FPGA Implementation of Optimized BIST Architecture for Testing of Logic Circuits

Ramya R[#1], Madhura R[#2]

[1] *Postgraduate Student, Department of Electronics and Communications Engineering, Dayananda Sagar College of Engineering, Bangalore, Karnataka, India*

[2]*Assistant Professor, Department of Electronics and Communication Engineering, Dayananda Sagar College of Engineering, Bangalore, Karnataka, India*

Abstract

Verification is used at each stage of VLSI design to ensure that the IC is working correctly, but most of the verification is done either at design time or at the time of designing or fabricating the IC. Along with verification at the design level, it is necessary to verify the operation of the chip after design and fabrication. Normally in such cases, it is done by placing it in the IC testing kit, which inserts all input combinations with comparing all output combinations to compare the correctness of the chip. But the cost of such kits is high and not easily available. As a result, it is essential to insert some extra logic inside the chip, which verifies the correctness of the chip. But it increases the area and power requirement of the chip. In this project, we have designed an efficient architecture of BIST to check the correctness of the design, which is simulated using the Xilinx ISE 14.5 design suite and VIVADO 2018.3 student version and is implemented on FPGA.

Keywords —*Verification, VLSI design, IC testing kit, Efficient architecture, Xilinx FPGA*

I. INTRODUCTION

As integrated chips are becoming larger and more complicated nowadays, exhaustive simulation is a very time-consuming process, and non-exhaustive simulation is done for a selected set of input patterns for which certain faults maybe failed to be detected. Formal verification is a solution to the above-stated problem. It is a technique that utilizes static analysis based on some mathematical transformations to find the correctness of hardware or software in alternate to dynamic verification techniques. It is very fast, and it takes less effort, and its performance majorly depends on the type of logic on which it is placed and the way it is applied.

In this technique, we deal with an abstract model of the system. To come up with the abstract model, we need formal methods as an abstract model of the system is easier to understand than the whole system. Here, we need a system model and specification/property to build an abstract model in which we can show that certain property holds good. In this technique, property that needs to be verified is decided first, and then the system model is designed and finally checks whether the property holds good when applied to the system model.

Formal verification includes a wide range of technologies, among them are Theorem proving, Equivalence checking, and Model checking. Theorem proving uses some axioms or rules to prove the correctness, whereas Equivalence checking always needs two designs, and it can be applied at or across various levels. This technology proves that for all possible input stimuli, their corresponding outputs are functionally equivalent.

Model checking is used for hardware verification. In this technology, predefined data are sent for which equivalent model is generated by the circuit under test, and then it is compared with the correct signature to find if there is any hardware fault in the design. BIST is similar to this technology. It is a design for testability method for testing the VLSI chips. The concept of BIST is to design a circuit to test Its own self (self-testing) and find whether the circuit has some fault or not. This technique makes electrical testing of a chip easier, faster, more efficient, and less expensive.

BIST is classified in a number of ways, but two major classifications are Logic BIST (LBIST) and Memory BIST (MBIST). MBIST is used for testing memory cores of the device, and LBIST is an inbuilt circuitry that will test the structural integrity of the chip after it is being manufactured. It not only decreases the cost of testing but also allows rapid testing of the circuit. We have proposed an optimized LBIST architecture for testing logic circuits, which reduces the overall hardware complexity of the circuit. The architecture consists of three major blocks, which are the Test Pattern Generator (TPG), Circuit Under Test (CUT), and Output Response Analyser (ORA). Standard 8-bit LFSR is designed as TPG for generating the pseudo-random test patterns, including the all-zero state. The generated test patterns are sent as input to the circuit under test. We have designed a 4*4 multiplier and used it as a circuit under test to verify if there are any hardware faults. In ORA, we have designed an 8-bit MISR as a signature finder, and we have designed a comparator to compare the signature value and output value of MISR. Based on this comparison result, we say a circuit is faulty or fault-free. Simulation and FPGA

implementation of the optimized BIST architecture is explained in detail.

II. REFERENCE STUDY

The BIST provides an automated test procedure to detect faults in memory cores and logic circuits. [1] describes a model of BIST that can be used as Memory BIST (MBIST) and Logic BIST (LBIST). By modifying the LFSR to a structure called Complete Linear Feedback Shift Register (CLFSR) they are able to generate both address sequence and test data with reduced hardware complexity. With CLFSR, Conventional methods that utilize two circuits for generating addressing sequence and test patterns are replaced by a single circuit, reducing the area overhead of the BIST. Xilinx ISE 14.2 tool is used to verify the functionality of CLFSR and BIST structure [1]. Here, they have implemented 8,16 and 32-bit LFSR on FPGA by using VHDL to study the performance and analysis of the behavior of randomness. The comparative study of 8,16, and 32- bit LFSR on FPGA is done to understand the chip verification [2].

In [3], test power reducing techniques are used using BIST for low power circuits as BIST is an alternate solution for the rising cost of external electrical testing and increasing complexity of the circuit. Here test patterns are generated using LFSR as it is more suitable for BIST architecture. The BIST technique with four LFSR based TPG is incorporated into Universal Asynchronous Receiver Transmitter (UART) design before the overall design is synthesized. Here they have implemented UART with the BIST technique using different LFSR techniques and compared these techniques in SPARTAN3 XC3S200-4FT256FPGA device. LBIST is a DFT technique in which part of the circuit on a chipboard/system is used to test itself logically [4]. Here BIST is used for the multiplier. Conventional TPG is based on normal polynomials, so test patterns may be repetitive, limiting the test coverage, but here they have designed a primitive polynomial based on the Galois field, which generates non-repetitive test vectors so that LBIST will cover a wide range of faults. VHDL implementation of logic BIST is done using Xilinx's 8.2i, and simulation is done on ModelSim 6.3F [4].

In pseudo-random BIST circuits, the test vectors will be generated by Linear Feedback Shift Register. This type of TPG will generate some repeated test patterns, which will increase the test power. [5] presents an approach called Low Power-Bit Complements Test Vector Generation (LP-BCTVG) technique with bipartite (half fixed) and bit insertion techniques. In order to reduce the test power, this technique inserts appropriate intermediate vectors between adjacent test vectors generated by LFSR. By inverting the output bits of LP-BCTVG, we can reduce the bulkiness of the TPG engine by half. This reduces the overall power consumption with better fault coverage. This technique has been tested on several ISCAS'85, ISCAS'89, and ISCAS'99 benchmark circuits [5]. In this paper, the performance of the test achieved with BIST implementation has proven adequate to offset the disincentive of the overhead of hardware created by an additional BIST circuit. This provides a short test time compared to externally applied test and permits the use of less test cost equipment during the production process. This is designed and implemented using Xilinx ISE 14.2 [6].

Here, UART with BIST capability is implemented via VHDL by using FPGA technology. This UART's architecture with BIST will test the UART for its correctness. Blocks of this architecture are coded in VHDL. This is functionally verified by simulating the code in ModelSim from Mentor Graphics. Synthesis is performed using the Xilinx ISE tool and implemented on SPARTAN 3E FPGA [7]. In this paper, the general architecture of the array, column, Wallace tree, and booth multipliers are designed. The verification of multipliers is done by using BIST. BIST will check the four multipliers by using input patterns obtained from TPG. Later they verify whether they are faulty or fault-free by using it. The logic design of these multipliers need not be verified further by any other means. The advantage of this type of verification is it does not require any third-party verification. The simulation is carried out using ModelSim EDA tool 10.0c and synthesis on Xilinx ISE 14.4 design suite [8].

Reference [9] focuses on the implementation of configurable LFSR in VHDL and checks its performance on the basis of logic, speed, and memory requirement in FPGA. The device used for implementing the configurable linear feedback shift register is Xilinx Virtex-4 FPGA. For simulation and synthesis, they have used the Xilinx ISE 9.2i tool. In paper [10], they have explained how the BIST logic controller can be restarted for a combinational circuit logic using VHDL. It permits to suspend the generation of the signature at any desired point. This controller will comprise hold logic and a signature generation element where hold logic will be implemented such as an external signal (HOLD) which can temporarily to suspend signature generation in the signature generation element at a particular time during BIST session [10]following is a breakdown of the internal design details of the hardware simulation

III. METHODOLOGY

A. Basic BIST architecture

BIST's basic structure consists of two functional blocks, Test Pattern Generator and Response Analyser operated by a test controller, as shown in Fig1. Testing time and complexity of the test circuits are two significant considerations

that must be taken into account during the development of the test rig.

Fig1: Generic BIST Architecture

A TPG is to generate the test vectors required for the test process. ORA analyses the CUT responses to these test vectors. Circuit Under test can be a logic circuit or a memory core. RA does its function with the need of a compacting signal circuit and a compactor. The test controller starts the test procedures and controls all test-related functions. CUT receives its input from other modules in normal operation and executes the task for which it was designed. During test mode, TPG applies CUT a series of test patterns, and test responses from the output of CUT are determined by the output response compactor.

B. Proposed BIST architecture

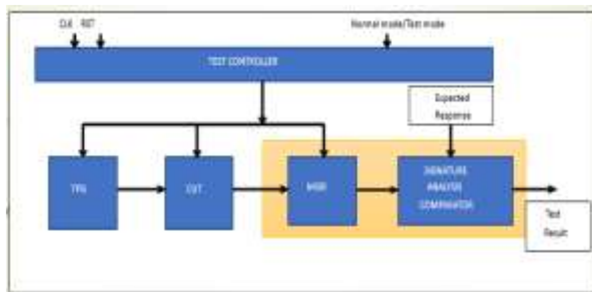

Fig2: Proposed BIST Architecture

We have proposed an optimized LBIST architecture for testing logic circuits, which reduces the overall hardware complexity of the circuit. The architecture shown in Fig2 consists of three major blocks, which are the Test Pattern Generator (TPG), Circuit Under Test (CUT), and Output Response Analyser (ORA). We have used LFSR as TPG as LFSR takes less delay than counters such that its internal circuit is fast. Standard 8-bit LFSR is designed as TPG for generating the pseudo-random test patterns, including an all-zero state. The generated test patterns are sent as input to the circuit under test during the test mode. CUT can be either a logic circuit or a memory core. We have designed a 4*4 multiplier and used it as a circuit under test to verify if there are any hardware faults. This multiplier is developed 4*1 multiplexer, and ripple carries adders. In ORA, we have designed an 8-bit MISR as a signature finder. Initially, MISR takes the output responses and then compresses the responses into a signature, and we have designed a comparator to compare the standard signature value and the output value of MISR. Based on this comparison result, we say a circuit is faulty or fault-free.

a) Test Pattern Generator

TPG generates test vectors and applies it as input to CUT during test mode. Some of the examples of pattern generators are ROM with stored patterns, a counter, and LFSR. The advantages of using LFSR as TPG are that the flip flops can be connected by few XOR gates. The internal circuit is very fast as the maximum delay is due to one XOR and one flip flop delay, takes less area when compared counters, and also provides high frequency. So here in our proposed architecture, we have used LFSR as Test Pattern Generator.

An LFSR is basically a shift register that, when clocked, shifts the signal through the register from one bit to the next most significant bit. Some of the outputs are combined in XOR configuration to form a feedback mechanism. Feedback around an LFSR comes from the selection of points called as taps in the register chain and leads to XORing these taps back into register. It is this feedback that makes the register to loop through repetitive sequences of pseudo-random value. LFSR makes extremely good pseudo-random pattern generators. When the outputs of flip flops of LFSR are loaded with the initial value (seed value), and when LFSR is clocked, it will generate a pseudo-random pattern of 1s and 0s.

There are two major types of LFSR. They are external exclusive-OR LFSR and internal exclusive-OR LFSR. The external XOR LFSR is also known as standard LFSR is a shift register XOR gates representing the tap positions of the feedback polynomials are concatenated to produce a new output bit. The single-bit output is given as feedback input to the last flip-flop in the structure. The internal XOR LFSR, also known as modular LFSR, is another form of LFSR generating the same pattern as that of the standard one. The only difference is that the connection of taps in feedback polynomials to the flip flops. The reason for using standard LFSR in our proposed architecture than any other type LFSR's is that it doesn't take more clock sequence and also in modular LFSR since the combinational circuit is used in between the flip flops, the possibility is more than setup and hold violations may occur. In conventional standard n-bit LFSR, the major disadvantage is that it can only produce 2^n-1 pseudo-random patterns. We have used an 8-bit standard Linear Feedback Shift Register that can generate $2^n = 2^8$ pseudo-random patterns and cycle repeats after a period of 2^n, as shown in Fig 3. To start an LFSR, we must supply the starting values for the registers. These starting values are called the 'seed'. Putting them into registers is called seeding. LFSR can generate different pseudo-random sequences based on the feedback polynomial. Feedback polynomial that is used to generate 8-bit standard LFSR for the period (2^8=256) is

$$x^8 + x^6 + x^5 + x^4 + 1$$

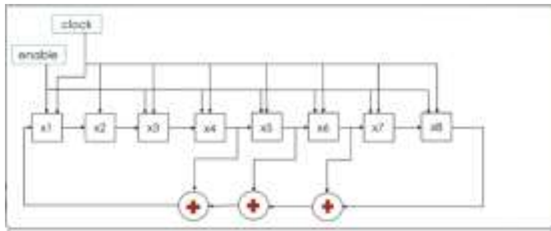

Fig3: 8-Bit StandardLFSR

The truth table of 8-bit standard LFSR for the first few states is as shown in Table I. When enable is 0 and clocked, outputs of the flip-flop are loaded with a seed value which is '11111111' and then when enable is made 1, LFSR will generate pseudo-random patterns of 1's and 0's. for example refer to the second row of a truth table. First the feedback polynomial value is calculated which is ($x^8 + x^6 + x^5 + x^4 + 1$ = 1 + 1 + 1 + 1 + 1 = 1), now these 1 is fed to x1 flip flop and then shifting of the LFSR takes place giving '01111111' state.

Table I: truth table of 8-bit standard LFSR

Clock	enable	Q 8	Q 7	Q 6	Q 5	Q 4	Q 3	Q 1	Q 0
1	0	1	1	1	1	1	1	1	1
2	1	0	1	1	1	1	1	1	1
3	1	0	0	1	1	1	1	1	1
4	1	0	0	0	1	1	1	1	1
5	1	0	0	0	0	1	1	1	1
6	1	1	0	0	0	0	1	1	1

b) Circuit Under Test (CUT)

We can choose any circuit to be circuit under test to check whether that circuit is faulty or fault-free. We have considered a 4*4 multiplier to be our circuit under test (CUT). The pseudo-random patterns generated by LFSR are given as input to the circuit under test (CUT). The output responses from the circuit under test are given as input to MISR(Multiple Input Signature Register).

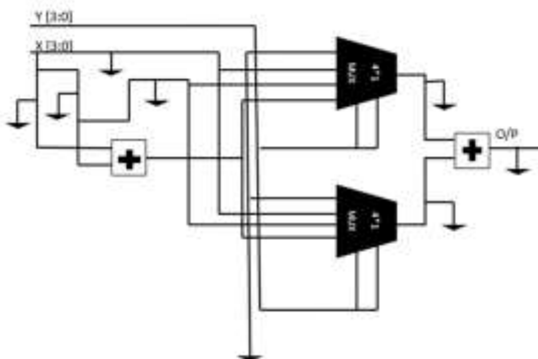

Fig4: Multiplexer based 4*4 Multiplier

Here we have designed multiplexer based 4*4 Multiplier as CUT as shown in Fig 4. The 8-bit test patterns are divided into two 4-bits i.e., lower 4-bit is considered as nibble 1 (X) input, and a higher 4-bit is considered as nibble 2 (Y) input for the CUT. This is done because we want both nibble 1 and nibble 2 inputs to be independent of each other. To design the 4*4 multiplier, we used 4*1 MUX, and ripple carry adders. The ripple carries adders are designed with full adders, and further full adders are designed with half adders. In Fig 4, The inputs of the Y group (4-bits) are partitioned into two sets of 2-bit pairs and separately feeds to two MUX as select lines. X values are the multiplexer inputs. Here it is realized that the multiplication number is created by a summation, the inputs X foe all the logic bits of the second inputs Y, taking into consideration the position of these ones in the select values Y. A 4*1 MUX will permit one of the four cases (00, 01, 10, and 11) to pass through it. When select lines S1S0 is (00), all bits are zero at the MUX output. When S1S0 is (01), the select lines allow the X digital values to pass through the MUX. Next, When S1S0 is (10), the duplicated version of the X values. By adding zero to the LSB of the X number, duplicates it. Finally, when the S1S0 is (11), the output will be the sum of X number and its duplicate value.

c) Multiple Input Shift Register (MISR)

The MISR takes the original output response from CUT and will generate a set of sequences that are non-repeated. It is implemented using Linear Feedback Shift Register as its linear in nature. The purpose of this MISR is to reduce the effort of storing standard response as storing responses needs memory, which in turn increases the area, and also it is hard to compress the responses.

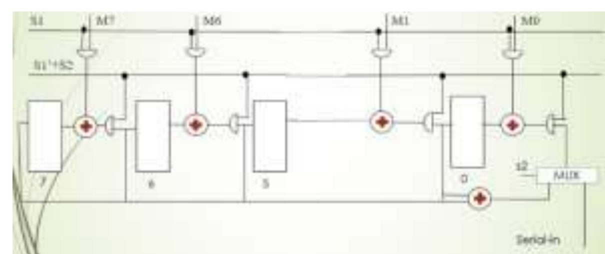

Fig5: Multiple Input Shift Register

Fig5 represents the circuit implementation of an eight-bit multiple input signature register. Here based on the select1 signal, the first MISR takes all the output responses from CUT as input, and secondly, it compresses these responses into a signature. In order to store the standard signature into ROM with which we need to compare the output responses, is achieved by applying the output responses of a properly working CUT to MISR, and creating a signature, then this is stored in ROM as a standard signature by using one-time generation

code. The main work of the response analyzer is to compare the actual responses of the circuit under test with a standard signature. The response analyzer takes the non-repeated sequences from MISR and gives it to a comparator which compares the output signature of MISR and standard signature and reveals that CUT is fault free and vice versa.

d) Comparator

The comparator performs data comparison where inputs are in the form of binary numbers. It finds whether a given number is equal, greater, or less than the other one in terms of magnitude. In our architecture, the comparator makes the comparison between the standard signature stored in ROM and the resultant signature obtained from MISR and checks whether they are equal. The equality between the values indicates a fault-free circuit. If the values are not the same, it confirms that the circuit is faulty.

Fig6: Black box diagram of a normal comparator

Comparator, as shown in fig6, acts as a signature analyzer where it compares the signature produced by MISR with the reference signature stored in ROM. If the signature output from MISR equal to the standard signature, then the output of the comparator is equal to one. It implies that the CUT is fault-free. If not, there is a fault present in CUT. The signature analyzer generates two distinct signature values for fault-free CUT and faulty CUT.

Fig7: Zybo Z7-10 Zynq 7000 board

IV. RESULTS AND SIMULATIONS

A. *RTL Schematic*

We are going to see the RTL schematic of each block of our proposed architecture like Test Pattern Generator (TPG), Circuit Under Test (CUT), Multiple Input Signature Register (MISR), and comparator by using Xilinx ISE 14.5 tool and then we are going to see the RTL schematic of the overall BIST architecture obtained by using Xilinx ISE 14.5 design suite and VIVADO 2018.3 student version.

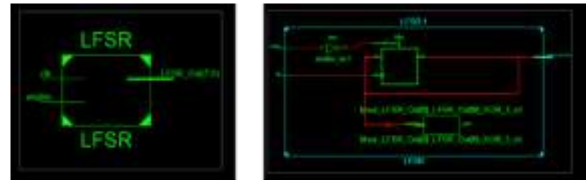

Fig8: RTL Schematic of a LFSR

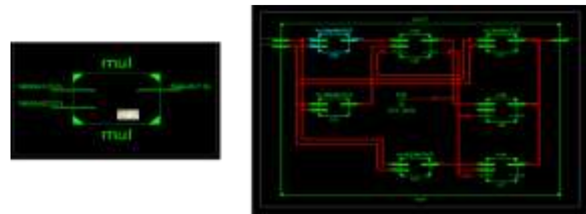

Fig9: RTL Schematic of 4*4 Multiplier

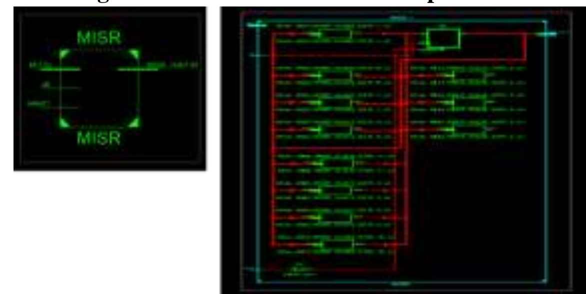

Fig10: RTL Schematic of MISR

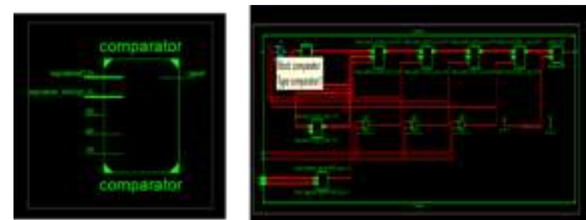

Fig 11: RTL Schematic of comparator

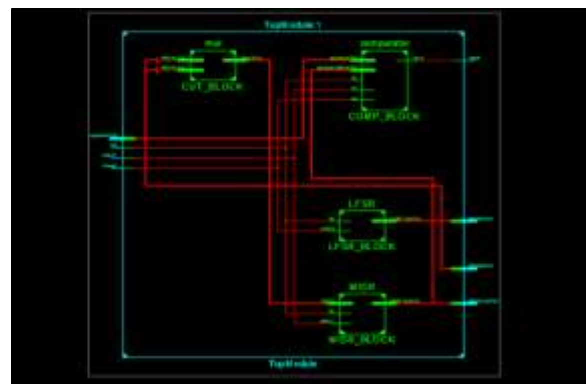

Fig12: RTL Schematic of BIST Architecture using Xilinx ISE 14.5

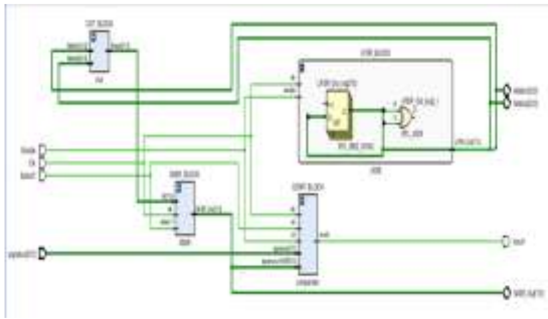

Fig13: RTL Schematic of BIST Architecture using VIVADO 2018.3

B. Simulation Results

Outputs obtained for both faulty and fault-free conditions applied to optimized BIST architecture is reported and analyzed, also the hardware utilization of the architecture and the architecture is shown here. Finally, our proposed LFSR is compared with an 8-bit LFSR designed in one of the reference papers, which is as shown in Table II.

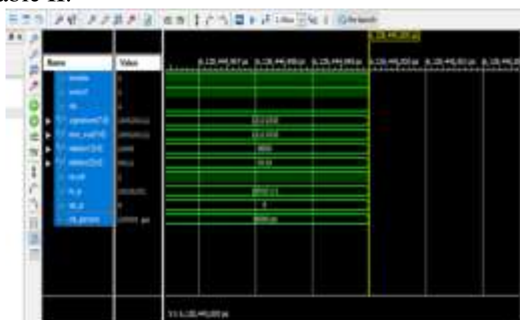

Fig 14: Using Xilinx ISE 14.5, when the signature of the circuit under test matches the standard signature, then the circuit under test is fault-free.

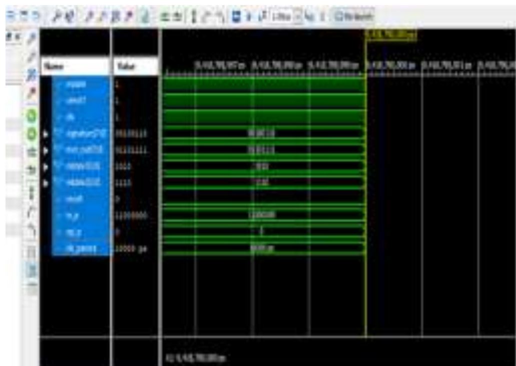

Fig15: Using Xilinx ISE 14.5.when the signature of the circuit under test does not match the standard signature, then the circuit under test is faulty

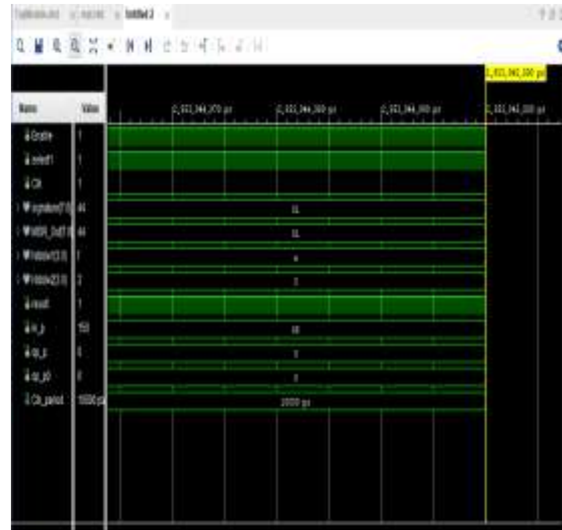

Fig16: Using VIVADO 2018.3, when the signature of the circuit under test matches the standard signature, then the circuit under test is fault-free.

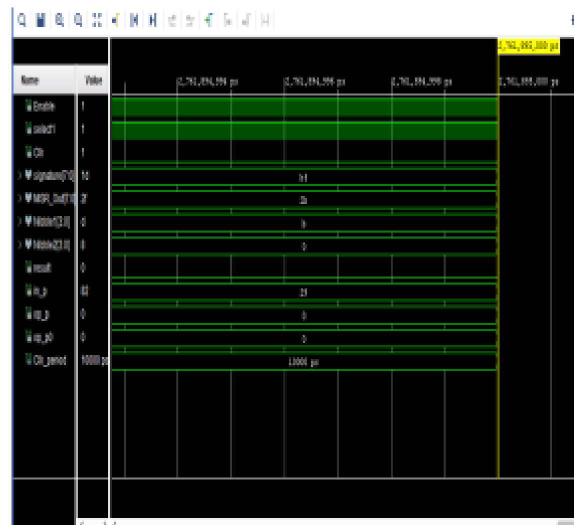

Fig 17: Using VIVADO 2018.3. when the signature of the circuit under test does not match the standard signature, then the circuit under test is faulty

TopModule Project Status [illegible]			
Project File:	TopModule.xise	Parser Errors:	No Errors
Module Name:	TopModule	Implementation State:	Synthesized
Target Device:	xc6slx45-3fgg484	•Errors:	
Product Version:	ISE 14.5	•Warnings:	
Design Goal:	Balanced	•Routing Results:	
Design Strategy:	Xilinx Default (unlocked)	•Timing Constraints:	
Environment:	System Settings	•Final Timing Score:	

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slice LUTs	24	27288	0%
Number of fully used LUT-FF pairs	0	24	0%
Number of bonded IOBs	16	316	5%

Summary Report of Proposed BIST architecture by simulating in Xilinx ISE 14.5.

Summary Report Proposed BIST architecture by simulating in VIVADO 2018.3.

The hardware utilization of an 8-bit standard Linear Feedback Shift Register simulated using Xilinx ISE 14.5 is compared with a reference paper, which is as shown in Table II.

Table II: Comparison Table

PARAMETERS	Proposed method Device: Spartan-6(XC6SLX45)	Amit Kumar Panda, Praveena Rajput Device: Spartan-3S1000
Number of slice Registers	8	8
Number of FF	0	8
Number of Slice LUTS	2	1

V. CONCLUSION

BIST architecture is implemented using minimum hardware requirements. The optimization is achieved by optimizing the LFSR and MISR by introducing clock gating. The architecture checks whether the CUT is faulty or fault-free. The hardware utilization is compared with one of the reference papers to show that our architecture is optimized. Each and every block of the architecture is designed and simulated using Xilinx ISE 14.5 design suite and VIVADO 2018.3 and implemented on Zybo Z7-10 Zynq 7000 board.

REFERENCES

[1] Preethy K John, Rony Antony P, "*Optimized BIST Architecture for Memory Cores and Logic Circuits using CLFSR*", IEEE,2017

[2] Amit Kumar Panda, Praveena Rajput, Bhawna Shukla," *FPGA Implementation of 8,16,32 bit LFSR with Maximum Length Feedback Polynomial using VHDL*". 2012.

[3] P. Ramesh, Dr. D.N Rao, Dr. K. Srinivasa Rao, "*Power Reduction Testing Techniques of BIST, LFSR & ATPG for Low Power Circuits*",2017.

[4] Pushpraj Singh Tanwar, Priyanka Srivatsava, "*VHDL Implementation of Logic BIST Architecture for Multiplier Circuit for Test Coverage in VLSI Chips*", 2014.

[5] J. Praveen, M.N. Shanmukha Swamy, "*BIST-Based Low Power Test Vector Generator and Minimising bulkiness of VLSI Architecture*", 2018.

[6] Ben John, Christy Mathew Philip, Agi Joseph, "*Design and Implementation of (Built in Self Test) BIST for VLSI Circuits using Verilog*", 2015.

[7] L. Supriya, J. Lingaiah, G.Kalyan "*FPGA Implementation of BIST (Built in Self Test) Enabled UART for Real Time Interface Applications*", 2015.

[8] Anju Rajput, "*Designing of BIST Architecture of Generic Multipliers*", 2014.

[9] Shivshankar Mishra, Ram Racksha Tripathi, Devendra Kr. Tripathi*," Implementation of Configurable Linear Feedback Shift Register in VHDL*", 2016.

[10] Jamuna S, Dr. V K Agarwal, "*Implementation of BIST Structure using VHDL for VLSI Circuits*", International Journal of Engineering and Technology, Issue no. 6, pp. 5041-5048, 2011.

[11] Mehboob Hasan Ahmed, Rutuja Jagtap, Roopal Pantode, and Prof. S. S. Phule, "*An FPGA Chip Identification Generator using Configurable Ring Oscillator*" SSRG International Journal of Electronics and Communication Engineering 3.4 (2016): 10-14.