On the Influence of Boolean Encodings in SAT-based ATPG for Path Delay Faults*

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

Abstract

Automatic Test Pattern Generation (ATPG) is an important task to ensure that a chip functions correctly. For high speed chips, testing for dynamic fault models such as the path delay fault model becomes more and more important. While classical algorithms for ATPG reach their limit, the significance of algorithms to solve the Boolean Satisfiability (SAT) problem grows due to recent developments of powerful SAT solvers. However, ATPG is not always a purely Boolean problem. For generating robust test patterns for delay faults, multiple-valued logics are needed. To apply a (Boolean) SAT solver on a problem modeled in multiple-valued logic, a Boolean encoding has to be used.

In this paper, we consider the problem of SAT-based ATPG for the robust path delay fault model where a 19-valued logic is used and provide a detailed study on the influence of the chosen Boolean encoding on the performance of test generation. Further, we show a method to identify efficient encodings and show the behavior of these encodings on ISCAS benchmarks and large industrial circuits.

1. Introduction

Ensuring the correctness of a chip is an important task before being delivered. Every chip has to pass a post production test, where the correctness is checked by applying test patterns that are generated by *Automatic Test Pattern Generation* (ATPG) algorithms. Due to the increased process variability, defects leading to timing violations are becoming dominant in modern chips. Such delay defects are tested by using dynamic fault models such as the *Path Delay Fault* (PDF) model [1].

A test for a PDF is a two pattern test, i.e. consists of a pair of test vectors v_1, v_2. The first test vector v_1 sets the initial value and the second test vector v_2 launches the desired transition, which can be either rising or falling. The transition is then propagated along a structural path to an output, where it can be observed, whether the acumulated delay along the path violates any timing constraints.

*Parts of this research work were supported by the BMBF in the Project MAYA under contract number 01M3172B and by DFG grant DR 287/15-1.

In [2], tests for PDFs have been classified into two different categories: *robust testable* and *non-robust testable*. Because robust tests guarantee the detection of delay faults in the presence of other delay faults, they are more desirable but also harder to obtain in terms of complexity. In the last decade, powerful engines to solve the *Boolean satisfiabiliy* (SAT) problem have been developed. Modern SAT solvers [3–6] incorporate techniques such as conflict-based learning, non-chronological backtracking and efficient search heuristics. Due to their efficiency, SAT solvers serve as a core engine for many problems in the field of *Computer-Aided Design* such as verification and ATPG.

SAT-based ATPG for PDFs was first introduced in [7]. In this approach, a 7-valued logic proposed in [8] with a fixed Boolean encoding was used to generate robust tests for PDFs. However, this approach was restricted to combinational circuits. Sequential behavior could not be modeled using this logic. Due to that, this approach is hardly feasible for today's circuits. Other approaches were presented, where SAT techniques were applied to path sensitization [9] and non-robust and semi-robust test [10], respectively. These approaches made only use of Boolean logic and cannot model static values that are necessary for generating robust tests.

In [11], robust test generation for PDFs in circuits with unknown values and tri-state elements was performed. A 19-valued logic and derivative logics with a smaller number of values were developed. The Boolean encoding for these logics was chosen randomly. However, in [12], it was shown, that the chosen encoding has significant influence on the SAT instance and therefore on the performance of the SAT solver. A detailed study on the influence of the chosen encoding in SAT-based ATPG for stuck-at faults is presented in [13]. But in this approach, only Boolean encodings for a four-valued logic are analyzed.

In this paper, the influence of the Boolean encodings for the 19-valued logic and its derivatives on the performance of the overall test generation algorithm are studied. Furthermore, the relation between the size of the SAT instance and the performance of robust test generation is evaluated and a method to identify efficient encodings is shown. Representative encodings have been developed and integrated into the test generation algorithm. Experimental results on ISCAS benchmarks

and industrial circuits containing multiple-valued logic are provided to show the impact of the chosen Boolean encoding.

This paper is structured as follows. In the next section, the basic concepts of SAT-based ATPG for PDFs and the usage of Boolean encodings are explained, whereas alternative Boolean encodings are discussed and analyzed in Section 3. Experimental results are presented in Section 4 and conclusions are drawn in the last section.

2. SAT-based ATPG for PDFs

In this section, the basic concepts of robust SAT-based ATPG for PDFs are briefly reviewed. For more details, we refer to [2] for the PDF model and to [11] for the SAT formulation. Section 2.1 introduces the PDF model with respect to robust and non-robust test classification, whereas in Section 2.2 the SAT formulation of robust test generation is presented. In Section 2.3, the usage of Boolean encodings is explained.

2.1. Path Delay Fault Model

The PDF model describes a distributed delay fault on a path from a (pseudo-)primary input to a (pseudo-) primary output of a circuit C. To detect such a fault, a transition which is either *rising* or *falling* is propagated along the path. Therefore, a PDF F can be defined as a tuple $F = (P, T)$ where P is a sequence of gates g_1,...,g_n with input g_1 and output g_n. The type of transition at g_1 is given by T. Note, that the transition is inverted after passing an inverting gate on the path, e.g. NAND or NOR. A test pattern which detects a PDF contains two test vectors v_1, v_2 that are applied in two consecutive time frames t_1, t_2. The test vector v_1 sets the initial value in t_1, whereas v_2 launches the transition in t_2. In case of a delay fault, the transition arrives at g_n not in the specified time, i.e. a timing violation occurs.

The task of ATPG for PDFs is to generate such two test vectors which detect a potential delay fault on P. According to [2], there exist two different categories of tests for PDFs: *robust* and *non-robust*. The difference between robust and non-robust tests is that robust tests guarantee the detection of a PDF even when other delay faults are present. This cannot be guaranteed by applying non-robust tests. Here, other delay faults can mask the considered PDF. From the technical point of view, both fault models differ in the constraints at the off-path inputs of P. An off-path input is defined as an input of a gate g_i on path P, that is not g_{i-1}.

The constraints at the off-path inputs are presented in Table 1. The value X1 (X0) on an off-path input signifies, that the final value in t_2 has to be 1(0), whereas S1 (S0) means, that both the initial value in t_1 and the final value in t_2 have to be 1(0) and no hazards occur between them, i.e. the signal is static.

2.2. SAT Formulation: Robust Tests

SAT solvers are working on a problem represented in *Conjunctive Normal Form* (CNF). A CNF Φ is a conjunction of clauses, whereas a clause is a disjunction of literals. A literal is a Boolean variable in its positive or negative from. The CNF Φ is *satisfiable*, if and only if there exists an assignment for which at least one literal in each clause evaluates to 1. If no such assignment exists, the CNF is *unsatisfiable*.

Table 1. Off-Path Constraints

gate type	robust		non-robust
	falling	rising	
AND/NAND	S1	X1	X1
OR/NOR	X0	S0	X0

Therefore, the problem must be converted to CNF. For a circuit C and a PDF $F = (P, T)$, the CNF Φ_F can be obtained by the following formula: $\Phi_F = \Phi_C \cdot \Phi_T \cdot \Phi_P$, where Φ_C is the characteristic function of C, Φ_T forces the transition and Φ_P sets the corresponding constraints at the off-path inputs of P. The characteristic function Φ_C is the conjunction of the characteristic function of each gate g in the circuit and can be obtained by $\Phi_C = \prod_{i=1}^{n} \Phi_g^i$, where $n = |g|$. The derivation of Φ_g depends on the applied logic. If Boolean logic is used, Φ_g can be easily derived using the method proposed in [14]. In case of a higher-valued logic L_m with m values and $m > 2$, a Boolean encoding is needed that maps each value into the Boolean domain. (cf. Section 2.3). To obtain a minimized CNF representation, the logic optimizer ESPRESSO of the SIS package [15] is used.

According to [11], for robust test generation in Boolean circuits a six-valued logic L_6 is needed, whereas for industrial circuits containing multiple-valued logic, a 19-valued logic L_{19} has to be used. Because a higher-valued logic typically results in more complex SAT instances and not all values of L_{19} can be assumed by each signal line, an algorithm was proposed which uses the higher-valued logics only where necessary. For most parts of the circuit, a lower-valued logic is sufficient. In this way, the size of the SAT instance is reduced (see [11] for more details).

2.3. Usage of Boolean Encodings

To apply a Boolean SAT solver to a multiple-valued ATPG problem, e.g. generating robust test patterns for path delay faults, the problem has to be transformed into a Boolean problem. This can be done by using a Boolean encoding η for each value in the multiple-valued logic L_m. The minimal number of Boolean variables n needed to encode this value depends on the number of values of L_m and is defined as follows: $n = \lceil \log_2 |L_m| \rceil$. The following study is restricted to these logarithmic encodings.

Consequently, three variables are needed to encode both logics L_6 and L_8, whereas L_{11} and L_{19} has to be encoded by four and five Boolean variables, respectively. More formally, the Boolean representation of signal s whose value is defined over L_m is given by $x_1^s ... x_n^s$.

Each value v in L_m is encoded by a unique assignment $A^v = a_1 ... a_n$ of $x_1 ... x_n$ where $a_{i|1 \leq i \leq n} \in \{0, 1\}$. The complete Boolean encoding η for L_m is defined as

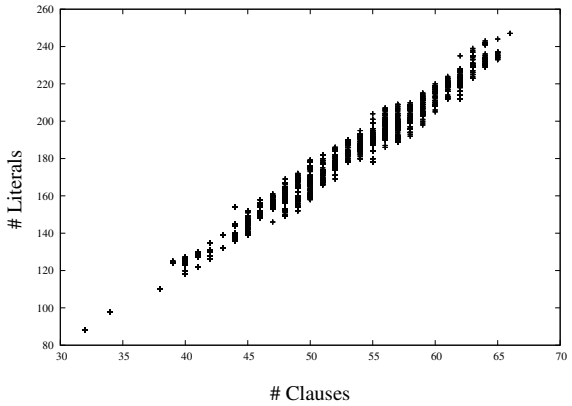

Figure 1. Distribution of the Compactness Values for Boolean Encodings of L_6

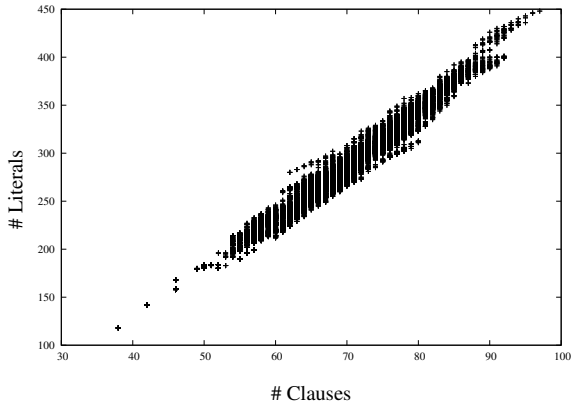

Figure 2. Distribution of the Compactness Values for Boolean Encodings of L_8

follows:

$$\eta = A^{v_1}, ..., A^{v_m}$$
$$\text{where} \quad A^{v_i} \neq A^{v_j} | 1 \leq i, j \leq m; i \neq j$$

However, the usage of different logics and Boolean encodings in a circuit requires a handling of logic transitions. A logic transition occurs, when signal lines of one gate are modeled in different logics. To avoid modifying the circuit structure, Boolean encodings can only be used together in a single circuit, if they are compatible to each other, i.e. each (partial) assignment of the signal's variables (determining a value v_i) does not exclude the interpretation as v_i in all other used logics, e.g. if a value in L_{11} is encoded by 0110, it has to be encoded by 011 in L_6 and L_8.

3. Analysis of Alternative Boolean Encodings

The Boolean encoding used in the initial approach was chosen rather randomly and its efficiency was not analyzed against other encodings. In this section, alternative Boolean encodings are discussed and analyzed.

Already for the CNF generation of a circuit modeled in L_6, one Boolean encoding out of $8!/2 = 20160$ has to be chosen. The number of potential Boolean encodings increases with the increasing number of values of the logic. For L_8, there are $8! = 40320$ Boolean encodings, whereas for L_{11}, there are more than one billion. Testing all possible encodings and selecting the most efficient is therefore not feasible. Some preselection must be done to identify efficient encodings. For studying the impact of the encodings, inefficient encodings have to be determined, too. Note, that preliminary experiments have shown that due to the small number of gates that have to be modeled in L_{19}, the change of the Boolean encoding of L_{19} had nearly no impact on the run time. Therefore, Boolean encodings for L_{19} are not discussed in the following.

3.1. Compactness of Boolean Representation

Typically, but not necessarily, a larger SAT instance results in higher run times of the SAT solver. Moreover, in the field of SAT-based ATPG, the SAT solver has to cope with thousands of smaller instances. Although the complexity of building a SAT instance is of linear size, the overhead is not negligible in the overall run time. Therefore, a Boolean encoding with a compact CNF representation is likely to perform well whereas a Boolean encoding with a large CNF representation has probably a poor performance.

Each gate type has a different CNF representation and a preliminary evaluation has shown that one single Boolean encoding may produce a compact representation for one gate type, whereas for other gate types (e.g. busdriver), it may be contrary. Due to the fact, that most gates in a circuit are primitive gates, e.g. AND, OR, and not higher-level, e.g. busdriver, we concentrate only on the size of the CNF representation of primitive gates in the following. Primitive gates have also the advantage that the size of their representation is very similar for a specific encoding.

Below, the CNF sizes of the Boolean encodings are analyzed. The compactness of the Boolean representation of each encoding e is denoted as C^e and is defined as a tuple $(|cls|, |lits|)$ that contains the accumulated number of clauses (cls) and the accumulated number of literals (lits) of the gate types AND and OR. The accumulation was done to obtain a good ratio of the compactness of both gate types.

The distribution of the compactness values of all possible Boolean encodings for L_6 and for L_8 are shown in Figure 1 and in Figure 2, respectively. The *Most Compact Encodings* (MCE) of L_6 have 32 clauses and 88 literals (accumulated for AND and OR), whereas the largest encodings have 67 clauses and 247 literals, which is more than two times the size of the MCEs; concerning the number of literals even nearly three times. The difference between most compact and largest encoding increases considering L_8. Here, the number of clauses (97) in the largest encoding is 2.6 times the most compact one (38) and the number of literals (448) 3.8 times larger than the most compact one (118).

Due to the very high number of possible encodings for L_{11}, the range of the compactness values for the encodings of L_{11} is determined with a simplified method. The compactness values of only those encodings of L_{11}

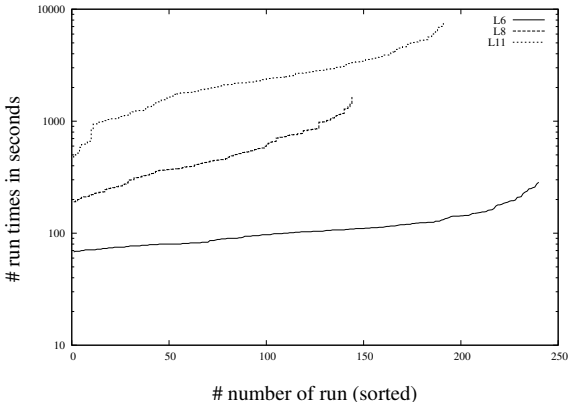

Figure 3. Run Time Distribution for c6288

are calculated that are compatible with the MCEs of L_6 and L_8. This is due to the fact that only compatible encodings (cf. Section 2.3) can be used. For this small subset of 64512 encodings, the number of clauses and the number of literals vary between 67 and 230, and 126 and 534, respectively.

Through an analysis of the MCEs of L_6 and L_8, we came to the result that all compatible encodings of the MCEs of L_8 are not among the MCEs of L_6. Moreover, those encodings of L_6 that are compatible to the MCEs of L_8 have a larger size than the MCEs of L_8. For example, consider the following compatible encodings η_e (L_6) and η_f (L_8). Whereas η_f with $C^f = (38, 118)$ is among the MCEs of L_8, η_e with $C^e = (40, 118)$ is not among the MCEs of L_6 and is larger than η_f.

It can be concluded that the chosen Boolean encoding has – independently from the logic used – an enormous impact on the size of the SAT instance. The usage of compatible encodings only, however, sets tight constraints on the usage of Boolean encodings and prevents the joint usage of the MCEs of each logic.

3.2. Efficiency of Most Compact Encodings

The size of the SAT instance is only one indicator for the efficiency of a Boolean encoding. Therefore, the MCEs of each logic are investigated according to their run time for a single circuit. To avoid influences from other encodings, the circuit must be modeled by only one single logic, i.e. either L_6, L_8 or L_{11}. The ISCAS '85 circuit c6288 representing a 16-bit multiplier was chosen to test the efficiency of the MCEs of each logic. All structural paths with a length of over 40 gates were identified and were set as targets (rising and falling) for robust test generation. This results in 3200 ATPG calls for each encoding.

The tests were carried out for each logic on a Dual Dual-Core Xeon (3000 MHz, 32768 MByte RAM) running GNU/Linux. In each of the three runs, the circuit is modeled completely with L_6, L_8 and L_{11}, respectively. For each logic, a set containing the MCEs is identified and for each encoding in the set, robust test generation was executed. In Table 2, statistical data and the overall results of the runs are given. The first column gives the logic, whereas in the next column, the number of runs are denoted. The third column

Table 2. Run times of MCEs for c6288

logic	runs	C^e	min	av.	max
L_6	240	(32,88)	68	113	285
L_8	144	(38-42,118-142)	191	544	1647
L_{11}	192	(67,126)	473	2608	7560

Table 3. Compactness Values of Encodings

enc.	C^e (L_6)	C^e (L_8)	C^e (L_{11})
η_{L6com}	(32,88)	(52,184)	(115,483)
η_{L6lar}	(64,241)	(78,347)	(161,759)
η_{L6med}	(48,162)	(60,226)	(117,465)
η_{L11com}	(32,88)	(42,142)	(67,230)
η_{L11lar}	(32,88)	(42,142)	(113,473)
η_{L8com}	(32,88)	(42,142)	(60,190)
η_{L8lar}	(32,88)	(70,287)	(105,413)
η_{L6mee}	(32,88)	(42,142)	(67,230)
η_{L11mee}	(32,88)	(42,142)	(67,230)
η_{L8mee1}	(40,118)	(38,118)	(56,166)
η_{L8mee2}	-	(38,118)	(60,190)

presents the compactness values of the chosen encodings. In the following columns *min*, *av.* and *max*, the smallest, the average and the highest run time, respectively, are given in CPU seconds.

In Figure 3, however, the run time distribution is shown for each logic in logarithmic scaling. The run times for each logic were sorted. The value on the x-axis defines the position in the sorted list and the value on the y-axis gives the run time in seconds. The upper curve denotes the run times of the MCEs of L_{11}, whereas the middle curve and the lower curve give the run times of the MCEs of L_8 and L_6, respectively.

For L_6 even the MCEs differ strongly regarding the run time behavior. The highest run time is over 4 times the minimal one, although they have equal compactness values. The range is even higher for the higher-valued logics L_8 and L_{11}. The highest run time for L_8 is eight times the minimal run time for L_8, whereas for L_{11}, the highest run time is nearly 16 times the minimal run time. While the curve of L_6 is increasing only very smoothly, the curves of L_8 and L_{11} are more steep, suggesting that encodings of L_8 and L_{11} have to be chosen more carefully. Note, that those encodings having the minimal run time for each logic are denoted as *Most Efficient Encodings* (MEE) in the following.

The application of the MCEs for robust test generation shows that, first, equal compactness values do not guarantee the same run time behavior, and second, the impact on the efficiency increases with a higher-valued logic.

4. Experiments

In this section, alternative Boolean encodings are experimentally evaluated. First in Section 4.1, four experiments with representative Boolean encodings are described. The experimental results of these encodings are shown in Section 4.2.

4.1. Encoding Selection

In this section, Boolean encodings are created to determine the influence of the ATPG run time. The compactness values of each encoding can be found in Table 3. Note, that in the following an encoding refers

to a set of compatible encodings for each logic rather than to a single encoding if no logic is explicitly named. Four different experiments are described below:

- Experiment 1 shows the behavior of two encodings from which one is likely to be very efficient, whereas the other is probably inefficient. For this, a compact encoding η_{L6com} (MCE of L_6) and a large encoding η_{L6lar} are chosen. Note, that the encoding of L_6 was first created and the compatible encodings are selected afterwards. Here, the most compact and the largest encodings, respectively, are selected among the compatible encodings. If not mentioned otherwise, this is the standard flow of choosing compatible encodings. Furthermore, an encoding η_{L6med} of medium size (applied in [11]) is selected.
- Experiment 2 shows the influence of the encoding selection for L_{11} on the ATPG performance. For this purpose, a compact encoding η_{L11com} (MCE of L_{11}) is created. Next, an encoding set η_{L11lar} is generated such that the encodings for L_6 and L_8 are equal, but instead of choosing an MCE of L_{11} the largest compatible encoding is selected.
- In Experiment 3, the influence of the encoding selection for L_8 is investigated. First, a compact encoding η_{L8com} is generated. Then, an encoding set η_{L8lar} is created containing the same encoding for L_6, but has different encodings of L_8 and L_{11}. Note that possible differences in run time cannot clearly be dedicated to the encoding of L_8, because the encoding for L_{11} also differs.
- In Experiment 4, the MEEs of each logic are evaluated for all circuits. This is to show that, for receiving a good overall performance, it is not sufficient to use an encoding optimized for one logic only. Therefore, the encodings η_{L6mee} (MEE of L_6), η_{L11mee} (MEE of L_{11}) and η_{L8mee1} (MEE of L_8) are created. As already stated in Section 3.1, all those encodings of L_6 that are compatible to the MCEs of L_8 have a larger size than the MCEs of L_8. Therefore, those parts of the circuit which are normally modeled in L_6 are modeled in L_8 using η_{L8mee2}. Otherwise η_{L8mee1} and η_{L8mee} are equal.

4.2. Experimental Results

In this section, the results of the four experiments are presented. The experiments were carried out on a AMD64 4200+ (2200 MHz, 2048 MByte RAM) running GNU/Linux. The program was implemented in C++ and the SAT solver MiniSat 1.14 [6] serves as core engine. As benchmarks, ISCAS '85 circuits and industrial circuits provided by NXP Semiconductors Hamburg, Germany were used. The name of the p-circuits roughly denotes their size, e.g. circuit p1330k has about 1.3 million gates. More statistical data about the circuits is given in Table 4. For each circuit the number of inputs (column *#PI*), the number of tri-state elements (column *#Tri*) and the number of flipflops (column *#FF*) are shown. Furthermore, the percentage of gates modeled in L_{11} and L_8 are given in column $\%L_{11}$ and column $\%L_8$, respectively. The number of paths for which robust test generation is executed is presented in column *#PUT*. As test targets, only paths with a length of over 40 gates are selected. The maximum number of test targets was set to 20100. The paths are chosen randomly, but to avoid testing paths of a small part of the circuit only, at least one path starts at every input (if such a long path exists).

Table 4. Circuit Statistics

circuit	#PI	#Tri	#FF	$\%L_{11}$	$\%L_8$	PUT
c1908	33	0	0	0	0	2264
c2670	157	0	0	0	0	1400
c3540	50	0	0	0	0	3700
c5315	178	0	0	0	0	4340
c6288	32	0	0	0	0	3200
c7552	206	0	0	0	0	6360
p44k	739	0	2175	0	0	20000
p49k	303	0	334	0	0	12390
p57k	8	0	2291	0.24	25.66	20000
p80k	152	0	3878	0	0	20078
p88k	403	412	4309	7.26	21.38	20052
p99k	167	0	5747	1.50	5.59	20026
p177k	768	560	10507	27.35	54.53	20028
p456k	1723	203	14900	25.52	73.50	20070
p462k	1815	597	29205	14.22	33.79	20006
p565k	996	169	32409	15.36	56.60	20088
p1330k	617	189	104630	8.86	15.14	20036

In Table 5, the results of the selected encodings are shown. Time is measured in minutes (m) and hours (h), respectively. The timeout for each target was set to 20 seconds, whereas the timeout for each ATPG run was 20 hours. The minimum run time of all encodings is marked bold for each circuit.

In Experiment 1, it is shown that the influence of the Boolean encoding is significant. The run time for the large encoding η_{L6lar} dramatically increases up to a factor of 56 (p80k) compared to η_{L6com} and up to a factor of 44 compared to η_{L6med}. In five out of eleven industrial circuits, η_{L6lar} even reaches the limit of 20 hours. Therefore, η_{L6lar} is not feasible for industrial practice. Comparing η_{L6com} and η_{L6med}, the compact encoding is in most cases only slightly better than η_{L6med} and in two cases (c6288, p57k) even worse.

In Experiment 2, the influence of the chosen encoding for L_{11} is evaluated. In those circuits with no or only few parts modeled in L_{11}, the run times are the same or even slightly better using the large encoding η_{L11lar}. In circuits with higher precentage of L_{11}, the maximum overhead is about 25% of run time (p1330k).

In Experiment 3, the influence of the chosen encoding for L_8 is investigated. The results are similar to those of Experiment 2, but the impact on the run times is scaled up. For p456k, where nearly two-thirds of the circuit is modeled in L_8, the run time is increased by a factor of 2.9 and for p57k a timeout occured.

In Experiment 4, the MEEs of each logic are investigated for all circuits. Encoding η_{L6mee} being MEE of L_6 is also the most efficient encoding for all other ISCAS circuits. But for the industrial circuits, η_{L6mee} provides the smallest run time only for p49k (completely in L_6) but not for any other circuit. Compared to each other, η_{L8mee2} has an advantage over η_{L8mee1} for the smaller circuits (with lesser percentage of L_8), whereas η_{L8mee1} is better for the larger ones and therefore preferable. Encoding η_{L11mee} (MEE of L_{11}) has

Table 5. Experimental Results – Alternative Boolean Encodings

	Exp. 1			Exp. 2		Exp. 3		Exp. 4			
circuit	η_{L6com}	η_{L6lar}	η_{L6med}	η_{L11com}	η_{L11lar}	η_{L8com}	η_{L8lar}	η_{L6mee}	η_{L11mee}	η_{L8mee1}	η_{L8mee2}
c1908	0:13m	0:28m	0:18m	0:13m	0:13m	0:12m	0:12m	**0:11m**	0:12m	0:15m	0:14m
c2670	0:16m	0:32m	0:24m	0:16m	0:16m	0:15m	0:15m	**0:14m**	**0:14m**	0:18m	0:18m
c3540	1:15m	2:38m	1:46m	1:12m	1:12m	1:11m	1:11m	**1:08m**	**1:08m**	1:25m	1:24m
c5315	0:51m	2:05m	1:14m	0:51m	0:51m	0:47m	0:47m	**0:46m**	0:47m	0:59m	0:59m
c6288	6:28m	3:21h	4:24m	**2:16m**	**2:16m**	2:17m	2:17m	**2:16m**	2:51m	4:48m	4:58m
c7552	0:40m	1:25m	0:57m	0:40m	0:40m	0:37m	0:37m	**0:36m**	0:37m	0:47m	0:46m
p44k	**2:07h**	$> 20h$	2:29h	2:25h	2:25h	2:15h	2:15h	2:16h	$> 20h$	2:16h	2:21h
p49k	3:13h	$> 20h$	4:42h	3:18h	3:18h	3:19h	3:19h	**3:07h**	3:09h	3:47h	3:46h
p57k	5:01h	$> 20h$	**4:28h**	9:00h	8:59h	12:00h	$> 20h$	9:59h	$> 20h$	5:03h	5:02h
p80k	**18:27m**	17:18h	23:20m	45:47m	45:47m	45:49m	45:49m	46:00m	1:12h	50:37m	50:20m
p88k	**21:50m**	$> 20h$	26:20m	22:44m	22:41m	22:37m	23:09m	22:37m	50:48m	23:15m	22:55m
p99k	11:31m	$> 20h$	13:20m	11:11m	**11:05m**	11:20m	11:06m	11:10m	12:06m	16:08m	17:13m
p177k	1:24h	10:30h	1:27h	1:11h	1:18h	1:11h	1:35h	1:13h	1:12h	**1:05h**	**1:05h**
p456k	1:10h	19:08h	1:15h	53:17m	1:04h	52:22m	1:51h	54:00m	53:25m	**46:01m**	46:48m
p462k	43:04m	1:53h	43:44m	39:45m	43:59m	40:13m	59:14m	40:54m	43:01m	36:05m	**36:00m**
p565k	8:12m	11:56m	8:59m	7:19m	7:18m	7:24m	9:58m	7:25m	7:27m	**6:54m**	6:55m
p1330k	1:05h	2:34h	1:07h	47:33m	1:03h	44:01m	1:11h	45:38m	46:55m	**40:14m**	40:42m

only minimal performance gain for those circuits with a large portion of L_{11} (e.g. p177k, p456k).

This experiment shows that the usage of an encoding which is optimized for one logic only is not optimal due to the different logic modeling of the circuits. Therefore, we propose the combination of multiple encodings depending on the percentage of the used logics. For instance, the combination of $\eta_{L8mee1}/\eta_{L8mee2}$ and η_{L6mee}, where η_{L6mee} is applied if the percentage of L_8 in the circuit is lower than 25% ($\eta_{L8mee1}/\eta_{L8mee2}$ otherwise), would provide the best result for 11 out of 17 circuits and is therefore more robust.

5. Conclusions

The influence of Boolean encodings in SAT-based ATPG for PDFs in which a set of multiple-valued logics is applied, has been studied in detail. First, it is shown that the size of the SAT instance strongly depends on the chosen encoding. Moreover, the effect increases the more values the logic has. Also, it is pointed out that the compactness of a Boolean encoding is only an indicator for the performance. Experiments have shown that the performance for encodings with equal size can vary by a factor of 4 for a lower-valued logic and by a factor of 16 for a higher-valued logic.

Representative encodings have been developed and their influence has been evaluated on ISCAS '85 circuits as well as on industrial circuits. According to the results, the influence is significant and the Boolean encodings have to be chosen carefully to avoid poor performing test generation. Moreover, it has been highlighted that the performance of the encoding is highly influenced by the logic modeling of the circuit. Thus, the best result is obtained by a combination of different encodings according to the circuit's logic modeling. It could be also observed that the usage of only compatible encodings sets tight constraints on the selection process. Therefore, studying the usage of different encodings in detail and the application of incompatible encodings is future work.

References

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

[2] K. Cheng and H. Chen, "Classification and identification of nonrobust untestable path delay faults." *IEEE Trans. on CAD*, vol. 15, no. 8, pp. 845–853, 1996.

[3] 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.

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

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

[6] N. Eén and N. Sörensson, "An extensible SAT solver," in *SAT 2003*, LNCS, vol. 2919, 2004, pp. 502–518.

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

[8] C.-J. Lin and S. Reddy, "On delay fault testing in logic circuits," *IEEE Trans. on CAD*, vol. 6, no. 5, pp. 694–703, 1987.

[9] J. Kim, J. Whittemore, J. P. Marques-Silva, and K. Sakallah, "On applying incremental satisfiability to delay fault testing," in *Design, Automation and Test in Europe*, 2000, pp. 380–384.

[10] K. Yang, K.-T. Cheng, and L.-C. Wang, "Trangen: a SAT-based ATPG for path-oriented transition faults," in *ASP Design Automation Conf.*, 2004, pp. 92–97.

[11] 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 *ACM & IEEE Int'l Conf. on Formal Methods and Models for Codesign*, 2007, pp. 181–187.

[12] C. Ansòtegui and F. Manyà, "Mapping many-valued CNF formulas to Boolean CNF formulas," in *Int'l Symp. on Multiple-Valued Logic*, 2005, pp. 290–295.

[13] G. Fey, J. Shi, and R. Drechsler, "Efficiency of multi-valued encoding in SAT-based ATPG," in *Int'l Symp. on Multiple-Valued Logic*, 2006, pp. 25–30.

[14] T. Larrabee, "Test pattern generation using Boolean satisfiability," *IEEE Trans. on CAD*, vol. 11, pp. 4–15, 1992.

[15] 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," University of Berkeley, Tech. Rep., 1992.