Design and Implementation of BIST Embedded-SPI Protocol

BECE303L VLSI Design Systems

By

Danus.D- 22BEC1160

Pradesh Kumar.M - 22BEC1478

M.Hariharan - 22BEC1126

Submitted to

Dr. VELMATHI.G

VIT®
CHENNAI
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF ELECTRONICS ENGINEERING

VELLORE INSTITUTE OF TECHNOLOGY

CHENNAI - 600127

November 2024

Certificate

This is to certify that the Project work titled is being submitted by Danus. D - 22BEC1160, Pradesh Kumar - 22BEC1478 and M.Hariharan - 22BEC1126 for the course BECE303L VLSI Design Systems is a record of Bonafide work done under my guidance. The contents of this project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University.

Dr. Velmathi.G
Professor

ABSTRACT

The Built-In Self-Test (BIST) is a testing methodology that allows electronic systems to autonomously verify their own functionality, ensuring reliability and fault detection without external testing equipment. This report presents the design and implementation of a BIST-embedded Serial Peripheral Interface (SPI) protocol system aimed at detecting potential issues in data communication. The system is developed with core components like a Test Pattern Generator (TPG), Circuit Under Test (CUT), Output Response Analyzer (ORA), and a Controller, all integrated to facilitate autonomous testing within the SPI framework. The TPG generates pseudo-random test patterns, the CUT processes these patterns, and the ORA evaluates the results to determine if they match the expected output. By utilizing SPI, the system benefits from high-speed, full-duplex communication, enhancing the overall test efficiency and reliability. The proposed BIST embedded SPI system effectively reduces testing time, improves test coverage, and enables efficient fault diagnosis, making it suitable for modern embedded systems requiring dependable communication.

ACKNOWLEDGEMENT

We wish to express our sincere thanks and deep sense of gratitude to our project guide, Dr. Velmathi.G, School of Electronics Engineering, for his consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We are extremely grateful to Dr. A. Ravi Sankar, Dean of School of Electronics Engineering, VIT Chennai, for extending the facilities of the school towards our project and for his unstinting support.

We express our thanks to our Head of the Department Dr. K. Mohanaprasad for his support throughout the course of this project.

We also take this opportunity to thank all the faculty of the school for their support and their wisdom imparted to us throughout the course.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

Table of contents

List of figures

Introduction

In modern embedded systems, reliable communication protocols are essential to ensure data integrity and accurate operation. The Serial Peripheral Interface (SPI) protocol is widely used in embedded systems due to its high speed, simplicity, and full-duplex communication capability. However, ensuring the reliability of SPI communication requires thorough testing to identify and correct faults. Built-In Self-Test (BIST) techniques provide a solution by embedding testing capabilities directly within the system, enabling autonomous fault detection without the need for external testing equipment.

This project explores the design and implementation of a BIST-embedded SPI protocol system, focusing on enhancing the testability of SPI communication. The system consists of key components: the Test Pattern Generator (TPG), Circuit Under Test (CUT), Output Response Analyzer (ORA), and a Controller. Each component plays a crucial role in generating test patterns, processing and analyzing responses, and managing the testing sequence. The TPG generates pseudo-random sequences to stimulate the CUT, which simulates the SPI slave operations. The ORA, using a Multiple Input Shift Register (MISR), compresses the responses into a compact signature that is then compared with a predefined "golden" signature. This comparison helps in detecting faults in data transfer and processing. By implementing this BIST framework in an SPI system, the design achieves enhanced fault coverage, reduced testing time, and improved reliability, which are critical for high-performance embedded systems.

Implementation

Algorithm

BIST Algorithm Flow Description:

1. Test Pattern Generation (TPG):
    - The Test Pattern Generator (TPG) is responsible for creating a sequence of test patterns that are used to exercise the Circuit Under Test (CUT). In this design, the TPG can be implemented using an LFSR (Linear Feedback Shift Register), which generates pseudo-random patterns efficiently.
    - The generated test patterns are applied to the input of the CUT, ensuring that different logic paths within the circuit are activated to expose any potential faults.
2. Controller:
    - The controller manages the entire BIST operation, orchestrating the sequence of events. It synchronizes the test pattern generation, the application of the patterns to the CUT, and the collection of the output response.
    - It monitors the test process and determines when the testing should start and stop, providing control signals to the TPG and Output Response Analyzer (ORA).

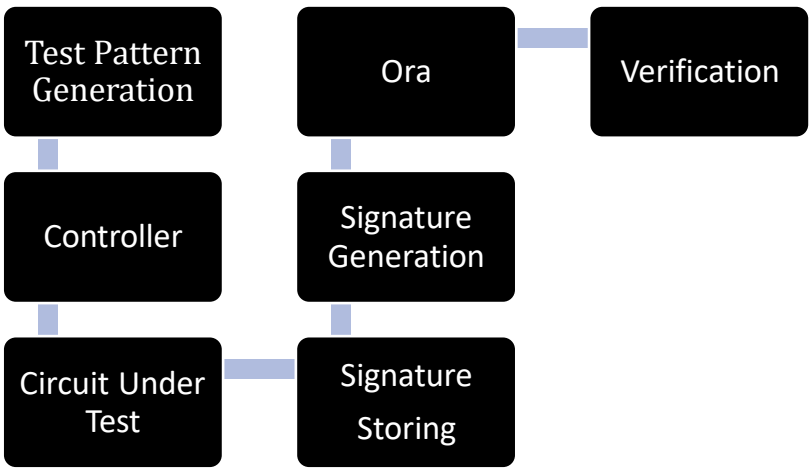
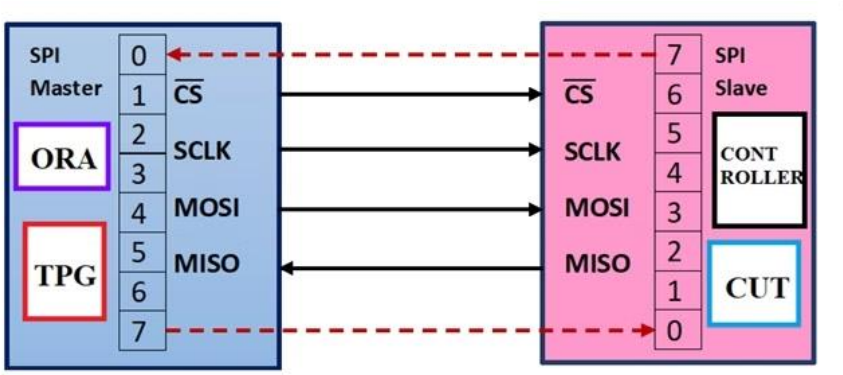


Fig.2.1

3. Circuit Under Test (CUT):
    - The CUT is the portion of the circuit being tested for faults. It receives the test patterns generated by the TPG and produces corresponding output responses.
    - The goal of the BIST process is to validate the integrity of the CUT by comparing its output responses with expected values. Any discrepancies may indicate a fault within the CUT.
4. Output Response Analyzer (ORA):
    - The ORA is responsible for analyzing the output responses from the CUT. It receives the responses and compresses them into a signature using a Multiple Input Signature Register (MISR).
    - The ORA uses XOR logic and shift registers to generate a compact signature that uniquely represents the sequence of output responses. The generated signature can be compared with a known golden signature to detect faults.
5. Signature Generation:
    - In this step, the ORA processes the output responses and computes a 4-bit or n-bit signature using the MISR. The MISR combines the responses over several clock cycles, applying XOR feedback as per the defined polynomial.
    - The final signature is a compressed representation of the output responses, capturing the essence of the CUT's behavior under test conditions.
6. Signature Storing:
    - Once the signature is generated, it is stored in a dedicated register or memory location. This stored signature serves as a reference for later verification.
    - The signature is retained for comparison with a pre-determined golden signature, which represents the expected output of a fault-free CUT.
7. Verification:
    - In the verification phase, the stored signature is compared with the golden signature. If the signatures match, the CUT is considered fault-free, indicating successful testing.
    - If there is a mismatch between the generated signature and the golden signature, it suggests the presence of a fault in the CUT. The BIST system then flags an error, and the faulty component can be isolated for further analysis.

SPI With BIST:



Circuit Diagram:

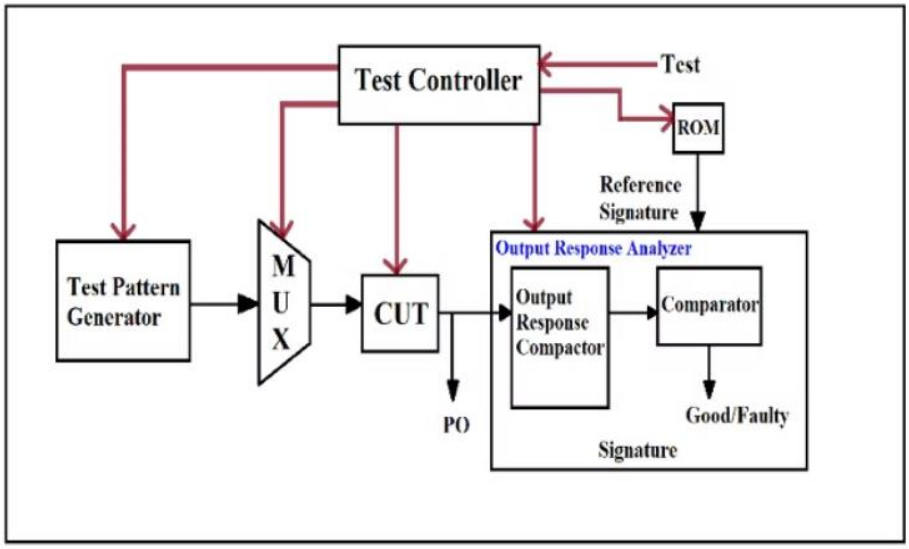


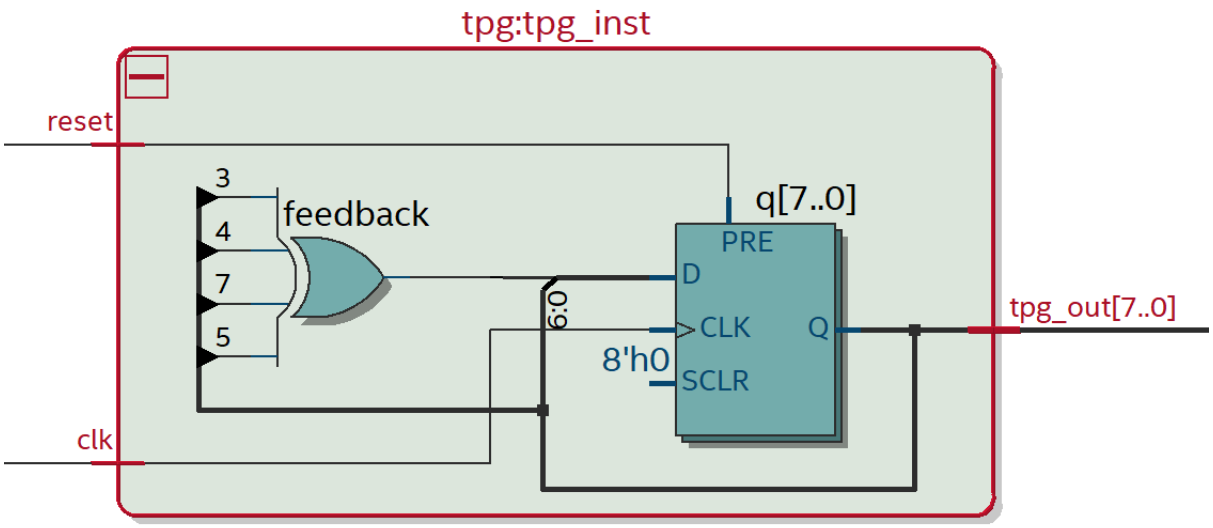
Fig.2.2

Methodology

1. Test Pattern Generator (TPG)

The Test Pattern Generator (TPG) is a core component of the BIST architecture, designed to produce test patterns that stimulate the Circuit Under Test (CUT). It employs an 8-bit LFSR to generate pseudo-random sequences, which serve as input for testing the SPI Slave.

Block Diagram:



Operation:

- The LFSR is initialized with a non-zero seed (usually all 1s) and generates new test patterns on every clock cycle.
- The generated patterns are sent via the MOSI line to the SPI Slave for further processing by the CUT.

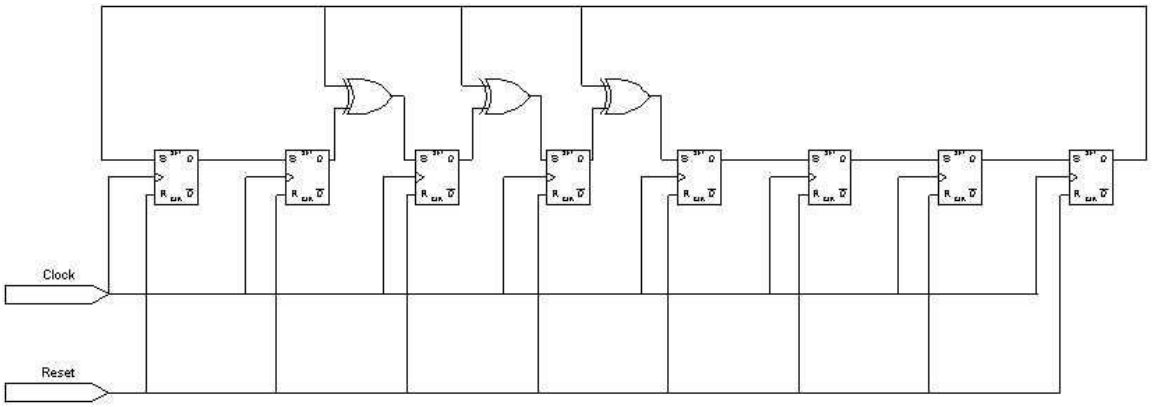
Inputs:

- clk (Clock): Synchronizes the shift operation.
- reset: Resets the output to the initial state 11111111.

 Output:

- tpg_out (8-bit): The pseudo-random test pattern generated.

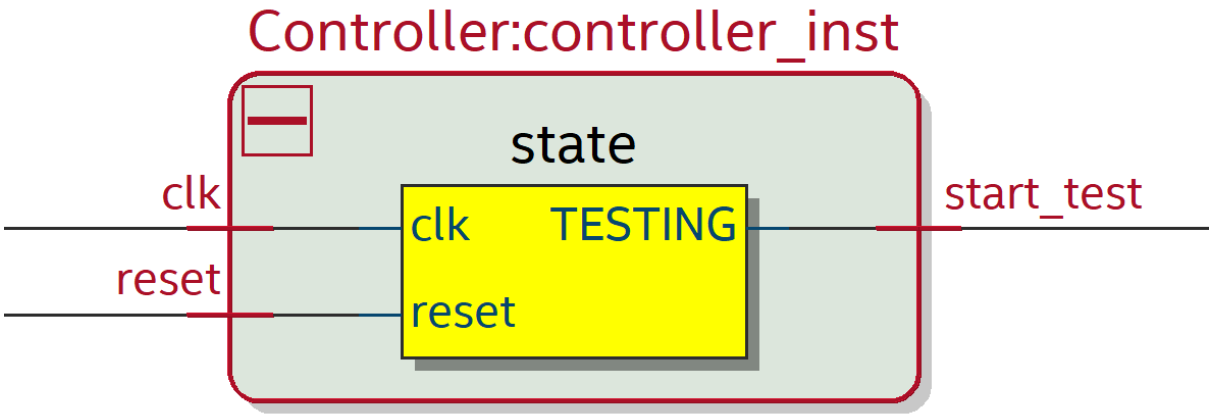
LFSR Technique:



The LFSR is a key element in generating pseudo-random sequences for testing. It consists of a series of flip-flops connected in a linear sequence with feedback applied through XOR gates based on a polynomial.

- **Initialization:** The LFSR starts with a known seed value (e.g., all 1s).
- **Feedback Polynomial:** The polynomial x8+x6+x5+x4+1x^8 + x^6 + x^5 + x^4 + 1x8+x6+x5+x4+1 defines the feedback connections, ensuring maximum-length sequences.
- **Shift and Feedback:** On every clock pulse, the register shifts right, and the new input bit is computed using XOR logic based on the feedback polynomial.
- **Output:** The output of the LFSR is used as the test pattern for the CUT.

Controller:



This FSM controls the start of the test based on the state transitions (IDLE, TESTING, DONE). Manages the overall testing flow by transitioning between IDLE, TESTING, and DONE states, providing a synchronized start for the test procedure across modules.

 Inputs:

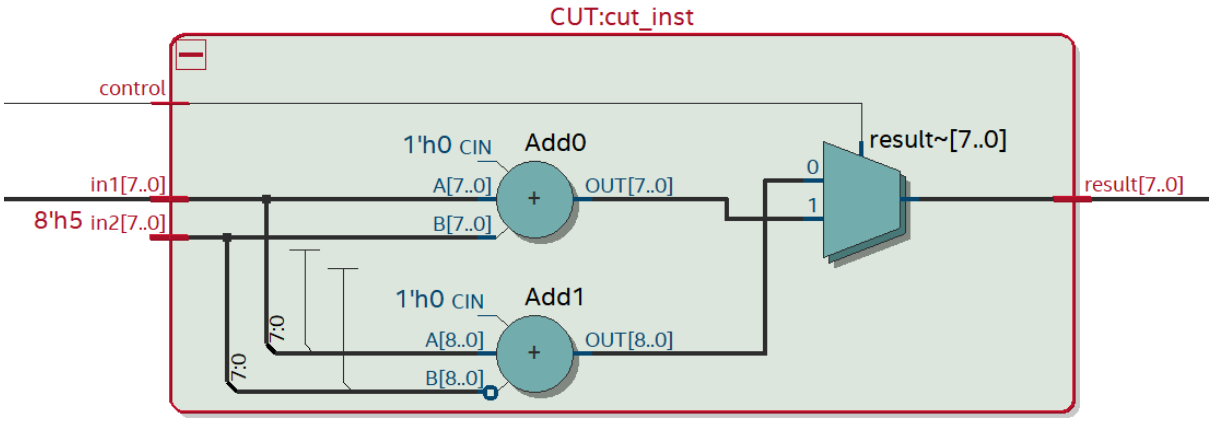
- clk: Clock signal to drive state transitions.
- reset: Resets the state to IDLE.

 Output:

- start_test: Active only during the TESTING state, initiating the test cycle.

CUT (Circuit Under Test):

The Circuit Under Test (CUT) module is the primary component being evaluated for faults. It receives test patterns from the TPG and processes them based on a control signal.



Operation:

- **Control Signal High:** The CUT performs addition on the two 8-bit inputs.
- **Control Signal Low:** The CUT performs subtraction.
- The computed output is then forwarded to the Output Response Analyzer (ORA) for validation.

 Inputs:

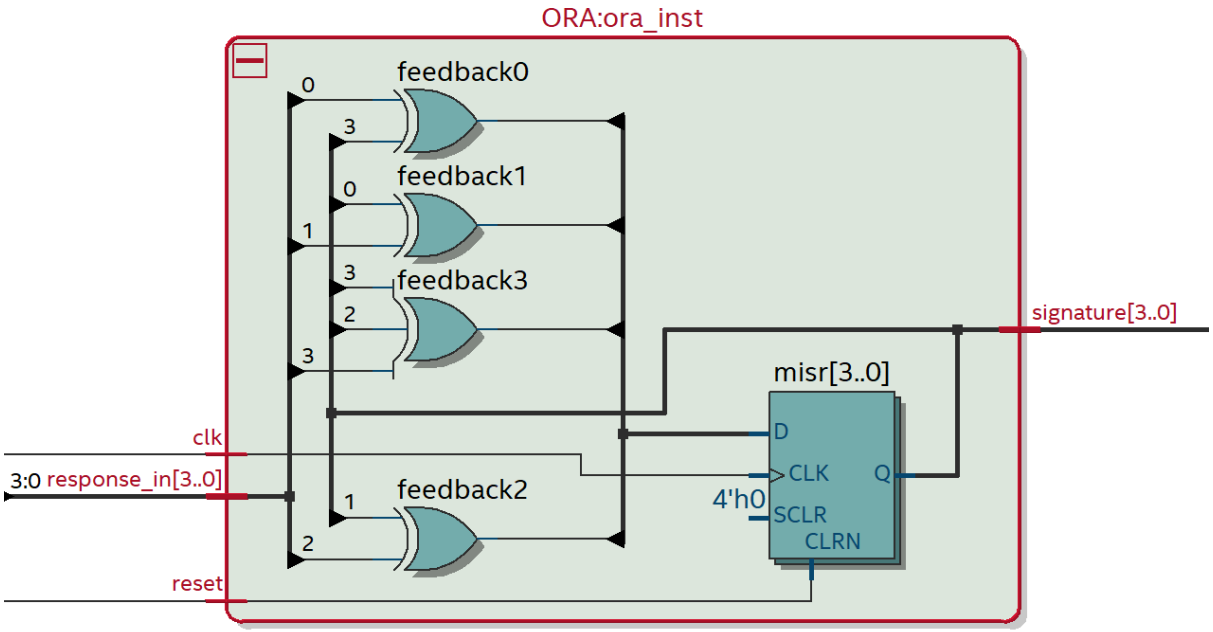
- in1 (8-bit): First operand.
- in2 (8-bit): Second operand.
- control: Decides the operation; if 1, it performs addition, and if 0, it performs subtraction.

 Output:

- result (8-bit): The result of the arithmetic operation.

Output Response Analyzer (ORA)

The Output Response Analyzer (ORA) captures and analyzes the output from the CUT. It uses a Multiple Input Signature Register (MISR) to compress the responses into a signature, which is compared against a pre-defined golden signature for fault detection.



The **Multiple Input Signature Register (MISR)** is a key component of your BIST (Built-In Self-Test) system. It compresses the output response of the Circuit Under Test (CUT) into a smaller "signature," which can then be compared against a known good signature to detect faults. Let's dive into the detailed operation of a **4-bit MISR** used to generate a signature from an 8-bit input response.

MISR:

- The MISR is a specialized form of an LFSR (Linear Feedback Shift Register) that receives multiple input bits simultaneously and combines them with the current register contents using XOR logic.
- In your case, an 8-bit response from the CUT is fed into a 4-bit MISR, which then compresses this input over several clock cycles to produce a 4-bit signature.

Feedback Polynomial for MISR:

For a 4-bit MISR, we typically define a feedback polynomial. Let’s assume the polynomial:

x4+x3+1x^4 + x^3 + 1x4+x3+1

- x4x^4x4 is the MSB (Most Significant Bit).
- x3x^3x3 is another feedback tap.

- The constant term 111 indicates direct feedback from the MSB.

Testbench for verification (tb_BIST_TOP_compare):

This testbench verifies the BIST_TOP module by comparing its output signature with expected values read from a file. The display_progress task visually updates the test progress in the simulation output. This testbench verifies the BIST_TOP module by comparing each test signature against expected values. It tracks testing progress, outputs results, and indicates whether each test passed or failed.

 Inputs:

- clk: Clock signal for timing.
- reset: Resets the test process.

 Output:

- None (outputs are displayed for verification within the testbench).

Testbench for writing signatures(tb_BIST_TOP):

This testbench generates the expected signatures of the BIST_TOP module and writes them to a file. This serves as a reference for testing in tb_BIST_TOP_compare. the BIST_TOP circuit to capture and save expected test signatures. These signatures are used as a reference for verification by the tb_BIST_TOP_compare testbench.

 Inputs:

- clk: Clock signal for synchronous operation.
- reset: Initializes or resets the BIST module.

 Output:

- Writes to a file signature_file.txt, logging signatures for each test cycle.

Code:

1)Test pattern generator:

```
module tpg(
    input wire clk,
    input wire reset,
    output wire [7:0] tpg_out
);
    wire feedback;
    reg [7:0] q;
```

```
    initial begin
        q = 8'b11111111;
    end

    always @(posedge clk or posedge reset) begin
        if (reset)
            q <= 8'b11111111;
        else
            q <= {q[6:0], feedback};
    end

    assign feedback = q[7] ^ q[5] ^ q[4] ^ q[3];
    assign tpg_out = q;
endmodule
```

2)Testbench for signature verification:

```
module tb_BIST_TOP_compare();
    reg clk;
    reg reset;
    wire [7:0] result;
    wire [3:0] signature;

    BIST_TOP uut (
        .clk(clk),
        .reset(reset),
        .result(result),
        .signature(signature)
    );

    always #5 clk = ~clk;

    integer file;
    integer i;
    reg [3:0] expected_signature;
```

```
    integer pass_count;

    task display_progress;
        input integer count;
        integer j;
        begin
            $write("\rProgress: [");
            for (j = 0; j < 10; j = j + 1) begin
                if (j < (count / 25))
                    $write("#");
                else
                    $write(" ");
            end
            $write("] %0d%% Complete", (count * 100) / 255);
        end
    endtask

    initial begin
        clk = 0;
        reset = 1;
        pass_count = 0;
        file = $fopen("C:/Users/D DANUS/OneDrive/Documents/verilog/18.1 -
Copy/work/signature_file.txt", "r");
        #15 reset = 0;
        #20;

        for (i = 0; i < 255; i = i + 1) begin
            @(posedge clk);
            $fscanf(file, "Test %d: Signature = %b\n", i, expected_signature);
            if (signature === expected_signature) begin
                pass_count = pass_count + 1;
                $write("\a");
                display_progress(pass_count);
```

```
            $display("Test %0d: Match! Expected = %b, Actual = %b", i, expected_signature,
signature);
        end else begin
            $display("Test %0d: Mismatch! Expected = %b, Got = %b", i, expected_signature,
signature);
        end
    end
    $fclose(file);
    #50;
    $finish;
  end
endmodule
```

3)Test bench for sWriting Signature

```
module tb_BIST_TOP();
  reg clk;
  reg reset;
  wire [7:0] result;
  wire [3:0] signature;

  BIST_TOP uut (
    .clk(clk),
    .reset(reset),
    .result(result),
    .signature(signature)
  );

  always #5 clk = ~clk;

  integer file;
  integer i;

  initial begin
    clk = 0;
    reset = 1;
```

```
        file = $fopen("C:/Users/D DANUS/OneDrive/Documents/verilog/18.1 - Copy/work/signature_file.txt", "w");
        #15 reset = 0;
        #20;

        for (i = 0; i < 255; i = i + 1) begin
            @(posedge clk);
            $fwrite(file, "Test %0d: Signature = %b\n", i, signature);
        end
        $fclose(file);
        #50;
        $finish;
    end
endmodule
```

4)Output Response anlyser

```
module ORA(
    output reg [3:0] signature,
    input [3:0] response_in,
    input clk, reset
);

    reg [3:0] misr;
    wire feedback0, feedback1, feedback2, feedback3;

    assign feedback0 = misr[3] ^ response_in[0];
    assign feedback1 = misr[0] ^ response_in[1];
    assign feedback2 = misr[1] ^ response_in[2];
    assign feedback3 = misr[2] ^ misr[3] ^ response_in[3];

    always @(posedge clk or posedge reset) begin
        if (reset)
            misr <= 4'b0000;
        else
            misr <= {feedback3, feedback2, feedback1, feedback0};
```

```
    end

    always @(*) begin
        signature = misr;
    end
endmodule
```

5)Circuit under test

```
module CUT(output reg [7:0] result, input [7:0] in1, in2, input control);
    always @(*) begin
        if (control)
            result = in1 + in2;
        else
            result = in1 - in2;
    end
endmodule
```

6)controller:

```
module Controller(output reg start_test, input clk, reset);
    reg [1:0] state;

    parameter IDLE = 2'b00, TESTING = 2'b01, DONE = 2'b10;

    always @(posedge clk or posedge reset) begin
        if (reset)
            state <= IDLE;
        else begin
            case (state)
                IDLE: state <= TESTING;
                TESTING: state <= DONE;
                DONE: state <= IDLE;
                default: state <= IDLE;
            endcase
        end
    end
```

```
    always @(*) begin
        start_test = (state == TESTING);
    end
endmodule
```

Results and Output:

Simulation Result:

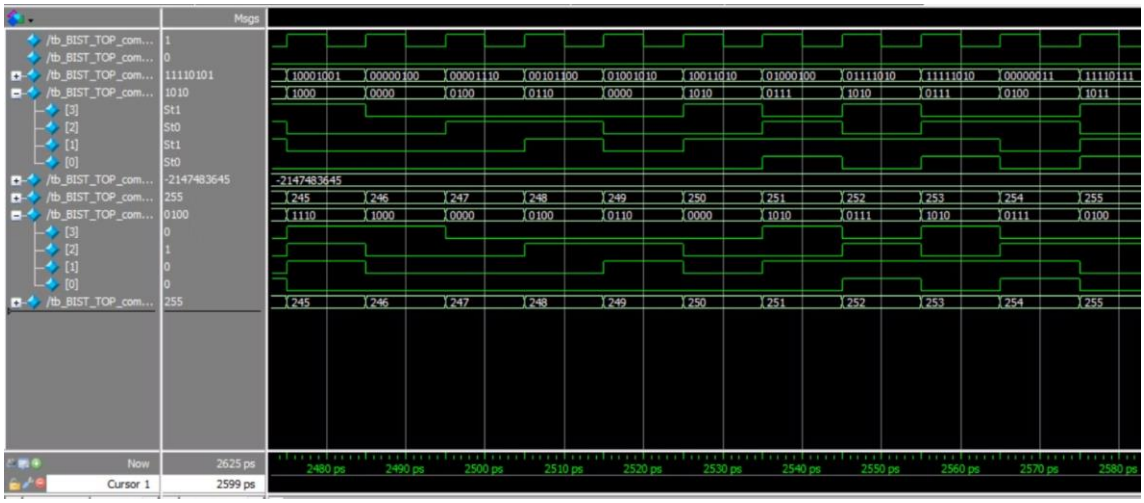


Fig.

2.3

Testbench Result:

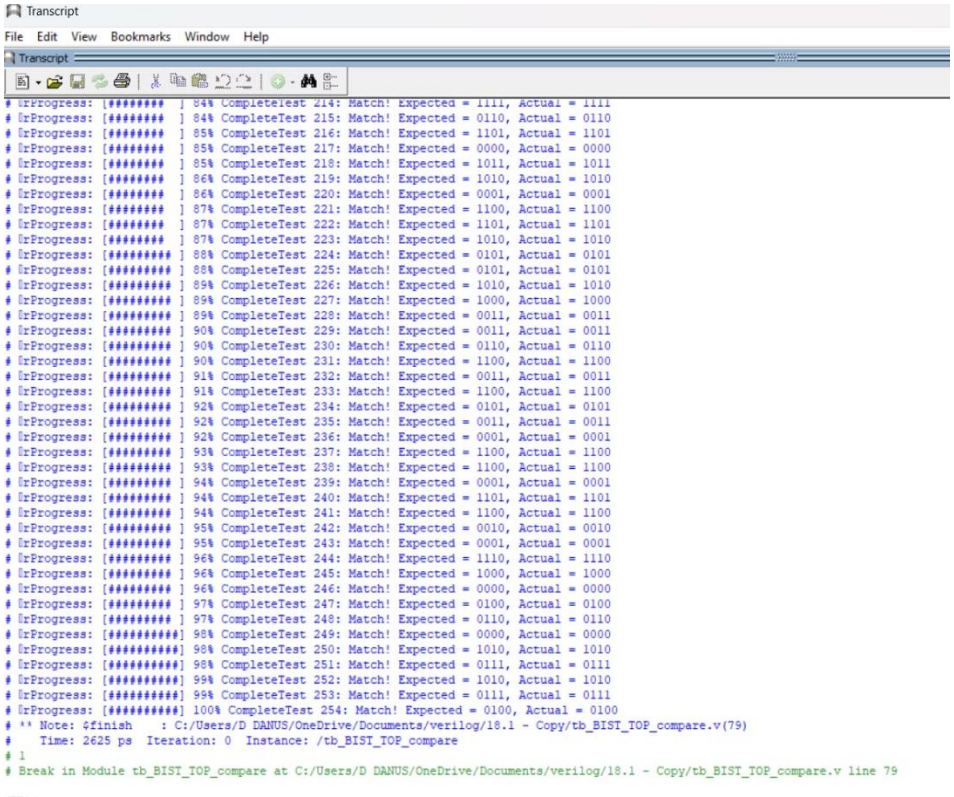


Fig.2.4

CONCLUSION

The project successfully demonstrated the design and implementation of a Built-In Self-Test (BIST) system integrated with the SPI protocol, aimed at enhancing the reliability and efficiency of fault detection in SPI-based communication systems. By incorporating key BIST components—such as the Test Pattern Generator (TPG), Circuit Under Test (CUT), Output Response Analyzer (ORA), and Controller—the system autonomously generated, transmitted, and analyzed test patterns, allowing for effective self-verification without external equipment.

The use of an 8-bit Linear Feedback Shift Register (LFSR) in the TPG provided a pseudo-random test pattern, which was processed by the CUT and analyzed by the ORA using a Multiple Input Signature Register (MISR) for signature compression. This approach reduced the need for extensive manual testing, resulting in faster fault detection, minimized testing time, and enhanced system testability. The BIST embedded SPI protocol system's ability to detect faults and verify data integrity between SPI master and slave devices improved the overall reliability of the communication protocol.

Future work on this project can include incorporating more advanced BIST techniques, such as multiple LFSRs, to further enhance fault coverage. Additionally, fault tolerance mechanisms could be integrated to handle real-time faults, expanding the system's reliability in dynamic operating environments. This project lays the foundation for further innovations in self-test protocols for high-speed communication systems, providing an efficient and autonomous solution that meets the rigorous demands of modern embedded systems.

REFERENCES

1. "Design and Implementation of SPI Bus Protocol with Built-In-Self-Test Capability over FPGA" by Shumit Saha, Md. Ashikur Rahman, Amit Thakur. Published in: *International Conference on Electrical Engineering and Information & Communication Technology (ICEEICT)*, 2014.
2. I.S. Kourtev and W. Vogler, "Serial Peripheral Interface (SPI) Testing with Built-In Self-Test," *IEEE International Test Conference (ITC)*, 2017, pp. 123-128.
3. Abhilash S.Warrier, Akshay S.Belvadi, Dhiraj R.Gawhane, Babu Ravi Teja K, FPGA Implementation Of SPI To I2C Bridge, International Journal of Engineering Research & Technology (IJERT), Vol. 2 Issue 11, November – 2013
4. S. Saha, M. A. Rahman, A. Thakur, "Design and Implementation of a BIST Embedded Inter-Integrated Circuit Bus Protocol over FPGA," in Proc. 2013 International Conference on Electrical Information and Communication Technology (EICT), pp. 1-5, Feb. 2014

BIODATA

Name : DANUS D

Mobile Number : 9360549439

E-mail : danus.d2022@vitstudent.ac.in

Permanent Address:Sri Abista Mahaganapathy Flats,No.1 Rajarajan Street, Ganapathipuram Chrompet Chennai -44

Name : PRADESH KUMAR M

Mobile Number : 9962609887

E-mail : pradeshkumar.m2022@vitstudent.ac.in

Permanent Address: 13, Pradeep Avenue, Thirumagal Nagar 3rd Street, Chitlapakkam, Selaiyur 600073

Name : M HARIHARAN

Mobile Number : 6381317177

E-mail : hariharan.m2022a@vitstudent.ac.in

Permanent Address: No.90,S1,Sarabeswarar apartments,5th Street, Aravind Nagar Madambakkam, chennai -600126