Exploring the impact of different circuits implementations in an ATPG system

Glória Denise Claro da Silva
Centro de Ciência Computacionais - C3
Universidade Federal do Rio Grande - FURG
Rio Grande, Brazil
gloria.claro1209@gmail.com

Gabriel Soares Porto
Centro de Ciências Computacionais - C3
Universidade Federal do Rio Grande - FURG
Rio Grande, Brazil
gabrielporto@furg.br

Paulo Francisco Butzen
Centro de Ciências Computacionais - C3
Universidade Federal do Rio Grande - FURG
Rio Grande, Brazil
paulobutzen@furg.br

***Abstract*—Over the years, the technology scaling increases the system complexity. Also, the nanometer circuits are more fault prone. Consequently, the testing process become a challenge to overcome. Due the high complexity circuits, ATPG (Automatic Test Pattern Generator) tools become very important to automatize the testing process. The test pattern generation process is highly linked to the circuit logic gates arrangement. A single logic function could be implemented in different circuit structures. With this in mind, it is interesting to analyze the impact of different circuits implementation in an ATPG (Automatic Test Pattern Generator) system. Experimental results show that circuits mapped with libraries that contain complex logic gates achieve better performance in most of cases. However, in some circuits the libraries with only basic functions create some exceptions.**

Keywords—Standard Cell, ATPG, SAT, ABC

I. Introduction

With the technology scaling over the years, Very-Large-Scale-Integration (VLSI) designs became a very complex task. With the performance increase due to technology scaling, digital systems became more fault prone. To ensure the proper system operation, there are several challenges to be overcome. One of them is the test process. In this context, Automatic Test Pattern Generator (ATPG) tools become essential to turn the test process possible.

The pattern generation process performance is linked to the logic gates arrangement. There are several circuits structures that can implement the same functionality. Each of these versions present different behavior in terms of test generation. Some of them have faults that can be easier to detect/observe than others, or also faults that are impossible to be tested.

In this context, the paper main objective is to evaluate the impact of different implementation of the ISCAS85 [1] benchmark set on test pattern generation. Three different libraries were used to map the benchmark set, an ATPG tool have provided the fault coverage and the test pattern set for each circuit.

For a better understanding of the analysis made in this paper, the background necessary to understand the concepts is shown in the Section II. Section III presents the used methodology, while Section IV discuss and analyze the obtained results. Final remarks are presented in Section V.

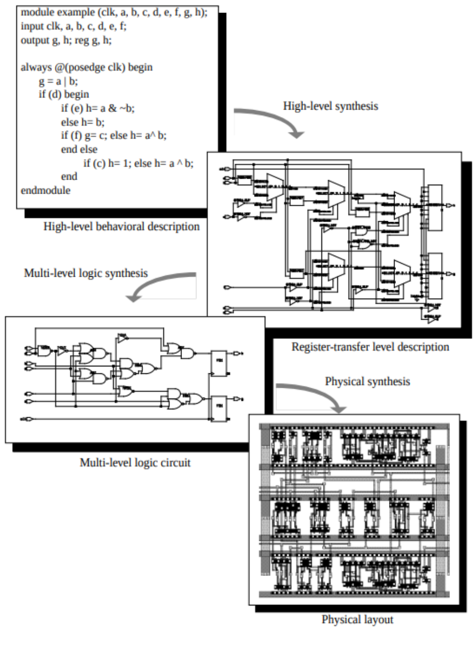
II. Background

In this Section, two relevant aspects are presented to a better understanding the analysis performed in this work. First, the most common circuit design flow, the standard cell design flow, is described. Later, the key points related to the ATPG process used in this work are presented.

A. Standard Cell Design Flow

Nowadays, the main methodology to develop an integrated circuit is called Standard Cell design flow. This methodology uses a set of standard cells to build the system structure. These standard cells are previously designed, characterized and tested. This flow provides a faster and more reliable project design.

Fig. 1 presents the complete Standard Cell design flow, starting from the circuit specification until the manufacturing process on silicon. This methodology can be described in three main steps: High Level synthesis, logic synthesis and physical synthesis [2].



```
module example (clk, a, b, c, d, e, f, g, h);
input clk, a, b, c, d, e, f;
output g, h; reg g, h;

always @(posedge clk) begin
     g = a | b;
     if (d) begin
          if (e) h= a & ~b;
          else h= b;
          if (f) g= c; else h= a^ b;
          end else
               if (c) h= 1; else h= a ^ b;
          end
endmodule
```

High-level behavioral description
High-level synthesis
Register-transfer level description
Multi-level logic synthesis
Multi-level logic circuit
Physical synthesis
Physical layout

Fig. 1. Major synthesis steps in the design of digital integrated circuits [2].

The High-Level synthesis aims to convert the circuit description behavior from a behavioral HDL (Hardware Description Language) to a structural representation at RTL (register-transfer level) description. The RTL structure contains elements like: data storage elements, functional modules and data steering logic.

The next step is the logic synthesis. It uses the RTL description created in previous step to generate a new description composed by logic gates. The first phase transfer the RTL description to a generic structure and optimization independent of the technology are performed. After, the generic logic gates are replaced by standard gates and finally, optimizations dependent of technology are processed. These standard gates are in a standard cell library that is already designed, characterized and tested. The logic synthesis step determines the logic gate arrangement in the circuit.

The last stage is the physical synthesis. This stage is responsible to convert the optimized mapped circuit into a physical structure. The main steps in this stage are the placement and routing. At the end of this stage, is made the signoff and the integrated circuit is ready to be sent to a foundry.

Depending on the used set of gates, a circuit could be easier or harder to generate a test pattern set and, consequently, affecting the quality of test process. According to the used standard cell library, different versions for the same circuit are synthesized and could be optimized to the test process.

B. *Automatic Test Pattern Generation Problem*

Due to the high complexity of VLSI designs, EDA (Electronic Design Automation) tools like ATPG become essential to make the testing process possible. The principal concepts of ATPG problem will be described as follow.

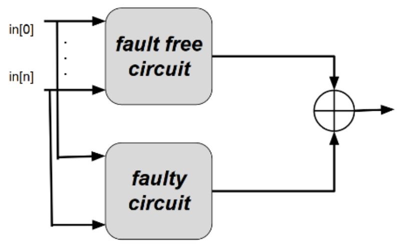


Fig. 2. Test Pattern Generation Circuit Example

The classical test pattern generation circuit example can be generally represented by the Figure 2. This representation consists in the comparison of the fault free and a fault prone circuits output. Usually, the behavior of the circuit with a fault *f* (faulty circuit) is different from the behavior of the circuit without faults (fault free circuit). The test pattern problem is addressed as the process to find a test pattern where the primary outputs (PO) of the fault-free and fault-prone circuits differentiate, i.e., the fault can be noticed in the circuit-under-test output. When the process finds a test pattern, it is said that the test pattern detects the current fault f. However, there are some cases where the behavior of the fault free and faulty circuit is the same. Faults that do not affect the circuit PO are called redundant faults [3].

There are some cases where it is impossible to find a test pattern that causes an unexpected output value. Depending which cells are used to map a logic circuit, the result could lead to a structure more or less controllable and/or observable. In other words, when there is not a combination of signals values that allow to control the fault *f* node and/or let the fault effect propagate from the fault site to at least one of the primary outputs, there is no test pattern that can test the fault *f*.

The main objective of and ATPG tool is to generate a reduced test set that guarantees a ratio between the number of detected faults and total faults reach (fault coverage) close to 100%. Depending on the standard cell library used to build the circuit structure, an ATPG tool can reach better or worse results, consequently, a faster or slower testing process.

III. METHODOLOGY

To analyze the impact of different circuits implementations in an ATPG system considering only combinational circuits, the ISCAS85 Benchmarks circuits [1] were mapped and described in Verilog through the ABC tool [4]. The libraries used were called *Max*, *Med* and *Min*, which are formed by the gates presented in Table I, ordered according to the number of functions and complexity. It is expected that large standard cell libraries result in circuits with reduced number of gates and logic depth [5]. Circuits with smaller number of gates and smaller logic depth, intuitively tend to present a better controllability and observability of internal signals. With this, it is expected to make easier the ATPG process.

TABLE I. GATES IN LIBRARIES MAX, MED AND MIN.

Library	Gates
Max	AND2/AND3/AND4/AOI21/AOI22/AOI33/BUFF/INV/MX2/NAND2/NAND3/NAND4/NOR2/NOR3/NOR4/OAI21/OAI22/OAI33/OR2/OR3/OR4/XOR2
Med	BUFF/INV/NAND2/NAND3/NAND4/NOR2/NOR3/NOR4
Min	BUFF/INV/NAND2/NOR2

After mapping, the circuits were tested in a SAT (Boolean Satisfiability Method) based ATPG developed in Java using the library SAT4J [6]. The ATPG tool was executed for the different versions of the circuits, resulting in a several numbers of vectors, a certain fault coverage, a CPU time and the number of unSAT and aborted faults. The unSAT faults are the faults that the SAT solver is not able to find the test pattern, consequently, there is no patterns that can test this set of faults. The aborted faults are the faults not solved in a specific time. In this work the time out value is set to 1s. This value has been chosen aiming a good fault coverage without compromise too much the CPU time.

As mention before, the ATPG tool generates three main factors: the CPU time, the number of vectors and the fault coverage. As expected results, the goal is to achieve a smaller number of test pattern set, a reduced CPU time and a fault coverage close to 100%. These factors can be related in a Figure of Merit metric (FoM), presented in Equation (1). With the FoM metric it is possible to compare the performance of ATPG methods. The goal is to minimize the FoM value. The metric enables to configure the relevance of each factor through the *m*, *n* and *p* exponents. This paper uses two configurations resulting in metrics represented by Equations (2) and (3). The first one prioritizes the pattern set and the second prioritizes the fault coverage. These two factors are the most important. The pattern set size influence directly the time of the testing process, and the fault coverage describes the test pattern set quality. Those FoM metrics are aiming to show in numbers which implementation is the best for each circuit.

$$FoM = \left(\frac{1}{F_c}\right)^m * (V)^n * (CPU_{time})^p \quad (1)$$

$$FoM_v = \frac{1}{FC} * (V)^2 * CPU_{time} \quad (2)$$

$$FoM_{FC} = \left(\frac{1}{FC}\right)^2 * V * CPU_{time} \quad (3)$$

IV. Results and Conclusions

The analysis of the results can be divided in two topics. The first one presents the features of the Benchmarks mapped with the ABC system and the second analyze the results obtained by the ATPG tool and discusses the conclusions formulated though the Figures of metrics.

The ISCAS85 Benchmark Circuits mapped by ABC system have different amounts of gates and levels according to the library used. The results for each circuit are presented in the Table II. The first column contains the ISCAS85 benchmark circuits and the subsequent columns show the number of gates and levels for each circuit mapped with the libraries *Max*, *Med* and *Min*. As expected, the libraries with more types of logic gates produce circuits with smaller number of gates and levels.

TABLE II. NUMBER OF GATES AND LEVELS.

Circuit	Library Min		Library Med		Library Max	
	#Gates	#Levels	#Gates	#Levels	#Gates	#Levels
c432	435	24	197	18	152	14
c499	680	20	571	16	314	11
c880	520	20	349	15	210	11
c1355	680	20	571	16	314	11
c1908	843	29	563	23	224	14
c2670	1154	18	807	14	501	10
c3540	1697	35	1002	29	642	18
c5315	2295	36	1471	33	980	18
c6288	4198	90	3352	89	1403	45
c7552	3055	29	2232	27	1227	15

The Table III shows the obtained data from the used ATPG tool: the CPU time, number of patterns, fault coverage, number of unSAT faults and the number of aborted faults for each ISCAS85 circuit implementation. The obtained data were used to calculate the metrics presented in Equation (2) and (3) for each circuits implementation. These results are presented in Table IV. The better FOMs are highlighted. We will use these metrics to try to identify the best implementation.

Analyzing the metric presented in Equation (3), which gives more importance to the fault coverage, one can notice that in 60% of the cases, the implementations using the Max library achieve the best result. This behavior is due to the smaller number of gates, that implies in a smaller fault set, consequently a smaller number of test patterns, a smaller or similar fault coverage to the other implementations and a considerably shorter runtime. In contrast, in 30% and 10% of the cases, implementations with the Min and Med library had a better performance, obtaining the best or equal results in the three aspects analyzed: fault coverage, run time and pattern set. From this perspective, the initial hypothesis has to the better investigated since depending of the logic of the circuit, an implementation mapped with a smaller set of gates available in the library can be a good alternative in question of testing process.

TABLE III. ATPG TOOL VALIDATION FOR THE ISCAS85 CIRCUITS IMPLEMENTATIONS.

Circuit	Library	CPU Time(s)	#Patterns	FC	#unSAT Faults	#Aborted Faults
c432	Min	848.8	94	89,94%	23	0
	Med	246.2	77	89,42%	68	0
	Max	161.4	65	96,36%	109	0
c499	Min	806.9	115	99,66%	6	0
	Med	877.4	146	99,56%	8	0
	Max	895.7	167	99,41%	8	0
c880	Min	576.6	108	100,00%	0	0
	Med	536.3	114	100,00%	0	0
	Max	572.9	121	100,00%	0	0
c1355	Min	883.4	112	99,66%	6	0
	Med	1106.2	167	99,56%	8	0
	Max	891.2	165	99,41%	8	0
c1908	Min	1956.4	163	99,65%	8	0
	Med	1769.6	181	99,43%	11	0
	Max	440.6	142	99,62%	4	0
c2670	Min	7846.1	331	94,70%	134	0
	Med	9338.0	350	96,74%	70	0
	Max	4141.8	408	98,15%	37	0
c3540	Min	32938.1	233	97,74%	72	16
	Med	11533.9	247	97,31%	77	8
	Max	7148.8	248	98,72%	31	5
c5315	Min	23364.2	317	99,60%	21	0
	Med	23776.2	437	99,08%	44	0
	Max	8326.2	295	99,35%	26	0
c6288	Min	47618.7	85	99,56%	17	35
	Med	54677.8	254	96,35%	235	134
	Max	3740.5	80	99,88%	1	6
c7552	Min	19993.8	297	98,81%	85	0
	Med	41231.0	304	97,68%	159	0
	Max	52745.5	304	98,77%	64	0

Through the analyze of the metric in Equation (2), which gives more relevance to the pattern set, one can notice that the previous behavior repeats. The same 60% for the same implementations with the Max library, reinforcing the previous conclusions that implementations which have more gates for mapping tend to obtain the best results for the three analyzed aspects. The remaining circuits, representing 40% of the circuits, got the best results for the Min library. Thus, it is possible to conclude that regardless of the metric used, (2) or (3), in most cases, libraries that have a greater number of gates, generate a smaller pattern set and a high fault coverage in function of their reduced number of faults. However, it is also seen that depending on the logic of the circuit, a mapping containing few types logical gates, such as the Min or Med library, even resulting in more gates and levels, can become a better alternative mapping for testing process.

TABLE IV. FIGURE OF MERIT FOR ISCAS85 BENCHMARK.

Circuit	Library	FOM_FC	FOM_V
c432	Min	9,86E+04	8,34E+06
	Med	2,37E+04	1,63E+06
	Max	**1,13E+04**	**7,08E+05**
c499	Min	**9,34E+04**	**1,07E+07**
	Med	1,29E+05	1,88E+07
	Max	1,51E+05	2,51E+07
c880	Min	6,23E+04	**6,73E+06**
	Med	**6,11E+04**	6,97E+06
	Max	6,93E+04	8,39E+06
c1355	Min	**9,96E+04**	**1,11E+07**
	Med	1,86E+05	3,10E+07
	Max	1,49E+05	2,44E+07
c1908	Min	3,21E+05	5,22E+07
	Med	3,24E+05	5,83E+07
	Max	**6,30E+04**	**8,92E+06**
c2670	Min	2,90E+06	9,08E+08
	Med	3,49E+06	1,18E+09
	Max	**1,75E+06**	**7,02E+08**
c3540	Min	8,03E+06	1,83E+09
	Med	3,01E+06	7,23E+08
	Max	**1,82E+06**	**4,45E+08**
c5315	Min	7,47E+06	2,36E+09
	Med	1,06E+07	4,58E+09
	Max	**2,49E+06**	**7,29E+08**
c6288	Min	4,08E+06	3,46E+08
	Med	1,50E+07	3,66E+09
	Max	**3,00E+05**	**2,40E+07**
c7552	Min	**6,08E+06**	**1,78E+09**
	Med	1,31E+07	3,90E+09
	Max	1,64E+07	4,94E+09

V. Final remarks

The paper introduced basic concepts about Standard Cell Design Flow and the Test Pattern Generation problem for a better understanding of the aimed analysis. Three cell libraries were used to map the ISCAS85 circuits aiming to analyze the impact in ATPG systems. Based on the obtained data from the ATPG tool used and the calculated metrics, it is concluded that large libraries obtained better results in 60% of the cases due to a smaller fault set. However, the minimum library can be an alternative too, representing 40%, depending of the logic of the circuit. From this perspective, the initial hypothesis has to the better investigated in the future.

References

[1] ISCAS85. (2016) Iscas85 combinational benchmark circuits. [Online]. Available: https://filebox.ece.vt.edu/ mhsiao/iscas85.html

[2] V. N. Kravets, "Constructive multi-level synthesis by way of functional properties," Ph.D. dissertation, Univ. Michigan, Ann Arbor, 2001.

[3] Y. Matsunaga, "An accelerating technique for sat-based atpg," in IPSJ Transactions on System LSI Design Methodology, Feb 2017, pp.39–44.

[4] Berkeley Logic Synthesis and Verification Group. (2013) ABC: A System for Sequential Synthesis and Verification, Release 30723. Available: *http://www.eecs.berkeley.edu/~alanmi/abc/*

[5] Binghong Guan and C. Sechen, "Large standard cell libraries and their impact on layout area and circuit performance," Proceedings International Conference on Computer Design. VLSI in Computers and Processors, Austin, TX, 1996, pp. 378-383.

[6] Université d'Artois. (2017) SAT4J, the boolean satisfaction and optimization library in Java. Disponível em:<http://www.sat4j.org/>.