# Efficiency of Multi-Valued Encoding in SAT-based ATPG*

Görschwin Fey Junhao Shi Rolf Drechsler

Institute of Computer Science
University of Bremen
28359 Bremen, Germany
{fey,junhao,drechsle}@informatik.uni-bremen.de

## Abstract

*Automatic Test Pattern Generation (ATPG) is one of the core algorithms in testing of digital circuits and systems. Due to recent advances in algorithms to solve Boolean Satisfiability (SAT), there is a renewed interest in SAT-based ATPG. While the early approaches only used two-valued logic, modern tools have to use multiple values to model unknown values and tri-state elements for buses.*

*In this paper we present a detailed study on how to chose the multi-valued encoding for SAT-based ATPG. The techniques have been implemented and evaluated on large industrial benchmarks.*

## 1. Introduction

Ensuring that a circuit functions correctly when being delivered is crucial for any circuit design company. For this reason a post-production test is carried out in the production flow. By applying a set of test patterns the correctness of a circuit is checked. These test patterns are calculated by dedicated algorithms for *Automatic Test Pattern Generation* (ATPG).

Due to the exponentially increasing number of gates integrated on a single chip the time needed for ATPG grows rapidly. As a result classical algorithms such as FAN [6] and PODEM [7] reach their limits.

An alternative approach is ATPG based on *Boolean Satisfiability* (SAT) as first proposed in [9]. Significant improvements in the transformation of the problem have been made in [16, 17] by including structural information in the SAT instance to aid the reasoning on the circuit. These first SAT-based approaches suffered from the hardness of the SAT problem. Meanwhile powerful engines for solving instances of the SAT problem have been developed [10, 11, 8, 5]. These solvers apply conflict-based learning, efficient implementation techniques and effective search heuristics. The SAT-based ATPG tool PASSAT [14] uses such a fast SAT solver to calculate test patterns for the stuck-at fault model. The efficiency on industrial benchmarks has been shown in [15].

While early approaches were very simple and only used a two-valued encoding to keep the SAT instance small, for practical purposes there is a need to model multiple values. This is for example necessary to take unknown values coming from the environment of a circuit into account or to model buses and tri-state elements. To use an efficient (Boolean) SAT solver for ATPG on such a multi-valued model a Boolean encoding is needed.

The way the encoding is done has a significant influence on the generated SAT instance and by this also influences the time needed to solve the problem [1]. Thus, SAT-based ATPG strongly depends on the encoding regarding memory requirement and run time.

In this paper we discuss alternative encodings for a SAT-based ATPG algorithm. The possible encodings are classified and the properties of the classes are discussed. Two representative encodings have been implemented and integrated in the ATPG tool PASSAT. Studies on industrial benchmarks show the significant influence of the chosen multi-valued encoding.

The paper is structured as follows: The next section explains how SAT-based ATPG is carried out in the tool PASSAT. In Section 3 the four-valued logic used for ATPG in PASSAT is introduced. The different encodings in the Boolean SAT instance are analyzed. Experimental results are given in Section 4. Conclusions are presented in the last section.

## 2. SAT-based ATPG

In the following the framework for SAT-based ATPG as used in the tool PASSAT [14] for the stuck-at fault model

*This work was funded in part by Philips Semiconductors GmbH, Hamburg, Germany and in part by DFG grant DR 287/16-1.

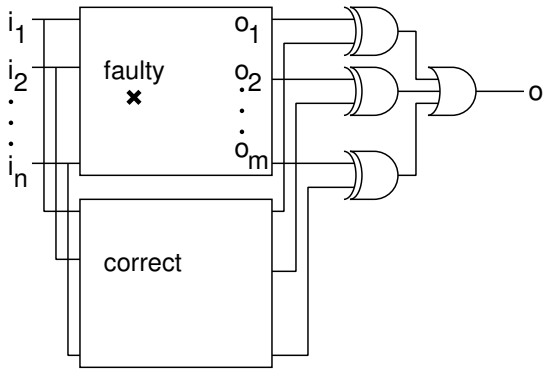


**Figure 1. Miter-circuit for test-pattern generation**

[3] is briefly reviewed. First, the general transformation of the ATPG problem into a SAT problem is considered. Then, the improvements by including structural information are discussed. Finally, the advanced SAT techniques that lead to the overall performance of PASSAT are explained.

### 2.1. SAT Formulation

For a given fault the problem of generating a test pattern is transformed into a Boolean SAT problem represented in *Conjunctive Normal Form* (CNF), i.e. a set of clauses. The fault is modeled in the gate-level circuit. Then, the faulty circuit and a non-faulty version are joined to form a miter circuit [2] as shown in Figure 1. This circuit is mapped into a CNF and the output is constrained to the value one. This constraint ensures that at least one output of the correct and faulty circuit differ. A satisfying assignment to this SAT instance exists if and only if the modeled fault is testable. This assignment also determines a test-vector for the fault. If the SAT instance is unsatisfiable the fault is redundant.

### 2.2. Structural Information

In practice some optimizations are done to improve the performance of such a SAT-based ATPG tool. First, only the output-cone and the input-cone of the modeled fault are included in the SAT instance. The input-cone is also shared between the correct and the faulty circuit, because values in this part can not differ.

Another important improvement is the inclusion of structural information in the SAT instance as suggested in [16]. This information is derived from arguments originally proposed for the D-algorithm in [12]:

- A gate $G$ is on a potential D-chain, if there exists a path between the error location and an output.
- For each gate $G$ on a potential D-chain a Boolean attribute $G_p$ is introduced. The attribute $G_p$ is 1 if and only if the error is propagated to a primary output via $G$.
- If the error is propagated via a gate $G$ it must also be propagated via at least one successor $H^1, \ldots, H^m$ of $G$. This is included as the following implication in the SAT instance:
$$G_p \rightarrow \cup_{i=1}^{m} H_p^i$$
- If the error is propagated via a gate $G$ the value $G_c$ of the gate in the correct circuit and the value $G_f$ in the faulty circuit differ:
$$G_p \rightarrow (G_c \neq G_f)$$

### 2.3. Advanced SAT Techniques

PASSAT is based on TEGUS [16]. An important improvement of PASSAT is the use of a modern SAT solver. Such a solver essentially enhances the backtrack search of the DLL procedure [4] by several techniques. This drastically improves the performance on large and difficult instances. The key techinques in modern SAT solvers are the following:

- Learning based on conflict analysis [10]: Each time a partial assignment is found not to satisfy the SAT instance this conflict is analyzed. The learned information is stored in form of a conflict-clause. Thus, the space without solutions is not reentered during the search.
- Efficient implementation techniques for fast implications [11]: Each time a value is assigned to a variable during the search resulting implications must be derived. Due to a watching scheme only those clauses have to be touched where an implication might occur.
- Efficient decision strategies [8]: The decision which variable and which value are assigned next during the search relies on decision strategies. These algorithms efficiently keep statistics that help to exploit previously learned information.

Additionally, PASSAT includes modified decision strategies that are adjusted to the problem of ATPG [14]. Essentially this allows to reduce the search space by using information about primary inputs or fanout points in the circuit similar to the classical ATPG-algorithms PODEM [7] and FAN [6], respectively.

Finally, PASSAT uses a multi-valued encoding scheme to handle ATPG problems that occur in practice. Only one fixed encoding was chosen in [14]. In the remaining sections the influence of different encodings on the performance of PASSAT is systematically analyzed.

**Table 1. Boolean encodings**

(a) Set 1

| $s$ | $x$ | $\overline{x}$ |
|---|---|---|
| 0 | $a$ | $b$ |
| 1 | $a$ | $\overline{b}$ |
| U | $\overline{a}$ | $b$ |
| Z | $\overline{a}$ | $\overline{b}$ |

(b) Set 2

| $s$ | $x$ | $\overline{x}$ |
|---|---|---|
| 0 | $a$ | $b$ |
| 1 | $a$ | $\overline{b}$ |
| U | $\overline{a}$ | $b$ |
| Z | $\overline{a}$ | $b$ |

(c) Set 3

| $s$ | $x$ | $\overline{x}$ |
|---|---|---|
| 0 | $a$ | $b$ |
| 1 | $\overline{a}$ | $\overline{b}$ |
| U | $\overline{a}$ | $b$ |
| Z | $a$ | $\overline{b}$ |

(d) Example: Set 1, $a = 0$, $b = 0$, $x = c_s$

| $s$ | $c_s$ | $c_s^*$ |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| U | 1 | 0 |
| Z | 1 | 1 |

**Table 2. AND-gate over** $\{0, 1, Z, U\}$

(a) 4-valued

| $t$ | $u$ | $s$ |
|---|---|---|
| 0 | $-$ | 0 |
| $-$ | 0 | 0 |
| 1 | 1 | 1 |
| $U$ | $\neq 0$ | $U$ |
| $Z$ | $\neq 0$ | $U$ |
| $\neq 0$ | $U$ | $U$ |
| $\neq 0$ | $Z$ | $U$ |

(b) Encoded

| $c_t$ | $c_t^*$ | $c_u$ | $c_u^*$ | $c_s$ | $c_s^*$ |
|---|---|---|---|---|---|
| 0 | 0 | $-$ | $-$ | 0 | 0 |
| $-$ | $-$ | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | $\neq 0$ | 0 | 1 | 0 |
| 1 | 1 | $\neq 0$ | 0 | 1 | 0 |
| $\neq 0$ | 0 | 1 | 0 | 1 | 0 |
| $\neq 0$ | 0 | 1 | 1 | 1 | 0 |

## 3. Multi-Valued Encoding

In this section the techniques to handle non-Boolean values in PASSAT are discussed. First, the use of multi-valued logic during ATPG is motivated and the four-valued logic is introduced. Then, different possibilities to encode the multi-valued problem in a Boolean SAT instance are discussed.

### 3.1. Four-Valued Logic

For practical purposes it is not sufficient to consider only the Boolean values 0 and 1 during test pattern generation as it has been done in [16]. This has mainly two reasons.

At first, industrial circuits usually have tri-state elements. From a modeling point of view the tri-state elements could be transformed into a Boolean structure with the same functionality, e.g. by inserting multiplexers. But during test pattern generation additional constraints apply to signals driven by tri-state elements. For example, no two drivers must drive the signal with opposite values or if all drivers are in the high impedance state the driven signal has an unknown value. The value $Z$ is used to properly model these constraints and the transition function of tri-state elements.

Environment constraints that apply to a circuit are another problem. Usually the circuit is embedded in a larger environment. As a result some inputs of the circuit may not be controllable. Thus, the value of such a non-controllable input is assumed to be unknown during ATPG. The logic value $U$ is used to model this situation. This has to be modeled explicitly in the SAT instance, because otherwise the SAT solver would also assign Boolean values to non-controllable inputs.

Therefore a four-valued logic over $\{0, 1, Z, U\}$ is considered in PASSAT.

### 3.2. Boolean Encoding

The multi-valued ATPG problem has to be transformed into a Boolean problem to make use of a modern Boolean SAT solver on the four-valued logic. Therefore each signal of the circuit is encoded by two Boolean variables. One encoding out of the $4! = 24$ mappings of four values onto two Boolean values has to be chosen. The chosen encoding determines which clauses are needed to model particular gates. This, in turn, influences the size of the resulting SAT instance and the efficiency of the SAT search.

All possible encodings are summarized in Tables 1(a)-1(c). The two Boolean variables are denoted by $x$ and $\overline{x}$, the letters $a$ and $b$ are placeholders for Boolean values. The following notations define the interpretation of the tables more formally:

- A signal $s$ is encoded by the two Boolean variables $c_s$ and $c_s^*$.
- $x \in \{c_s, c_s^*\}, \overline{x} \in \{c_s, c_s^*\} \setminus \{x\}$
- $a \in \{0, 1\}, \overline{a} \in \{0, 1\} \setminus \{a\}$
- $b \in \{0, 1\}, \overline{b} \in \{0, 1\} \setminus \{b\}$

**Example 1** *Consider Table 1(a) and the following assignment: $a = 0$, $b = 0$, $x = c_s$. Then, the encoding in Table 1(d) results.*

Thus, a particular encoding is determined by choosing values for $a$, $b$ and $x$. Each table defines a set of eight encodings.

Note, that for encodings in Set 1 or Set 2 one Boolean variable is sufficient to decide, if the value of $s$ is in the Boolean domain, i.e. in $\{0, 1\}$, or in the non-Boolean domain, i.e. in $\{U, Z\}$. In contrast encodings in Set 3 do not have this property. This observation will be important when the efficiency of a particular encoding for SAT solving is determined.

**Table 3. Number of clauses for each encoding**

| Set | $x$ | $a$ | $b$ | NAND | NOR | AND | BUS | BUS0 | BUS1 | BUSDR. | XOR | NOT | OR | All |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | $c_s$ | 0 | 0 | 8 | 9 | 9 | 10 | 11 | 10 | 9 | 5 | 5 | 8 | 100 |
| | $c_s$ | 0 | 1 | 8 | 9 | 9 | 10 | 11 | 10 | 9 | 5 | 5 | 8 | 100 |
| | $c_s$ | 1 | 0 | 8 | 9 | 9 | 10 | 11 | 10 | 9 | 5 | 5 | 8 | 100 |
| | $c_s$ | 1 | 1 | 8 | 9 | 9 | 10 | 11 | 10 | 9 | 5 | 5 | 8 | 100 |
| | $c_s^*$ | 0 | 0 | 8 | 9 | 9 | 10 | 11 | 10 | 9 | 5 | 5 | 8 | 100 |
| | $c_s^*$ | 0 | 1 | 8 | 9 | 9 | 10 | 11 | 10 | 9 | 5 | 5 | 8 | 100 |
| | $c_s^*$ | 1 | 0 | 8 | 9 | 9 | 10 | 11 | 10 | 9 | 5 | 5 | 8 | 100 |
| | $c_s^*$ | 1 | 1 | 8 | 9 | 9 | 10 | 11 | 10 | 9 | 5 | 5 | 8 | 100 |
| 2 | $c_s$ | 0 | 0 | 9 | 8 | 8 | 10 | 10 | 11 | 9 | 5 | 5 | 9 | 100 |
| | $c_s$ | 0 | 1 | 9 | 8 | 8 | 10 | 10 | 11 | 9 | 5 | 5 | 9 | 100 |
| | $c_s$ | 1 | 0 | 9 | 8 | 8 | 10 | 10 | 11 | 9 | 5 | 5 | 9 | 100 |
| | $c_s$ | 1 | 1 | 9 | 8 | 8 | 10 | 10 | 11 | 9 | 5 | 5 | 9 | 100 |
| | $c_s^*$ | 0 | 0 | 9 | 8 | 8 | 10 | 10 | 11 | 9 | 5 | 5 | 9 | 100 |
| | $c_s^*$ | 0 | 1 | 9 | 8 | 8 | 10 | 10 | 11 | 9 | 5 | 5 | 9 | 100 |
| | $c_s^*$ | 1 | 0 | 9 | 8 | 8 | 10 | 10 | 11 | 9 | 5 | 5 | 9 | 100 |
| | $c_s^*$ | 1 | 1 | 9 | 8 | 8 | 10 | 10 | 11 | 9 | 5 | 5 | 9 | 100 |
| 3 | $c_s$ | 0 | 0 | 11 | 11 | 11 | 8 | 9 | 9 | 11 | 5 | 6 | 11 | 108 |
| | $c_s$ | 0 | 1 | 11 | 11 | 11 | 8 | 9 | 9 | 11 | 5 | 6 | 11 | 108 |
| | $c_s$ | 1 | 0 | 11 | 11 | 11 | 8 | 9 | 9 | 11 | 5 | 6 | 11 | 108 |
| | $c_s$ | 1 | 1 | 11 | 11 | 11 | 8 | 9 | 9 | 11 | 5 | 6 | 11 | 108 |
| | $c_s^*$ | 0 | 0 | 11 | 11 | 11 | 8 | 9 | 9 | 11 | 5 | 6 | 11 | 108 |
| | $c_s^*$ | 0 | 1 | 11 | 11 | 11 | 8 | 9 | 9 | 11 | 5 | 6 | 11 | 108 |
| | $c_s^*$ | 1 | 0 | 11 | 11 | 11 | 8 | 9 | 9 | 11 | 5 | 6 | 11 | 108 |
| | $c_s^*$ | 1 | 1 | 11 | 11 | 11 | 8 | 9 | 9 | 11 | 5 | 6 | 11 | 108 |

### 3.3. Transformation to SAT Instance

The clauses to model a particular gate type can be determined when a particular encoding and the truth-table of the gate are given. This set of clauses can be reduced by two-level logic-optimization. The tool Espresso contained in SIS [13] was used for this purpose. Espresso is capable of calculating a minimal representation. The following example illustrates this flow.

**Example 2** *Table 2(a) shows the truth-table of an AND-gate $s = t \cdot u$ over $\{0, 1, Z, U\}$. The truth-table is mapped onto the Boolean domain using the encoding from Example 1. The encoded truth-table is shown in Table 2(b) (for compactness the notation "$\neq 0\ 0$" is used to denote that at least one of two variables must be different from $0$). A CNF is extracted from this truth-table and optimized by Espresso.*

Results for all possible encodings are presented in Table 3. For each gate type the number of clauses needed to model the gate are given. Besides the well-known Boolean gates (*AND*, *OR*, ...) also non-Boolean gates are considered. The gate *BUSDRIVER* is a tri-state buffer that assumes the value $Z$ when not being driven and propagates the input value otherwise. A *BUS* resolves the value coming from several tri-state buffers. If no buffer has a value different from $Z$ the bus also assumes the value $Z$. Similarly, *BUS0* and *BUS1* are buses that assume values $0$ and $1$, respectively, when not being driven. The last column *All* in the table gives the sum of the numbers of clauses for all gate types.

All encodings of a given set lead to clauses that are isomorphic to each other. By mapping the polarity of literals and the choice of variables the other encodings of the set are retrieved. Particularly, Boolean gates are modeled efficiently by encodings from Set 1 and Set 2. The sum of clauses needed for all gates is equal for both sets. The difference is that for example the encodings of one set are more efficient for *NAND*-gates, while the encodings of the other set are more efficient for *NOR*-gates. In our benchmarks both gate types occur with a similar frequency (see next section). The same observation is true for the other gates where the efficiency of the encodings differs. Therefore no significant trade-off for the encodings occurs on the benchmarks.

In contrast more clauses are needed to model Boolean gates when an encoding of Set 3 is used. At the same time this encoding is more efficient for non-Boolean gates. In most practical circuits the number of non-Boolean gates is much smaller than the number of Boolean gates. Therefore more compact SAT instances will result when an encoding from Set 1 or Set 2 is used. The behavior of the SAT solver

**Table 4. Number of gates for each type**

| circ. | IN | OUT | FANO. | NOT | AND | NAND | OR | NOR | BUS | BUSDR. |
|---|---|---|---|---|---|---|---|---|---|---|
| p44k | 2356 | 2232 | 6845 | 16869 | 12365 | 528 | 5484 | 1128 | 0 | 0 |
| p88k | 4712 | 4565 | 14560 | 20913 | 27643 | 2838 | 16941 | 5883 | 144 | 268 |
| p177k | 11273 | 11031 | 33605 | 48582 | 49911 | 5707 | 30933 | 5962 | 0 | 560 |

**Table 5. Memory and run time for different encodings**

| circ. | no. | enc. | clauses | cls. % | variables | memory | mem. % | CNF | CNF % | solve | solve % |
|---|---|---|---|---|---|---|---|---|---|---|---|
| p44k | 1 | A | 173,987 | | 56,520 | 13,713 | | 41 | | 14 | |
| | | B | 220,375 | 127 | 56,520 | 14,087 | 103 | 49 | 120 | 78 | 557 |
| p44k | 2 | A | 174,083 | | 56,542 | 13,713 | | 43 | | 16 | |
| | | B | 220,493 | 127 | 56,542 | 14,088 | 103 | 51 | 119 | 79 | 494 |
| p44k | 3 | A | 174,083 | | 56,542 | 13,713 | | 43 | | 15 | |
| | | B | 220,493 | 127 | 56,542 | 14,088 | 103 | 52 | 121 | 79 | 527 |
| p88k | 1 | A | 33,406 | | 10,307 | 2,824 | | 8 | | 4 | |
| | | B | 41,079 | 123 | 10,307 | 3,410 | 121 | 10 | 125 | 7 | 175 |
| p88k | 2 | A | 33,501 | | 10,328 | 2,824 | | 9 | | 4 | |
| | | B | 41,188 | 123 | 10,328 | 3,411 | 121 | 9 | 100 | 8 | 200 |
| p88k | 3 | A | 33,517 | | 10,289 | 2,825 | | 8 | | 8 | |
| | | B | 41,321 | 123 | 10,289 | 3,412 | 121 | 9 | 113 | 8 | 100 |
| p177k | 1 | A | 96,550 | | 34,428 | 8,900 | | 23 | | 23 | |
| | | B | 119,162 | 123 | 34,428 | 9,082 | 102 | 25 | 107 | 247 | 1074 |
| p177k | 2 | A | 96,536 | | 34,425 | 8,900 | | 25 | | 28 | |
| | | B | 119,145 | 123 | 34,425 | 9,082 | 102 | 29 | 116 | 234 | 836 |
| p177k | 3 | A | 96,550 | | 34,428 | 8,899 | | 25 | | 20 | |
| | | B | 119,162 | 123 | 34,428 | 9,082 | 102 | 29 | 116 | 237 | 1185 |

does not necessarily depend on the size of the SAT instance, but when the same problem is encoded in a much smaller instance also a better performance of the SAT solver can be expected. These hypotheses are strengthened by the experimental results reported in the following section.

## 4. Experimental Results for ATPG

Benchmark results for some industrial benchmarks from Philips Semiconductors GmbH, Hamburg, Germany are reported in the following. All experiments were carried out on an AMD Athlon 3000 (2.1GHz, 1GB, Linux).

Table 4 shows some information about the circuits. The first column reports the name of the circuit. Here, p44k means that the circuit contains about 44k gates. The succeeding columns give the numbers of different types of gates in the circuit. Inputs, outputs and fanout gates are reported for completeness. In the SAT instance an input is modeled as an unconstrained signal. An output is not explicitly modeled, but the output signal of the preceding gate is used instead. Similarly, no additional constraints are necessary in the SAT instance for fanout gates. These gates mark fanout points in the circuit. If no error is modeled on one of the branches the whole gate is modeled by a single signal. All other gate types are explicitly modeled using clauses as discussed in Section 3. As can be seen the Boolean gates make up the vast majority of all gates. Only a few or no tri-state elements are contained in the benchmark circuits.

In the following we compare two representative encodings in detail. Encoding A belongs to Set 2 with parameters $x = c_s^*$, $a = 0$, and $b = 0$. Encoding B belongs to Set 3 with the same parameters. Thus, for Encoding A a single Boolean variable "naturally" differentiates between Boolean and non-Boolean values. In contrast this is not possible for Encoding B.

Detailed results are given in Table 5. Considered are individual faults in the previously introduced circuits with respect to the two encodings. Reported are the number of clauses and the number of variables in the SAT instance. The memory needed is reported in KB. Then, the run times to create the SAT instance and the run time to solve the SAT instance are shown in columns "CNF" and "solve", respectively. All run times are given in $1/100$ seconds. The ratio of clauses, memory, and run time needed by Encoding B compared to Encoding A is always given in percent in the column following the absolute values.

The number of variables remains the same for both en-

codings. But the number of clauses increases significantly when Encoding B is used instead of Encoding A. In the same way the memory needed is significantly larger for Encoding B. But the overhead is not as large as for the clauses. This is due to the additional memory needed to store the circuit, test-patterns, learned information, etc. Also the run time to generate the CNF is always larger for Encoding B, again because more clauses have to be produced for this case.

Finally, the most significant overhead occurs for the time needed to solve the SAT instances. In case of p44k and p177k no tri-state elements are needed. Thus, the more complex representation of such gates in Encoding A does not produce any overhead in the SAT instance. In contrast Encoding B leads to more complex representations for Boolean gates. As a result the run time increases usually by 5 times for p44k and up to 11 times for p177k. But when p88k is considered the run time overhead for Encoding B is reduced. Here, also tri-state elements are contained and therefore the improved representation of those gates in Encoding B can be exploited during the solving process. The run time increases by at most two times for the considered faults. In one case both run times are the same.

In summary, Encoding A performed significantly better on the benchmarks than Encoding B with respect to memory consumption and run time. Moreover, the same behavior can be expected for most circuits as usually many more Boolean than non-Boolean gates are contained in the circuits.

## 5. Conclusions

The encoding of a four-valued ATPG problem into a Boolean SAT instance has been studied in detail. All different encodings have been considered and classified into three groups. For two groups one Boolean variable allows to "naturally" differentiate between Boolean and non-Boolean values of the original four-valued logic. Such encodings lead to a more compact CNF representation of the ATPG problem. Experimental studies show that the natural encoding also outperforms the other encoding with respect to run time on the considered industrial benchmarks. The same behavior can be expected in general when the number of Boolean gates in the circuit is much larger than the number of tri-state elements and unknown input values.

## Acknowledgement

The authors would like to thank Andreas Glowatz, Friedrich Hapke and Jürgen Schlöffel from Philips Semiconductors GmbH, Hamburg, Germany for helpful discussions.

## References

[1] C. Ansòtegui and F. Manyà. Mapping many-valued CNF formulas to Boolean CNF formulas. In *Int'l Symp. on Multi-Valued Logic*, pages 290–295, 2005.

[2] D. Brand. Verification of large synthesized designs. In *Int'l Conf. on CAD*, pages 534–537, 1993.

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

[4] M. Davis, G. Logeman, and D. Loveland. A machine program for theorem proving. *Comm. of the ACM*, 5:394–397, 1962.

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

[6] H. Fujiwara and T. Shimono. On the acceleration of test generation algorithms. *IEEE Trans. on Comp.*, 32:1137–1144, 1983.

[7] P. Goel. An implicit enumeration algorithm to generate test for combinational logic. *IEEE Trans. on Comp.*, 30:215–222, 1981.

[8] E. Goldberg and Y. Novikov. BerkMin: a fast and robust SAT-solver. In *Design, Automation and Test in Europe*, pages 142–149, 2002.

[9] T. Larrabee. Test pattern generation using Boolean satisfiability. *IEEE Trans. on CAD*, 11:4–15, 1992.

[10] J. Marques-Silva and K. Sakallah. GRASP – a new search algorithm for satisfiability. In *Int'l Conf. on CAD*, pages 220–227, 1996.

[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] J. Roth. Diagnosis of automata failures: A calculus and a method. *IBM J. Res. Dev.*, 10:278–281, 1966.

[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.

[15] J. Shi, G. Fey, R. Drechsler, A. Glowatz, J. Schlöffel, and F. Hapke. Experimental studies on SAT-based test pattern generation for industrial circuits. In *ASICON*, pages 967–970, 2005.

[16] P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli. Combinational test generation using satisfiability. *IEEE Trans. on CAD*, 15:1167–1176, 1996.

[17] P. Tafertshofer, A. Ganz, and K. Antreich. Igraine - an implication graph based engine for fast implication, justification, and propagation. *IEEE Trans. on CAD*, 19(8):907–927, 2000.