**[Power-Efficient BIST Design with High Fault Coverage Using Verilog]**

**Submitted**

**By**

**Project Team:**

* **JAGADEESHWAR REDDY BUSIREDDY-BU21EECE0100177**
* **NAVEEN BADDILA- BU21EECE0100412**
* **VAMSI KOPPALA- BU21EECE0100085)**

**Under the Guidance of**

**(Dr.KARTHICK S, Assistant Professor)**

**(Duration: 14/08/2024 to 19/03/2025)**



**Department of Electrical, Electronics and Communication Engineering**

**GITAM School of Technology**

**GITAM**

**(DEEMED TO BE UNIVERSITY)**

**(Estd. u/s 3 of the UGC act 1956)**

**NH 207, Nagadenehalli, Doddaballapur taluk, Bengaluru-561203 Karnataka, INDIA.**

**DECLARATION**

**I/We declare that the project work contained in this report is original and it has been done by me under the guidance of my project guide.**

**Name:**

**Date: Signature of the Student**

**JAGADEESHWAR REDDY BUSIREDDY**

**NAVEEN BADDILA**

**VAMSI KOPPALA**

**Department of Electrical, Electronics and Communication Engineering GITAM School of Technology, Bengaluru-561203**

****

**CERTIFICATE**

**This is to certify that (Student Name) bearing (Regd. No.:) has satisfactorily completed Mini Project Entitled in partial fulfillment of the requirements as prescribed by University for VIIIth semester, Bachelor of Technology in “Electrical, Electronics and Communication Engineering” and submitted this report during the academic year 2024-2025.**

**[Signature of the Guide] [Signature of HOD**

**Table of contents**

[**Chapter 1: Introduction 1**](#_heading=h.gjdgxs)

[1.1 Overview of the problem statement 1](#_heading=h.30j0zll)

[1.2 Objectives and goals 1](#_heading=h.1fob9te)

[**Chapter 2 : Literature Review 2**](#_heading=h.3znysh7)

[**Chapter 3 : Strategic Analysis and Problem Definition 3**](#_heading=h.2et92p0)

[3.1 SWOT Analysis 3](#_heading=h.tyjcwt)

[3.2 Project Plan - GANTT Chart 3](#_heading=h.1t3h5sf)

[**Chapter 4 : Methodology 4**](#_heading=h.17dp8vu)

[4.1 Description of the approach 4](#_heading=h.3rdcrjn)

[4.2 Tools and techniques utilized 4](#_heading=h.26in1rg)

[4.3 Design considerations 4](#_heading=h.lnxbz9)

[**Chapter 5 : Implementation 5**](#_heading=h.1ksv4uv)

[5.1 Description of how the project was executed 5](#_heading=h.44sinio)

[5.2 Challenges faced and solutions implemented 5](#_heading=h.2jxsxqh)

[**Chapter 6:Results 6**](#_heading=h.z337ya)

[6.1 outcomes 6](#_heading=h.3j2qqm3)

[6.2 Interpretation of results 6](#_heading=h.1y810tw)

[6.3 Comparison with existing literature or technologies 6](#_heading=h.2xcytpi)

[**Chapter 7: Conclusion 7**](#_heading=h.1ci93xb)

[**Chapter 8 : Future Work 8**](#_heading=h.2bn6wsx)

[Here write Suggestions for further research or development Potential improvements or extensions 8](#_heading=h.qsh70q)

[**References 9**](#_heading=h.1pxezwc)

# **Chapter 1: Introduction**

## **1.1 Overview of the problem statement**

The ever-increasing number of chip transistors and decreasing feature sizes pose manufacturing challenges, leading to defects. Testing is vital for Very-Large-Scale-Integration (VLSI designs), especially with high-capacity embedded memories on system chips. Design For Testability (DFT) techniques facilitate fault detection. One effective method for testing memory modules is using Built-In Self-Test (BIST) technology. In this work, power efficiency & high fault coverage Built-In-Self-Test (BIST) will be designed and implemented to test combinational logic. The developed technique will be tested on standard combinational circuits and yield promising results. our approach will achieve fault coverage, and a reduced power compared to conventional methods by the proposed design.

## **1.2 Objectives and goals**

**Objective**:

* Implementing BIST for combinational circuits.
* Analyse different types of LFSR for fault coverage.
* To design power efficient VLSI circuit design.

**Main Goals:**

* Bring fault reduction.
* Testing the implemented BIST for various Combinational circuits to know the accuracy of fault reduction.

**Additional Goals:**

* Trying to implement algorithms for BIST to measure accuracy.

# 

# **Chapter 2: Literature Review**

**Key Publications**

**Google scholar research best 3-5 papers**

* **Design and Implementation of BIST Architecture for low power VLSI Applications using Verilog** – 2023 - Mr. Aadesh Gonda VLSI and Embedded Systems 2111706@reva.edu.in REVA University, Bangalore

Design and Implementation of BIST Architecture for Low Power VLSI Applications using Verilog" is the title of this research effort, which tries to reduce the power consumption during the testing phase of Very Large-Scale Integration (VLSI) circuits. This is a BIST architecture that tries to balance high fault coverage with the minimum utilization of energy. Several power reduction techniques which are considered in this research are test pattern compression, selective clock gating, and power-aware test scheduling.

**Key features:**

BIST Overview This is the on-chip testing method, used in proving the quality and reliability of ICs; it's essential for efficient energy use by electronic devices.

Power Optimization The paper deals with the various low-power strategies used during testing and ensures it complies with the Verilog synthesis for practical deployment.

FPGA Implementation: The architecture is proven on an FPGA in order to test the concept of the proof of its performance.

- \*\*Memory Testing\*\*: It talks about the problems of memory testing in SoC designs and highlights the necessity of reducing power during test pattern generation.

Some of the future improvements suggested are as follows: self-repairing in memory BIST and techniques for testing in embedded DRAM.

* **BIST (Built-In Self-Test) Memory by Using VERILOG -2023** - B. Hanumanthu1, N.

Sathvika2, A. Manikanta3, A. Praveen4, Ch. Abhiram5

The research document "Design and Implementation of BIST Architecture for Low Power VLSI Applications using Verilog" aimed at lowering the power consumption in the testing phase of VLSI circuits. It has high fault coverage but very low energy usage. Some of the major power reduction techniques used here are test pattern compression, selective clock gating, and power-aware test scheduling.

Key Highlights:

BIST Overview: BIST is an on-chip testing technique that verifies the quality and reliability of integrated circuits. In addition, it is also the prerequisite for creating energy-efficient electronic devices.

Power Optimization: The paper covers several low power techniques that occur in testing and are compatible with Verilog synthesis for practical implementation.

FPGA Implementation: The architecture is tested on a field-programmable gate array for feasibility and performance.

- \*\*Memory Testing\*\*: This paper addresses the issue of memory testing in SoC designs, which deals with low power during the creation of test patterns. My paper "BIST (Built-In Self-Test) Memory by Using Verilog" provides research on the implementation of BIST architecture to perform memory testing using Verilog. BIST is a technique which allows a system the ability to test itself for any kind of errors independently. This technique is invaluable in applications where the reliability of the system under test is the most critical factor and external testing is not possible.

Key features are:

- \*\*BIST Overview\*\*: BIST architecture creates pseudo-random test patterns and compares the output with expected results to verify memory functions. The dependency on the external test equipment gets minimized and the testing speed also increases.

- \*\*Memory Testing\*\*: BIST is targeted at memory systems, whose architecture makes them complex because they are used in many critical applications. BIST makes the memory test less complicated because it only requires a minimum of additional pins and a clocking signal for the test of the whole memory IC.

\*\*Verilog Implementation\*\*: The BIST-enabled RAM is developed with Verilog so as to illustrate how hardware description languages can aid self-testing mechanisms.

- \*\*Advantages\*\*: BIST has proved to improve the fault coverage efficiently detecting various faults, such as those concerning stuck-at and transition faults, and also it saves time and cost in the testing.

Conclusion

Conclusion In this kind of research, the findings are that the BIST improves the system reliability as well as efficiency while applying it on memory testing. Also, verilog is heavily used in the design and integration of BIST into memory architecture.

Future work

The two primary enhancements deal with future work: the first one is to support self-repair functionality for memory BIST, and the other one deals with the extension of the techniques on embedded DRAM testing.

* **Design and Implementation of Power Efficient Logic BIST with High Fault Coverage Using Verilog** – 2018 - Akhila K UG Student, Department of ECE BNM Institute of Technology Bengaluru, India [akhilarao1996@gmail.com](mailto:akhilarao1996@gmail.com)

The paper "Design and Implementation of Power Efficient Logic BIST With High Fault Coverage Using Verilog" tries to design a BIST architecture for combinational logic circuits which enhance the fault coverage, and reduces their power dissipation. The design is done using a Verilog tool and validated on an FPGA platform. Major highlights encompass:

- \*\*Power Efficiency\*\*: The proposed design is able to achieve 12.25% power savings compared to the conventional design by optimizing the structure of the LFSR and other logic components.

- \*\*High Fault Coverage\*\*: The modified LFSR ensures 100% fault coverage as it will create all the possible states, thus detecting faults in the forms of stuck-at, bridging, and multiple faults.

-Simulation and Validation: The design was simulated with Xilinx ISIM simulator and verified on Spartan 6 FPGA boards to validate the architecture. Thus, it proved efficiency in both normal modes and testing modes.

This architecture further shows improvement with tremendous progress in the context of power efficiency and fault-detection capabilities.

* **VLSI Implementation of Linear Feedback Shift Register (LFSR) based Test Pattern Generator for Pseudo Exhaustive Testing** – 2020 - S. Sridhar, K. Mounika, M. Anjali, G. Venkatesh, M. Murali Krishna

This paper dwells on the use of LFSRs in the testing process and self-testing involving digital systems, with special reference to BIST techniques. The advantages of LFSRs include high speed, better encoding efficiency as well as low power consumption, which makes them suitable for test pattern generation in integrated circuits.

Main parts of the paper are:

Abstract: Provides a short description of the purpose and conclusion of the research.

Introduction: Examines the role of good test mechanisms in digital systems, along with the role of VLSI testing.

Methodology: Overview of the configuration and function of LFSRs, classification of different types with applications.

Proposed Architecture: Design and test of various modular units, such as cut cell unit, counter unit, LFSR unit.

Experimental Analysis: Presents experimental results and waveforms of the proposed design.

Conclusion: summarizes the outcome of this work and presents further improvements that can be done for the LFSR method.

**Key Resources – Whitepaper| Application Notes | Datasheet| Others**

* <https://youtu.be/IRSwVbKPrZU?si=gBUWw9370TdIGZvE> , test pattern generation.
* <https://youtu.be/dVdnFQvHJ74?si=CoWy0CKppY8QetQJ>, <https://youtu.be/sfxtYY-PpiA?si=NljJIHef9N3I3BX4> , Bist.
* <https://youtu.be/4Q8AaevnAaU?si=mpO_wImfbKWVrhYX> , <https://youtu.be/lb0lwriIMjw?si=3PL9oIa3JJXP4mYq> , fault-simulation.

**Existing Implementations – Products| Opensource| GitHub etc**

* <https://github.com/nihargowdakm/BIST-Design>

# **Chapter 3: Strategic Analysis and Problem Definition**

## 3.1 SWOT Analysis

**Strengths**

1. **High Fault Coverage**:

* The proposed BIST method, modified LFSR is designed to achieve high fault coverage. This ensures that a wide range of faults, trying to including complex linked faults, can be detected effectively.

1. **Power Efficiency**:
   * Another strength of the proposed method is that it addresses the power level efficiency .. As the proposed method increases the test patterns and decreases the power consumed in the testing, it tackles one of the most formidable problems in VLSI design testing.
2. **Integration with Existing Designs**:
   * The BIST approach can be seamlessly integrated into existing VLSI designs without requiring significant changes to the design process. This makes it a practical solution for real-world applications.
3. **Scalability**:
   * The technique is not limited to small designs or certain classes of circuits, thus the results obtained are likely to be relevant in many VLSI designs for SoC chips.

**Weaknesses**

1. **Complex Implementation**:
   * The modified LFSR may increase the level of difficulty towards the design of the BIST. Some of the major advantages of the proposed BIST can include. This could translate into longer development cycles and greater time for the engineers to train while on the job.
2. **Initial Cost**:
   * The proposed BIST technique is likely to, require higher initial costs essentially during the design and verification phase as is the case with most complex designs. For some small firms or projects, or those with limited funds, this might prove to be a limitation.
3. **Resource Consumption**:
   * Although power efficiency defines a strength, the method can be rather time and computationally intensive during testing of quite large or heavily complex circuits. This could affect the total system performance during the testing phase of the systems integration Lifecyle.

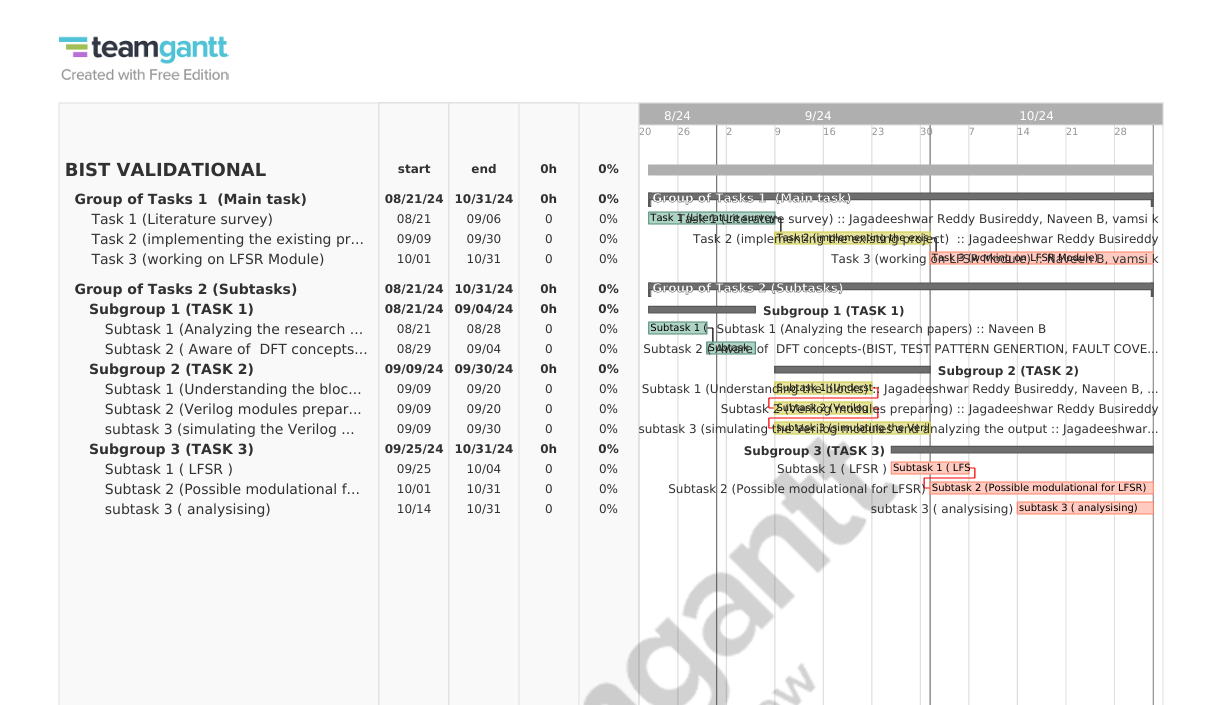
**Opportunities**

1. **Growing Demand for Reliable VLSI Testing**:
   * As VLSI designs become more complex and are increasingly used in critical applications (e.g., automotive, healthcare, aerospace), there is a growing demand for reliable and efficient testing methods. The proposed BIST method is well-positioned to meet this demand.
2. **Advancements in VLSI Technology**:
   * Ongoing advancements in VLSI technology, such as the development of new semiconductor materials and manufacturing processes, provide opportunities to further enhance the BIST method, potentially improving its efficiency and effectiveness.
3. **Adoption in Emerging Markets**:
   * Emerging markets, particularly in regions investing heavily in semiconductor manufacturing, represent a significant opportunity for the adoption of advanced testing techniques like the proposed BIST method.
4. **Collaboration with Industry Leaders**:
   * Collaborating with industry leaders and academic institutions could lead to further refinements of the method and its wider adoption in the industry.

**Threats**

1. **Competition from Established Methods**:
   * There are already well-established BIST methods and algorithms in the industry. The proposed method may face competition from these existing solutions, especially if they are perceived as more mature or easier to implement.
2. **Technological Obsolescence**:
   * Rapid technological advancements could render the proposed BIST method obsolete. For example, the development of new testing technologies or methodologies could outpace the adoption of this approach.
3. **Regulatory and Standardization Challenges**:
   * Changes in industry standards or regulatory requirements could pose a threat to the adoption of the proposed method. Ensuring compliance with evolving standards is critical for its success.
4. **Economic Factors**:
   * Economic downturns or budget constraints in the semiconductor industry could impact the adoption of new testing technologies, including the proposed BIST method.

### 3.2 Project Plan - GANTT Chart

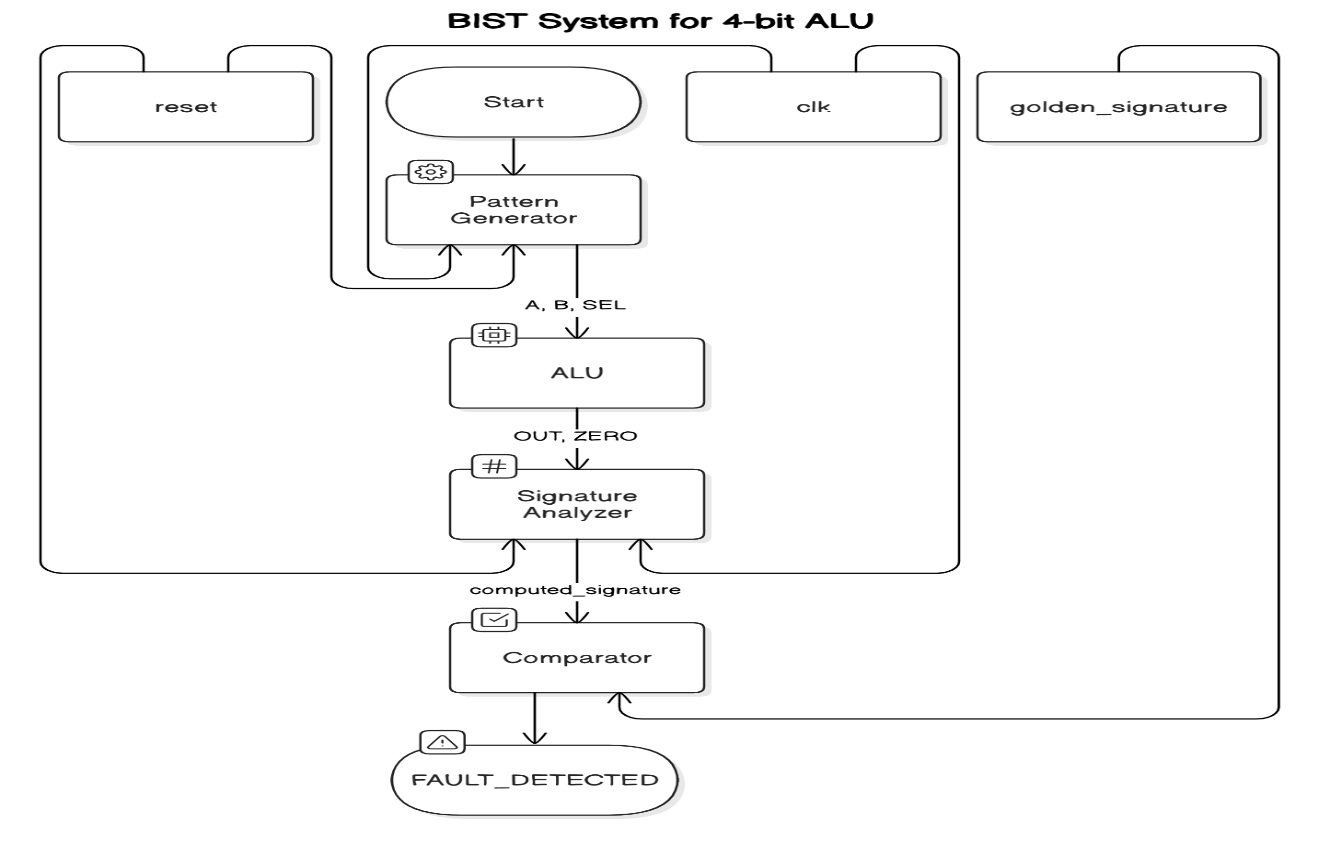


# **Chapter 4: Methodology**

## 4.1 Description of the approach

A diagram of a process

Description automatically generated



### 4.2 Tools and techniques utilized

* XILINX VIVADO 2024.1
* MODELSIM
* VERILOG HDL

#### 4.3 Design considerations

Design Aspects while Implementing BIST

1. Component Selection:

LFSR (Linear Feedback Shift Register): Good LFSR configuration should be selected that will be capable enough to produce a wide number of test patterns, so that it covers almost all possible faults.

CUT (Circuit Under Test): Circuit or module to be tested should be clearly defined; so that it can interface correctly with the components of BIST.

MISR (Multiple Input Signature Register): Design or select a good MISR configuration so that it can correctly compress the test responses into a signature.

2. Power and Resource Management:

Power Consumption: The BIST additional logic LFSR and MISR should not have any impact on the overall power consumption of the circuit due to it.

Resource Usage: The BIST logic must be optimized such that the usage of ancillary gates and interconnects is kept minimum to have least impact on the area of the circuit.

3. Signal Integrity:

Clock Management : The CUT should be properly clocked and synchronized with the BIST components so that no timing violations are encountered.

Noise and Interference: The BIST should be designed with minimal noise and interference that will disturb the normal functioning of CUT.

4. Test Coverage:

Fault Models: What faults should be detected, for instance, stuck-at faults or transition faults, and that can be detected by the BIST should be determined.

Pattern Generation: The LFSR should be designed to provide adequate unique test patterns so that all possible fault scenarios are covered.

Signature Analysis: Ensure that the MISR is able to generate and compare signatures reliably enough to detect changes.

5. Timing and Performance:

Test Time: Optimize the BIST sequence for the minimum amount of test time without sacrificing fault coverage.

Performance Impact: Ensure that the incorporation of BIST logic does not adversely impact the CUT's performance in nominal mode.

6. Design Integration:

Modularity: Design BIST components as modular blocks that can be easily fitted into different types of circuits.

Interface: Ensure that BIST components interface well with the CUT by clearly defining the signals and their timings.

7. Verification and Validation

Simulation: The BIST operation has to be simulated in great detail so that verification can be made that test patterns generated are indeed correct and that faults are detected as expected.

Hardware Testing: Where it is possible, the BIST shall be shown using real hardware and that it also works in real conditions.

Fault Injection Testing the BIST shall be tested under fault injection conditions to ensure that the range of fault conditions covered by the BIST is identified correctly.

These considerations will ensure that your BIST design works effectively, efficiently, and reliably. They give comprehensive guidelines for designing and integrating a robust BIST system into your digital circuit. Need help exploring any specific part?

# **Chapter 5: Implementation**

## 5.1 Description of how the project was executed

1**.** Planning and Initial Setup

Objective Setting: The primary objective was to design a Built-In Self-Test (BIST) system for a digital circuit, specifically a combinational circuit (full adder).

Research: Gathered necessary information on BIST, LFSR, CUT, and MISR.

2. Design Phase

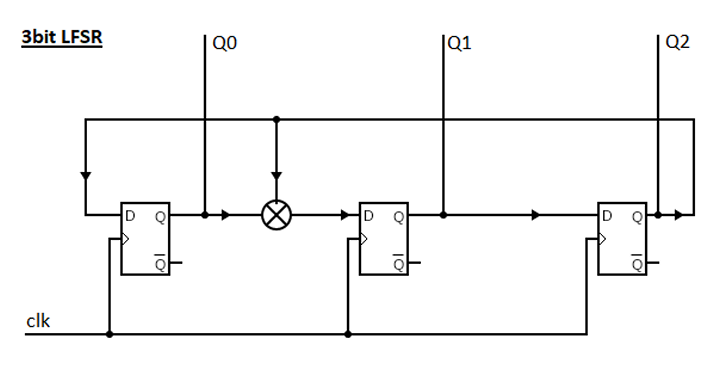
LFSR Design:

Implemented the Linear Feedback Shift Register (LFSR) for generating pseudo-random test patterns. The LFSR was designed to shift and combine bits to produce a unique sequence of binary values.

Polynomial Equation : 1 + x + x 3

A screenshot of a computer

Description automatically generated



The LFSR has been seeded to 001. The LFSR has period of 7 (= **23**-1)

**Next State Equations:**

Q0\* = Q2

Q1\* = Q0 + Q2

Q2\* = Q1

**Test Patterns:**

**A table with numbers and text

Description automatically generated**

**TPG for ALU:**

**Polynomials Derived from Feedback Logic**

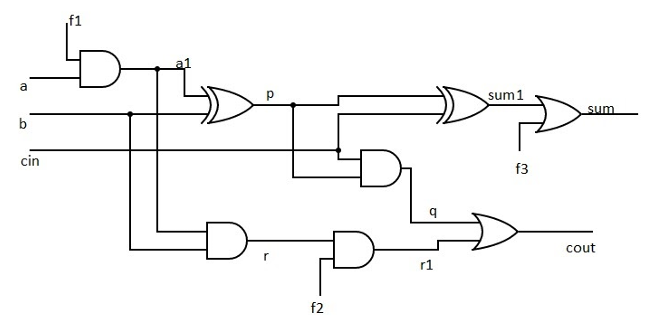
1. **For** lfsr\_a **(4-bit LFSR):**
   * The feedback logic is lfsr\_a[3] ^ lfsr\_a[2].
   * This corresponds to the polynomial: P(x)=x4+x3+1P(x) = x^4 + x^3 + 1.
2. **For** lfsr\_b **(4-bit LFSR):**
   * The feedback logic is lfsr\_b[3] ^ lfsr\_b[1].
   * This corresponds to the polynomial: P(x)=x4+x2+1P(x) = x^4 + x^2 + 1.
3. **For** lfsr\_sel **(3-bit LFSR):**
   * The feedback logic is lfsr\_sel[2] ^ lfsr\_sel[0].
   * This corresponds to the polynomial: P(x)=x3+x+1P(x) = x^3 + x + 1.

A screenshot of a computer

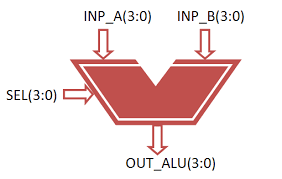
AI-generated content may be incorrect.

**CUT Integration:**

Integrated the Circuit Under Test (CUT), which is the full adder. The CUT was designed to receive inputs from either the LFSR (in test mode) or normal inputs. The CUT is a 1- bit full adder. Faults have been injected at 3 points in the design i.e. a@0, sum@1, r@0. The response from the MISR is compared with the golden signature to check whether the fault is detected or goes undetected.



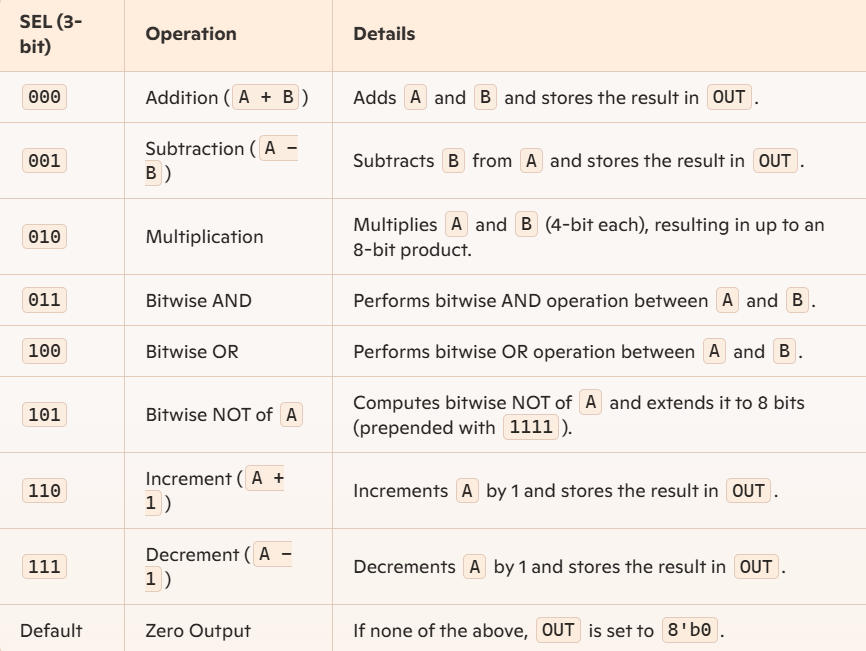
**CUT AS ALU:**



**Operations Supported**

The ALU processes two 4-bit inputs (A and B) and produces an 8-bit result (OUT). Additionally, it sets the ZERO flag if the output is zero. Below is a breakdown of the operations based on SEL:

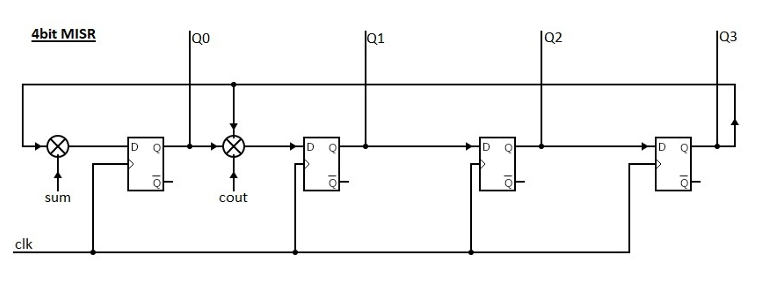
* + Compares OUT with zero and sets the ZERO flag.



**ZERO Flag:** The ZERO output is asserted (1) whenever the result in OUT is zero.

**MISR Implementation:**

Designed the Multiple Input Signature Register (MISR) to collect the outputs of the CUT and compress them into a signature that can be compared against a known good signature (golden signature).



**SIGNALANALYZER FOR ALU:**

The feedback logic used in this design corresponds to a polynomial that defines the LFSR's behavior. From the XOR feedback in the code:

signature\_next = signature[7] ^ OUT[0] ^ OUT[1] ^ OUT[2] ^ OUT[3] ^ ZERO

* This feedback is applied to form the least significant bit (LSB) of the updated signature.
* Polynomial Representation: Assuming the bits of the signature represent the coefficients of the polynomial, your LFSR is governed by a custom polynomial:

P(x)=x8+x7+D(x)P(x) = x^8 + x^7 + D(x)

Where D(x)D(x) depends on the combination of OUT bits and ZERO. This is not a standard LFSR polynomial but serves the purpose of compressing data from OUT and ZERO.

| **signature[7]** | **OUT[0]** | **OUT[1]** | **OUT[2]** | **OUT[3]** | **ZERO** | **signature\_next** |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

**Coding and Simulation**

* **Verilog Coding for (FULLADDER):**

Wrote the Verilog code for each module, ensuring proper interfacing and communication between LFSR, CUT, and MISR.

//////////////////////////////////////////////////////////////////////////////////

module patterngenerator (

input clk,

input reset,

output reg [3:0] A,

output reg [3:0] B,

output reg [2:0] SEL

);

// Internal LFSR registers

reg [3:0] lfsr\_a;

reg [3:0] lfsr\_b;

reg [2:0] lfsr\_sel;

always @(posedge clk or posedge reset) begin

if (reset) begin

lfsr\_a <= 4'b0001;

lfsr\_b <= 4'b0010;

lfsr\_sel <= 3'b001;

end else begin

// LFSR logic for pseudo-random pattern generation

lfsr\_a <= {lfsr\_a[2:0], lfsr\_a[3] ^ lfsr\_a[2]};

lfsr\_b <= {lfsr\_b[2:0], lfsr\_b[3] ^ lfsr\_b[1]};

lfsr\_sel <= {lfsr\_sel[1:0], lfsr\_sel[2] ^ lfsr\_sel[0]};

end

end

// Assign outputs

always @(\*) begin

A = lfsr\_a;

B = lfsr\_b;

SEL = lfsr\_sel;

end

endmodule

//////////////////////////////////////////////////////////////////////////////////

module ALU (

input [3:0] A,

input [3:0] B,

input [2:0] SEL,

output reg [7:0] OUT,

output reg ZERO

);

always @(\*) begin

case (SEL)

3'b000: OUT = A + B;

3'b001: OUT = A - B;

3'b010: OUT = A \* B;

3'b011: OUT = A & B;

3'b100: OUT = A | B;

3'b101: OUT = ~A;

3'b110: OUT = A + 1;

3'b111: OUT = A - 1;

default: OUT = 8'b0; // Default case

endcase

ZERO = (OUT == 8'b0);

end

endmodule

//////////////////////////////////////////////////////

module signalanalyzer (

input clk,

input reset,

input [7:0] OUT,

input ZERO,

output reg [7:0] signature

);

always @(posedge clk or posedge reset) begin

if (reset) begin

signature <= 8'b0;

end else begin

// XOR-based LFSR for signature compression

signature <= {signature[6:0], signature[7] ^ OUT[0] ^ OUT[3] ^ ZERO};

end

end

endmodule

//////////////////////////////////////////////////////

module comparator (

input [7:0] computed\_signature,

input [7:0] golden\_signature,

output reg FAULT\_DETECTED

);

always @(\*) begin

if (computed\_signature == golden\_signature) begin

FAULT\_DETECTED = 0; // No fault

end else begin

FAULT\_DETECTED = 1; // Fault detected

end

end

endmodule

////////////////////////////////////////////////////////

**Testbench Development:**

Created a comprehensive testbench to simulate the behavior of the BIST system. Included multiple scenarios to validate the functionality of the system.

////

module TB\_ALU;

reg [3:0] A, B;

reg [2:0] SEL;

wire [7:0] OUT;

wire ZERO;

// Instantiate the ALU

ALU uut (

.A(A),

.B(B),

.SEL(SEL),

.OUT(OUT),

.ZERO(ZERO)

);

initial begin

$monitor("A=%b, B=%b, SEL=%b, OUT=%b, ZERO=%b", A, B, SEL, OUT, ZERO);

// Test Addition

A = 4'b0011; B = 4'b0001; SEL = 3'b000; #10;

// Test Subtraction

A = 4'b0101; B = 4'b0010; SEL = 3'b001; #10;

// Test Multiplication

A = 4'b0011; B = 4'b0010; SEL = 3'b010; #10;

// Test AND

A = 4'b1100; B = 4'b1010; SEL = 3'b011; #10;

// Test OR

A = 4'b1100; B = 4'b1010; SEL = 3'b100; #10;

// Test NOT

A = 4'b1100; SEL = 3'b101; #10;

// Test Increment

A = 4'b0111; SEL = 3'b110; #10;

// Test Decrement

A = 4'b1000; SEL = 3'b111; #10;

$finish;

end

endmodule

//

module TB\_BIST\_ALU();

reg clk;

reg reset;

wire FAULT\_DETECTED;

// Clock generation

initial begin

clk = 0;

forever #5 clk = ~clk;

end

// Instantiate BIST\_ALU module

BIST\_ALU uut (

.clk(clk),

.reset(reset),

.FAULT\_DETECTED(FAULT\_DETECTED)

);

initial begin

reset = 1;

#10 reset = 0;

#1000;

$display("Fault detected status: %b", FAULT\_DETECTED);

$stop;

end

endmodule

**Simulation and Debugging:**

Ran simulations using a hardware description language simulator. Debugged and fixed any issues found during simulations to ensure the system operated correctly under different conditions.

**Validation and Testing:**

Functional Testing: Verified the system functionality in both normal and test modes. Ensured that the LFSR generated correct test patterns, the CUT performed accurate computations, and the MISR correctly compressed the outputs.

**Fault Injection:**

The CUT is a 1- bit full adder. Faults have been injected at 3 points in the design i.e. a@0, sum@1, r@0. The response from the MISR is compared with the golden signature to check whether the fault is detected or goes undetected.

Introduced faults intentionally to test the fault detection capability of the BIST. Confirmed that the system could detect discrepancies and raise fault flags as expected.

**Verilog Coding for (ALU):**

///TPG

module patterngenerator(

input clk,

input reset,

output reg [3:0] A,

output reg [3:0] B,

output reg [2:0] SEL

);

// Internal LFSR registers

reg [3:0] lfsr\_a;

reg [3:0] lfsr\_b;

reg [2:0] lfsr\_sel;

always @(posedge clk or posedge reset) begin

if (reset) begin

lfsr\_a <= 4'b0001;

lfsr\_b <= 4'b0010;

lfsr\_sel <= 3'b001;

end else begin

// LFSR logic for pseudo-random pattern generation

lfsr\_a <= {lfsr\_a[2:0], lfsr\_a[3] ^ lfsr\_a[2]};

lfsr\_b <= {lfsr\_b[2:0], lfsr\_b[3] ^ lfsr\_b[1]};

lfsr\_sel <= {lfsr\_sel[1:0], lfsr\_sel[2] ^ lfsr\_sel[0]};

end

end

// Assign outputs

always @(\*) begin

A = lfsr\_a;

B = lfsr\_b;

SEL = lfsr\_sel;

end

endmodule

//CUT

module ALU(

input [3:0] A,

input [3:0] B,

input [2:0] SEL,

output reg [7:0] OUT,

output reg ZERO

);

always@(\*) begin

case(SEL)

3'b000: OUT = A + B;

3'b001: OUT = A - B;

3'b010: OUT = A \* B;

3'b011: OUT = A & B;

3'b100: OUT = A | B;

3'b101: OUT = {4'b1111, ~A}; // Extend 4-bit ~A to 8 bits with 1s (if signed)

3'b110: OUT = A + 1;

3'b111: OUT = A - 1;

default: OUT = 8'b0; //default case

endcase

ZERO = (OUT == 8'b0);

end

endmodule

//MISR

module signalanalyzer( //signature analyzer

input clk,

input reset,

input [7:0] OUT,

input ZERO,

output reg [7:0] signature

);

always @(posedge clk or posedge reset) begin

if (reset) begin

signature <= 8'b0;

end else begin

// XOR-based LFSR for signature compression

signature <= {signature[6:0], signature[7] ^ OUT[0] ^ OUT[1] ^ OUT[2] ^ OUT[3] ^ ZERO};

end

end

endmodule

//COMPARATOR

module comparator(

input [7:0] computed\_signature,

input [7:0] golden\_signature,

output reg FAULT\_DETECTED

);

always @(\*) begin

// Fault detection logic

if (computed\_signature == golden\_signature) begin

FAULT\_DETECTED = 0; // No fault

end else begin

FAULT\_DETECTED = 1; // Fault detected

end

end

endmodule

//TOP MODULE

module BIST\_ALU(

input clk,

input reset,

output FAULT\_DETECTED

);

wire [3:0] A, B;

wire [2:0] SEL;

wire [7:0] OUT;

wire ZERO;

wire [7:0] computed\_signature;

parameter [7:0] golden\_signature = 8'hA5;

// Instantiate PatternGenerator

patterngenerator Pattern\_Generator (

.clk(clk),

.reset(reset),

.A(A),

.B(B),

.SEL(SEL)

);

// Instantiate ALU

ALU alu (

.A(A),

.B(B),

.SEL(SEL),

.OUT(OUT),

.ZERO(ZERO)

);

// Instantiate SignalAnalyzer

signalanalyzer Signature\_Analyzer (

.clk(clk),

.reset(reset),

.OUT(OUT),

.ZERO(ZERO),

.signature(computed\_signature)

);

// Instantiate Comparator

comparator Comparator (

.computed\_signature(computed\_signature),

.golden\_signature(golden\_signature),

.FAULT\_DETECTED(FAULT\_DETECTED)

);

endmodule

//testbench for top module

module TB\_BIST\_ALU();

// Testbench signals

reg clk;

reg reset;

wire FAULT\_DETECTED;

// Clock generation: 10ns period

initial begin

clk = 0;

forever #5 clk = ~clk; // Toggle clock every 5ns

end

// Instantiate the BIST\_ALU module

BIST\_ALU uut (

.clk(clk),

.reset(reset),

.FAULT\_DETECTED(FAULT\_DETECTED)

);

// Simulation setup

initial begin

// Initialize reset and simulate

reset = 1;

#10; // Hold reset high for 10ns

reset = 0; // Release reset

#2000; // Simulate for 1000ns

// Check for fault detection manually

$display("Fault detected status: %b", FAULT\_DETECTED);

#100; // Extra time for observation

$stop; // End simulation

end

endmodule

**Result Analysis**:

Analyzed simulation waveforms to ensure correct operation and accurate fault detection.

### 5.2 Challenges faced and solutions implemented

**1. Design Complexity**

Challenge:

Integrating multiple modules (LFSR, CUT, MISR) with different functionalities into a cohesive BIST system was complex.

Solution: Broke down the project into smaller, manageable tasks. Focused on designing and testing each module individually before integrating them. Modular design helped in isolating and fixing issues quickly.

**2. Fault Detection Accuracy**

Challenge:

Ensuring the BIST system accurately detects faults without false positives or negatives.

Solution:

Implemented comprehensive testing, including fault injection to simulate different types of faults. Validated the accuracy of the golden signature and MISR logic to ensure reliable fault detection.

**3. Signal Synchronization**

Challenge:

Synchronizing various signals across the modules, particularly with different clock cycles and reset conditions.

Solution:

Carefully designed the state machines and control logic to handle synchronization. Used flip-flops and clock gating techniques to ensure all modules operate in harmony.

**4. Debugging and Simulation**

Challenge:

Debugging the Verilog code during simulation was time-consuming, especially for subtle timing issues.

Solution:

Utilized waveforms extensively to visualize signal transitions and states. Employed a systematic approach to debug by isolating sections of code and validating each part incrementally.

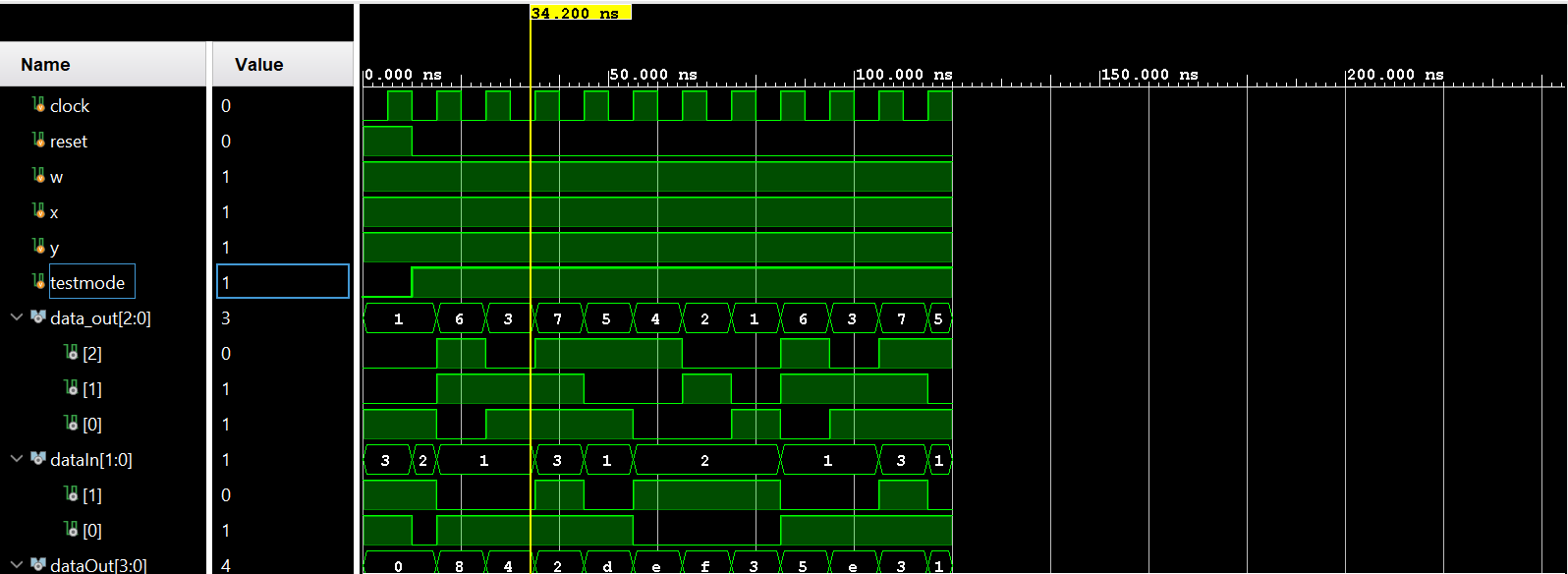
**Outcome:**

These challenges required a blend of strategic planning, methodical testing, and continuous learning. Each challenge encountered was a valuable learning experience, contributing to the successful execution and completion of the BIST project.

# **Chapter 6: Results**

## 6.1 outcome

**BIST ARCHITECTURE Waveform (FULLADDER):**

****

**A screenshot of a video game

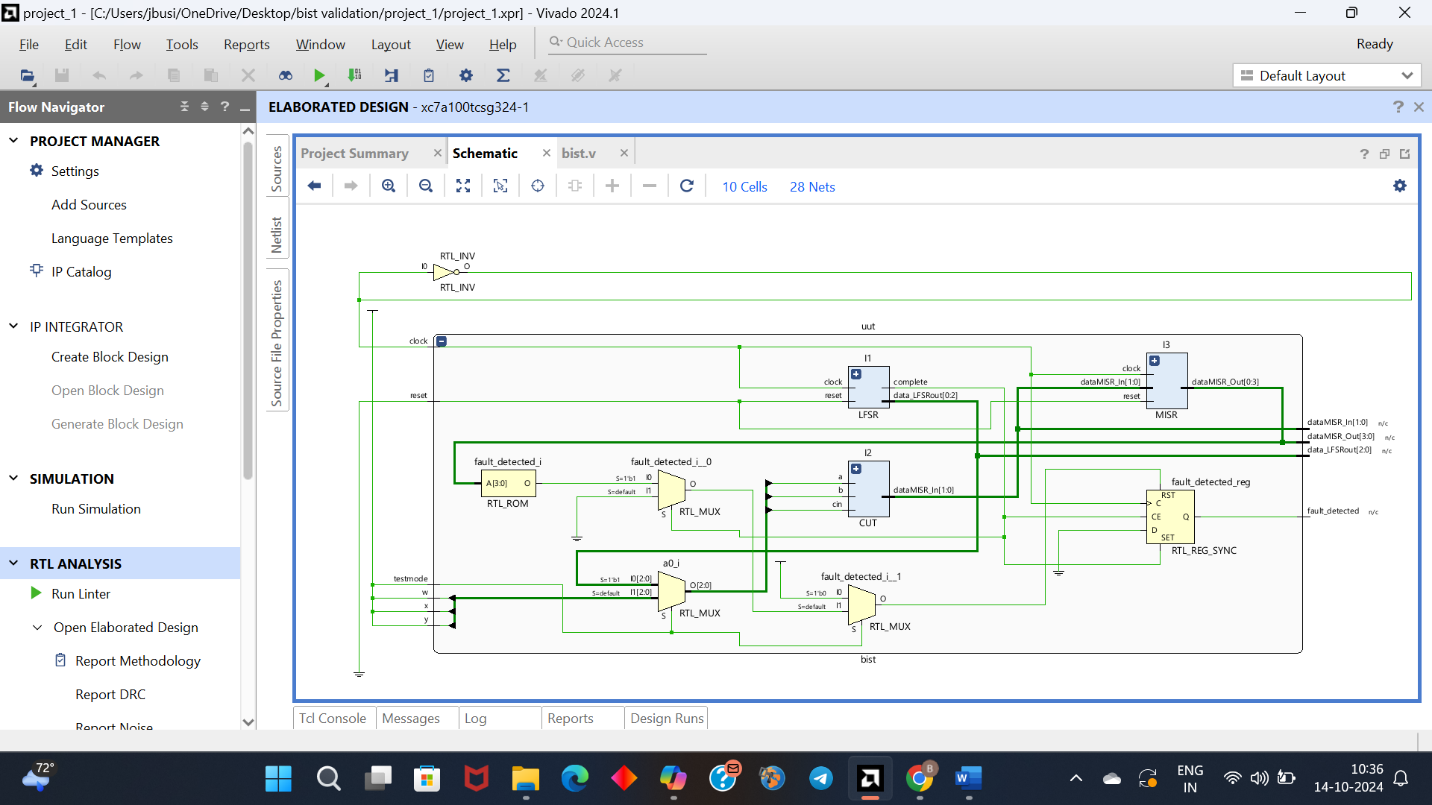
Description automatically generated**

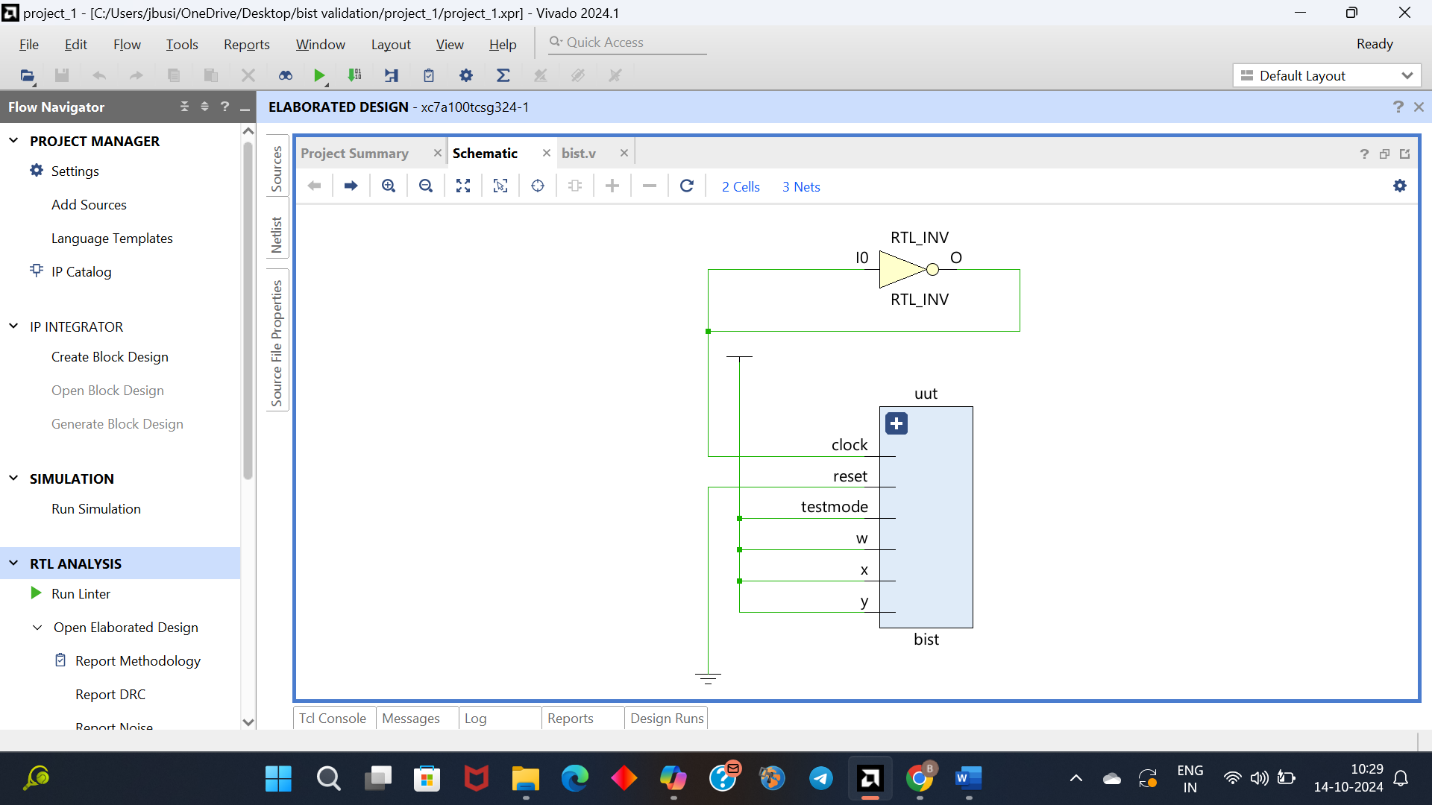
**FULLADDER WAVEFORM:**

**A screenshot of a computer

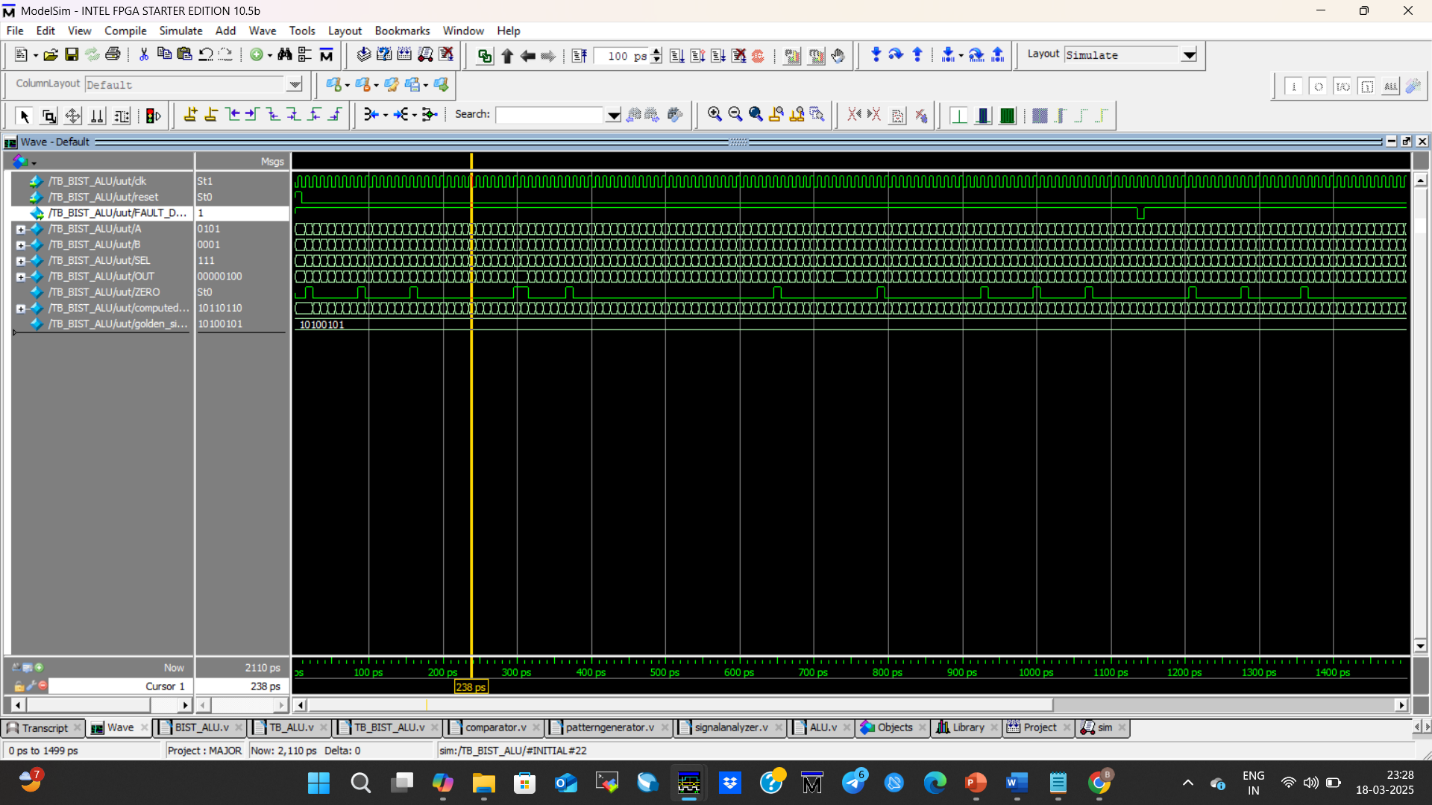
Description automatically generated**

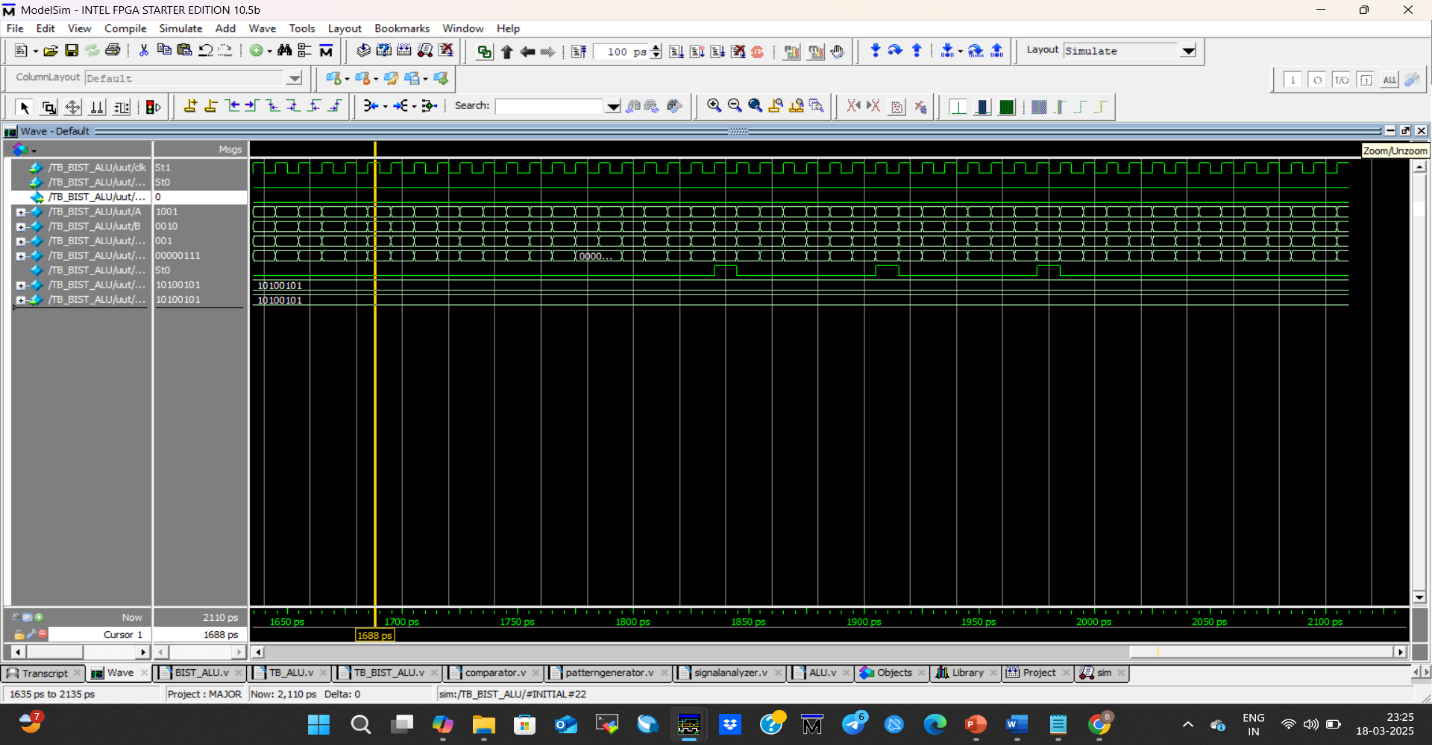
**RTL Design:**

****

****

**BIST FOR ALU:**

****

****

### A diagram of a computer program6.2 Interpretation of results

 **Test Pattern Generation**:

* A Linear Feedback Shift Register (LFSR) or a similar Test Pattern Generator (TPG) generates pseudo-random input patterns.
* These patterns are applied to the inputs of the ALU, testing its logic extensively.

 **Response Generation**:

* The ALU processes the input patterns and produces outputs according to its implemented logic.

 **Output Capturing**:

* The outputs of the ALU are collected by a **Response Analyzer**, such as a Multiple Input Signature Register (MISR).
* This analyzer compresses the output data into a unique signature (a hash-like value).

 **Expected Signature Comparison**:

* A "golden" or reference signature is precomputed for the ALU when it is functioning without faults. This is stored in the BIST system.
* During testing, the generated signature from the MISR is compared to this stored golden signature.

 **Fault Detection**:

* If the generated signature matches the golden signature, the ALU is considered fault-free.
* If there’s a mismatch, it indicates a fault in the ALU.

 **Fault Localization (Optional)**:

* In advanced BIST implementations, the system may attempt to locate which specific area or operation in the ALU caused the fault.

#### 6.3 Comparison with existing literature or technologies

**1. Design for Testability (DFT)**

* **DFT**: Focuses on incorporating test features during the design process to support efficient testing
* **BIST**: Adds self-testing logic to the circuit, enabling periodic self-tests without external equipment2.
* **Comparison**: While DFT prepares the design for easier testing, BIST provides an autonomous testing mechanism, reducing the need for external testing hardware.

**2. Automatic Test Pattern Generation (ATPG)**

* **ATPG**: Generates test patterns to detect faults in digital circuits1.
* **BIST**: Uses LFSR to generate pseudo-random test patterns, which are more efficient and repeatable3.
* **Comparison**: ATPG requires external tools to generate and apply test patterns, whereas BIST integrates the pattern generation within the circuit itself, making it more efficient for regular testing.

**3. Memory BIST (MBIST)**

* **MBIST**: Specialized BIST for memory testing, generating patterns to test memory cells2.
* **Logic BIST**: Focuses on testing combinational and sequential logic circuits2.
* **Comparison**: MBIST is tailored for memory testing, while Logic BIST is used for general logic circuits. Both are subsets of BIST, each addressing specific testing needs.

**4. Hardware Diffusers**

* **Hardware Diffusers**: Generate pattern clusters around stored patterns in ROM for testing3.
* **BIST**: Uses LFSR for generating pseudo-random patterns, which are algorithmically produced and repeatable3.
* **Comparison**: Hardware diffusers rely on stored patterns, while BIST's LFSR approach is more flexible and scalable for various test scenarios.

**5. Binary Counters**

* **Binary Counters**: Generate test patterns by counting through binary sequences3.
* **BIST**: Uses LFSR for generating pseudo-random patterns, which are more efficient and cover a broader range of input scenarios3.
* **Comparison**: Binary counters are simpler but less efficient compared to the LFSR-based approach in BIST.

In summary, BIST offers a more integrated and efficient approach to testing compared to traditional methods like DFT, ATPG, and binary counters. It provides autonomous testing capabilities, making it suitable for regular and reliable fault detection in digital circuits.

# **Chapter 7: Conclusion**

**Suggestions for Further Research or Development**

1. **Enhanced Fault Coverage**: Investigate methods to increase fault coverage using advanced algorithms for test pattern generation and fault simulation.
2. **Adaptive Testing**: Develop adaptive testing techniques that dynamically adjust test patterns based on detected fault types to improve efficiency and accuracy.
3. **Integration with AI**: Explore integrating AI and machine learning for intelligent fault prediction and diagnosis, leveraging historical data to identify potential issues before they occur.
4. **Scalability**: Research ways to scale the BIST system to more complex circuits, including multi-core processors and large-scale integration systems.
5. **Energy Efficiency**: Study energy-efficient BIST designs to minimize power consumption during testing, especially for portable and battery-operated devices.
6. **Real-Time Monitoring**: Implement real-time monitoring capabilities for continuous assessment and immediate feedback on circuit health during operation.

**Potential Improvements or Extensions**

1. **Optimized LFSR Sequences**: Fine-tune the LFSR sequences for more efficient and faster test pattern generation, reducing test time.
2. **Robustness Against Variability**: Enhance the BIST system to be more robust against environmental changes such as temperature and voltage variations.
3. **Multi-Mode Testing**: Introduce multi-mode testing capabilities that can switch between different types of tests based on the operational context of the circuit.
4. **Comprehensive Reporting**: Develop detailed reporting mechanisms that provide insights into test coverage, detected faults, and potential reliability issues.
5. **User Interface**: Create a user-friendly interface for configuring and running BIST, allowing easier integration and usage by engineers.
6. **Advanced Fault Models**: Implement advanced fault models beyond stuck-at faults, such as bridging faults and delay faults, to enhance the comprehensiveness of the testing.

These suggestions and potential improvements aim to push the boundaries of current BIST technology, ensuring more reliable and efficient testing for a wide range of digital circuits and systems.

# **Chapter 8: Future Work**

#### Here write Suggestions for further research or development Potential improvements or extensions​

​

​

**Future Work**​

* We are going to work on different digital circuits.​
* We will try to implement the algorithms for circuits.​
* We will use different equation for LFSR.
* We will try to do research on technology further.

# References

* **Design and Implementation of BIST Architecture for low power VLSI Applications using Verilog** – 2023 - Mr. Aadesh Gonda VLSI and Embedded Systems 2111706@reva.edu.in REVA University, Bangalore
* **Design and Implementation of Power Efficient Logic BIST with High Fault Coverage Using Verilog** – 2018 - Akhila K UG Student, Department of ECE BNM Institute of Technology Bengaluru, India [akhilarao1996@gmail.com](mailto:akhilarao1996@gmail.com)
* **BIST (Built-In Self-Test) Memory by Using VERILOG -2023** - B. Hanumanthu1, N. Sathvika2, A. Manikanta3, A. Praveen4, Ch. Abhiram5
* **VLSI Implementation of Linear Feedback Shift Register (LFSR) based Test Pattern Generator for Pseudo Exhaustive Testing** – 2020 - S. Sridhar, K. Mounika, M. Anjali, G. Venkatesh, M. Murali Krishna

-