Combining Multi-Valued Logics in SAT-based ATPG for Path Delay Faults

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

Andreas Glowatz Friedrich Hapke Juergen Schloeffel
NXP Semiconductors Germany GmbH
21147 Hamburg, Germany
{andreas.glowatz,friedrich.hapke,juergen.schloeffel}@nxp.com

Abstract

Due to the rapidly growing speed and the decreasing size of gates in modern chips, the probability of faults caused by the production process grows. Already small variations lead to functional failures. Therefore, dynamic fault models like the Path Delay Fault Model (PDFM) have become more important in the last years. At the same time, classical algorithms for test pattern generation reach their limits due to the steadily increasing complexity of modern circuits.

In this work, a SAT-based approach to calculate robust and non-robust test patterns for Path Delay Faults (PDF) is presented. In contrast to previous approaches, the sequential behavior of a circuit is modeled adequately. Moreover, tri-state elements and environment constraints that occur in industrial practice can be handled. The encoding to apply a Boolean SAT solver for this problem is motivated and explained in detail. Experimental results for large industrial circuits show the efficiency of this approach.

1. Introduction

Because of the rapidly growing speed and the decreasing size of gates in modern chips, strict constraints on the physical realization are needed to guarantee a functionally correct chip. Already very small variations in the manufacturing process, which violate the constraints, may cause a functional fault. Such a fault may not influence the combinational function, but infringe the dynamic behavior of the chip at speed, e.g. the propagation of signal values is delayed.

Such delay effects are tested by using dynamic fault models on the gate level representation like the *Path Delay Fault Model* (PDFM). The PDFM checks whether the accumulation of delay along a structural path violates the timing constraints. Testing all structural paths is generally desirable. But due to the very large number of possible paths in a circuit, only a subset is considered in the practical application. Usually, critical and long paths are chosen for test pattern generation.

A test for a *Path Delay Fault* (PDF) propagates a transition along the path that is to be tested. Therefore, the dynamic behavior of the circuit during two time frames has to be considered. Different faults may mask each other; therefore the definition of robustness has been introduced [12, 9, 3]. A robust test detects a PDF independently of the presence or absence of other PDFs. In contrast, when using a non-robust test a fault effect may be masked. In our approach, we consider the definition of [3] that guarantees robustness by forcing static, hazard-free values at side inputs of a path. Robust tests are more desirable but also more complex to generate.

Additionally, the number of gates in modern circuits grows steadily. As a result, classical ATPG algorithms reach their limit. Meanwhile solvers for the *Boolean satisfiability* (SAT) problem have been significantly improved in the last years. This is mainly due to the integration of powerful techniques like dynamic learning, non-chronological backtracking and efficient search heuristics (see e.g. [10, 11, 4]). Based on these engines, very powerful ATPG tools have been developed that show the efficiency of SAT-based ATPG for varying fault models (see e.g [14, 6]).

A first SAT-based approach for generating test patterns for PDFs was presented in [2]. But the dynamic behavior of a circuit during two time frames was not modeled adequately. The application of enhanced SAT-based learning techniques was studied in [8, 1]. But only path sensitization was considered in [8], while the approach in [1] only focused on the classification of non-robust untestable paths.

Robust as well as non-robust test generation was considered in [5]. In this approach, no structural analysis was applied and therefore, the model was less compact. But – as in all other previous approaches – only purely Boolean circuits were handled. Tri-state elements or restrictions from the environment were not modeled.

In this work, we propose a SAT-based approach to compute robust and non-robust test patterns for the PDFM. In contrast to previous SAT approaches, the proposed technique models the sequential behavior and handles constraints that result from industrial applications. Moreover, it can be applied to circuits with tri-state elements.

For this purpose, a number of multi-valued logics is developed. The logic that is applied to model a certain part of a circuit depends on the values that are needed, e.g. a tri-state element is modeled in a different logic than a Boolean gate. This procedure ensures a small size of the Boolean SAT instance. Experimental results on large industrial and ISCAS circuits show the efficiency of the approach.

The paper is structured as follows. In the next section, the PDFM and its formulation in SAT for Boolean circuits are explained. In Section 3, the transformation for PDFs for industrial circuits is shown. Moreover, the multi-valued logic is presented. Section 4 provides experimental results. Conclusions are drawn in Section 5.

2. Preliminaries

2.1. Path Delay Fault Model

The PDFM describes a distributed delay on a path from a (pseudo) primary input to a (pseudo) primary output of a circuit. Formally, a PDF is described by $F = (P, T)$, where $P = (g_1, \dots, g_n)$ is a path from an input g_1 to an output g_n. The type of transition is given by $T \in \{R, F\}$, where R denotes a rising transition and F a falling transition.

To detect a fault, two test vectors v_1, v_2 are needed to propagate a transition along the path P during two consecutive time frames t_1, t_2. Note, that the transition must be inverted after an inverting gate on the path. The initial vector v_1 sets the initial value of the transitions in time frame t_1, whereas the final vector v_2 causes the transition in t_2 at operating speed. In case of a delay fault, the new value cannot be observed at g_n.

The quality of the tests can be classified by the definition of robustness. A test is called *robust* iff it detects the fault independently from other delay faults in the circuit. *Non-robust* tests guarantee the detection of a fault, if there are no other delay faults in the circuit. If there is neither a non-robust nor a robust test, the PDF is untestable.

Robust and non-robust tests differ in the constraints on the off-path inputs of the path as shown in Table 1. The values correspond to the 7-valued logic presented in [2]. The second position in the value marks the signal's value in the final time frame, whereas the first position defines, whether the signal is static (S) – that means *hazard-free* – during two time frames or the signal value is unknown (X).

For example, $S1$ means, that there must be a static 1 on the signal line and $X1$ means, that the value has to be 1 in the final time frame. Applying static values to the off-path inputs avoids that other delay faults on the inputs of the gate may have influence on the value of the output. Therefore, a robust test is also a non-robust test but not vice versa.

This model does not always provide the exact value of a signal line at t_1. Therefore, in case of pseudo primary outputs, the behavior of flip flops cannot be fully modeled.

Table 1. Off-path input constraints

	rising rob.	*falling rob.*	*non-robust*
AND/NAND	$X1$	$S1$	$X1$
OR/NOR	$S0$	$X0$	$X0$

Table 2. Encoding of L_{4B}

L_{4B}	0	01	10	1
x_1	0	0	1	1
x_2	0	1	0	1

2.2. SAT Formulation: Boolean Circuits

To apply a SAT solver to an ATPG problem, the problem must be transformed into *Conjunctive Normal Form* (CNF). A CNF Φ on n binary variables is a conjunction of m clauses. Each clause is a disjunction of literals. A literal is a variable in its positive or negative form.

Each signal line x in a given circuit C is represented by a Boolean variable x and each single gate g in C can be converted into a CNF Φ_g by building the characteristic function of g or truthtable of g, respectively. The CNF for C is therefore the conjunction of the constraints for each gate:

$$\Phi_C = \prod_{g \in C} \Phi_g$$

For the PDFM, two time frames must be considered. The 4-valued logic $L_{4B} = \{0, 01, 10, 1\}$ is used, which represents the value of a signal line x in both time frames. The value 0 (1) describes the value 0 (1) on x in t_1 and t_2, whereas 01 (10) represents the rising (falling) transition.

To apply a SAT solver, the multi-valued signal x must be encoded by two Boolean variables x_1, x_2. A possible encoding for L_{4B} is presented in Table 2. Therefore, the CNF formula of each single gate g is created by duplicating the clauses for one time frame using the respective variables. The CNF of the circuit modeled in L_{4B} is given by Φ_{C4B}.

Additional constraints are added to guarantee the equivalence of the value of a pseudo primary output in t_1 and the value of the corresponding pseudo primary input in t_2. These constraints are described by Φ_t.

Finally, given a fault $F = (P, T)$, a constraint Φ_T forces the transition according to T and a constraint Φ_P sets the off-path inputs to non-controlling values in t_2. The conjunction of all these constraints is the CNF formula Φ_P^{C4B}:

$$\Phi_P^{C4B} = \Phi_{C4B} \cdot \Phi_t \cdot \Phi_T \cdot \Phi_P$$

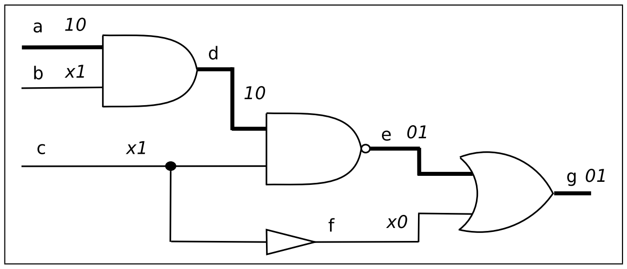

Figure 1. Example circuit C with path P

Any satisfying assignment for Φ_P^{C4B} determines the two test patterns. If the formula is unsatisfiable, the PDF is not testable.

Using this formulation, only non-robust tests can be generated, because forcing a signal to be static cannot be modeled adequately. Modeling static signals is described in Section 3.4. The following example demonstrates the procedure.

Example 1 *A non-robust test for the path $P = (a, d, e, g)$ with a falling edge, shown in Figure 1, should be calculated. For this reason, the circuit must be transformed into the CNF Φ_{C4B} which is derived from the conjunction of the CNFs of all gates modeled with L_{4B}.*

$$\Phi_{C4B} = \Phi_{\text{AND}}^d \cdot \Phi_{\text{NAND}}^e \cdot \Phi_{\text{NOT}}^f \cdot \Phi_{\text{OR}}^g$$

For modeling the fault, Φ_T corresponds to the values on P shown in Figure 1 and can directly be determined by applying the Boolean encoding.

According to Table 1, for a non-robust test, the off-path inputs of P must be set to $X1$ or $X0$, respectively. For example, the signal b has to adopt the value $X1$, i.e. the value 01 or 1 of L_{4B} which is encoded by:

$$\bar{b}_1 b_2 + b_1 b_2 = b_2$$

Subformula Φ_P summarizes all constraints for off-path inputs:

$$\Phi_P = (b_2) \cdot (c_2) \cdot (\bar{f}_2)$$

A corresponding test is:

$$\begin{aligned} v_1 &= \{a_1 = 1, b_1 = x, c_1 = x\} \\ v_2 &= \{a_2 = 0, b_2 = 1, c_2 = 1\} \end{aligned}$$

3. SAT Formulation: Industrial Circuits

For industrial applications, a number of additional requirements has to be met to allow for robust and non-robust PDF test generation. For this purpose, unknown signal values and the high impedance state must be modeled as well as the exact behavior of a signal during two time frames.

This means a large overhead for a Boolean encoding of the problem. In the proposed approach this is handled by using a set of multi-valued logics that provide different sets of values. A particular signal of the circuit is modeled by the logic that provides only those values that may be adopted by the signal. As a result, a compact CNF instance for robust test pattern generation is created.

The multi-valued logics are further motivated and introduced in Section 3.1. In Section 3.2, the application and difficulties of multiple logics in one single circuit are described and discussed. Section 3.3 deals with the Boolean encoding of the problem. Fault modeling is explained in Section 3.4.

3.1. Multi-Valued Logic

There are several requirements for a multi-valued logic L so that it can be applied to the robust test generation for industrial circuits.

- *Representation of two time frames* – For modeling sequential circuits, it must be possible to determine the value of the signal line in each single time frame.
- *Static signals* – For the generation of robust tests, the logic must be able to represent whether a signal is static.
- *Additional values* – For handling tri-state elements and restrictions, the logic must be able to represent the additional values U and Z in each time frame.
- *Compactness* – Generally, the size of the CNF for each element in L is much bigger, if there are more values in the logic. Therefore, the logic should have as few values as possible.

In [14], the 4-valued logic $L_4 = \{0, 1, U, Z\}$ is applied to represent a single time frame for stuck-at test pattern generation. For two time frames, the Cartesian product of L_4 is needed to describe all possible assignments. This leads to the 16-valued logic L_{16}:

$$\begin{aligned} L_{16} = \{&\bar{0}, 01, 10, \bar{1}, 0Z, 1Z, Z0, Z1, \\ &0U, 1U, U0, U1, ZU, UZ, \bar{Z}, \bar{U}\} \end{aligned}$$

The name determines the signal's behavior in both time frames. The first position gives the value of the signal line in the initial time frame, whereas the second position describes the value in the final time frame.

In case of having only one position, the signal is equal in both time frames. Overlined values signify, that it is not known, whether the signal is static or not.

Due to that, the logic L_{16} is only suitable for non-robust test generation. For robust test generation, three additional values 0, 1, Z must be included, which guarantee static signals. A static value for U is not needed. The following logic L_{19s} (s means static) contains these static values.

$$\begin{aligned} L_{19s} = \{&0, \bar{0}, 01, 10, \bar{1}, 1, 0Z, 1Z, Z0, Z1, \\ &0U, 1U, U0, U1, ZU, UZ, \bar{Z}, Z, \bar{U}\} \end{aligned}$$

Table 3. Number of Boolean variables

		AND		bus-driver	
logic	# Var	# cls	# lits	# cls	# lits
L_{19s}	5	-	-	321	1794
L_{11s}	4	148	690	-	-
L_{8s}	3	38	152	-	-
L_{6s}	3	39	151	-	-
L_4	2	-	-	12	35
L_3	2	15	38	-	-
L_2	1	3	7	-	-

In principle, L_{19s} can be used to model the ATPG problem. But in a circuit typically only a few signal lines can adopt all values in L_{19s} or L_{16}, respectively. To create a compact Boolean CNF, a certain signal is modeled by a logic that only contains those values, which can be adopted. All lines which cannot adopt the value Z can therefore be represented by logic L_{11s}:

$$L_{11s} = \{0, \overline{0}, 01, 10, \overline{1}, 1, 0U, 1U, U0, U1, \overline{U}\}$$

The 8-valued logic L_{8s} can be applied, if the value U can only be adopted in the final time frame.

$$L_{8s} = \{0, \overline{0}, 01, 10, \overline{1}, 1, 0U, 1U\}$$

If the value U can never be adopted, the line can be described by L_{6s}.

$$L_{6s} = \{0, \overline{0}, 01, 10, \overline{1}, 1\}$$

For those signal lines which are only needed to calculate the value of a pseudo primary input, i.e. that are only considered in t_1, the logics $L_4 = \{0, 1, U, Z\}$, $L_3 = \{0, 1, U\}$ and the Boolean logic L_2 are used. The logics for non-robust tests can be assembled by excluding the static values $0, 1, Z$.

3.2. Combining Multi-Valued Logics

Logics with less values are generally more compact in their CNF representation than logics with more values. This can be seen in Table 3. The table shows the number of clauses (# cls) and literals (# lits) for the CNF representation of an AND gate and of a bus-driver shown in column *AND* and column *bus-driver*, respectively. Therefore, a certain signal should be modeled by a logic that only provides those values that can be adopted by the signal.

For example, a busdriver can only be represented in L_{19s} and L_4 to model the value Z correctly. An AND gate interprets the value Z as unknown U. For this reason, an encoding with L_{19s} and L_4 is not required.

To classify the signal lines according to the required logic, a preprocessing step on the circuit is needed. This step is executed only once for each circuit. The complexity of this step is linear in the number of gates. Therefore, the overhead is negligible. All signal lines, which can adopt the value Z, are identified and classified to use L_{19s}. All lines of the fanout cone of each element using L_{19s} are marked to use L_{11s} due to the interpretation of the value Z as U by regular gates.

The fanout cone of each pseudo primary input, whose associated pseudo primary output is marked as using L_{11s} is marked itself as using L_{8s}, because the value U can be propagated into the final time frame. All other lines are using L_{6s}, because they can only adopt the Boolean values including the static ones.

Moreover, those elements which are only used for computing the value of a pseudo primary input during the first time frame but are irrelevant in the second time frame, can be modeled with L_4, L_3 or the Boolean logic L_2.

As a result, multiple logics are applied to model a single circuit. When changing from one logic to some other logic a *logic transition* occurs. To guarantee consistent signals on the logic transitions, additional constraints are added to the CNF.

A logic transition occurs, when the signal lines of a gate g use different logics. Then, the lines with the lower-valued logic must be converted to the higher-valued logic of g by adding additional clauses and variables that exclude illegal values.

In the following, these constraints are described by Φ_{lt}. For example, the Boolean input value on the data input of a busdriver would be described in L_2, because it cannot adopt the values U and Z, whereas the busdriver is modeled in L_4. Therefore, the lower-valued logic L_2 on the data input is converted to the higher-valued logic L_4. To guarentee consistent signals, the values U and Z which are not contained in L_2 must be excluded on the data input.

The corresponding constraints are applied at the Boolean level and depend on the Boolean encoding for the multi-valued logics which is presented in the next section.

3.3. Boolean Encoding

To apply a Boolean SAT solver, the values must be encoded by Boolean variables. This encoding should allow a compact representation of logic transitions as explained above. The chosen encoding for L_{19s} is presented in Table 4. Five Boolean variables $x_1, ..., x_5$ are needed to represent the 19 values.

All lower-valued logics that model static values are derived from L_{19s}. The encoding of a logic that has x values is determined by the first $n = \lceil \log_2 x \rceil$ Boolean variables of L_{19s}. The exact number of needed Boolean variables is given in in Column *#Var* in Table 3. As a result, the encodings are compatible to each other, i.e. the logic transition can be realized compactly. This is demonstrated by the following example.

Example 2 *The value* 1 *in* L_{19s} *is encoded by :*

$$\{x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0, x_5 = 0\}$$

Table 4. Boolean Encoding of L_{19s}

L_{19s}	0	$\overline{0}$	01	10	$\overline{1}$	1	$0Z$	$1Z$	$Z0$	$Z1$	$0U$	$1U$	$U0$	$U1$	ZU	UZ	$\overline{Z}$	Z	$\overline{U}$
x_1	0	0	0	1	1	1	0	1	0	1	0	1	0	0	0	1	0	1	1
x_2	0	0	1	0	1	1	1	0	0	0	1	0	0	1	0	1	0	1	1
x_3	0	1	0	0	1	0	1	1	1	0	1	1	0	0	0	1	0	0	1
x_4	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	1
x_5	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0

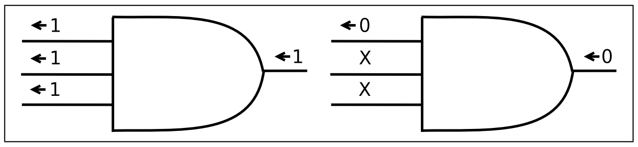

Figure 2. Static value propagation – AND gate

For L_{11s}, only four variables are needed to encode a value. Therefore, x_5 is omitted and the value 1 is encoded by:

$$\{x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0\}$$

Using L_{8s} and L_{6s}, the value is encoded by using the first three variables of the above encoding:

$$\{x_1 = 1, x_2 = 1, x_3 = 0\}$$

In case of a logic transition from e.g. L_{11s} to L_{19s}, it is sufficient to assign the value 0 to x_5, because all values which are contained in L_{19s} but not in L_{11s} are the only one that are encoded with the value 1 at x_5 (cf. Table 4).

The encoding for a non-robust test can be created in a similar way using L_{16}. Once the encoding is determined, a CNF can be created for each gate type by building a truthtable of the function of the gate. Similarly, the constraints to model flip flops are included in the CNF. As introduced in Section 2, this is described by the constraint Φ_t. To minimize the CNF, ESPRESSO [13] was used.

3.4. Fault Modeling

Up to now, the modeling of the circuit during two consecutive time frames was presented. This section deals with the fault modeling in CNF especially with respect to guarenteeing static signals.

The CNF Φ_{C19s} models a circuit during two consecutive time frames, where $logic(g)$ determines the logic used for a gate g as described in Section 3.1:

$$\Phi_{C19s} = (\prod_{g \in C} \Phi_{logic(g)}) \cdot \Phi_{lt} \cdot \Phi_t$$

Then, the CNF for the PDF test is described by the following equation:

$$\Phi_P^{C19s} = \Phi_{C19s} \cdot \Phi_T \cdot \Phi_P$$

The constraint Φ_T is derived in a similar way as described in Section 2.2; according to the logic used and the encoding constraints are added that force the rising (falling) transition.

Using L_{19s}, it is possible to generate robust tests. According to Table 1, the static values $S0$ and $S1$ must be forced at the off-path inputs of path P. These values correspond to the values 0 and 1 in L_{19s} and are contained in Φ_P.

Static values at the off-path inputs of P originate from static values on one or more of the inputs of the circuit. Therefore, a static value must be traced towards the (pseudo) primary inputs. This is guaranteed by the functional description of each gate and therefore inherently included in Φ_{C19s}.

In case of a static controlling value at the output of a gate g, it is sufficient to force one input of g to the static controlling value of g. In contrast, if a static non-controlling value is set to the output of g, all inputs must be forced to the static non-controlling value of g. This is shown in Figure 2 for an AND gate. If Φ_P^{C19s} is satisfiable, the test pattern is given by the assignment of the variables which encode the values of the (pseudo) primary inputs.

Example 3 *Given the circuit C shown in Figure 1 and the falling path $P = (a, d, e, g)$. In contrast to the presented off-path constraints for non-robust test generation, the off-path inputs must be forced to static 1 at the inputs of the AND/NAND gates and to static 0 at the input of the OR gate.*

Assuming that all values use L_{6s}, i.e. all signal lines can only adopt Boolean values, the constraints Φ_T and Φ_P are given by the following equations:

$$\begin{aligned} \Phi_T &= (a_1) \cdot (\overline{a}_2) \cdot (\overline{a}_3) \cdot (d_1) \cdot (\overline{d}_2) \cdot (\overline{d}_3) \\ &\quad \cdot (\overline{e}_1) \cdot (e_2) \cdot (\overline{e}_3) \cdot (\overline{g}_1) \cdot (g_2) \cdot (\overline{g}_3) \\ \Phi_P &= (b_1) \cdot (b_2) \cdot (\overline{b}_3) \cdot (c_1) \cdot (c_2) \cdot (\overline{c}_3) \\ &\quad \cdot (\overline{f}_1) \cdot (\overline{f}_2) \cdot (\overline{f}_3) \end{aligned}$$

A corresponding robust test is:

$$\begin{aligned} v_1 &= \{a_1 = 1, b_1 = 1, c_1 = 1\} \\ v_2 &= \{a_2 = 0, b_2 = 1, c_2 = 1\} \end{aligned}$$

Table 5. General circuit information

circuit	#ff	#paths	non-rob	rob
p393	20	2492	1156	1108
p57k	2291	2290	717	636
p292k	10101	360	243	17 + x
p823k	31745	108	108	x
s344	15	710	259	253
s444	21	1070	166	148
s526	21	820	147	141
s641	19	3444	1100	1007
s713	19	43624	1097	831
s838	32	2018	656	656
s838.1	32	3428	756	756
s953	29	2312	961	954
s1196	18	6196	3406	3229
s1238	18	7118	3365	3233
s1423	74	89452	8057	5853

Table 6. MV-logic statistics

	#elems				classification t	
circuit	L_{19s}	L_{11s}	L_{8s}	L_{6s}	with	without
p393	0	0	0	393	2.94	21.82
p57k	13	8363	22613	22209	1136.94	>20000
p823k	167	733	10297	825028	10770.97	>40000

4. Experimental Results

In this section, experimental results for industrial circuits and for ISCAS benchmarks are presented and discussed. The industrial circuits were provided by NXP Semiconductors Germany GmbH, Hamburg, Germany. The methods presented in this paper are implemented in C++ and executed on a Pentium 4 system (GNU/Linux, x86_64, 3.6 GHz) with 4 GByte RAM.

As SAT solver, MiniSat v1.14 [4] was used. All time data is given in CPU seconds. For each path a time limit of 30 seconds is given. Only while classifying *p823k* the limit was increased to 250 seconds due to the circuit's larger depth.

General information about the circuits is given in Table 5. This table is divided into two parts. Details about industrial circuits are given in the upper part, whereas details about ISCAS benchmarks can be found in the lower part. Column *circuit* gives the circuit's name. The name of a circuit roughly denotes the number of gates contained in the circuit, e.g. *p823k* has about 823 thousand gates. In column *#ff* the number of flip flops within the circuit is provided. The number of tested paths is given in column *#paths*. For ISCAS benchmarks, all structural paths were considered, whereas for industrial circuits only critical and long paths were set to be tested.

The number of non-robustly testable paths can be found in column *non-rob* and the number of robustly testable paths in column *rob*. The variable x is used when the exact number is unknown because of aborted paths.

The influence of using multiple logics versus using only the highest-valued logic was studied in a first experiment. Table 6 summarizes the results for some industrial circuits. In column *#elem*, the number of elements modeled by each logic are given. Most gates of the circuits are modeled with a lower-valued logic.

The total run times for non-robust test generation are presented in column *classification t*. Column *with* describes the optimized flow combining multi-valued logics while column *without* presents the flow without classification. Only the highest-valued logic is used in the latter case. Clearly, the optimized flow combining multiple logics is mandatory to handle large industrial circuits.

The influence of several configurations to calculate robust and non-robust tests is considered in the next series of experiments. Every benchmark was executed with four different configurations:

1. Only generating non-robust test patterns (using L_{16} and derivatives).

2. Only generating robust test patterns (using L_{19s} and derivatives).

3. Firstly, a non-robust test is generated. In case of success, a robust test pattern is generated.

4. Firstly, a robust test will be generated. In case of failure, a non-robust test pattern is generated.

The results of each configuration are presented in Table 7. For each configuration, the total run time (column *t*), the run time which is needed for creating the CNF (column *CNF*) and the memory needs (column *mem* in MByte) are provided.

As expected, due to the higher complexity of robust test generation (*configuration 2*), the run times for the generation of non-robust tests (*configuration 1*) are lower than the run times of robust test generation for all benchmarks. Concerning the ISCAS benchmarks, the difference in the memory needs is negligible. But for industrial circuits, the memory needs to generate robust tests are significantly larger than for non-robust test generation. This is due to the larger number of clauses and literals when higher-valued logics are needed (cf. Table 3).

Using *configuration 3* and *configuration 4*, the test patterns with the highest quality are generated for each path. Considering the ISCAS benchmarks, *configuration 3* is faster than *configuration 4*. But for the more important case of industrial circuits, the opposite can be observed. The total run times of *configuration 3* are – in spite of shorter run times in CNF generation – higher than the run times of *configuration 4*. An analysis of the run times for single paths shows, that they are higher for satisfiable instances than for unsatisfiable instances. Due to more calls, which lead to satisfiable instances in *configuration 4*, the total run times are shorter.

Table 7. Experimental results for the different configurations

	configuration 1			configuration 2			configuration 3			configuration 4		
circuit	t	CNF	mem	t	CNF	mem	t	CNF	mem	t	CNF	mem
p393	2.94	1.74	3.85	6.27	4.33	4.33	6.00	3.64	4.34	7.51	5.18	4.54
p57k	1136.04	341.26	133.56	2291.76	440.73	348.07	3033.08	463.77	329.14	2746.19	701.47	384.77
p292k	2517.34	198.87	328.61	4804.11	228.60	1005.19	7256.78	398.81	1069.48	7219.28	440.31	1108.67
p823k	10770.97	310.95	1053.47	>40000	-	-	>40000	-	-	>40000	-	-
s344	0.45	0.31	2.03	1.00	0.70	2.98	0.50	0.32	3.09	1.01	0.68	3.11
s444	0.52	0.43	2.29	0.80	0.58	3.37	1.02	0.79	3.48	1.85	1.41	3.52
s526	0.57	0.38	2.36	0.73	0.56	3.52	0.90	0.62	3.67	1.36	1.06	3.68
s641	2.88	2.00	3.76	7.08	5.31	5.02	5.17	3.73	5.32	7.72	5.93	5.38
s713	29.73	24.31	29.59	91.55	78.25	31.42	35.09	29.24	31.76	131.48	110.76	35.86
s838	2.86	2.01	3.29	8.16	4.94	5.07	7.89	4.57	5.32	9.49	5.85	5.25
s838.1	4.47	3.81	4.77	11.98	7.64	5.70	10.24	6.51	5.98	16.66	10.98	5.79
s953	3.08	2.12	3.56	6.06	4.30	5.06	6.47	4.13	5.39	8.15	5.75	5.24
s1196	8.44	5.07	4.83	29.71	18.58	7.41	28.11	16.04	7.93	36.13	23.59	7.63
s1238	9.54	6.24	4.89	38.25	25.27	7.82	32.15	18.81	8.46	44.66	30.61	8.63
s1423	195.23	155.72	47.98	425.01	344.83	50.99	282.87	221.93	50.96	629.09	500.34	58.36

It can be concluded, that non-robusts tests can be generated efficently using *configuration 1*, whereas the more desirable robust tests can be generated with only small overhead with *configuration 2*. To obtain tests of highest possible quality, *configuration 4* is preferable to *configuration 3* due to lower run times.

In summary, the proposed approach is very efficient for generating both non-robust and robust test patterns. Due to combining multiple logics, the approach is feasible even for the classification of PDFs in large industrial circuits.

5. Conclusions and Future Works

In this paper, a SAT-based approach for generating non-robust and robust test patterns for the PDFM in industrial circuits containing tri-state elements was presented. A multi-valued logic and the transformation to Boolean SAT was described. Furthermore, a significant reduction of the complexity of the SAT instance by applying a structural analysis is applied. Experimental results show the efficiency even on large industrial circuits.

The integration of more powerful learning techniques into this approach as proposed in [8, 1, 7] is focus of future work.

6. Acknowledgement

This research work was supported in part by the German Federal Ministry of Education and Research (BMBF) in the Project MAYA under the contract number 01M3172B.

References

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

[2] C. Chen and S. K. Gupta. A satisfiability-based test generator for path delay faults in combinational circuits. In *Design Automation Conf.*, pages 209–214, 1996.

[3] K. Cheng and H. Chen. Classification and identification of nonrobust untestable path delay faults. *IEEE Trans. on CAD*, 15(8):845–853, 1996.

[4] N. Eén and N. Sörensson. An extensible SAT solver. In *SAT 2003*, volume 2919 of *LNCS*, pages 502–518, 2004.

[5] S. Eggersglüß, G. Fey, and R. Drechsler. SAT-based ATPG for path delay faults in sequential circuits. In *IEEE International Symposium on Circuits and Systems*, 2007.

[6] 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. of the 37th Int'l Symp. on Multiple-Valued Logic (ISMVL'07)*, 2007.

[7] G. Fey, T. Warode, and R. Drechsler. Reusing learned information in SAT-based ATPG. In *VLSI Design*, pages 69–76, 2007.

[8] J. Kim, J. Whittemore, K. Sakallah, and J. Marques Silva. On applying incremental satisfiability to delay fault testing. In *Design, Automation and Test in Europe*, pages 380–384, 2000.

[9] C.-J. Lin and S. Reddy. On delay fault testing in logic circuits. *IEEE Trans. on CAD*, 6(5):694–703, 1987.

[10] J. Marques-Silva and K. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. on Comp.*, 48(5):506–521, 1999.

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

[12] A. Pramanick and S. Reddy. On the design of path delay fault testable combinational circuits. In *Int'l Symp. on Fault-Tolerant Comp.*, pages 374–381, 1990.

[13] E. Sentovich, K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. Technical report, University of Berkeley, 1992.

[14] J. Shi, G. Fey, R. Drechsler, A. Glowatz, F. Hapke, and J. Schlöffel. PASSAT: Effcient SAT-based test pattern generation for industrial circuits. In *IEEE Annual Symposium on VLSI*, pages 212–217, 2005.