A BIST GENERATOR CAD TOOL FOR NUMERIC INTEGRATED CIRCUITS

Chiraz Khedhiri[1], Mouna Karmani[1] and Belgacem Hamdi[1,2]

[1]Electronic & Microelectronics'LAB, Monastir, Tunisia
[2]ISSAT, Sousse, Tunisia
chirazkhedhiri@yahoo.fr
mouna.karmani@yahoo.fr
Belgacem.Hamdi@issatgb.rnu.tn

ABSTRACT

This paper describes a training and research tool for learning basic issues related to BIST (Built-In Self-Test) generator. The main didactic aim of the tool is presenting complicated concepts in a comprehensive graphical and analytical way. The paper describes a computer-aided design (CAD) that is used to generate automatically the BIST to any digital circuit. This software technique attempts to reduce the amount of extra hardware and cost of the circuit.

In order to make our software being easily available, we used the Java platform, which is supported by most operating systems. The multi-platform JAVA runtime environment allows for easy access and usage of the tool.

KEYWORDS

BIST generator, test, Computer-Aided Design, Tool, Linear Feedback Shift Register, Single/Multiple Input Signature Register, fault.

1. INTRODUCTION

In recent years, the development of integrated circuit technology has accelerated rapidly. Accordingly, digital systems are built with more and more complexity; the fault testing and diagnosis of digital circuits becomes an important and indispensable part of the manufacturing process [1].

The difficulty found in testing today's Very Large Scale Integrated (VLSI) circuits is essentially due to the inaccessibility of the internal nodes for probing. In the last decades when electronic circuits were realized with discrete components, each component was easily accessible and the signals at its pins could be checked by a test probe. Now the internal nodes of a VLSI circuit are only controlled and observed through the pins of the chip by sensitizing a path from these pins to the nodes. There would not have been any problem were it not that sensitive paths cannot always be found for each node either because they do not exist or because finding them is too time–consuming.

This obstacle can be overcome by inserting memory elements on some of the nodes and then connecting these memory elements – in the form of a scan chain – to one of the outputs during test mode which renders internal nodes totally controllable and observable. Thus, test patterns can be scanned in to exercise the internal nodes and response patterns are scanned out for comparison. A better solution would be to use the memory elements, combined with some additional logic, as a Test Pattern Generator (TPG) so as to produce the test patterns inside the circuit itself. Response analysis would also be done on-chip by an Output Response Analyzer (ORA) so as to produce a "PASS/FAIL" signal.

This last technique referred to Built-In Self-Test (BIST) [2], is regarded today as the solution for tomorrow's test problems as, owing to the increasing complexity and density of the circuits, testing can be effectively carried out only by dedicating part of the circuit to it. In this way, Automatic Test Equipments (ATEs) will be greatly simplified; furthermore, at-speed testing and in-system checks for maintenance purposes would be made possible [3].

Logic built-in self-test is a design for testability (DFT) technique in which a portion of a circuit on chip, board, or system is used to test the digital logic circuit itself. Logic BIST is crucial for many applications, in particular for life-critical and mission-critical applications. These applications commonly found in the aerospace/defence, automotive, banking, computer, healthcare, networking, and telecommunications industries require on-chip, on board, or in system self-test to improve the reliability of the entire system, as well as the ability to perform remote diagnosis [4].

At present, the major barriers that prevent design engineers from using BIST extensively are test generation time and hardware cost of the TPGs. This is the reason why design for testability (DFT) and test are receiving more and more attention in the electronics community. Hence, the curricula in computer and electronics engineering should also incorporate a greater emphasis on DFT and on the concepts of digital testing. Young engineers should be trained on integrating design and test. Teaching in this domain should be facilitated by having integrated CAD tools that support test pattern generation, fault simulation and fault grading, testability analysis, DFT and built-in self-testing (BIST).

In this paper, we describe a knowledge-based computer-aided design (CAD) tool to plan the use of built-in self-test (BIST) in VLSl design and specially the use of BIST in digital circuits. Software is being developed using the Java platform.

The software is an illustrative tool explaining the problems of fault modelling and simulation in digital systems as well as test generation and fault diagnosis problems. Unlike hardware testing techniques, the use of the CAD tool attempts to reduce the amount of extra hardware and cost of the circuit.

The work is organised as follows. Section 2 describes the general BIST structure. In section 3, we present the proposed CAD TOOL and we conclude in section 4.

2. BIST STRUCTURE

Logic testing of integrated circuits (ICs) is important to ensure a high level of quality in product functionality in both commercially and privately produced products. In the modern System-ona-Chip (SoC) design, many cores are integrated into a single chip. Some of them are embedded, and cannot be accessed directly from the outside of the chip. Such SOC designs make the test of these embedded cores become a great challenge. The Built-In-Self-Test (BIST) is one of most popular test solutions to test the embedded cores [5].

As the digital circuit technology is moving to high densities of integration, built-in self-testing has become a primary issue in the realm of VLSI circuit design. Techniques for design for testability and built-in self-test consider the testing problem during the design stage of digital devices and have been found to be extremely effective. The central idea behind built-in selftesting or BIST is to have the chip test itself. This technique generates test patterns and evaluates output responses inside the chip [6, 7, 8].

Built-in Self-test is gaining popularity as a means to address test issues at the different packaging levels of digital systems. One of the benefits of BIST is the fact that no patterns need to be stored in the test equipment, which is simply required to provide a clock and a few control signals. This is especially important when high performance systems are being tested. BIST also makes the chip/board/system more independent of the specific test resources available at each manufacturing stage. BIST is also a convenient way of applying more test patterns to compensate for the weaknesses of the stuck-at fault model [9].

BIST can significantly improve the testability of VLSI chips and save testing time as well [10]. BIST is a design-for-testability technique that places the testing functions physically with the CUT, as illustrated in Fig. 1. In normal operating mode, the CUT receives its inputs *X* from other modules and performs the function for which it was designed. In test mode, a Linear Feedback Shift Register (LFSR) applies a sequence of test patterns to the CUT, and a multiple input signature register (MISR) compacts test responses received from primary output. The response signatures are compared with reference signatures generated or stored on-chip, and the error signal indicates any discrepancies detected.

The basic blocks that forms the BIST are: **LFSR** (Linear Feedback Shift Register), **CUT** (Circuit Under Test), **SISR/MISR** (Single/Multiple Input Signature Register), **BIST controller** and **signature analyzer**.

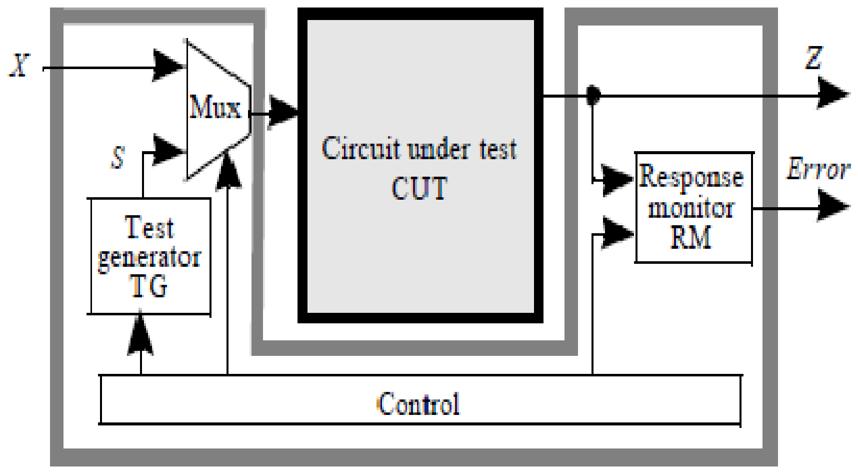


Figure 1: General BIST structure

2.1. LFSR (Linear Feedback Shift Register)

The main part of BIST is LFSR which can generate random numbers that are used to identify the physical faults in the Integrated Circuit (IC).

In this section we consider briefly the mathematical definitions and theorems related to an LFSR. Details can be found in literatures [11, 12].

A Linear Feedback Shift Register is a sequential shift register with combinational logic. It pseudo-randomly cycles through a sequence of binary values. There are two ways to implement LFSRs : Internal feedback (Fig. 2) and External feedback (Fig. 3). These techniques differ in the way feedback is applied. All the flip-flops that feed an XOR gate are known as 'taps'. These taps decide the patterns generated by the LFSR and hence define the characteristic polynomial of an LFSR.

In Fig. 2, a common internal structure of LFSR (type 1) is given. It consists of D flip-flops connected in series and forming a simple shift register and a special kind of feedback loops collected by XOR gates. The presence or absence of the feedback loops is described by a socalled generator polynomial. The state of the LFSR at the beginning of test generation is determined by its initial state parameter (***Seed***).

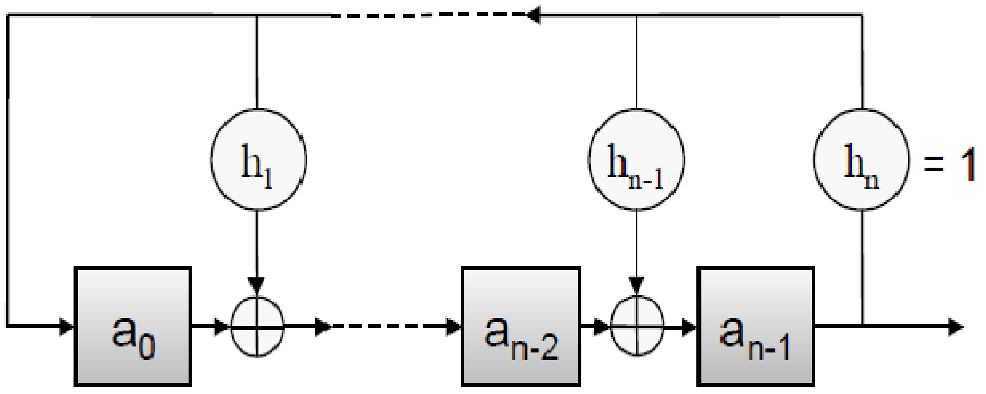


Figure 2: LFSR with internal feedback (type 1)

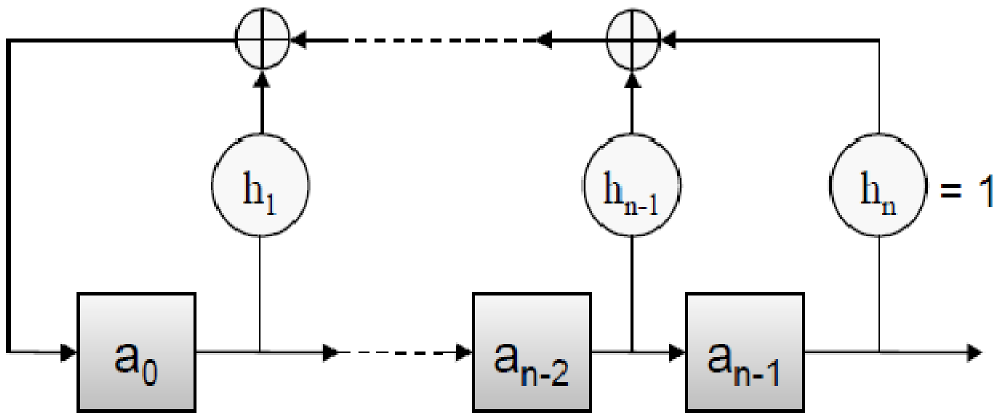


Figure 3: LFSR with external feedback (type 2)

The main useful property of LFSR circuits is such that if clocked repeatedly, they go through a fixed sequence of states, which has a number of explicit properties of randomness and can be used, therefore, as a TPG in a BIST scheme [13, 14]. The maximum number of such states is ($2_n - 1$), where n is the length of the LFSR (no. of flip-flops). However, the actual length of the sequence depends on the selected polynomial and in some cases on the initial state as well. These two LFSR parameters have direct influence to the resulting test quality and, therefore, they play an important role in TPG.

Every LFSR has a characteristic polynomial that describes its behaviour. The characteristic polynomial of an n-bit LFSR has the form:

$$P(x) = 1 + h_1.x + \ldots + h_{n-1}.x^{n-1} + h_n.x^n . \quad (1)$$

Where h_i are either 1 or 0 [15].

Table 1 provides a primitive polynomial for values of n between 1 and 36 [16]. For example, for degree 30 we have the primitive polynomial :

$$P(x) = x^{30} + x^{16} + x^{15} + x + 1 .$$

Table 1: example of primitive polynomial

degree					**degree**					**degree**				
1	0				**13**	4	3	1	0	**25**	3	0		
2	1	0			**14**	12	11	1	0	**26**	8	7	1	0
3	1	0			**15**	1	0			**27**	8	7	1	0
4	1	0			**16**	5	3	2	0	**28**	3	0		
5	2	0			**17**	3	0			**29**	2	0		
6	1	0			**18**	7	0			**30**	16	15	1	0
7	1	0			**19**	6	5	1	0	**31**	3	0		
8	6	5	1	0	**20**	3	0			**32**	28	27	1	0
9	4	0			**21**	2	0			**33**	13	0		
10	3	0			**22**	1	0			**34**	15	14	1	0
11	2	0			**23**	5	0			**35**	2	0		
12	7	4	3	0	**24**	4	3	1	0	**36**	11	0		

2.2. CUT (Circuit Under Test)

The system that is to be tested is termed as CUT. It is the circuit of the IC (Integrated Circuit) that is going to be checked for any defects after its manufacturing. Any digital design represented in one of the Hardware Description Languages (HDL's) is used as a CUT.

2.3. MISR (Multiple Input Signature Register)

BIST requires ability to capture the test results without the need for an external tester. This is often achieved by using a multi-input signature register (MISR) to capture individual test results and compress these into an overall value called the test signature. A MISR is quite similar to an in-tapping LFSR, it consists of *n* memory cells ($M_1 \ldots M_n$) with linear feedbacks from cell M_n (n is the number of output of the CUT).

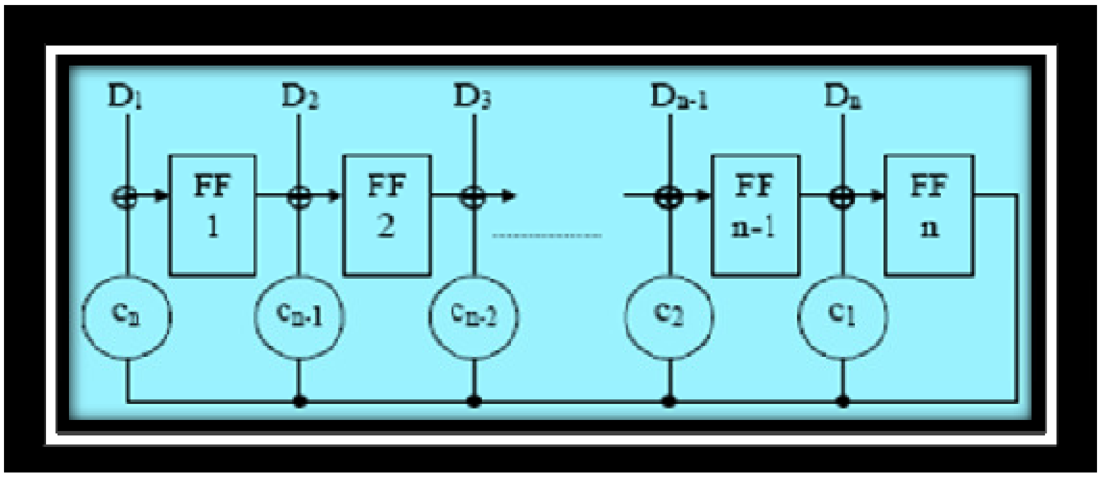


Figure 4: An n-bit MISR model [17]

Like the LFSR, the MISR can be characterized by the polynomial given in Eq. (1). Given the Characteristic Polynomial and the initial state of the MISR, every cycle, the MISR will generate a new state, based on the current state, the Characteristic Polynomial and the response of the CUT. The state of the MISR is referred to as the *signature*. Given the initial state, the Characteristic Polynomial and the set of responses from the CUT to a given test, the signature of the MISR is determined after the complete test can be determined. This signature is compared to the signature of a simulated fault-free circuit. When these signatures do not match, the CUT is not fault-free.

2.4. Signature Analyzer

For BIST operations, it is impossible to store all output responses on-chip, on-board, or in system to perform bit by bit comparison. An output response analysis technique must be employed such that output responses can be compacted into a ***signature*** and compared with a golden signature for the fault-free circuit either embedded on-chip or stored off-chip. If they are equal, then it means the IC is good and if not it is bad.

2.5. BIST Controller

BIST controller coordinates the operations of different blocks of the BIST. Based on the test mode input to the controller, the system either operates in the normal mode or in the test mode.

3. PROPOSED CAD TOOL

A digital circuit module CUT (circuit under test) is tested for faults by applying a set of inputs (test set) and processing the resulting outputs (test verification). Circuits with built-in test generate the test set on-chip as well as verify on-chip [18, 19].

This paper describes a computer-aided design (CAD) that is used to generate automatically the BIST to any digital circuit. The software tool generates automatically built-in self-test blocks into VHDL models of digital circuits by giving the suitable values of initial seed and primitive polynomial of the LFSR. In order to make our software being easily available, we used the Java platform, which is supported by most operating systems.

The different GUI modules of the BIST Generator are shown in Fig. 5.

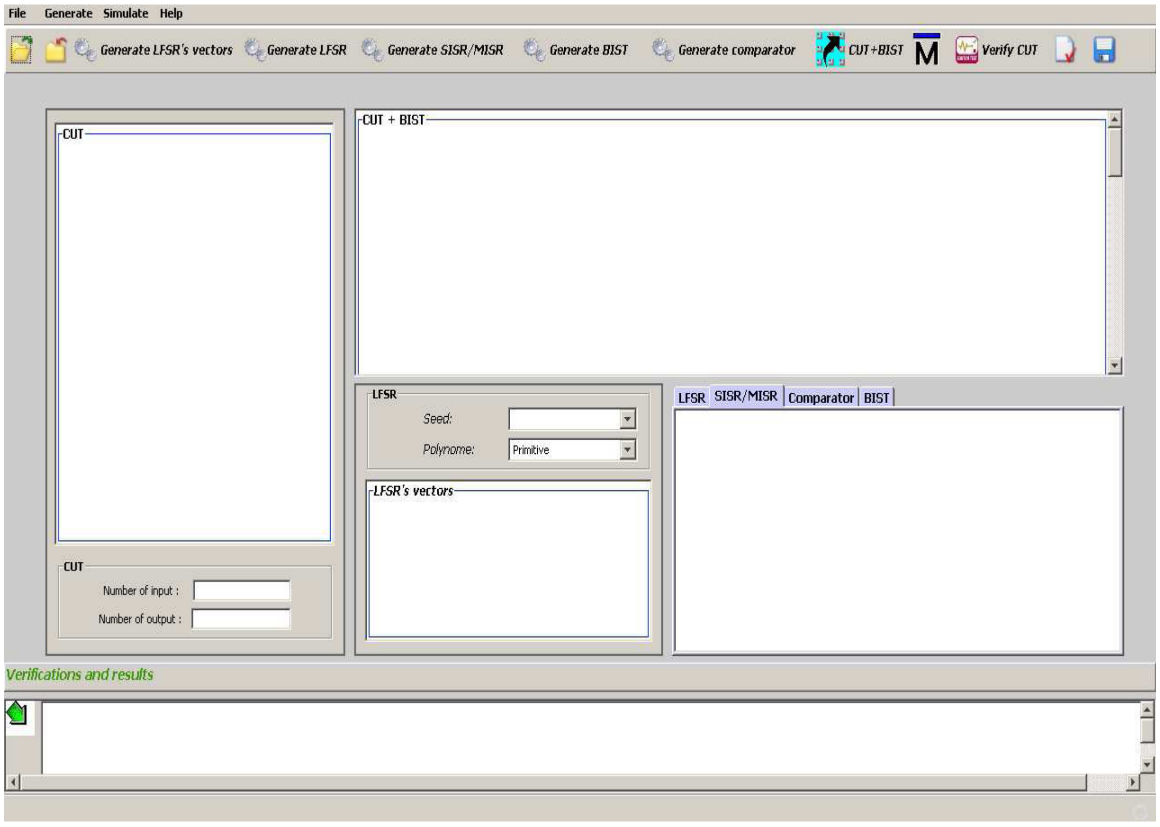
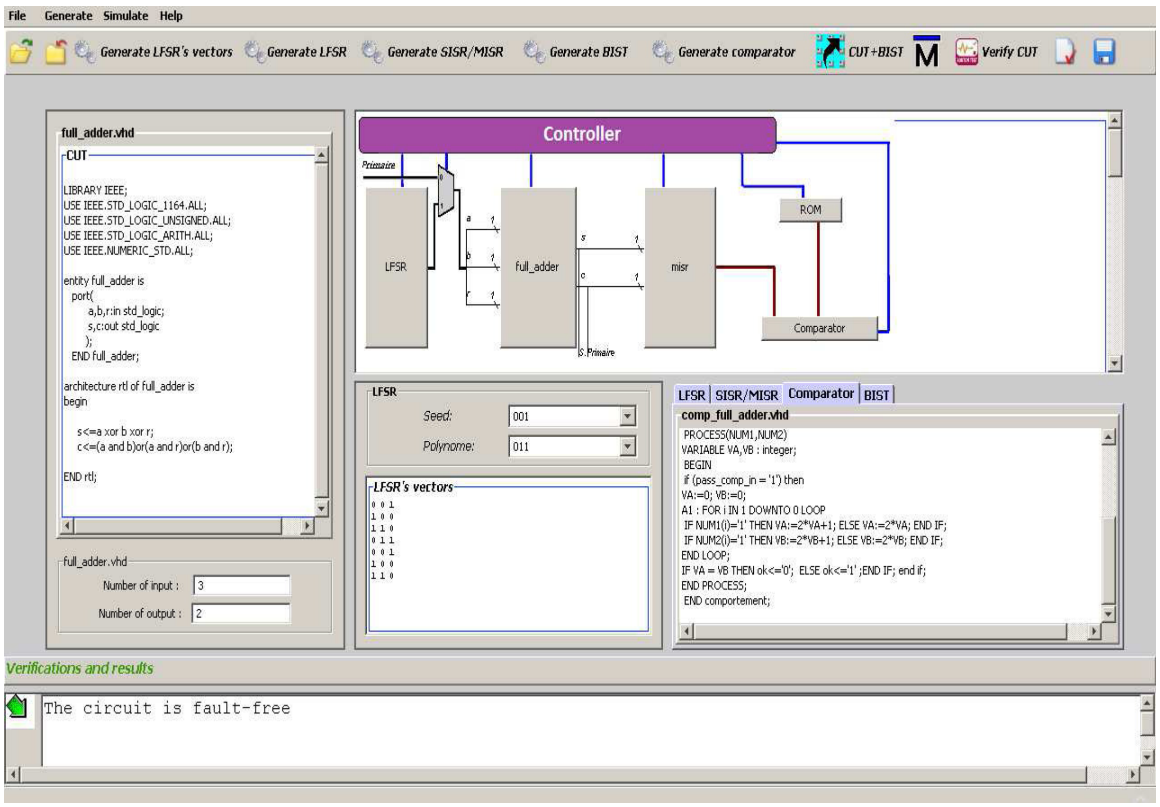


Figure 5: The BIST Generator CAD TOOL

3.1. Case Study Of A Fault-Free Circuit (Fault-Free Full Adder)

The interface region includes five control panels (Fig. 6).



full_adder.vhd

CUT

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.NUMERIC_STD.ALL;

entity full_adder is
  port(
      a,b,r:in std_logic;
      s,c:out std_logic
     );
  END full_adder;

architecture rtl of full_adder is
begin

   s<=a xor b xor r;
   c<=(a and b)or(a and r)or(b and r);

END rtl;
```

full_adder.vhd

Number of input : 3

Number of output : 2

LFSR

Seed: 001

Polynome: 011

LFSR's vectors

```
0 0 1
1 0 0
1 1 0
0 1 1
0 0 1
1 0 0
1 1 0
```

comp_full_adder.vhd

```
 PROCESS(NUM1,NUM2)
VARIABLE VA,VB : integer;
 BEGIN
 if (pass_comp_in = '1') then
VA:=0; VB:=0;
A1 : FOR i IN 1 DOWNTO 0 LOOP
 IF NUM1(i)='1' THEN VA:=2*VA+1; ELSE VA:=2*VA; END IF;
 IF NUM2(i)='1' THEN VB:=2*VB+1; ELSE VB:=2*VB; END IF;
END LOOP;
IF VA = VB THEN ok<='0';  ELSE ok<='1' ;END IF; end if;
END PROCESS;
 END comportement;
```

```
The circuit is fault-free
```

Figure 6: The BIST Generator CAD TOOL (case of a fault-free circuit)

The first panel is used to insert the VHDL code of the numeric circuit (CUT) which will be tested (in this case we have chosen a full-adder as an example). From the given VHDL code, the tool calculates the number of inputs (3 in the case of full-adder) and outputs (2 in this case) of the CUT (full-adder). Than, the BIST's schema is automatically posted in the second panel.

By giving the suitable value of the seed and the primitive polynomial of the LFSR, the LFSR's vectors can be calculated.

Here, the initial state of the LFSR is (001). The initial state repeats after 7 cycles. The same sequence will repeat the next cycle and thereafter. If one examine the sequence carefully, one will notice that the period of the recurrence is 7 or $2_3 - 1$. The recurrence cycle contains all possible combinations except all zeros (000). Here, we would like to look into the characteristic polynomial and the patterns it generates in more detail.

The characteristic polynomial is: $P(x) = x^3 + x^2 + 1$ (011)

Fig.7 shows an example of the three stage LFSR.

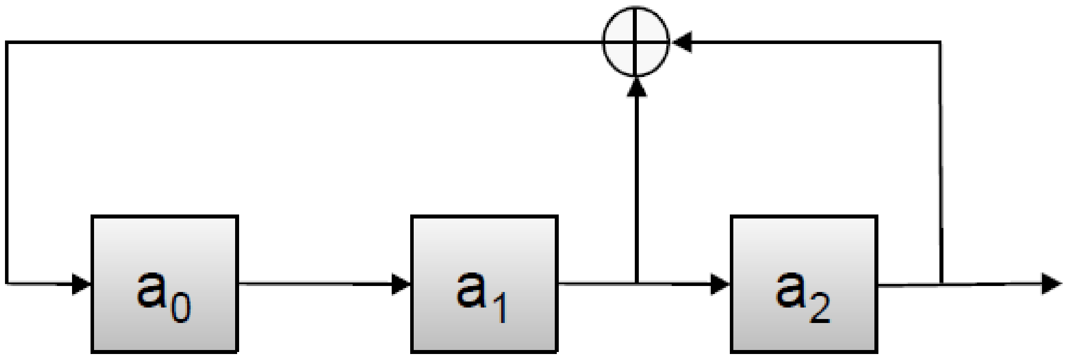


Figure 7: Example of a three-stage LFSR

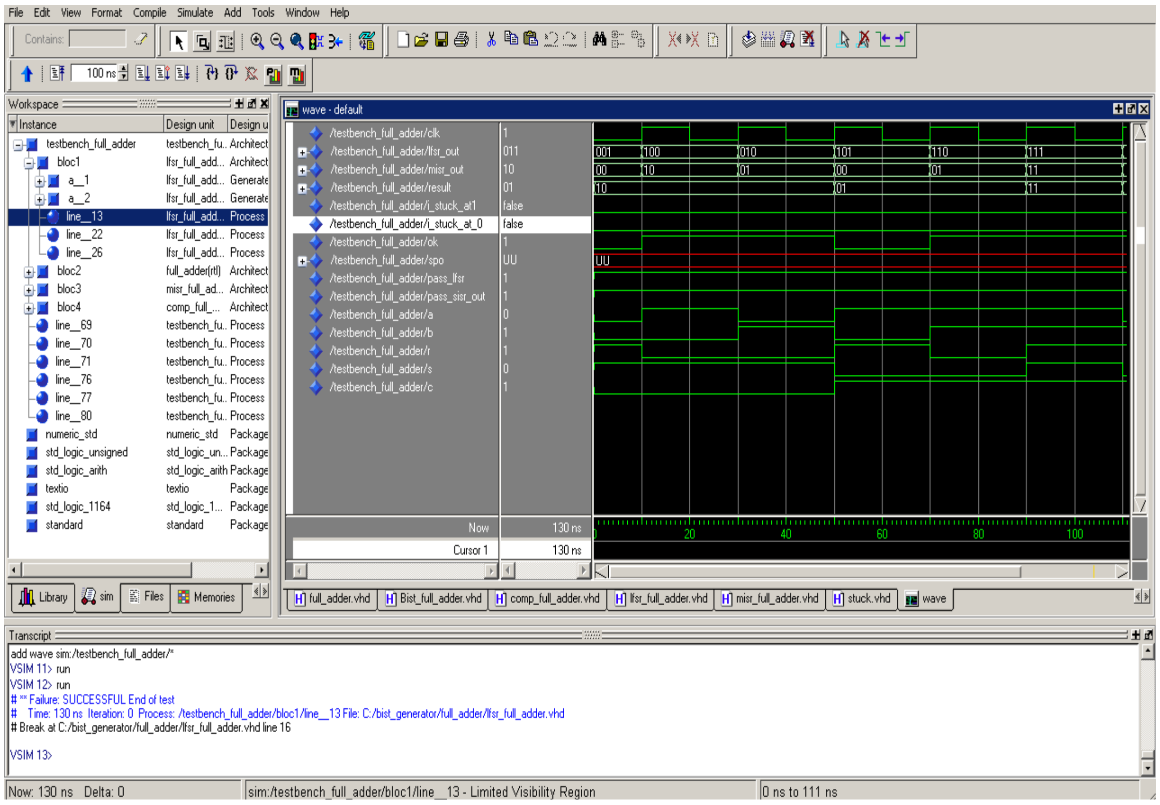
The patterns generated by the LFSR are:

Table 2: vectors generated by the LFSR

V2	V1	V0
0	0	1
1	0	0
1	1	0
0	1	1
0	0	1
1	0	0
1	1	0
0	0	1

In the fourth panel, the software tool will automatically generate the VHDL code of different blocks of the CUT (LFSR.vhd, SISR/MISR.vhd, comparator.vhd and BIST.vhd).

After simulating the different blocks of the BIST_full-adder using the MODELSIM tool (Fig. 8), and by clicking on the button *verify* the result of the test will be posted in the fifth column (Fig. 6).



```
/testbench_full_adder/clk              1
/testbench_full_adder/lfsr_out         011
/testbench_full_adder/misr_out         10
/testbench_full_adder/result           01
/testbench_full_adder/i_stuck_at1      false
/testbench_full_adder/i_stuck_at_0     false
/testbench_full_adder/ok               1
/testbench_full_adder/spo              UU
/testbench_full_adder/pass_lfsr        1
/testbench_full_adder/pass_sisr_out    1
/testbench_full_adder/a                0
/testbench_full_adder/b                1
/testbench_full_adder/r                1
/testbench_full_adder/s                0
/testbench_full_adder/c                1
Now                                    130 ns
Cursor 1                               130 ns
```

```
add wave sim:/testbench_full_adder/*
VSIM 11> run
VSIM 12> run
# ** Failure: SUCCESSFUL End of test
#    Time: 130 ns  Iteration: 0  Process: /testbench_full_adder/bloc1/line__13 File: C:/bist_generator/full_adder/lfsr_full_adder.vhd
# Break at C:/bist_generator/full_adder/lfsr_full_adder.vhd line 16

VSIM 13>
```

```
Now: 130 ns   Delta: 0   sim:/testbench_full_adder/bloc1/line__13 - Limited Visibility Region   0 ns to 111 ns
```

Figure 8: Simulation results of the full-adder using the Modelsim tool

The mechanism of testing is shown in Fig. 9. A set of input stimuli is applied to the CUT and the output response of that circuit is compared to the known good output response, or expected response, to determine if the circuit is "good" or "faulty".

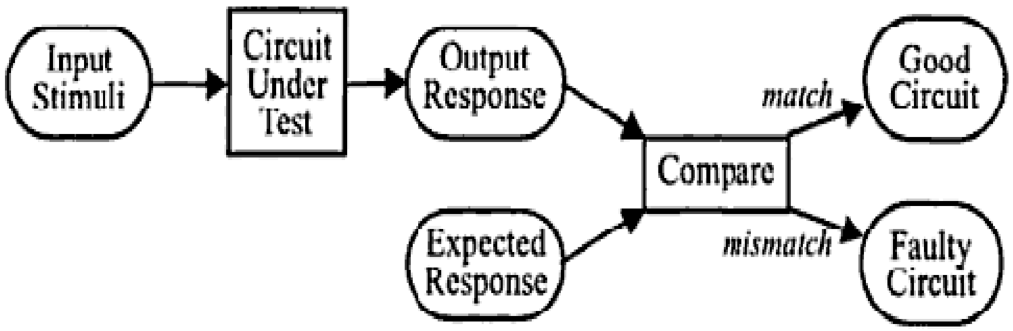


Figure 9: The basic testing mechanism [20]

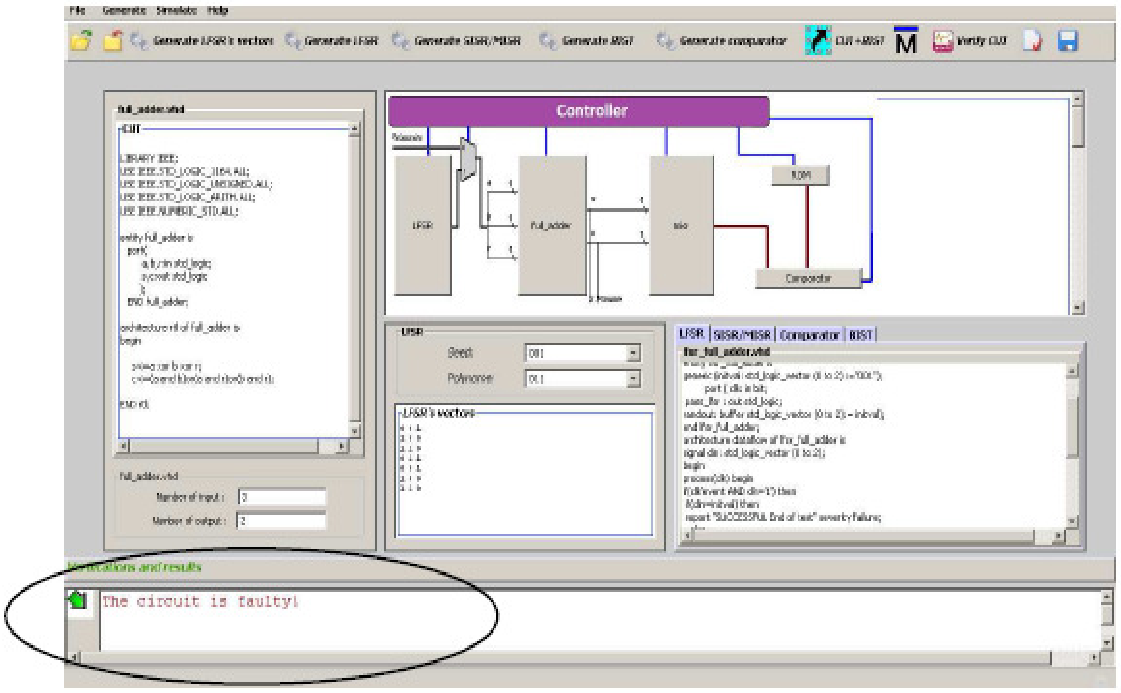
As we can see in Fig. 6, the full-adder tested is *fault-free*. There is no fault found in the circuit.

3.2. Case Study Of A Faulty Circuit (Full Adder)

In order to prove the efficiency of the proposed CAD tool, we simulate the CUT in the presence of faults. Faults are voluntary injected in the inputs of the circuit under test.

Fault is that defect which affects the circuit operation [21].

After injecting a Stuck-At-0 fault in the primary input 'a' of the full-adder, the BIST generator indicates that the circuit is faulty (Fig. 10)



full_adder.vhd

CUT

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.NUMERIC_STD.ALL;

entity full_adder is
  port(
      a,b,cin:in std_logic;
      s,cout:out std_logic
      );
  END full_adder;

architecture rtl of full_adder is
begin

   s<=a xor b xor ci;
   c<=(a and b)or(a and ci)or(b and ci);

END rtl;
```

full_adder.vhd

Number of input : 3

Number of output : 2

LFSR

Seed: 001

Polynomial: 011

LFSR's vectors

```
[illegible]
```

lfsr_full_adder.vhd

```
generic (inival : std_logic_vector (0 to 2) :="001");
       port ( clk : in bit;
pass_lfsr : out std_logic;
randout: buffer std_logic_vector (0 to 2) := inival);
end lfsr_full_adder;
architecture dataflow of lfsr_full_adder is
signal clk : std_logic_vector (0 to 2);
begin
process(clk) begin
if(clk'event AND clk='1') then
 if(clk=inival) then
report "SUCCESSFUL End of test" severity failure;
```

```
The circuit is faulty!
```

Figure 10: The BIST Generator CAD TOOL (case of a faulty circuit)

4. CONCLUSIONS

This paper describes a computer-aided design (CAD) that is used to generate automatically the BIST to any digital circuit. The software tool generates automatically built-in self-test blocks into VHDL models of digital circuits by giving the suitable values of initial seed and primitive polynomial of the LFSR. The training tool is developed in order to turn learning of the BIST generation technique into an easy task.

As the test of VLSI circuits is the most complicated and time-consuming problem in digital design, the development of CAD presents an efficient solution for this problem. So, by using this software technique, the detection of the faulty design becomes more reliable, faster, and requires less space.

REFERENCES

[1] SM. Thamarai, K.Kuppusamy and T. Meyyappan, "Heuristic approach to optimize the number of test cases for simple circuits". International journal of VLSI design & Communication Systems (VLSICS) Vol.1, No.3, September 2010.

[2] T.W. Williams and K..P. Parker, "Design for Testability – A Survey". IEEE Trans. On Computers, Vol. C–31, N°1, pp. 2–15, 1982.

[3] Christian DUFAZA, Hélène VIALLON and Cyril CHEVALIER, "BIST Hardware Generator for Mixed Test Scheme ". This work is supported by the Commission of the European Communities DGXIII under ESPRIT Basic Research Action 7107 Archimedes.

[4] Laung-Terng Bang, Cheng-wen wu, Xiaoqing Wen, "VLSI TEST PRINCIPLES AND ARCHITECTURES", Design for testability, 2006.

[5] Charles E. Stroud, "A Designer's Guide to Built-in Self-Test", Kluwer Academic Publishers, 2002.

[6] Das, S. R. (Oct. 1991). "Built-in self-testing of VLSI circuits", IEEE Potentials, 10, pp. 23-26.

[7] McCluskey, E. J. (April 1985). "Built-in self-test techniques", IEEE Design and Test of Computers, 2, pp. 21- 28.

[8] Savir, J. and Bar,dell, P. H. (March 1993). "Built-in self-test: milestones and challenges", VLSI Design, 1, pp. 23-44.

[9] Pancholy A, Rajski J., McNaughton L. J.. "Empirical Failure Analysis and Validation of Fault Models in CMOS VLSI", International Test Conference '90, Washington D.C., 10-12. September 1990, pp. 938- 947.

[10] SUNIL R. DAS, NITA GOEL, WEN B. JONE and AMIYA R. NAYAK. "Syndrome Signature in Output Compaction for VLSI Built-in Self-Test", VLSI DESIGN, Vol. 7, No. 2, pp. 191-201, 1998.

[11] Peterson, W.W., and Weldon, J.J., "Error Correcting Codes". MIT Press, Cambridge, London, 1972.

[12] Ahmad, A., "Development of State Model Theory for External Exclusive NOR Type LFS Structures". Enformatika, Volume 10, December 2005, pp. 125 – 129, 2005.

[13] A. Crouch, "Design-for-test for Digital IC's and Embedded Core Systems", Prentice Hall, p 347,1999.

[14] M. Bushnell, V. Agrawal, "Essentials of Electronic Testing for Digital Memory & Mixed Signal VLSI Circuits", Kluwer Academic Publishers, p 712, 2000.

[15] B. W. Lee, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: "ANALOG AND DIGITAL SIGNAL PROCESSING", VOL. 45, NO. 3, MARCH 1998.

[16] P.H. Bardell, W.H. McAnney and J. Savir, "Built-In-Test for VLSI : Pseudorandom Techniques", John Wiley & Sons, 1987.

[17] Afaq Ahmad, "A Simulation Experiment on a Built-In Self Test Equipped with Pseudorandom Test Pattern Generator and Multi-Input Shift Register (MISR)". International journal of VLSI design & Communication Systems (VLSICS) Vol.1, No.4, December 2010.

[18] T.W. Williams and K.P.Parker, "Design for Testability – A Survey", IEEE Trans. Computers, vol. C-31, pp. 2-15, Jan. 1982.

[19] "Built-in Test – Concepts and Techniques", IEEE Computer Society Publications, October 17, 1983.

[20] Vinay Kumar, "ANALYSIS, VERIFICATION AND FPGA IMPLEMENTATION OF VEDIC MULTIPLIER WITH BIST CAPABILITY", Master of Technology in VLSI Design & CAD, Department of Electronics and Communication Engineering, THAPAR university, June, 2009.

[21] Ms. Shweta S. Meshram and Ms. Ujwala A. Belorkar, "DESIGN APPROACH FOR FAULT TOLERANCE IN FPGA ARCHITECTURE". International Journal of VLSI design & Communication Systems (VLSICS) Vol.2, No.1, March 2011.