2M	0	3:50m
b17	0	17:52m	+15M	0	16:50m	+5M	0	15:52m	+9M	0	16:08m	+1M	0	15:56m	+6M	0	16:07m
b18	3	1:53h	+132M	0	2:36h	+52M	0	2:07h	+101M	0	2:36h	+5M	0	1:48h	+37M	0	2:02h
p44k	59	5:35h	+17M	0	1:22h	+9M	0	1:20h	+13M	0	1:20h	+1M	2	3:53h	+8M	0	1:29h
p57k	275	1:03h	+79M	6	41:10m	+26M	13	34:41m	+49M	10	37:15m	+1M	13	34:18m	+21M	11	33:43m
p77k	5421	1:18h	+30M	0	27:49m	+3M	0	27:39m	+9M	0	26:46m	+22M	6690	1:57h	+18M	0	26:58m
p80k	30	11:26m	+17M	4	7:48m	+4M	4	8:10m	+8M	6	7:52m	-2M	6	7:57m	+9M	6	7:45m
p88k	0	34:02m	+6M	0	27:13m	+2M	0	26:15m	+3M	0	26:26m	+0M	0	26:47m	+4M	0	27:04m
p99k	81	22:41m	+49M	9	16:39m	+10M	11	15:05m	+27M	13	15:46m	+1M	36	18:06m	+16M	32	15:46m
p177k	23422	85:26h	+556M	14	38:03h	+217M	48	24:08h	+338M	22	27:56h	+11M	1652	32:23h	+92M	241	21:59h
p456k	35970	16:12h	+1184M	488	19:17h	+611M	1049	15:52h	+821M	516	15:28h	-1M	2120	7:59h	+135M	721	8:34h
p462k	554	6:26h	+9M	2	5:27h	+1M	12	5:36h	+5M	9	5:31h	-1M	58	5:44h	+4M	3	5:25h
p565k	638	4:45h	+140M	14	4:36h	+70M	126	4:45h	+101M	52	4:37h	+4M	71	4:32h	+43M	12	4:33h
p1330k	3	10:50h	+29M	0	10:29h	+10M	0	10:23h	+17M	0	10:26h	+4M	3	10:30h	+15M	2	10:32h
total	66456	128:23h	+2269M	537	84:15h	+1022M	1263	66:26h	+1505M	628	70:11h	+47M	10651	70:36h	+412M	1028	56:50h

Table 4. Experimental Results - Post-Classification Phase

	Path Delay Fault Model							Transition Delay Fault Model						
	w/o DL	all			DL 20			w/o DL	all			DL 20		
Circ	Ab.	Time	Diff	Ab.	Time	Diff	Ab.	Ab.	Time	Diff	Ab.	Time	Diff	Ab.
p44k	0	–	–	0	–	–	0	59	–	–	0	–	–	0
p57k	1539	+4:00m	-89	4	+3:14m	-99	9	275	+0:09m	-1	5	+0:12m	-2	9
p77k	0	–	–	0	–	–	0	5421	–	–	0	–	–	0
p80k	547	+5:07m	-98	42	+4:08m	-101	54	30	+0:16m	0	4	+0:19m	-2	4
p88k	0	–	–	0	–	–	0	0	–	–	0	–	–	0
p99k	34	+0:02m	-2	1	+0:02m	-3	2	81	+0:02m	-8	1	+0:11m	-6	26
p177k	223	+0:40m	-16	0	+0:50m	-21	0	23422	12:14m	-7	7	+16:38m	-27	214
p456k	171	+0:12m	-13	3	+0:06m	-6	1	35970	+1:16h	-201	287	+37:36m	-200	521
p462k	0	–	–	0	–	–	0	554	+0:31m	-1	1	+0:37m	-2	1
p565k	115	+0:26m	-23	8	+0:21m	-25	7	638	+0:06m	-13	1	+0:04m	-12	0
p1330k	0	–	–	0	–	–	0	3	–	–	0	+0:01m	-2	0
total	2629	+10:27m	-241	58	+8:41m	-255	73	66453	+1:29h	-231	306	+55:38m	-253	775

the integration can reduce the number of aborts significantly. Therefore, the proposed approach is well suited to cope with the high delay fault coverage demands of the chip industry.

Acknowledgment

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

References

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

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

[3] P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli, "Combinational test generation using satisfiability," *IEEE Trans. on Comp.-Aided Design for Circ. and Syst.*, vol. 15, pp. 1167–1176, 1996.

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

[5] E. Gizdarski and H. Fujiwara, "SPIRIT: A highly robust combinational test generation algorithm," *IEEE Trans. on Comp.-Aided Design for Circ. and Syst.*, vol. 21, no. 12, pp. 1446–1458, 2002.

[6] 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 Comp.-Aided Design for Circ. and Syst.*, vol. 27, no. 7, pp. 1329–1333, 2008.

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

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

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

[10] G. Fey, T. Warode, and R. Drechsler, "Reusing learned information in SAT-based ATPG," in *Proc. Int'l Conf. on VLSI Design*, 2007, pp. 69–76.

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

[12] G. Smith, "Model for delay faults based upon paths," in *Proc. Int'l Test Conf.*, 1985, pp. 342–349.

[13] A. Krstić and K.-T. Cheng, *Delay Fault Testing for VLSI Circ.*. Kluwer Academic Publishers, Boston, MA, 1998.

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

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

[16] S. Eggersglüß, D. Tille, G. Fey, R. Drechsler, A. Glowatz, F. Hapke, and J. Schloeffel, "Experimental studies on SAT-based ATPG for gate delay faults," in *Proc. Int'l Symp. on Multiple-Valued Logic*, 2007.

[17] S. Eggersglüß, G. Fey, R. Drechsler, A. Glowatz, F. Hapke, and J. Schloeffel, "Combining multi-valued logics in SAT-based ATPG for path delay faults," in *Proc. ACM & IEEE Int'l Conf. on Formal Methods and Models for Codesign*, 2007, pp. 181–187.

[18] W. Kunz and D. Pradhan, "Accelerated dynamic learning for test pattern generation in combinational circuits," *IEEE Trans. on Comp.-Aided Design for Circ. and Syst.*, vol. 12, no. 5, pp. 684–694, 1993.

[19] E. Goldberg and Y. Novikov, "BerkMin: a fast and robust SAT-solver," in *Proc. Design, Automation and Test in Europe*, 2002, pp. 142–149.

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

[21] L. Zhang, C. Madigan, M. Moskewicz, and S. Malik, "Efficient conflict driven learning in a Boolean satisfiability solver," in *Proc. Int'l Conf. on Comp.-Aided Design*, 2001, pp. 279–285.