IDENTIFICATION OF STUCK-AT-FAULTS IN A MODULE USING TESTING DEVICES

A Major Project Report submitted to
JNTU Hyderabad in partial fulfillment
of the requirements for the award of the degree

BACHELOR OF TECHNOLOGY

In

ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted by

TEJASRI GURRALA	**21S11A0498**
SAIPRASAD REDDY AKKENAPALLY	**21S11A0487**
SAKETHBABU VARAGANI	**21S11A0491**
SAI KUMAR REDDY MANDAPATI	**21S11A0485**

Under the Guidance of

Ms. G. HARITHA
B.Tech., M.Tech.
Assistant Professor of ECE

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
MALLA REDDY INSTITUTE OF TECHNOLOGY & SCIENCE
(Approved by AICTE New Delhi and Affiliated to JNTUH)
(Accredited by NBA& NAAC with "A" Grade)
An ISO 9001: 2015 Certified Institution
Maisammaguda, Medchal (M), Hyderabad-500100, T. S.

JULY 2025

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

MALLA REDDY INSTITUTE OF TECHNOLOGY & SCIENCE

(Approved by AICTE New Delhi and Affiliated to JNTUH)

(Accredited by NBA & NAAC with "A" Grade)

An ISO 9001: 2015 Certified Institution

Maisammaguda, Medchal (M), Hyderabad-500100, T. S.

JULY 2025

CERTIFICATE

This is to certify that the Major project entitled **"IDENTIFICATION OF STUCK-AT-FAULTS IN A MODULE USING TESTING DEVICES"** has been submitted by **TEJASRI GURRALA (21S11A0498), SAIPRASAD REDDY AKKENAPALLY (21S11A0487), SAKETH BABU VARAGANI (21S11A0491) and SAI KUMAR REDDY MANDAPATI (21S11A0485)** in partial fulfillment of the requirements for the award of **BACHELOR OF TECHNOLOGY** in **ELECTRONICS AND COMMUNICATION ENGINEERING**. This record of bonafide work carried out by them under my guidance and supervision. **The result embodied in this Major project report has not been submitted to any other University or Institute for the award of any degree.**

Ms. G. Haritha
Assistant professor

Mrs. G. Subhashini
Head of the Department

External Examiner

ACKNOWLEDGEMENT

The Major Project work carried out by our team in the Department of Electronics and Communication Engineering, Malla Reddy Institute of Technology and Science, Hyderabad. ***This work is original and has not been submitted in part or full for any degree or diploma of any other university.***

We wish to acknowledge our sincere thanks to our project guide **Ms. G. Haritha**, Assistant professor of Electronics and Communication Engineering for formulation of the problem, analysis, guidance and her continuous supervision during the course of work.

We acknowledge our sincere thanks to **Dr. Vaka Murali Mohan,** Principal and **Mrs. G. Subhashini**, Head of the Department and Coordinator, faculty members of ECE Department for their kind cooperation in making this Case study work a success.

We extend our gratitude to **Sri. Ch. Malla Reddy**, Founder Chairman MRGI and **Sri. Ch. Mahender Reddy,** Secretary MRGI, **Dr.Ch. Bhadra Reddy**, President MRGI, **Sri. Ch. Shalini Reddy**, Director MRGI, **Sri. P. Praveen Reddy**, Director MRGI, for their kind cooperation in providing the infrastructure for completion of our Major Project.

We acknowledge our special thanks to the entire teaching faculty and non-teaching staff members of the Electronics and Communication Engineering Department for their support in making this project work a success.

TEJASRI GURRALA **21S11A0498**_____

SAIPRASAD REDDY AKKENAPALLY **21S11A0487**_____

SAKETH BABU VARAGANI **21S11A0491**_____

SAI KUMAR REDDY MANDAPATI **21S11A0485**_____

INDEX

ABSTRACT

Advanced CMOS devices are a critical issue due to the shrinking process of the size of the transistor. The number of transistors packed in an IC increases with the latest technology. Manufacturing defects in the chip happen due to the interconnection of wires. This leads to unexpected outputs. The process of testing confirms that the chip is fault free. This work discusses the number of test vectors needed to find the faults present in a full adder. An adder circuit is realized in this project. Test vectors are also implemented in LFSR which is fed as inputs to the full adder. Thus generated outputs are tested with the expected outputs to identify the faulty position(s) in the full adder. Spartan3e xc3s100e-5vq100 chip was used to realize the testing.

LIST OF FIGURES

ABBREVIATIONS

LCG : Linear congruential Generator

LFSR : Linear-Feedback Shift Register

NVMs : Non-Volatile Memories

ECC : Error Correction Codes

IOT : Internet of Things

PRBG : Pseudorandom Bit Generator

CLCG : Coupled Linear Congruential Generator

CVLCG : Coupled-Variable input LCG

TAP : Test Access Port

NIST : National Institute of Standard Technology

FPGA : Field-programmable Gate Array

DDSM : Digital Delta-Sigma Modulator

BIST : Built-In Self-Test

LUTs : Lookup Tables

TPG : Test Pattern Generator

SRAM : Static RAM

CLBs : Configurable Logic Blocks

RTL : Register Transfer Level

CHAPTER-1: INTRODUCTION

Now a days verification very important for VLSI designs. Manufacturing defects in the chip happen due to the interconnection of wires. This leads to unexpected outputs. The process of testing confirms that the chip is fault free. before manufacturing we need to test the design, so we need to apply test patterns for circuit under test. Basically LCG and LFSR (linear feedback shift register) are used for test pattern generation, in this project we are selected LCG(Linear congruential generator).

As technology continues to evolve at an unprecedented pace, the demand for high performance and reliable computing systems has never been greater. Central to these systems is main memory, a critical component responsible for storing and managing vast amounts of data. With the advent of next-generation applications and the exponential growth in data volume, ensuring the reliability and sustainability of main memories has become a paramount concern. Next-generation main memories, such as DDR5, HBM, and emerging non-volatile memories (NVMs) like MRAM and ReRAM, promise significant improvements in speed, capacity, and energy efficiency. However, these advancements also introduce new challenges in fault management and error detection. As memory systems become denser and more complex, they are increasingly susceptible to a variety of faults and errors, ranging from transient and soft errors to permanent hardware failures.

1.1 OBJECTIVE OF THE STUDY:

The objective of this study is to Identify Stuck-at-Fault in a module using testing devices. As memory technologies evolve to meet the demands of high-performance computing and data-intensive applications, ensuring data integrity and system reliability becomes paramount. This research aims to identify and analyze key vulnerabilities in emerging memory architectures, proposing sustainable solutions that minimize energy consumption while effectively managing faults and errors. By leveraging advanced algorithms and fault-tolerant designs, the study seeks to enhance the resilience of memory systems, enabling them to withstand and recover from various failure scenarios. Ultimately, the objective is to contribute to the advancement of

memory technologies that are not only reliable but also environmentally sustainable, supporting the future of computing in a resource-conscious manner.

1.2 SCOPE OF THE STUDY:

The scope of this study encompasses the exploration of Stuck-at-Faults. The research will cover a range of fault management strategies, from traditional error correction codes (ECC) to more advanced machine learning-based approaches, evaluating their effectiveness in real-time applications. Additionally, the study will investigate the integration of these techniques into memory controllers and system-level architectures to ensure seamless operation and minimal overhead. By focusing on energy-efficient solutions that balance performance and reliability, this research aims to provide a comprehensive framework that addresses both immediate challenges and long-term sustainability in memory technology, ultimately contributing to the development of robust computing environments.

CHAPTER-2: LITERATURE VIEW

J. Zhou, Z. Cao, X. Dong, and A. V. Vasilakos, "Security and privacy for cloud-based IoT: Challenges,". The Internet of Things (IoT) is becoming the next Internet-related revolution. It allows billions of devices to be connected and communicate with each other to share information that improves the quality of our daily lives. On the other hand, Cloud Computing provides on- demand, convenient and scalable network access which makes it possible to share computing resources, indeed, this, in turn, enables dynamic data integration from various data sources. There are many issues standing in the way of the successful implementation of both Cloud and IoT. The integration of Cloud Computing with the IoT is the most effective way on which to overcome these issues. The vast number of resources available on the Cloud can be extremely beneficial for the IoT, while the Cloud can gain more publicity to improve its limitations with real world objects in a more dynamic and distributed manner. This paper provides an overview of the integration of the Cloud into the IoT by highlighting the integration benefits and implementation challenges. Discussion will also focus on the architecture of the resultant Cloud-based IoT paradigm and its new applications scenarios. Finally, open issues and future research directions are also suggested.

E. Zenner, "Cryptanalysis of LFSR-based pseudorandom generators-A survey," Pseudorandom bit generator (PRBG) is an essential component for securing data during transmission and storage in various cryptography applications. Among popular existing PRBG methods such as linear feedback shift register (LFSR), linear congruential generator (LCG), coupled LCG (CLCG), and dual-coupled LCG (dual-CLCG), the latter proves to be more secure. This method relies on the inequality comparisons that lead to generating pseudorandom bit at a non- uniform time interval. Hence, a new architecture of the existing dual CLCG method is developed that generates pseudo-random bit at uniform clock rate. However, this architecture experiences several drawbacks such as excessive memory usage and high-initial clock latency, and fails to achieve the maximum length sequence. Therefore, a new PRBG method called as "modified dual- CLCG" and its very

large-scale integration (VLSI) architecture are proposed in this project to mitigate the aforesaid problems. The novel contribution of the proposed PRBG method is to generate pseudorandom bit at uniform clock rate with one initial clock delay and minimum hardware complexity. Moreover, the proposed PRBG method passes all the 15 benchmark tests of NIST standard and achieves the maximal period of 2 n. The proposed architecture is implemented using Verilog-HDL and prototyped on the commercially available FPGA device.

J. Stern, "Secret linear congruential generators are not cryptographically secure," The dual- coupled linear congruential generator (LCG) (dual-CLCG) is a secure pseudorandom bit generator (PRBG) method among various linear feedback shift register (LFSR), LCG, and chaotic- based 3 PRBG methods for generating a pseudorandom bit sequence. The hardware implementation of this method has a bottleneck due to the involvement of inequality equations. Initially, a direct architectural mapping of the dual-CLCG method is performed. Since two inequality equations are involved for coupling, it generates pseudorandom bit at unequal interval of time that leads to large variation in output latency. In addition, it consumes a large area and fails to achieve the maximal period. Hence, to overcome the aforesaid drawbacks, a new efficient PRBG method, i.e., "coupled- variable input LCG (CVLCG)," and its architecture are proposed. The novelty of the proposed method is the coupling of two newly formed variable input LCGs that generates pseudorandom bit at every uniform clock rate, attains maximum length sequence, and reduces one comparator area as compared to the dual-CLCG architecture. The proposed architecture is implemented using Verilog-HDL and prototyped on the commercially available field-programmable gate array (FPGA) device. Furthermore, the sequences are captured through the logic analyzer and evaluated for randomness using the National Institute of Standard and Technology (NIST) standard test tool. The experimental result reports that the proposed PRBG method passes all the randomness tests with a high degree of consistency. W. B. Jones and D. C. Huang and S. C. Wu and K. J. Lee, An efficient BIST method for small buffers IEEE Standard 1687-2014 - IEEE Standard for Access and Control of Instrumentation Embedded within a Semiconductor Device, A methodology for accessing instrumentation

embedded within a semiconductor device, without defining the instruments or their features themselves, via the IEEE 1149.1(TM) test access port (TAP) and/or other signals, is described in this standard. The elements of the methodology include a hardware architecture for the on- chip network connecting the instruments to the chip pins, a hardware description language to describe this network, and a software language and protocol for communicating with the instruments via this network. This standard develops a methodology for access to embedded instrumentation, without defining the instruments or their features themselves, via the IEEE 1149.1(TM) test access port (TAP) and additional signals that may be required. The elements of the methodology include a description language for the characteristics of the features and for communication with the features, and requirements for interfacing to the features.

D. Bronzi, Y. Zou, F. Villa, S. Tisa, A. Tosi, and F. Zappa, "Automotive three-dimensional vision through a single-photon counting SPAD camera," Linear-feedback shift register (LFSR) counters have been shown to be well suited to applications requiring large arrays of counters. However, significant logic is required to decode the count order into binary, causing system- on-chip designs to be unfeasible. This paper presents a counter design based on multiple LFSR 4 stages that retains the and can improve the area and performance compared with conventional binary counters. advantages of a single-stage LFSR but only requires decoding logic that scales logarithmically with the number of stages rather than exponentially with the number of bits as required by other methods. A four-stage four-bit LFSR proof of concept was fabricated in 130-nm CMOS and was characterized in a time-to-digital converter application at 800 MHz I.

Vornicu, R. Carmona-Galán, and A. Rodríguez-Vázquez, "A CMOS 0.18 μm 64×64 single photon image sensor with in-pixel 11 b time to- digital converter, "Linear-feedback shift register (LFSR) counters have been shown to be well suited to applications requiring large arrays of counters and can improve the area and performance compared with conventional binary counters. However, significant logic is required to decode the count order into binary, causing system-on- chip designs to be unfeasible. This paper presents

a counter design based on multiple LFSR stages that retains the advantages of a single-stage LFSR but only requires decoding logic that scales logarithmically with the number of stages rather than exponentially with the number of bits as required by other methods.

H. Mo and M. P. Kennedy, "Masked dithering of MASH digital delta sigma modulators with constant inputs using multiple linear feedback shift registers," This paper shows that applying a linear feedback shift register (LFSR) dither to a digital delta-sigma modulator (DDSM) cannot always increase its fundamental period. For some DDSMs, the LFSR dither may reduce its period in some cases, instead of increasing it, which worsens the output spectrum. Hence, the paper calculates the dithered DDSM's period and analyzes the influence of LFSR dither on the period. Furthermore, for such kind of DDSM, the paper explains how to add the LFSR dither to increase the period for a full input range. Finally, experiment is performed to confirm the analysis. D. Xiang, M. Chen, and H. Fujiwara, "Using weighted scan enable signals to improve test Effectiveness of scan-based BIST," FPGAs have become popular in present because of its features like high logic capacity, configurability and regular architecture with low area cost. However, FPGAs are prone to faults, testing is one of the important process for designers. Hence, the best method for testing faults in FPGAs is Built-In Self-Test (BIST). BIST is a design technique that allows a system to test itself. The proposed method of a BIST design for fault detection and fault diagnosis of Static-RAM (SRAM) based FPGAs can test Lookup Tables (LUTs) in the Configurable Logic Blocks (CLBs).There are three major blocks in the system to detect and diagnose the faults. They are Test Pattern Generator (TPG), Output Response Analyzer (ORA) and Block Under Test (BUT).

CHAPTER-3: EXISTING SYSTEM

3.1 LINEAR CONGRUENTIAL GENERATOR:

In fact, any higher level programming language offers at least one form of random number generator. The generation of random numbers, however, is not an easy task for a computer, since the computer is a deterministic machine with no built-in randomness. Thus it is impossible to create true random numbers without any additional hardware. What can be done, is to create **pseudo random numbers** which behave almost like random numbers but which are repeated after a fixed (mostly quite long) period.

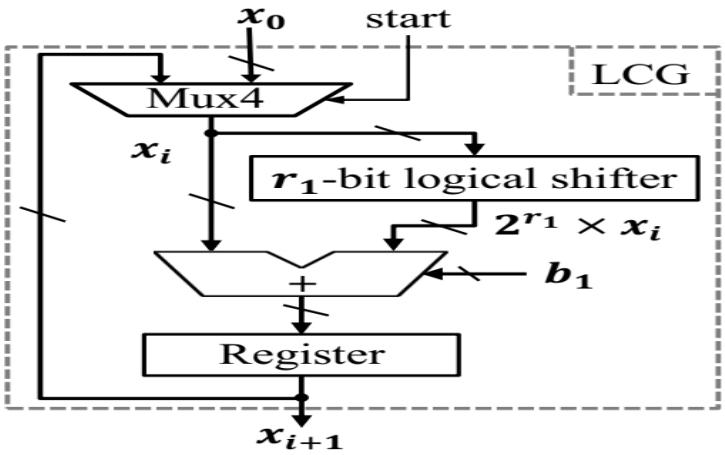

Fig:3.1 Architecture of the linear congruential generator

These pseudo random numbers are generated by **linear congruential generators** (**LCG**). The principle of an LCG is quite simple: a new pseudo random number is generated on the basis of the previous random number by adding a certain offset and wrapping the result if it exceeds a certain limit. The process can be denoted by the following equation: The linear congruential method produces a sequence of integers $X_0, X_1, X_2 \ldots$ between zero and $m-1$ according to the following recursive relationship:

$x_{i+1} = a_1 \times x_i + b_1 \bmod 2^n$ (1)

Here, a1, b1 are the constant parameters; x_i is the initial seed.

Following are the necessary conditions to get the maximum period.

b_1 is relatively prime with 2^n(m).

a_1-1) must be divisible by 4.

LCG method is developed to generate pseudorandom bit at an equal interval of time for encrypting continuous data stream in the stream cipher.

The architecture is designed with two comparators, four LCG blocks, one controller unit and memory (flip-flops) as shown in Fig:4.1. The LCG is the basic functional block in the dual-CLCG architecture that involves multiplication and addition processes to compute n-bit binary random number on every clock cycle. The multiplication in the LCG equation can be implemented with shift operation, when a is considered as (2^r+1). Here, r is a positive integer, $1 < r < 2^n$. Therefore, for the efficient computation of xi+1, the equation (1) can be rewritten as,

$xi+1 = (a1 \times xi + b1) \bmod 2^n = [(2^{r1} + 1)xi + b1] \bmod 2^n$

$=[(2^{r1} \times xi)+xi+b1] \bmod 2^n$

The architecture of Linear congruential generator (LCG) shown in Fig:3.1 is implemented with a 3-operand modulo 2n adder, 2×1 n-bit multiplexer and n-bit register.

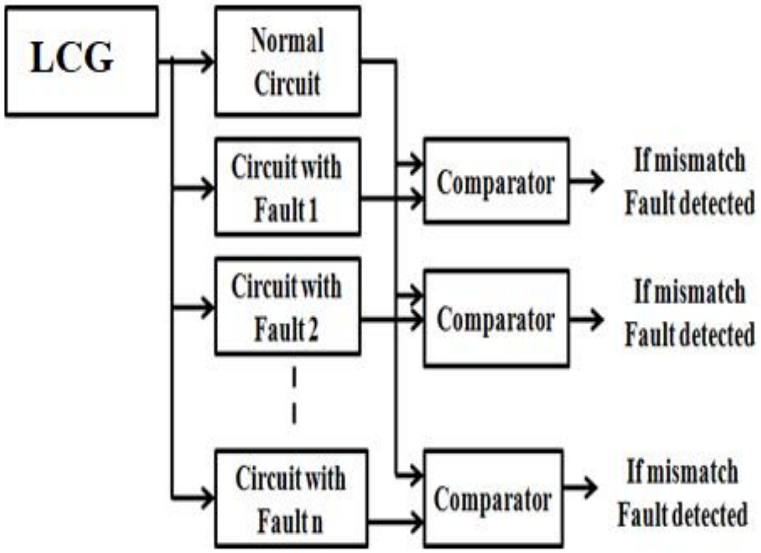

Fig:3.2 BIST Architecture of Existing System

LCG generates a random n-bit binary equivalent to integer number in each clock cycle. Other three LCG equations can also be mapped to the corresponding architecture similar to the LCG equation.

CHAPTER-4: PROPOSED SYSTEM

4.1 SUSTAINABLE FAULT TESTING ALGORITHM:

The evolution of integrated circuits started with the introduction of the microprocessor. Proper functioning of a chip is tested before it is manufactured. Various types of the verification involved in the process of chip testing are IP verification, RTL verification, timing verification, etc., Testing becomes a headache in the process of IC fabrication as it consumes 80% of the manufacturing time. A faulty product is generated due to incomplete specification problem, incorrect design, faulty fabrication process, and wrong test method.

Testing is normally done using a computer with the device under test (DUT). A suitable data transfer medium transfers the test vectors to the DUT and receives the outputs generated by it. Now, the algorithm analyzes the outputs and identifies the presence of the fault(s). FPGA is known for its unique advantage called concurrent execution. This motivates the researchers to develop applications using FPGA. It is flexible and re-programmable. Latest FPGAs have in-built features like DDR, DSP, MAC etc.,

The two types of stuck-at-faults are Stuck-at-1 (SA1) and Stuck-at-0 (SA0). Stuck-at fault identifies the faults that are available in an integrated circuit. It can occur either at the input to the gate or at the output of the gate. When the value of the line or cell is always '0', it is identified as a stuck-at-0 fault (SA0). When the value of the line or cell is always '1', it is identified as a stuck-at-1 fault (SA1).

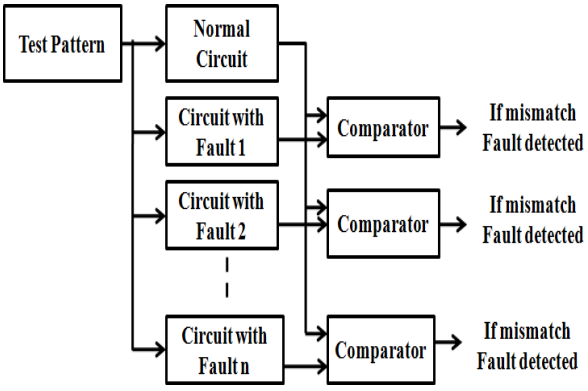

Fig:4.1 Basic Principle of Fault Simulation

Different kinds of algorithms used for fault simulation are serial fault simulation, parallel fault simulation, concurrent fault simulation, and deductive fault simulation. This work concentrates on parallel fault simulation. Parallel fault simulation is the most effective technique when a circuit is designed using logic gates and modeled with stuck-at- faults. The testing flow diagram shown in Fig:4.2. The flow will continue till find a required test vectors.

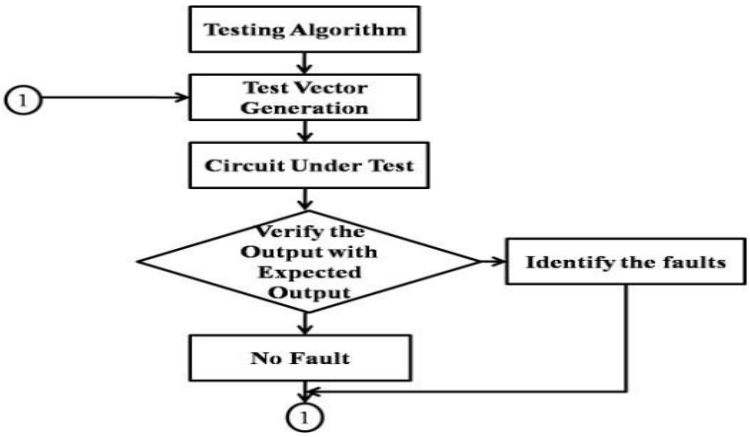

Fig:4.2 Testing Flow Diagram

This work uses the application and testing algorithm in the same chip. It can be re- configured at any point of time with the help of hardware description languages (HDL). The bit file generated after synthesis is implemented into the Xilinx. Xilinx realizes the HDL coding in the form of logic gates. Path sensitizing method of testing has been a powerful approach for test generation of any combinational circuit. Path Sensitization is based on the assumption that the failure mechanism in a gate results in its inputs or outputs being stuck-at-1 or stuck-at-0. Realization of path sensitizing in a full adder circuit is shown in Fig:4.3.

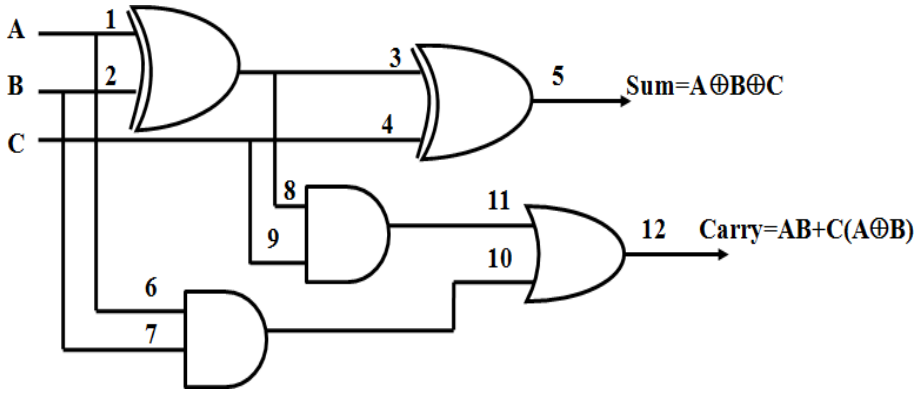

Fig:4.3 Single bit Full adder

The Boolean difference method is a well known mathematical concept which has found significant application in the single fault analysis of combinational logic circuits. Consider the output 'sum' which has an input of a, b, and c. Now, let line 1 has a stuck-at-0 fault. Boolean difference method helps to identify this fault. The output of the full adder circuit is Sum= A>B>C. Let the target fault be a, s-a-0 or s-a-1.

A fault present in the circuit is represented as stuck-at-0 (SA0) and stuck-at-1(SA1). Most of the faults found in VLSI design/ circuits are stuck at fault because the electrical failures occur due to the physical mechanism of the circuit. So, concern stuck at fault which occurs due to broken wires, short to ground or to VCC, shorted transistors and diodes. The use of all possible inputs for testing is known as exhaustive testing and it is straight forward, but impractical. Fault table displays the output of the circuit for a set of faults and fault-free test inputs. The Boolean difference method is applied and tested for a full adder circuit. All the possible faults of full adder were determined and executed using Xilinx 14.7 ISE design Suite. Each line of full adder was analyzed for either stuck- at-1 or stuck-at-0 and the required test vectors to find multiple faults were identified as test vectors. The complete coding was done using Verilog HDL code.

4.2 LINEAR-FEEDBACK SHIFT REGISTER:

In computing, a LFSR is a shift register whose input bit is a linear function of its previous state. The most commonly used linear function of single bits is exclusive-or (XOR). Thus, an LFSR is most often a shift register whose input bit is driven by the XOR of some bits of the overall shift register value. The initial value of the LFSR is called the seed, and because the operation of the register is deterministic, the stream of values produced by the register is completely determined by its current (or previous) state. Likewise, because the register has a finite number of possible states, it must eventually enter a repeating cycle. However, an LFSR with a well-chosen feedback function can produce a sequence of bits that appears random and has a very long cycle. Applications of LFSRs include generating pseudo-random numbers, pseudo-noise sequences, fast digital counters, and whitening sequences. Both hardware and software implementations of

LFSRs are common. The mathematics of a cyclic redundancy check, used to provide a quick check against transmission errors, are closely related to those of an LFSR LFSRs can be implemented in hardware, and this makes them useful in applications that require very fast generation of a pseudo-random sequence, such as direct-sequence spread spectrum radio. LFSRs have also been used for generating an approximation of white noise in various programmable sound generators.

Uses as pattern generator: The repeating sequence of states of an LFSR allows it to be used as a clock divider or as a counter when a non-binary sequence is acceptable, as is often the case where computer index or framing locations need to be machine-readable. LFSR counters have simpler feedback logic than natural binary counters or Gray-code counters, and therefore can operate at higher clock rates. However, it is necessary to ensure that the LFSR never enters an all-zeros state, for example by presetting it at start-up to any other state in the sequence.

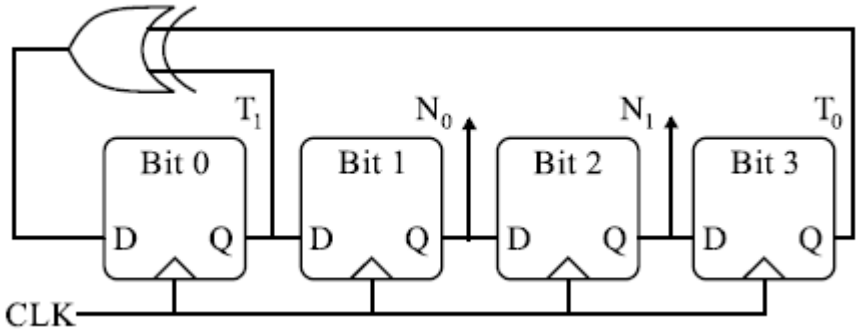

Fig:4.4 Structure of many-to-one 4-LFSRs.

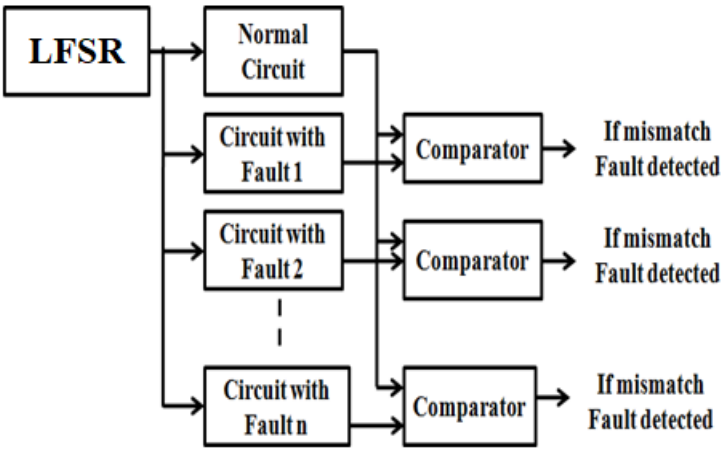

Fig:4.5 BIST Architecture of Proposed System

Uses as counter: The repeating sequence of states of an LFSR allows it to be used as a clock divider or as a counter when a non-binary sequence is acceptable, as is often the case where computer index or framing locations need to be machine-readable.[7] LFSR counters have simpler feedback logic than natural binary counters or Gray-code counters, and therefore can operate at higher clock rates. However, it is necessary to ensure that the LFSR never enters an all-zeros state, for example by presetting it at start-up to any other state in the sequence. The table of primitive polynomials shows how LFSRs can be arranged in Fibonacci or Galois form to give maximal periods. One can obtain any other period by adding to an LFSR that has a longer period some logic that shortens the sequence by skipping some states.

Uses in cryptography: LFSRs have long been used as pseudo-random number generators for use in stream ciphers. However, an LFSR is a linear system, leading to fairly easy cryptanalysis. For example, given a stretch of known plaintext and corresponding cipher text, an attacker can intercept and recover a stretch of LFSR output stream used in the system described, and from that stretch of the output stream can construct an LFSR of minimal size that simulates the intended receiver by using the Berlekamp-Massey algorithm. This LFSR can then be fed the intercepted stretch of output stream to recover the remaining plaintext.

Uses in circuit testing: LFSRs are used in circuit testing for test-pattern generation (for exhaustive testing, pseudo-random testing or pseudo-exhaustive testing) and for signature analysis.

Test-pattern generation: Complete LFSR are commonly used as pattern generators for exhaustive testing, since they cover all possible inputs for an n-input circuit. Maximal-length LFSRs and weighted LFSRs are widely used as pseudo-random test-pattern generators for pseudo-random test applications.

Signature analysis: In built-in self-test(BIST) techniques, storing all the circuit outputs on chip is not possible, but the circuit output can be compressed to form a signature that will later be compared to the golden signature (of the good circuit) to detect faults. Since this compression is lossy, there is always a possibility that a faulty output also generates the same signature as the golden signature and the faults cannot be detected. This

condition is called error masking or aliasing.

BIST is accomplished with a multiple-input signature register (MISR or MSR), which is a type of LFSR. A standard LFSR has a single XOR or XNOR gate, where the input of the gate is connected to several "taps" and the output is connected to the input of the first flip-flop. A MISR has the same structure, but the input to every flip-flop is fed through an XOR/XNOR gate. For example, a 4-bit MISR has a 4-bit parallel output and a 4-bit parallel input. The input of the first flip-flop is XOR/XNOR with parallel input bit zero and the "taps". Every other flip-flop input is XOR/XNOR with the preceding flip-flop output and the corresponding parallel input bit. Consequently, the next state of the MISR depends on the last several states opposed to just the current state. Therefore, a MISR will always generate the same golden signature given that the input sequence is the same every time.

Uses in digital broadcasting and communications: To prevent short repeating sequences (e.g., runs of 0s or 1s) from forming spectral lines that may complicate symbol tracking at the receiver or interfere with other transmissions, the data bit sequence is combined with the output of a linear-feedback register before modulation and transmission. This scrambling is removed at the receiver after demodulation. When the LFSR runs at the same bit rate as the transmitted symbol stream, this technique is referred to as scrambling. When the LFSR runs considerably faster than the symbol stream, the LFSR-generated bit sequence is called chipping code. The chipping code is combined with the data using exclusive or before transmitting using binary phase-shift keying or a similar modulation method. The resulting signal has a higher bandwidth than the data, and therefore this is a method of spread-spectrum communication. When used only for the spread-spectrum property, this technique is called direct-sequence spread spectrum; when used to distinguish several signals transmitted in the same channel at the same time and frequency, it is called code division multiple access. Neither scheme should be confused with encryption or encipherment; scrambling and spreading with LFSRs do not protect the information from eavesdropping. They are instead used to produce equivalent streams that possess convenient engineering properties to allow robust and efficient

modulation and demodulation.

Digital broadcasting systems that use linear-feedback registers:

ATSC Standards (digital TV transmission system – North America)

DAB(Digital Audio Broadcasting system – for radio)

DVB-T (digital TV transmission system – Europe, Australia, parts of Asia)

NICAM (digital audio system for television)

Other digital communications systems using LFSRs:

INTELSAT business service (IBS)

Intermediate data rate (IDR)

SDI (Serial Digital Interface transmission)

Data transfer over PSTN (according to the ITU-T V-series recommendations)

CDMA (Code Division Multiple Access) cellular telephony

100BASE-T2 "fast" Ethernet scrambles bits using an LFSR

1000BASE-T Ethernet, the most common form of Gigabit Ethernet, scrambles bits using an LFSR

PCI Express 3.0

SATA

Serial attached SCSI (SAS/SPL)

USB 3.0

IEEE 802.11a scrambles bits using an LFSR

Bluetooth Low Energy Link Layer is making use of LFSR (referred to as whitening)

Satellite navigation systems such as GPS and GLONASS. All current systems use LFSR outputs to generate some or all of their ranging codes (as the chipping code for CDMA or DSSS) or to modulate the carrier without data (like GPS L2 CL ranging code). GLONASS also uses frequency-division multiple access combined with DSSS.

CHAPTER-5: SOFTWARE USED

5.1 XILINX VIVADO:

Xilinx software is used by the VHDL/VERILOG designers for performing Synthesis operation. Any simulated code can be synthesized and configured on FPGA. Synthesis is the transformation of VHDL code into gate level net list.

5.2 ALGORITHM FOR USING VIVADO:

Start the ISE Software by clicking the XILINX ISE icon.

Create a New Project and find the following properties displayed.

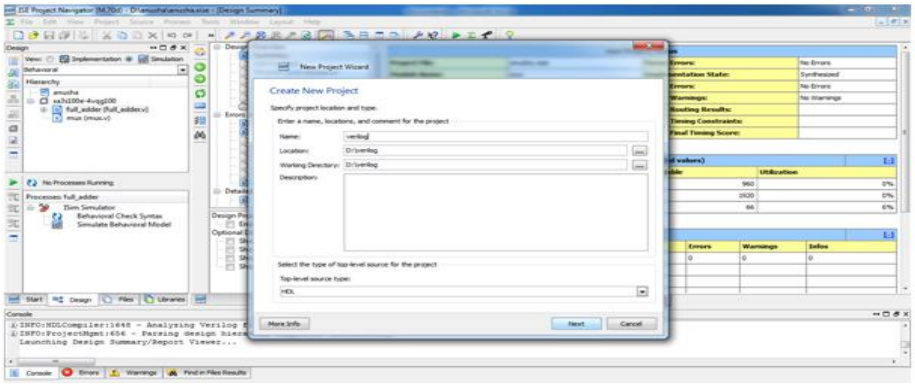

Fig :5.1 Create new folder for design

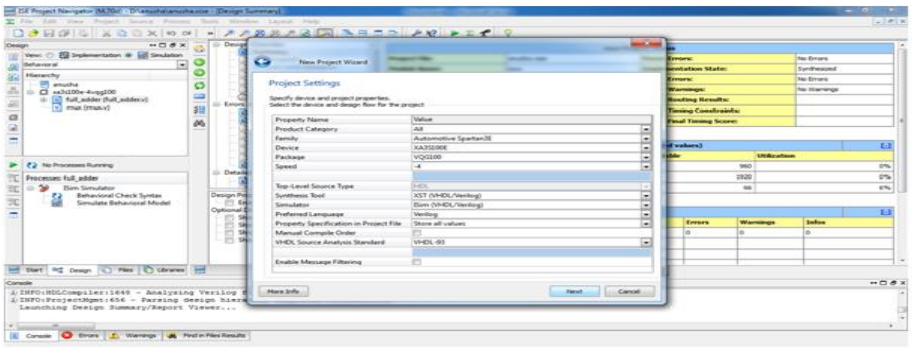

Fig:5.2 Set family and device before design a project

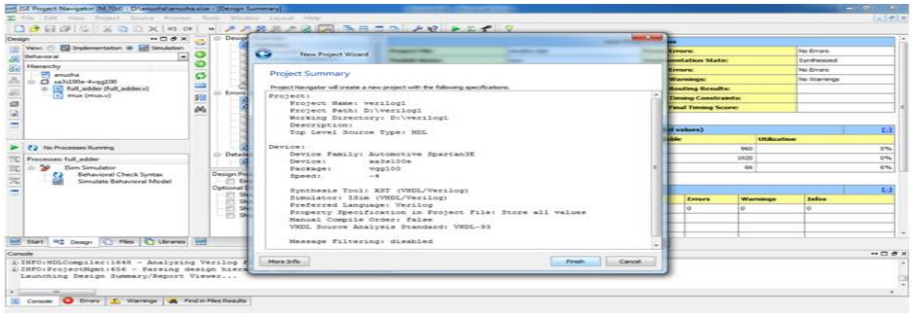

Fig:5.3 Finishing new folder, Set family & device

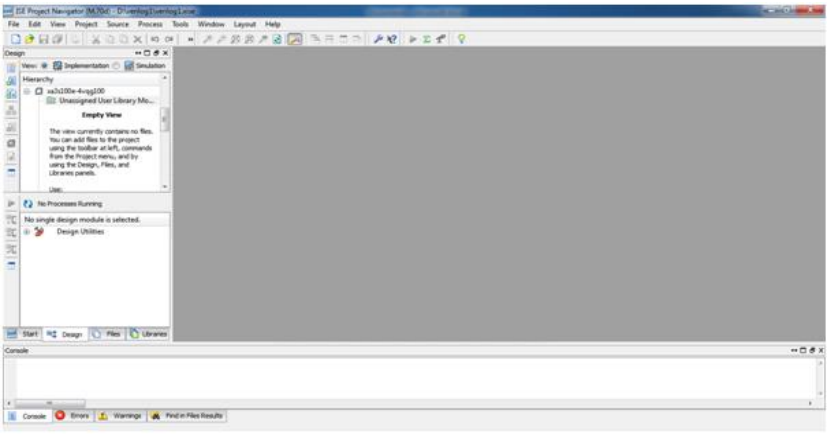

Fig:5.4 Ready to design a project

Create a HDL Source formatting all inputs, outputs and buffers if required. which provides a window to write the HDL code, to be synthesized.

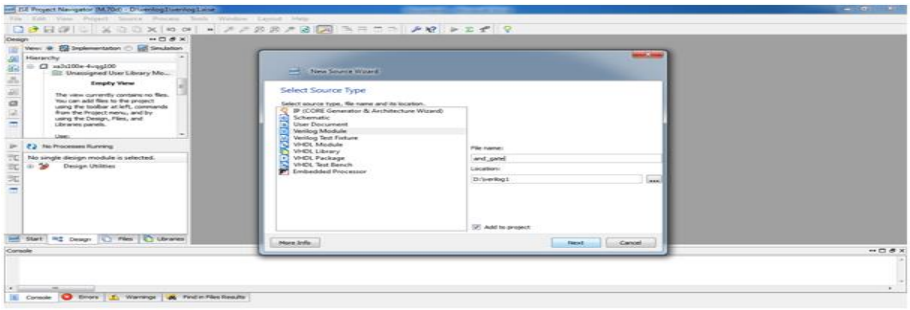

Fig:5.5 create module name (.v files names) for design

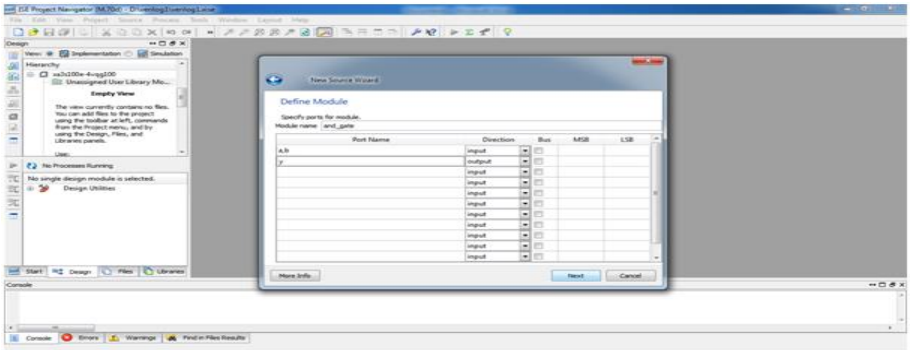

Fig:5.6 Declaration of input and output ports with bit lengths

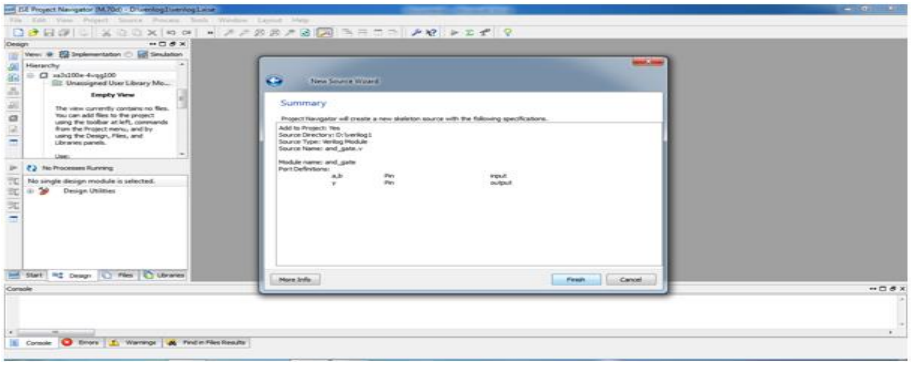

Fig:5.7 The schematic was created by its ports

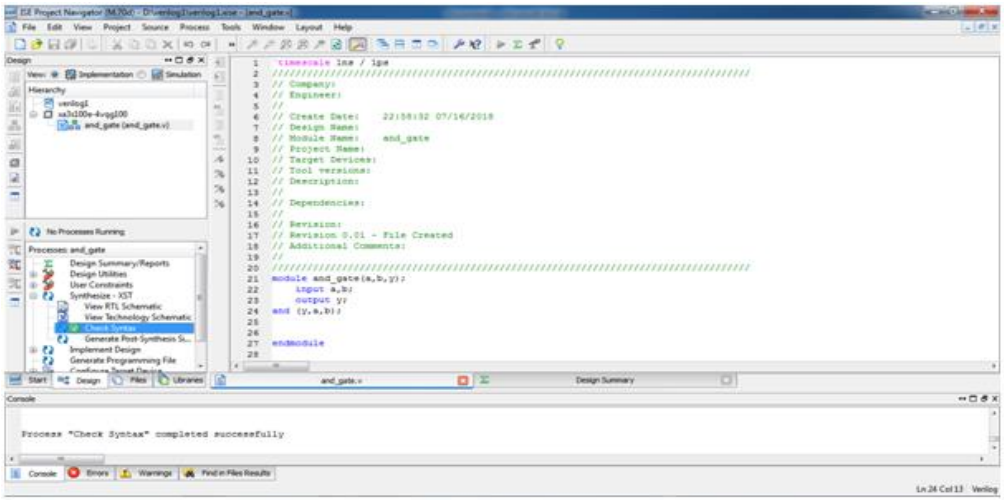

Fig:5.8 Check syntax for the Design

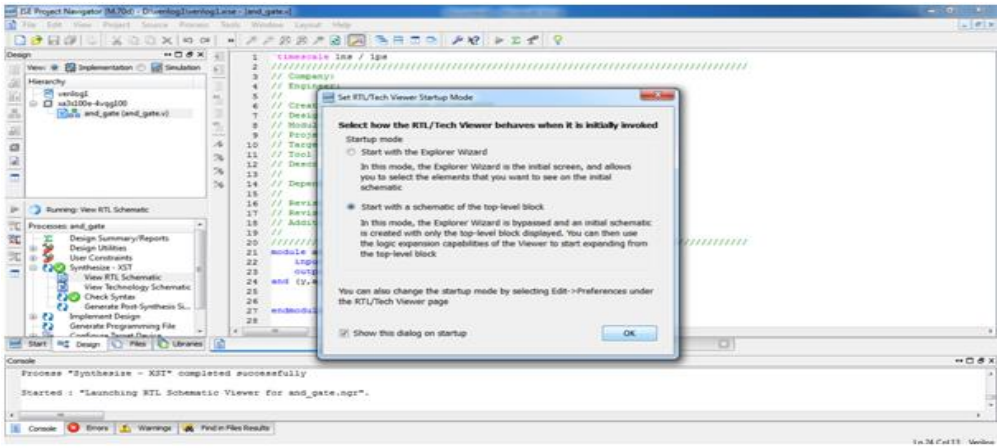

Fig:5.9 check for RTL schematic view

For RTL (register transfer logic) view, which is also known as designer view because it is look like what the designer thinks in his mind.

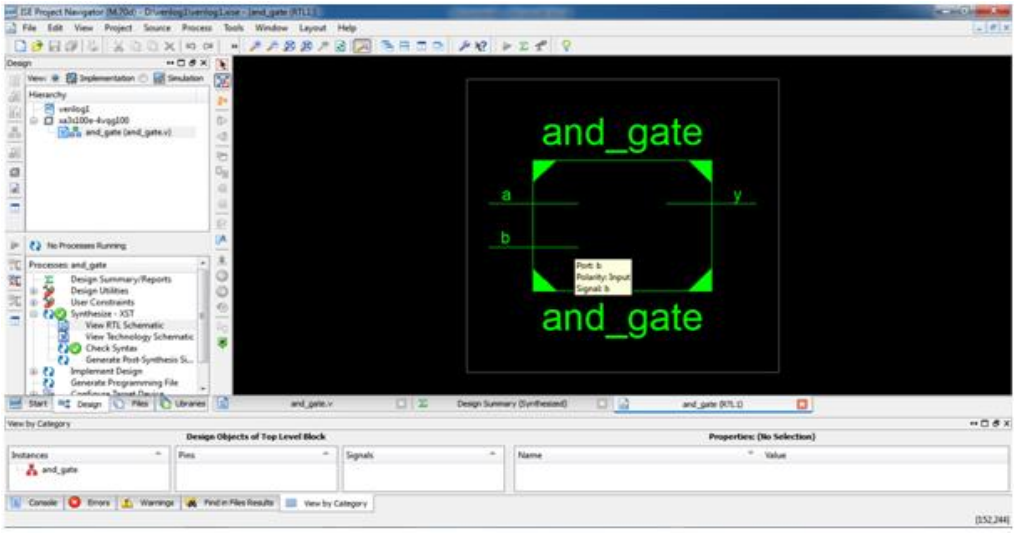

Fig:5.10 RTL schematic view of design (AND gate)

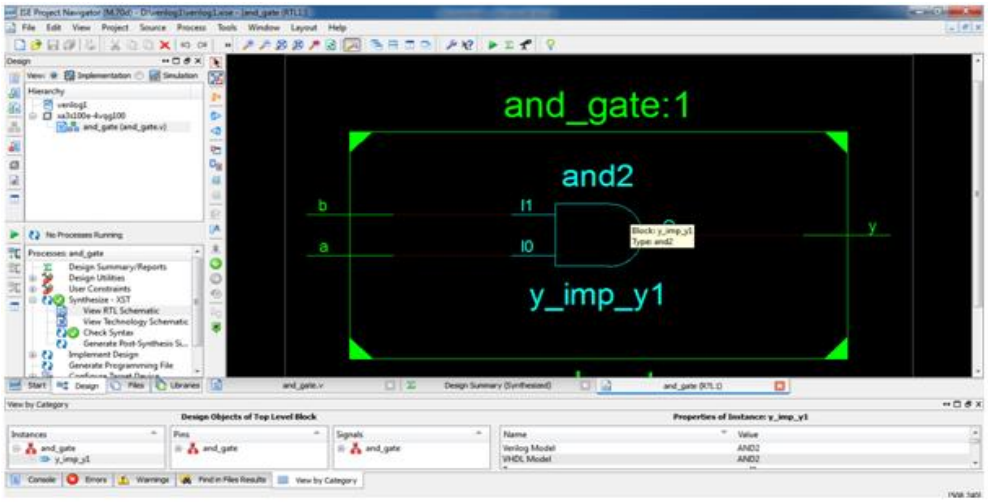

Fig:5.11 Internal structure of RTL schematic view of design

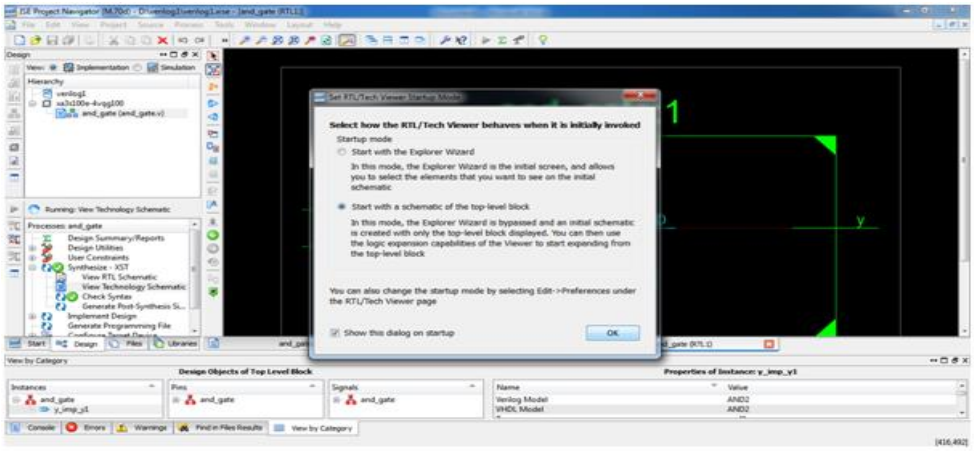

Fig:5.12 Check for Technology schematic view of the project

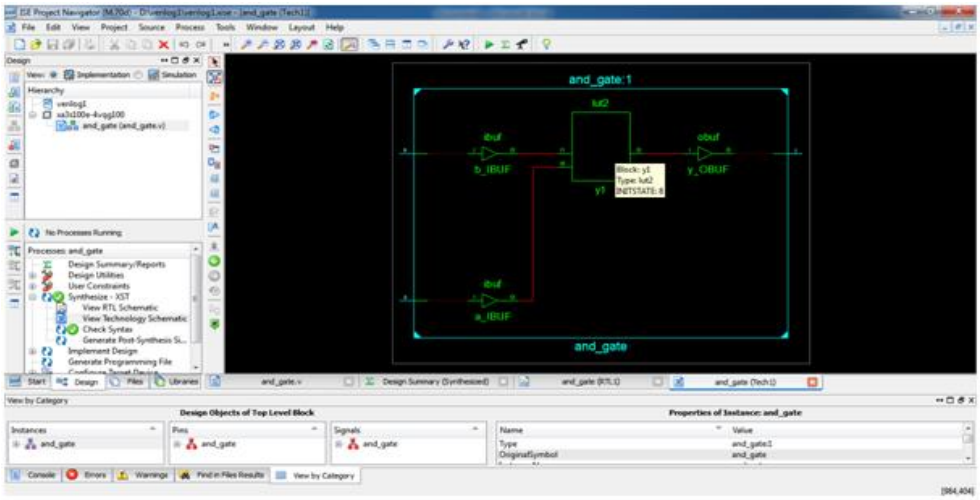

Fig:5.13 Internal structure of Technology schematic view

View technology schematic of design (and gate), here LUTs are displayed, LUT are considered as area of the design.

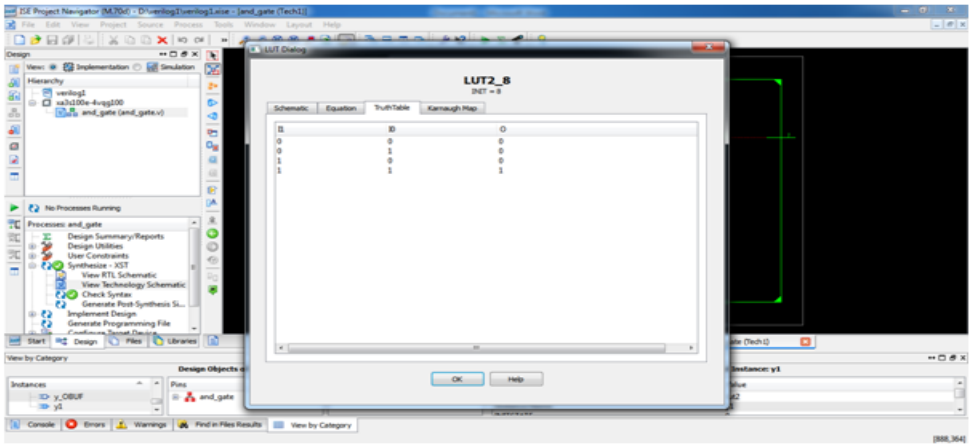

Fig:5.14 The truth table, schematic and k-map of design.

In Xilinx tool there is a availability to get truth table, schematic of design and k-map

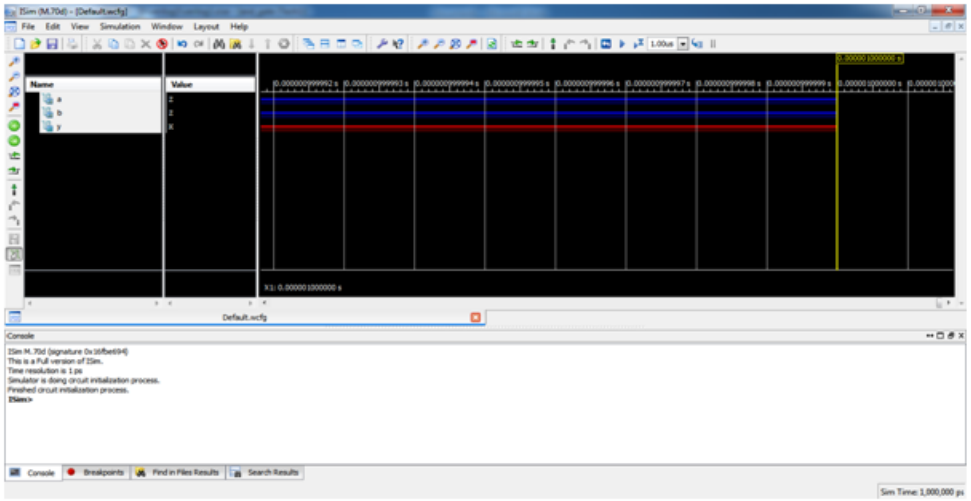

Fig:5.15 Simulation of design to verifying the logics of design.

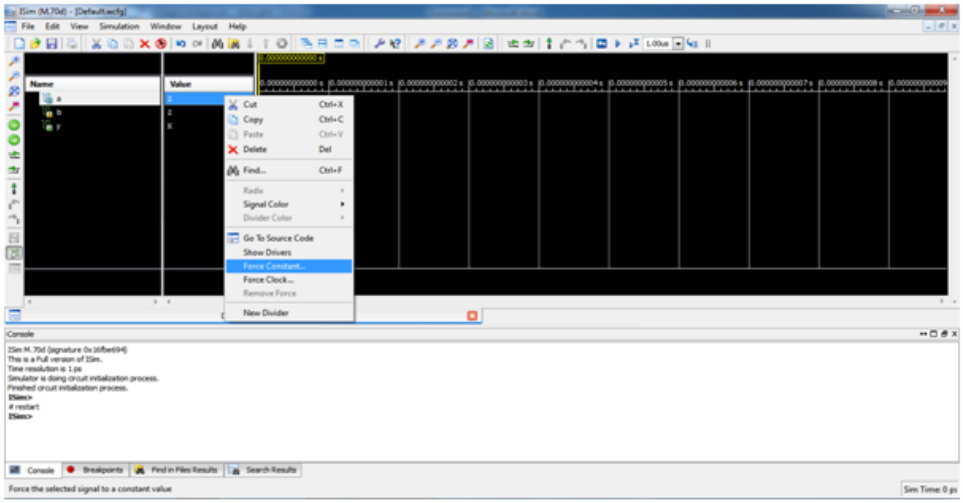

Fig:5.16 Apply inputs through force clock for input signals

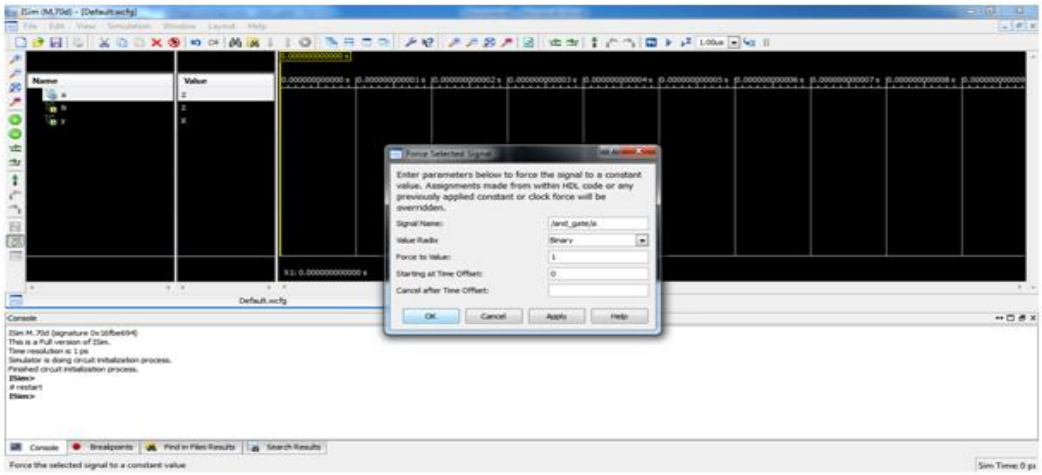

Fig:5.17 Apply force to value

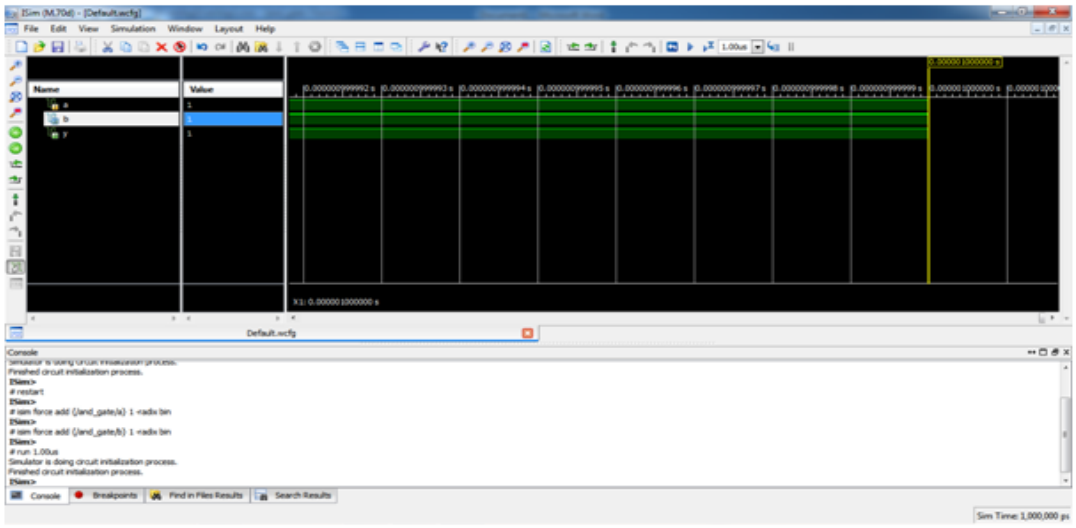

Fig:5.18 Run the design after applying inputs

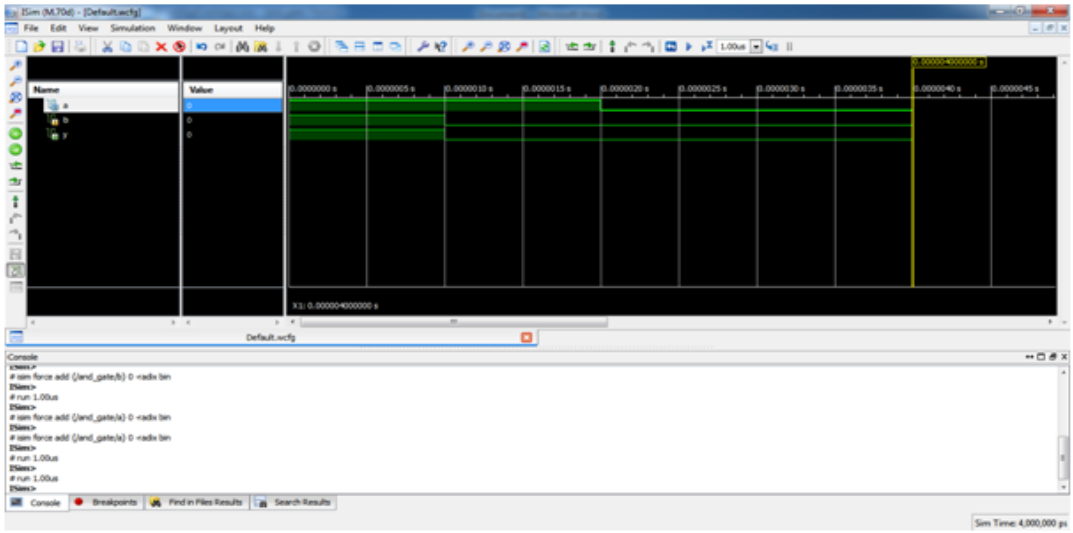

Fig:5.19 Show all values for the simulation

CHAPTER-6: RESULTS AND DISCUSSIONS

6.1 RTL SCHEMATIC:

The RTL schematic is abbreviated as the register transfer level it denotes the blue print of the architecture and is used to verify the designed architecture to the ideal architecture that we are in need of development. The HDL language is used to convert the description or summery of the architecture to the working summery by use of the coding language i.e verilog, VHDL. The RTL schematic even specifies the internal connection blocks for better analyzing. The figure represented below shows the RTL schematic diagram of the designed architecture.

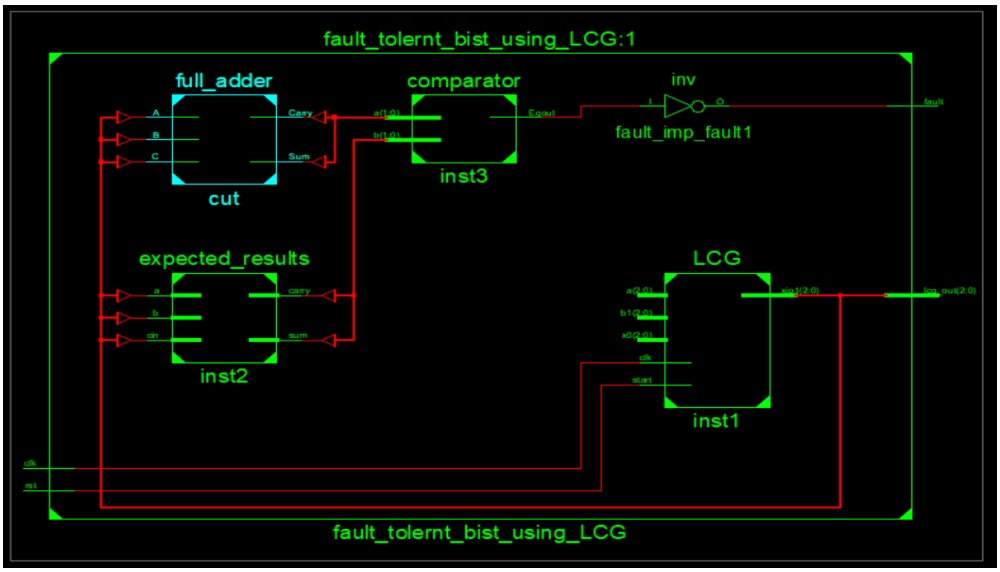

Fig:6.1 RTL Schematic of Existing Fault tolerant BIST.

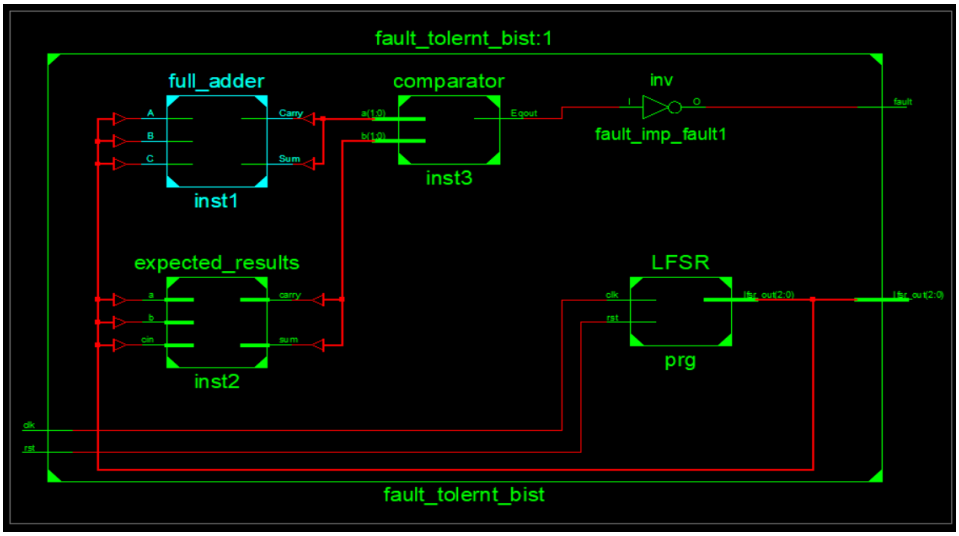

Fig:6.2 RTL Schematic of proposed Fault tolerant BIST.

6.2 TECHNOLOGY SCHEMATIC:

The technology schematic makes the representation of the architecture in LUT format, where the LUT is consider as the parameter of area that is used in VLSI to estimate the architecture design. The LUT is consider as an square unit the memory allocation of the code is represented in there LUT s in FPGA.

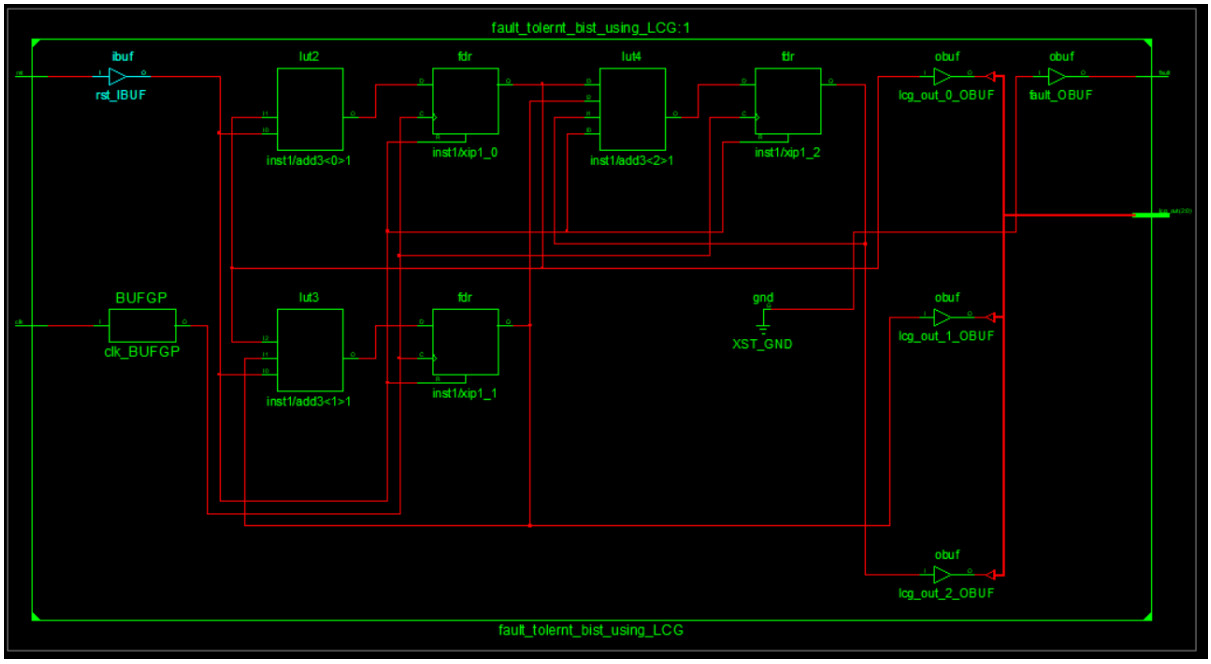

Fig:6.3 View Technology Schematic of existing Fault tolerant BIST.

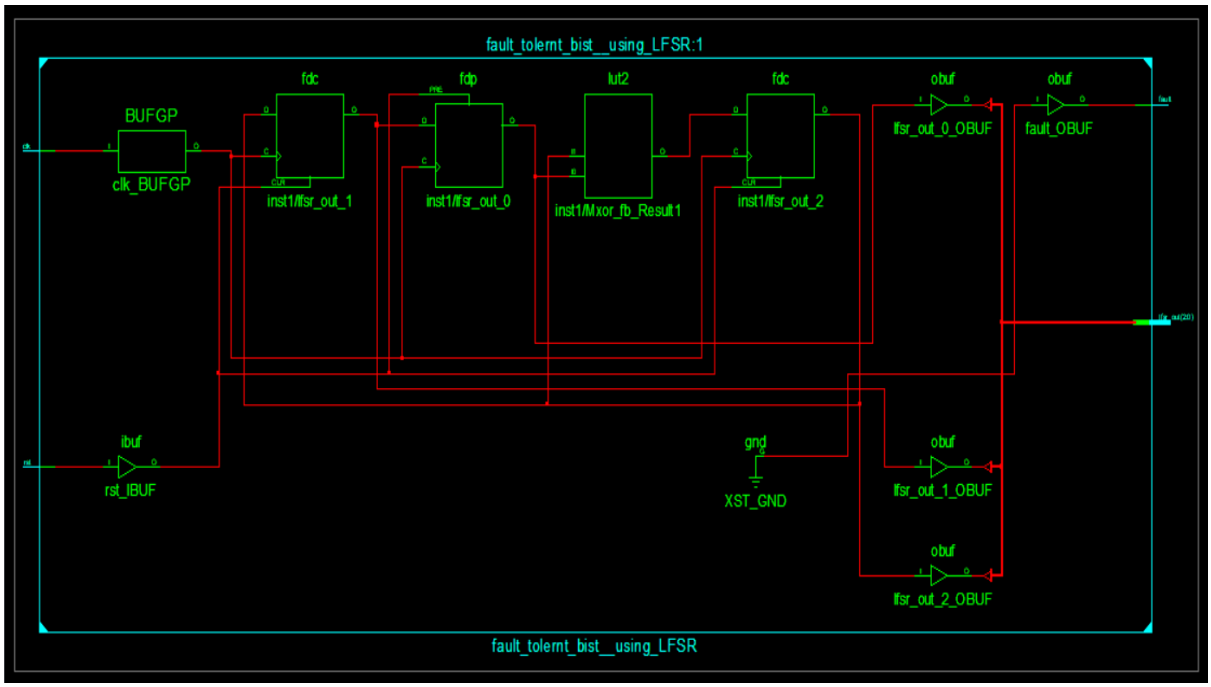

Fig:6.4 View Technology Schematic of proposed Fault tolerant BIST.

6.3 SIMULATION RESULT:

The simulation is the process which is termed as the final verification in respect to its working where as the schematic is the verification of the connections and blocks. The simulation window is launched as shifting from implementation to the simulation on the home screen of the tool, and the simulation window confines the output in the form of wave forms output. Here it has the flexibility of providing the different radix number systems.

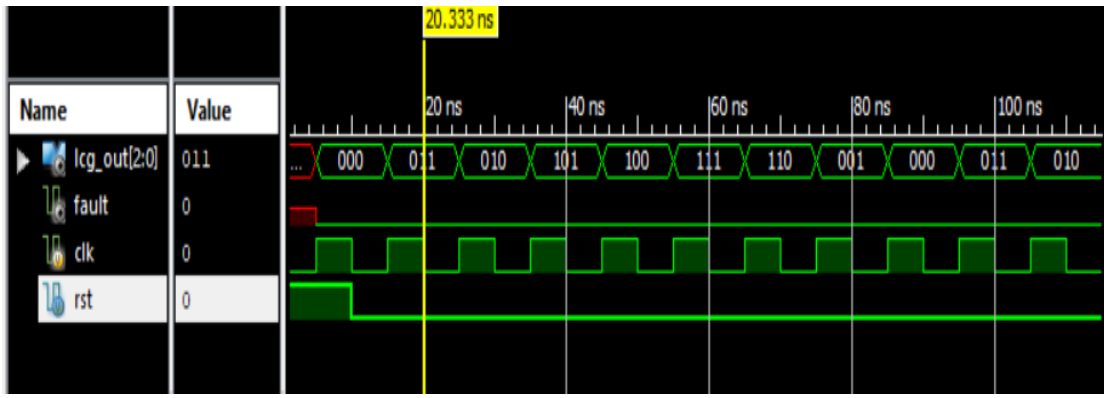

Fig:6.5 Simulated of existing design Fault free BIST.

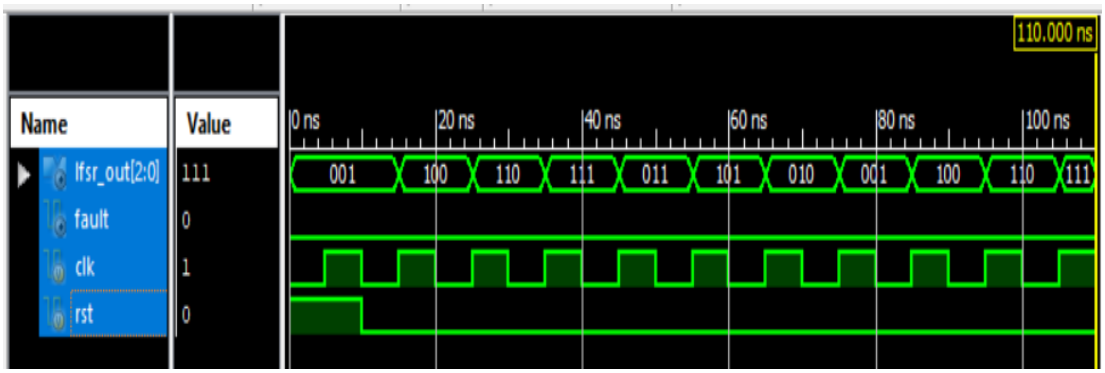

Fig:6.6 Simulated of proposed existing Fault free BIST.

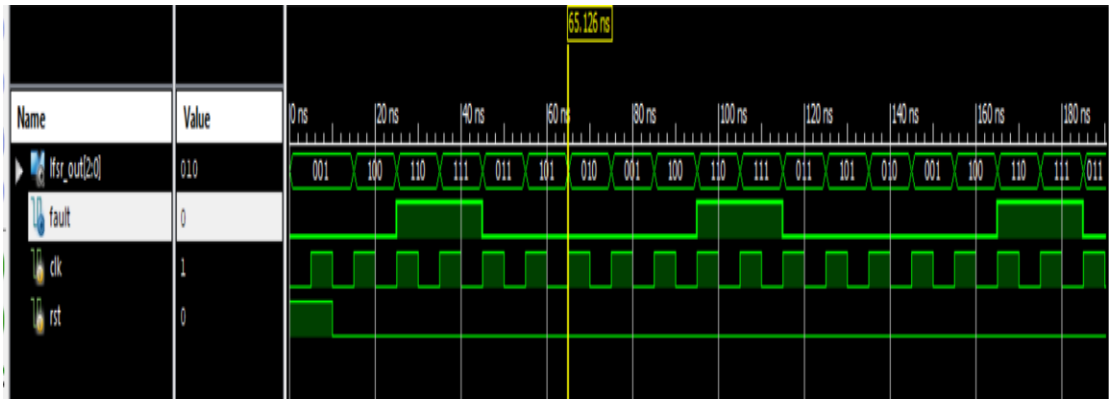

Fig:6.7 Simulated of proposed design Faulty BIST.

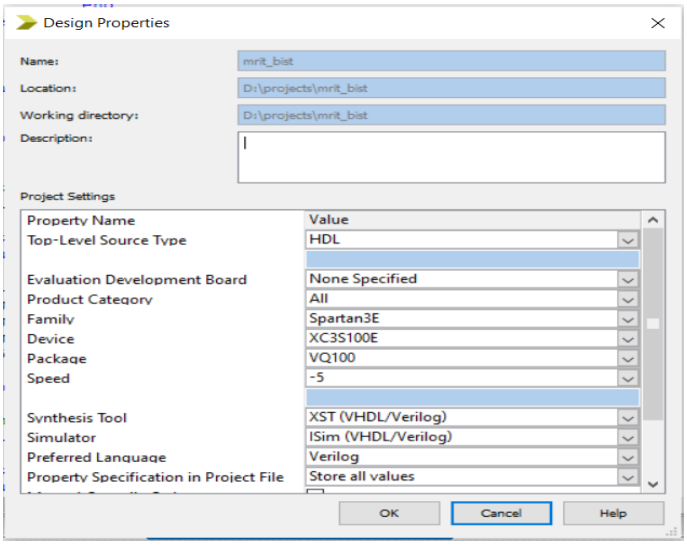

Fig:6.8 family used for synthesis

6.4 PARAMETERS OF EXISTING SYSTEM:

Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	2	960	0%
Number of Slice Flip Flops	3	1920	0%
Number of 4 input LUTs	3	1920	0%
Number of bonded IOBs	6	66	9%
Number of GCLKs	1	24	4%

Minimum period: 1.962ns (Maximum Frequency: 509.697MHz)

Minimum input arrival time before clock: 2.707ns

Maximum output required time after clock: 4.182ns

Maximum combinational path delay: No path found

Timing Detail:

All values displayed in nanoseconds (ns)

==========

==========

Timing constraint: Default period analysis for Clock 'clk'

Clock period: 1.962ns (frequency: 509.697MHz)

Total number of paths / destination ports: 6 / 3

Delay: 1.962ns (Levels of Logic = 1)

Source: inst1/xip1_0 (FF)

Destination: inst1/xip1_0 (FF)

Source Clock: clk rising

```
    Destination Clock: clk rising
    Data Path: inst1/xip1_0 to inst1/xip1_0
                              Gate     Net
      Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
      ----------------------------------------  ------------
       FDR:C->Q             4   0.514   0.568  inst1/xip1_0 (inst1/xip1_0)
       LUT2:I1->O           1   0.612   0.000  inst1/add3<0>1 (inst1/add3<0>)
       FDR:D                    0.268          inst1/xip1_0
      ----------------------------------------
      Total                    1.962ns (1.394ns logic, 0.568ns route)
                                   (71.1% logic, 28.9% route)
  ========================================================
==============
  Timing constraint: Default OFFSET IN BEFORE for Clock 'clk'
    Total number of paths / destination ports: 6 / 6
  ------------------------------------------------------------------------
  Offset:              2.707ns (Levels of Logic = 2)
    Source:            rst (PAD)
    Destination:       inst1/xip1_2 (FF)
    Destination Clock: clk rising
    Data Path: rst to inst1/xip1_2
                              Gate     Net
      Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
      ----------------------------------------  ------------
       IBUF:I->O            6   1.106   0.721  rst_IBUF (rst_IBUF)
       LUT2:I0->O           1   0.612   0.000  inst1/add3<0>1 (inst1/add3<0>)
       FDR:D                    0.268          inst1/xip1_0
      ----------------------------------------
      Total                    2.707ns (1.986ns logic, 0.721ns route)
                                   (73.4% logic, 26.6% route)
  ========================================================
==============
  Timing constraint: Default OFFSET OUT AFTER for Clock 'clk'
```

Total number of paths / destination ports: 3 / 3

Offset: 4.182ns (Levels of Logic = 1)

Source: inst1/xip1_0 (FF)

Destination: lcg_out<0> (PAD)

Source Clock: clk rising

Data Path: inst1/xip1_0 to lcg_out<0>

Cell:in->out	fanout	Gate Delay	Net Delay	Logical Name (Net Name)
FDR:C->Q	4	0.514	0.499	inst1/xip1_0 (inst1/xip1_0)
OBUF:I->O		3.169		lcg_out_0_OBUF (lcg_out<0>)

Total 4.182ns (3.683ns logic, 0.499ns route)

(88.1% logic, 11.9% route)

==========

6.5 PARAMETERS OF PROPOSED SYSTEM:

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	2	960	0%
Number of Slice Flip Flops	3	1920	0%
Number of 4 input LUTs	1	1920	0%
Number of bonded IOBs	6	66	9%
Number of GCLKs	1	24	4%

Minimum period: 1.926ns (Maximum Frequency: 519.224MHz)

Minimum input arrival time before clock: No path found

Maximum output required time after clock: 4.134ns

Maximum combinational path delay: No path found

Timing Detail:

All values displayed in nanoseconds (ns)

==========

Timing constraint: Default period analysis for Clock 'clk'

```
  Clock period: 1.926ns (frequency: 519.224MHz)
  Total number of paths / destination ports: 4 / 3
-------------------------------------------------------------------------
Delay:               1.926ns (Levels of Logic = 1)
  Source:            inst1/lfsr_out_0 (FF)
  Destination:       inst1/lfsr_out_2 (FF)
  Source Clock:      clk rising
  Destination Clock: clk rising
  Data Path: inst1/lfsr_out_0 to inst1/lfsr_out_2
                             Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDP:C->Q             2   0.514   0.532  inst1/lfsr_out_0 (inst1/lfsr_out_0)
     LUT2:I0->O           1   0.612   0.000  inst1/Mxor_fb_Result1 (inst1/fb)
     FDC:D                     0.268          inst1/lfsr_out_2
    ----------------------------------------
    Total                      1.926ns (1.394ns logic, 0.532ns route)
                                       (72.4% logic, 27.6% route)
=========================================================================
===============
  Timing constraint: Default OFFSET OUT AFTER for Clock 'clk'
  Total number of paths / destination ports: 3 / 3
-------------------------------------------------------------------------
Offset:              4.134ns (Levels of Logic = 1)
  Source:            inst1/lfsr_out_2 (FF)
  Destination:       lfsr_out<2> (PAD)
  Source Clock:      clk rising
  Data Path: inst1/lfsr_out_2 to lfsr_out<2>
                             Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDC:C->Q             3   0.514   0.451  inst1/lfsr_out_2 (inst1/lfsr_out_2)
     OBUF:I->O                 3.169          lfsr_out_2_OBUF (lfsr_out<2>)
```

Total 4.134ns (3.683ns logic, 0.451ns route)
(89.1% logic, 10.9% route)

==========

==========

CHAPTER-7: CONCLUSION AND FUTURE SCOPE

7.1 CONCLUSION:

The method for reducing test vectors and performing testing at speed testing is discussed in this paper. A circuit can be simulated in the presence of faults. The fault may be stuck-at-0 or stuck at 1 simulates at a functional level. The simulator is used for design verification and verifying its timing analysis. The process to find test vectors for single stuck at fault using path sensitization method and for multiple stuck-at faults using the Boolean difference method through forming a fault table. The number of test vectors can be minimized to find each line of fault in the circuit. Test vector becomes a major issue for the power consumption of the circuit. The methods for reducing test vector is designed and implemented by Hardware Description Language (HDL) simulation tools.

7.2 FUTURE SCOPE:

The future scope for identifying stuck-at-faults in a module using Testing Devices is a dynamic and crucial area of research. As memory technologies evolve, advanced error correction codes (ECC) like LDPC and polar codes will enhance reliability, while adaptive ECC can optimize performance based on real-time workload conditions. Fault-tolerant architecture utilizing redundant memory cells and hierarchical management systems can help mitigate failures. Moreover, integrating machine learning for predictive maintenance and anomaly detection will enable proactive fault management. Energy-efficient solutions, including low-power ECC and green computing initiatives, will address growing energy constraints. Additionally, tailored strategies for emerging memory technologies like 3D NAND and MRAM will be essential. Implementing self-healing mechanisms and robust system designs will further enhance resilience. Establishing comprehensive testing protocols and standardization for fault management will ensure reliability across diverse systems. Lastly, a cross-layer approach that includes collaboration between hardware and software teams will create more robust memory systems, ultimately supporting the demands of an increasingly data-driven world while minimizing environmental impact.

CHAPTER-8: BIBILOGRAPHY

[1]. L.Bushnell, and Vishwani D.Agrawal, "Essentials of Electronic Testing for Digital, Memory & Mixed–Signal VLSI Circuits",:Springer, 2002

[2]. MironAbramovici, Melvin A.Breuer, and Arthur D.Friedman. " Digital Testing and Testable Design", Piscataway-New Jersey: IEEE Press,1994.

[3]. N. Weste and K. Eshragian, “Principles of CMOS VLSI Design”, A Systems Perspective, second ed. Addison-Wesley, 1994.

[4]. Vinod Kumar Khera , R.K. Sharma, and A.K. Gupta " A heuristic fault based optimization approach to reduce test vectors count in VLSI testing", Journal of King Saud University – Computer and Information Sciences (2019)

[5]. K.L.V.Ramana Kumari, M.Asha Rani and N. Balaji "Design Verification And Test Vector Minimization Using Heuristic Method Of A Ripple Carry Adder", International Journal on Cybernetics & Informatics (IJCI) vol. 5, No. 4, pp. 307-313, August 2016.

[6]. Dhanabalan G and Tamil Selvi S, “Design of parallel conversion multichannel analog to digital converter for scan time reduction of programmable logic controller using FPGA”, Computer standards and interfaces, vol. 39, pp. 12 – 21, 2015.

[7]. Mary D. Pulukuri and Charles E. Stroud " On Built-In Self-Test for Adders", J Electron Test (2009) 25:343–346.

[8]. Ashok Kumar, Rahul Raj Choudhary, and Pooja Bhardwaj, "Universal Pattern Set for Arithmetic Circuits", International Journal of Computer Applications (0975 – 8887) vol. 40– No.15, February 2012.

[9]. S Jayanthy, and MC Bhuvaneswari, “Delay Fault Testing of VLSI Circuits”, – Springer 2019, Test Generation of Crosstalk Delay Faults, chapter-2, pp 15-35.

[10]. Matthias Sauer, Jie Jiang, Sven Reimer, Kohei Miyase, Xiaoqing Wen, Bernd Becker and Ilia Polian, “On Optimal Power-aware Path Sensitization”, IEEE 25th Asian Test Symposium, 2016, pp. 179-184.

[11]. Nelson, Nagle, Carroll, and Irwin",Digital Logic Circuit Analysis & Design" Prentice-Hall,1995, Chapter 12, pp. 739 to 757.

APPENDIX

INTRODUCTION TO VLSI:

Very-large-scale integration (VLSI) is the method of making included circuits by combining thousands of transistor-primary based circuits into a single chip. VLSI started inside the 1970s whilst complex semiconductor and conversation technology had been being developed. The microprocessor is a VLSI device., As chips have expanded in complexity into the masses of hundreds of thousands of transistors.

Overview:

The first semiconductor chips held one transistor. Subsequent advances added increasingly more transistors, and, as a consequence, greater individual features or systems have been integrated through the years. The first integrated circuits held just a few gadgets, possibly as many as ten diodes, transistors, resistors and capacitors, making it viable to fabricate one or greater logic gates on a single device. Now recognized retrospectively as "small-scale integration" (SSI), improvements in method brought about devices with hundreds of logic gates, called large-scale integration (LSI), i.e. Systems with as a minimum a thousand logic gates. Current generation has moved far past this mark and state-of-the-art microprocessors have many tens of millions of gates and loads of tens of millions of man or woman transistors.

At one time, there was an effort to name and calibrate various ranges of big-scale integration above VLSI. Terms like Ultra-large-scale Integration (ULSI) have been used. But the massive quantity of gates and transistors available on not unusual devices has rendered such first-class distinctions moot. Terms suggesting greater than VLSI tiers of integration are not in much use. Even VLSI is now fairly old fashioned, given the not unusual assumption that every one microprocessors are constructed in VLSI .

As of early 2008, billion-transistor processors are commercially available, an example of which is Intel's Montecito Itanium chip. This is predicted to end up extra common as semiconductor fabrication actions from the

contemporary generation of sixty five nm techniques to the next forty five nm generations (whilst experiencing new demanding situations including extended variation across procedure corners). Another excellent example is NVIDIA's 280 series GPU.

This microprocessor is precise in the fact that its 1.4 Billion transistor remember, able to a teraflop of overall performance, is nearly totally devoted to logic (Itanium's transistor depend is largely due to the 24MB L3 cache). Current designs, as opposed to the earliest gadgets, use good sized design automation and automatic logic synthesis to lay out the transistors, enabling better degrees of complexity within the resulting logic capability. Certain excessive-performance logic blocks just like the SRAM mobile, however, are nevertheless designed by hand to ensure the best performance (on occasion by using bending or breaking mounted design policies to gain the final little bit of overall performance by buying and selling stability).

What is VLSI?

VLSI stands for "Very Large Scale Integration". This is the field which involves packing more and more logic devices into smaller and smaller areas.

VLSI

1. Simply we say Integrated circuit is many transistors on one chip.
2. Design/manufacturing of extremely small, complex circuitry using modified semiconductor material
3. Integrated circuit (IC) may contain millions of transistors, each a few mm in size
4. Applications wide ranging: most electronic logic devices

History of Scale Integration:

The first transistors are invented in late 40s at Bell Labs, and in late 50s First IC (JK-FF by Jack Kilby at TI), followed in 60's SMALL SCALE INTEGRATION is done with two transistors.

By following the moore's integration law the integration of transistors has advanced and the MEDIUM SCALE INTEGRATION has evolved with 100

transistors. In late 70's LARGE SCALE INTEGRATION has evoloved with integration of 10,000 transistors. Later VERY LARGE SCALE INTEGRATION has evolved with more than 1 million transistors integration on a single chip.

Advantages of ICs over discrete components:

While we will deal with incorporated circuits, the homes of integrated circuits-what we will and can not efficiently installed an incorporated circuit-in large part determine the structure of the whole system. Integrated circuits improve device traits in numerous crucial approaches. ICs have 3 key advantages over virtual circuits built from discrete components:

- ✓ Size: Integrated circuits are a lot smaller-each transistors and wires are gotten smaller to micrometer sizes, compared to the millimeter or centimeter scales of discrete additives. Small length results in blessings in velocity and power consumption, considering that smaller components have smaller parasitic resistances, capacitances, and inductances.
- ✓ Speed: Signals can be switched between logic zero and logic 1 a great deal quicker inside a chip than they can between chips. Communication within a chip can occur loads of times faster than communication between chips on a printed circuit board. The excessive velocity of circuits on-chip is because of their small size-smaller additives and wires have smaller parasitic capacitances to slow down the signal.
- ✓ Power consumption: Logic operations inside a chip additionally take a whole lot much less power. Once more, decrease power intake is largely due to the small size of circuits on the chip-smaller parasitic capacitances and resistances require much less electricity to force them.

VLSI and systems:

These advantages of integrated circuits translate into advantages at the system level:

- Smaller bodily length. Smallness is regularly a bonus in itself-do not forget transportable televisions or hand held mobile phones.

- Lower power intake. Replacing a handful of popular parts with a single chip reduces total power consumption. Reducing strength intake has a ripple effect at the rest of the gadget: a smaller, cheaper power deliver can be used; for the reason that less power consumption approach less heat, a fan may additionally no longer be essential; a less difficult cupboard with much less protective for electromagnetic protective may be possible, too.
- Reduced value. Reducing the wide variety of components, the strength supply requirements, cupboard charges, and so forth, will necessarily lessen system cost. The ripple effect of integration is such that the fee of a gadget built from custom ICs may be less, even though the individual ICs fee greater than the usual parts they replace.

  Understanding why integrated circuit technology has such profound influence on the design of digital systems requires understanding both the technology of IC manufacturing and the economics of ICs and digital systems.

  - Electronic system in cars.
  - Digital electronics control VCRs
  - Transaction processing system, ATM
  - Personal computers and Workstations
  - Medical electronic systems.

Applications of VLSI:

Electronic structures now perform a huge type of duties in daily life. Electronic systems in a few instances have changed mechanisms that operated mechanically, hydraulically, or with the useful resource of different way; electronics are typically smaller, greater flexible, and much less complex to provider. In other instances digital systems have created truly new applications. Electronic structures perform a diffusion of obligations, some of them seen, a few more hidden:

- ✓ Personal input management systems such as portable MP3 players and DVD players perform sophisticated algorithms with remarkably little power.

- ✓ VLSI involvement in car is used in stero systems and fuel injections and ANTILOCK BREAK SYSTEMS(ABS). Even new evolutions are making use of accident acknowledgement to nearer hospitals and police stations and even in automatic mode driving.
- ✓ The video compression is done in digital signal processing for transmission of the data from one place to another.
- ✓ Low cost browsers are developed for fast and easy usage to users.
- ✓ Personal and laptops are the most often used systems , which perform the multitasking and provides a great flexibility for users.

The growing sophistication of programs continually pushes the layout and manufacturing a of included circuits and electronic systems to new levels of complexity. And perhaps the maximum super characteristic of this collection of structures is its variety-as systems emerge as more complex, we build no longer some popular-reason computers however an ever wider variety of special-motive systems. Our capacity to accomplish that is a testament to our growing mastery of each integrated circuit manufacturing and design, but the increasing needs of clients keep to test the bounds of layout and production.

VERILOG:

- Verilog synthesis gear can create logic-circuit structures without delay from Verilog behavioral description and goal them to a particular generation for attention (i.e., translate Verilog to actual hardware).
- Using Verilog, we will layout, simulate and synthesis something from a simple combinational circuit to a entire microprocessor on chip.
- Verilog HDL has evolved as a preferred hardware description language. Verilog HDL offers many beneficial capabilities for hardware layout.
- Verilog HDL is a trendy-reason hardware description language that is straightforward to analyze and easy to use. It is similar in syntax to the C programming language. Designers with C programming enjoy will locate it smooth to research Verilog HDL.

- Verilog HDL permits distinctive levels of abstraction to be combined within the equal version. Thus, a designer can define a hardware model in terms of switches, gates, RTL, or behavioral code. Also, a designer wishes to examine handiest one language for stimulus and hierarchical design.
- The Programming Language Interface (PLI) is a effective function that allows the user to jot down custom C code to engage with the inner facts systems of Verilog. Designers can customize a Verilog HDL simulator to their desires with the PLI.

History Of Verilog HDL: Verilog turned into initial of all as a proprietary hardware modeling language by using Gateway Design Automation Inc. Round 1984. It is rumored that the original language turned into designed by way of taking features from the maximum popular HDL language of the time, known as HiLo, as well as from traditional pc languages which includes C. At that point, Verilog become not standardized and the language changed itself in nearly all of the revisions that got here out inside 1984 to 1990.

- Verilog simulator became first used beginning in 1985 and was extended notably via 1987. The implementation changed into the Verilog simulator offered via Gateway. The first main extension became Verilog-XL, which brought some functions and implemented the notorious "XL set of rules" which become a very green technique for doing gate-degree simulation.

- The time was late 1990. Cadence Design System, whose primary product at that point blanketed Thin film manner simulator, determined to accumulate Gateway Automation System. Along with other Gateway merchandise, Cadence now have become the proprietor of the Verilog language, and continued to marketplace Verilog as each a language and a simulation.
- OVI did a large amount of paintings to enhance the Language Reference Manual (LRM), clarifying matters and making the language specification as dealer-unbiased as viable.

- Soon it was found out that if there have been too many organizations in the marketplace for Verilog, probably each person would like to do what Gateway had performed to date - changing the language for his or her very own benefit. This might defeat the main reason of freeing the language to public area. As a result in 1994, the IEEE 1364 operating institution become shaped to show the OVI LRM into an IEEE preferred. This attempt turned into concluded with a successful ballot in 1995, and Verilog have become an IEEE trendy in December 1995.
- When Cadence gave OVI the LRM, numerous agencies began running on Verilog simulators. In 1992, the primary of those have been announced, and by 1993 there have been numerous Verilog simulators available from organizations other than Cadence.
- After a few years, new capabilities had been introduced to Verilog, and the new version is referred to as Verilog 2001. This version seems to have fixed a variety of issues that Verilog 1995 had. This model is called 1364-2001.

Program structure: The basic unit and programming in Verilog is ”MODULE”(a text file containing statements and declarations).A Verilog module has declarations that describes the names and types of the module inputs and outputs as well as local signals, variables, constants and functions that are used internally to the module, are not visible outside. Verilog is a case-sensitive language like C. Thus sense, Sense, SENSE, sense, etc., are all treated as different entities / quantities in Verilog.

Syntax:

Module Module_Name(port list);

Port declaration

Function declaration

Endmodule

module ← signifies the beginning of a module definition.

endmodule ←signifies the end of a module definition.

VERILOG OPERATORS:

Arithmetic Operators: These perform arithmetic operations. The **+** and **-** can be used as either unary (-z) or binary (x-y) operators.

+ (addition)

- (subtraction)

* (multiplication)

/ (division)

% (modulus)

Relational Operators: Relational operators compare two operands and return a single bit 1or 0. These operators synthesize into comparators.

< (less than)

<= (less than or equal to)

> (greater than)

>= (greater than or equal to)

== (equal to)

!= (not equal to)

Bit-wise Operators: Bit-wise operators do a bit-by-bit comparison between two operands. However see “Reduction Operators”.

~ (bitwise NOT)

& (bitwise AND)

| (bitwise OR)

^ (bitwise XOR)

~^ or ^~ (bitwise XNOR)

Logical Operators: Logical operators return a single bit 1 or 0. They are the same as bit-wise operators only for single bit operands. They can work on expressions, integers or groups of bits, and treat all values that are nonzero as “1”. Logical operators are typically used in conditional (**if** … **else**) statements since they work with expressions.

! (logical NOT)

&& (logical AND)

|| (logical OR)

Reduction Operators: Reduction operators operate on all the bits of an operand vector and return a single-bit value. These are the unary (one argument) form of the bit-wise operators above.

& (reduction AND)

| (reduction OR)

~& (reduction NAND)

~| (reduction NOR)

^ (reduction XOR)

~^ or ^~ (reduction XNOR)

Shift Operators: Shift operators shift the first operand by the number of bits specified by the second operand. Vacated positions are filled with zeros for both left and right shifts (There is no sign extension).

<< (shift left)

>> (shift right)

Concatenation Operator: The concatenation operator combines two or more operands to form a larger vector

{ } (concatenation)

Replication Operator: The replication operator makes multiple copies of an item.

Literals: Literals are constant-valued operands that can be used in Verilog expressions. The two common Verilog literals are:

(a) String: A string literal is a one-dimensional array of characters enclosed in double quotes(“”).

(b) Numeric: constant numbers specified in binary, octal, decimal or hexadecimal.

Number Syntax

n’Fddd…, where

n - integer representing number of bits

F - one of four possible base formats:

b (binary), o (octal), d (decimal),h (hexadecimal). Default is d.

NET: Verilog actually has two classes of signals

1. nets.
2. variables.

- Nets represent connections between hardware elements. Just as in real circuits, nets have values continuously driven on them by the outputs of devices that they are connected to.

- The default net type is wire, any signal name that appears in a module input /output list, but not in a net declaration is assumed to be type wire.
- Nets are one-bit values by default unless they are declared explicitly as vectors. The terms wire and net are often used interchangeably.
- Note that net is not a keyword but represents a class of data types such as wire, wand, wor, tri, triand, trior, trireg, etc. The wire declaration is used most frequently.
- The syntax of verilog net declaration is similar to an input/output declaration.

  Syntax:

  Wire identifier ,……. identifier;
  Wire [msb:lsb] identifier ,……. identifier;
  tri identifier ,……. identifier;
  tri [msb:lsb] identifier ,……. identifier;

- The keyword tri has a function identical to that of wire. When a net is driven by more than one tri-state gate, it is declared as tri rather than as wire. The distinction is for better clarity.

VARIABLE: Verilog variables stores the values during the program execution, and they need not have Physical significance in the circuit.

- They are used in only procedural code (i.e,behavioral design).A variable value can be used in a expression and can be combined and assign to other variables, as in conventional software programming language.
- The most commonly used variables are REG and INTEGERS.

Syntax:

Reg identifier ,…….identifier;
Reg [msb:lsb] identifier,…….identifier;
Integer identifier ,…….identifier;

- A register variable is a single bit or vector of bits , the value of 1-bit reg variable is always 0,1,X,Z. the main use of reg variables is to store values in Verilog procedural code.
- An integer variable value is a32-bit or larger integer ,depending on the word length on the word length used by simulator .An integer variable is typically used to control a repetitive statements ,such as loop, in Verilog procedural code.

PARAMETER: Verilog provides a facility for defining named constants within a module ,to improve readability and maintainability of code. The parameter declaration is

Syntax:

Parameter identifier =value;
Parameter identifier =value,
: :
identifier =value;

- An identifier is assigned to a constant value that will be used in place of the identifier throughout the current module.
- Multiple constants can be defined in a single parameter declaration using a comma –separated list of arguments.
- The value in the parameter declaration can be simple constant ,or it can be a constant expression.
- An expression involving multiple operators and constants including other parameters ,that yields a constant result at compile time. The parameter scope is limited to that module in which it is defined.

ARRAYS: Arrays are allowed in Verilog for reg, integer, time, and vector register data types. Arrays are not allowed for real variables. Arrays are accessed by <array_name> [<subscript>]. Multidimensional arrays are not permitted in Verilog.

Syntax:

Reg identifier [start:end];
Reg [msb:lsb] identifier [start:end];
Integer identifier [start:end] ;

DATAFLOW DESIGN ELEMENTS: Continuous assignment statement allows to describe a combinational circuit in terms of the flow of data and operations on the circuit. This style is called "dataflow design or description".

Syntax:

Assign net-name=expression;
Assign net-name[bit-index]=expression;
Assign net-name[msb:lsb]=expression;
Assign net-concatenation =expression;

- "**Assign**" is the keyword carrying out the assignment operation. This type of assignment is called a continuous assignment.
- The keyword "assign "is followed by the name of a net, then an"="sign and finally an expression giving the value to be assigned
- If a module contains two statements "assign X=Y" and "assign Y=~X", then the simulation will loop "forever"(until the simulation times out).

For example:

assign c = a && b;

- a and b are operands – typically single-bit logic variables.
- "&&" is a logic operator. It does the bit-wise AND operation on the two
- operands a and b.
- "=" is an assignment activity carried out.
- c is a net representing the signal which is the result of the assignment.

STURCTURAL DESIGN (OR) GATE LEVEL MODELING: Structural Design Is the Series of Concurrent Statement. The Most Important Concurrent Statement In the module covered like instance statements, continuous – assignment statement and always block. These gives rise to three distinct styles of circuit design and description.

- Statement of these types, and corresponding design styles, can be freely intermixed within a Verilog module declaration.
- Each concurrent statement in a Verilog module "executes" simultaneously with other statements in the same module declaration.

- In Verilog module, if the last statement updates a signal that is used by the first statement, then the simulator goes back to that first statement and updates its result.
- In fact, the simulator will propagate changes and updating results until the simulated circuit stabilizes.
- In structural design style, the circuit description or design individual gates and other components are instantiated and connected to each other using nets.

Syntax of Verilog instance statements:

Component_name
instance-identifier(expression……expresssion);
Component_name instance-identifier (**.**port-name(expression),
: **:**
.port-name(expression));

BEHAVIORAL MODELING: Behavioral level modeling constitutes design description at an abstract level. One can visualize the circuit in terms of its key modular functions and their behavior. The constructs available in behavioral modeling aim at the system level description. Here direct description of the design is not a primary consideration in the Verilog standard. Rather, flexibility and versatility in describing the design are in focus [IEEE].Verilog provides designers the ability to describe design functionality in an algorithmic manner. In other words, the designer describes the behavior of the circuit. Thus, behavioral modeling represents the circuit at a very high level of abstraction. Design at this level resembles C programming more than it resembles digital circuit design. Behavioral Verilog constructs are similar to C language constructs in many ways.

FOREVER LOOPS: The forever statement executes an infinite loop of a statement or block of statements. To avoid combinational feedback during synthesis, a forever loop must be broken with an@(posedge/negedge clock) statement. For simulation a delay inside the loop will suffice. If the loop contains only one statement, the *begin ... end* statements may be omitted.

Syntax

forever

begin

… statements …

End

Example

forever begin

@(posedge clk); // or use a= #9 a+1;

a = a + 1;

end

REPEAT: The repeat statement executes a statement or blocks of statements a fixed number of times. repeat CONSTRUCT The repeat construct is used to repeat a specified block a specified number of times. The quantity a can be a number or an expression evaluated to a number. As soon as the repeat statement is encountered, a is evaluated. The following block is executed "a" times. If "a" evaluates to 0 or x or z, the block is not executed.

Syntax:

repeat (number_of_times)

begin

… statements …

End

SOURCE CODE:

Existing system:

```
module fault_tolernt_bist(clk,rst,lcg_out ,fault );
input clk,rst;
output fault ;
output [2:0] lcg_out ;//pattern generator output
//x0== initial seed …1
parameter a=4'd5,b=4'd3 ;
//a-1 is must be devisible by 4
//bis reletively prime with 16=2^4
wire [2:0] p ;//p[2],p[1],p[0]
wire tsum,tcarry ;
wire Esum,Ecarry ;
```

```
wire Eqout ;
LCG inst1(.x0(3'd1),.start(rst),.clk(clk),.a(a),.b1(b),.xip1(p));
full_adder cut(.A(p[2]),.B(p[1]),.C(p[0]),.Sum(tsum),.Carry(tcarry));
expected_results inst2(.a(p[2]),.b(p[1]),.cin(p[0]),.sum(Esum),.carry(Ecarry));
comparator  inst3(.a({tsum,tcarry}),.b({Esum,Ecarry}),.Eqout(Eqout));
assign lcg_out = p ;
assign fault = Eqout ? 0 : 1 ;
endmodule
//expected output
module expected_results(a,b,cin,sum,carry);
input a,b,cin;
output reg sum,carry;
always @*
begin
                        case({a,b,cin})
                        3'd1 : begin//001
                                        sum = 1 ;
                                        carry = 0 ;
                                        end
                        3'd2 : begin//010
                                        sum = 1 ;
                                        carry = 0 ;
                                        end
                        3'd3 : begin
                                        sum = 0 ;
                                        carry = 1 ;
                                        end
                        3'd4 : begin
                                        sum = 1 ;
                                        carry = 0 ;
                                        end
                        3'd5 : begin
                                        sum = 0 ;
```

```
                    carry = 1 ;
                    end
            3'd6 : begin
                    sum = 0 ;
                    carry = 1 ;
                    end
            3'd7 : begin
                    sum = 1 ;
                    carry = 1 ;
                    end
            default  begin
                    sum = 0 ;
                    carry = 0 ;
                    end
            endcase
end
endmodule
//cut (full adder)
module full_adder(A,B,C,Sum,Carry);
input A,B,C;
output Sum,Carry ;
xor g1(S1,A,B);////
and g2(w1,A,B);//B
and g3(w2,S1,C);
xor g4(Sum, S1,C);
or g5(Carry,w1,w2);
endmodule
//comparator
module comparator (a,b,Eqout);
input [1:0] a,b;
output Eqout ;
assign Eqout = a == b ;
endmodule
```

```
//lcg...linear congruential generator for test patterns
module LCG(x0,start,clk,a,b1,xip1);
input [2:0]x0;
input [2:0]a,b1;
input start,clk;
output reg [2:0]xip1;
wire [2:0]xi,lsr,add3 ;
wire [1:0]r ;
//parameter b1 = 4'd1;
assign xi = start ? x0 : xip1 ;//mux4 logic
//a= 2^r+1 --> a=5;r=2
rgen rg1(.a(a),.r(r));
assign lsr = xi<<r ;//r bit logical shifting
assign add3 = xi + lsr + b1 ;
always @(posedge clk)
begin
                    if(start) xip1 <= 4'd0 ;
                    else xip1 <= add3 ;
end
endmodule
//r genaration
module rgen(a,r);
input [2:0]a;//2^8>9
output reg [1:0]r;
always @(a)
begin
                    case (a)
                    4'd5 : r = 2 ;
                    default : r = 0 ;
                    endcase
end
endmodule
```

```
module bist_tb;
                    // Inputs
                    reg clk=0;
                    reg rst=1;
                    // Outputs
                    wire [2:0] lcg_out;
                    wire fault;
                    // Instantiate the Unit Under Test (UUT)
                    fault_tolernt_bist uut (
                      .clk(clk),
                      .rst(rst),
                      .lcg_out(lcg_out),
                      .fault(fault)
                    );
always #5 clk = !clk ;
initial #10 rst = 0 ;
endmodule
```

proposed system:

```
module fault_tolernt_bist(clk,rst,lfsr_out ,fault );
input clk,rst;
output fault ;
output [2:0] lfsr_out ;
wire [2:0] p ;//p[2],p[1],p[0]
wire tsum,tcarry ;
wire Esum,Ecarry ;
wire Eqout ;
LFSR inst1(.clk(clk),.rst(rst),.lfsr_out(p));

full_adder cut(.A(p[2]),.B(p[1]),.C(p[0]),.Sum(tsum),.Carry(tcarry));
expected_results inst2(.a(p[2]),.b(p[1]),.cin(p[0]),.sum(Esum),.carry(Ecarry));
comparator  inst3(.a({tsum,tcarry}),.b({Esum,Ecarry}),.Eqout(Eqout));
assign lfsr_out = p ;
```

```
assign fault = Eqout ? 0 : 1 ;
endmodule
//expected output
module expected_results(a,b,cin,sum,carry);
input a,b,cin;
output reg sum,carry;
always @*
begin
                    case({a,b,cin})
                    3'd1 : begin
                                    sum = 1 ;
                                    carry = 0 ;
                                    end
                    3'd2 : begin
                                    sum = 1 ;
                                    carry = 0 ;
                                    end
                    3'd3 : begin
                                    sum = 0 ;
                                    carry = 1 ;
                                    end
                    3'd4 : begin
                                    sum = 1 ;
                                    carry = 0 ;
                                    end
                    3'd5 : begin
                                    sum = 0 ;
                                    carry = 1 ;
                                    end
                    3'd6 : begin
                                    sum = 0 ;
                                    carry = 1 ;
                                    end
```

```
                3'd7 : begin
                                sum = 1 ;
                                carry = 1 ;
                                end
                default  begin
                                sum = 0 ;
                                carry = 0 ;
                                end
                endcase
end
endmodule
//cut (full adder)
module full_adder(A,B,C,Sum,Carry);
input A,B,C;
output Sum,Carry ;
xor g1(S1,A,B);////
and g2(w1,A,B);//B
and g3(w2,S1,C);
xor g4(Sum, S1,C);
or g5(Carry,w1,w2);
endmodule
//comparator
module comparator (a,b,Eqout);
input [1:0] a,b;
output Eqout ;
assign Eqout = a == b ;
endmodule
//lfsr
module LFSR(clk,rst,lfsr_out);
input clk ,rst ;
output reg [2:0] lfsr_out ;
assign fb = lfsr_out[0] ^ lfsr_out[2] ;
always @(posedge clk or posedge rst )
```

```
begin
                    if(rst)lfsr_out <= 3'd1 ;
                    else lfsr_out <= { fb , lfsr_out[2:1] } ;
end
endmodule
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////
//////////////////
module tb;
                    // Inputs
                    reg clk =0;
                    reg rst=1;
                    // Outputs
                    wire [2:0] lfsr_out;
                    wire fault;
                    // Instantiate the Unit Under Test (UUT)
                    fault_tolernt_bist uut (
                      .clk(clk),
                      .rst(rst),
                      .lfsr_out(lfsr_out),
                      .fault(fault)
                    );
always #5 clk = ! clk ;
                    initial begin
                      // Initialize Inputs
                      //clk = 0;
                      #10 rst = 0;
                      // Wait 100 ns for global reset to finish
                      #100;
                      // Add stimulus here
$stop ;
                    end
endmodule
```

YUKTI CERTIFICATE

MoE's INNOVATION CELL (GOVERNMENT OF INDIA)

INSTITUTION'S INNOVATION COUNCIL
MOE'S INNOVATION CELL

INSTITUTION'S INNOVATION COUNCIL (Ministry of Education Initiative)

Institute Name:
Malla Reddy Institute of Technology & Science

Title of the Innovation/Prototype:
Identification of Stuck-at-Faults in a module using Testing Devices

Team Lead Name:	Team Lead Email:	Team Lead Phone:	Team Lead Gender:
Tejasri Gurrala	gurralatejasri8@gmail.com	7702710629	Female
FY of Development:	**Developed as part of:**	**Innovation Type:**	**TRL LEVEL:**
2024-25	Academic Requirement/Study Project	Process	3

Theme:
IoT based technologies (e.g. Security & Surveillance systems etc),Manufacturing,

Define the problem and its relevance to today's market / sociaty / industry need:
The problem is the limitations of traditional artificial intelligence (AI) systems in simulating human-like cognition and mental states. Current AI systems lack the ability to understand and replicate human mental events and objects, hindering their ability to interact and collaborate with humans effectively. This limitation is relevant to today's market, society, and industry needs, as there is a growing demand for AI systems that can understand and respond to human emotions, intentions, and needs, particularly in areas like customer service, healthcare, and education.

Describe the Solution / Proposed / Developed:
The proposed solution is an advanced artificial intelligence (AI) system that simulates human-like cognition and mental states by integrating mental events and objects. This system utilizes cognitive architectures and machine learning algorithms to replicate human mental processes, enabling it to understand and respond to human emotions, intentions, and needs. The system is designed to be adaptable and scalable, making it applicable to various industries, including customer service, healthcare, and education, where human-AI interaction is critical.

Explain the uniqueness and distinctive features of the (product / process / service) solution:
The uniqueness of the proposed AI system lies in its ability to simulate human-like cognition and mental states, enabling it to understand and respond to human emotions, intentions, and needs. Distinctive features include: - Integration of cognitive architectures and machine learning algorithms - Adaptive and scalable design - Ability to replicate human mental processes - Emotion recognition and response capabilities - Human-like interaction and collaboration These features set the system apart from traditional AI solutions, enabling more natural and effective human-AI interaction.

How your proposed / developed (product / process / service) solution is different from similiar kind of product by the competitors if any:
Our proposed AI system differs from similar products by competitors in several ways: - Deeper cognitive capabilities: Our system integrates cognitive architectures and machine learning algorithms to replicate human mental processes, whereas competitors' products focus on narrow AI applications. - Emotion recognition and response: Our system can recognize and respond to human emotions, creating a more natural and empathetic interaction experience. - Adaptive and scalable design: Our system can adapt to various industries and applications, making it a more versatile solution than competitors' products.

Is there any IP or Patentable Component associated with the Solution?:
No

Has the Solution Received any Innovation Grant/Seefund Support?:
No

Are there any Recognitions (National/International) Obtained by the Solution?:
No

***Is the Solution Commercialized either through Technology Transfer or Enterprise Development/Startup?:**
No

Had the Solution Received any Pre-Incubation/Incubation Support?:
No

Video URL:
https://drive.google.com/file/d/1XkUghu8ySYxIgv4P2hUco7_m2OIO5yVl/view?usp=sharing

Downloaded on: 11-03-2025

This report is electronically generated against Yukti - National Innovation Repository Portal.