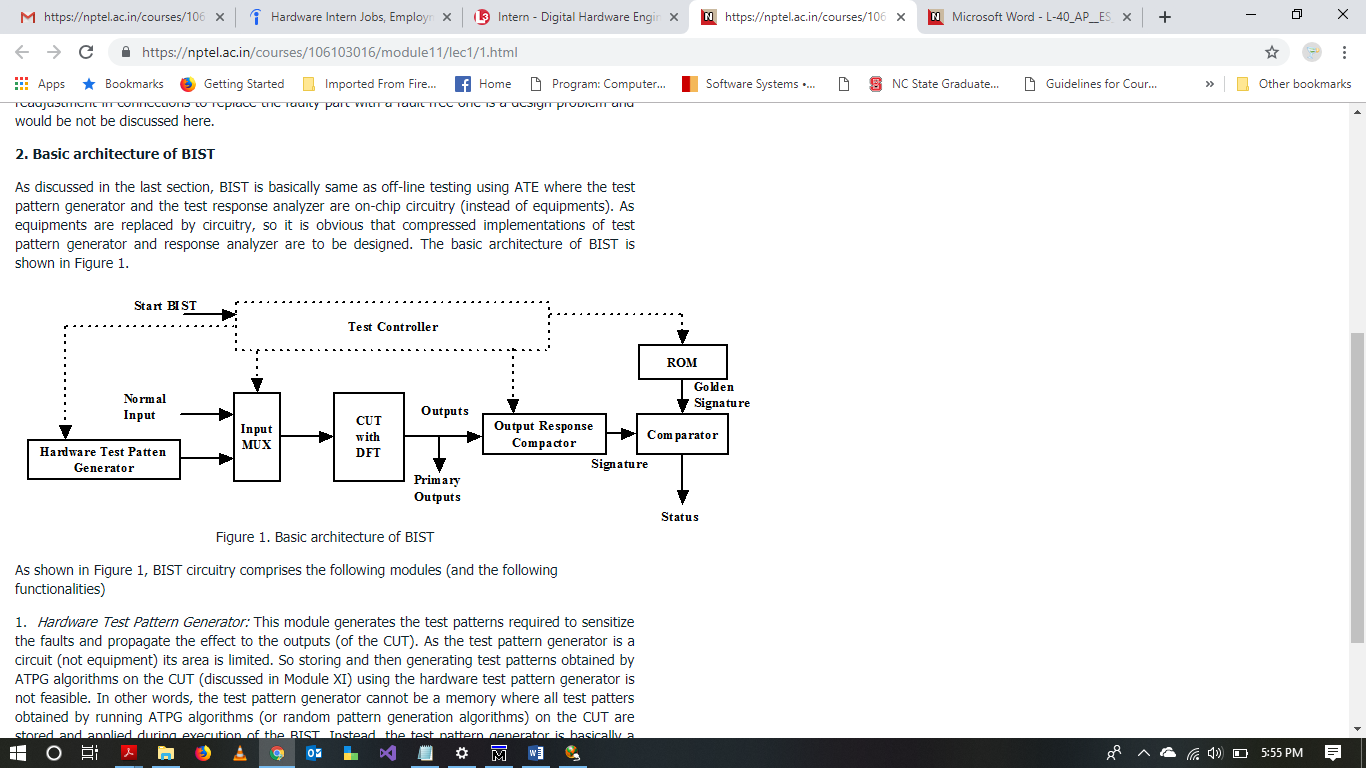
**1. Block diagram and explanation of your complete design with CUT and BIST Logic. Please**

**include your selection criteria of BIST components and how you calculated the golden**

**(expected) signature.**

**Built-In Self-test (BIST)**

BIST is a design-for-testability (DFT) technique that adds some additional hardware along with the chip for testing. As, we know that faults may occur even after a chip is fabricated, and for detecting such kind of faults we use BIST. BIST would test a circuit every time before it starts.



*Basic Architecture of BIST*

The basic BIST architecture requires the addition of three hardware blocks to a digital circuit: a test pattern generator, a response analyzer, and a test controller.

Test Pattern Generator (TPG generates the test patterns required to sensitize the faults and propagate the effect to the outputs of the CUT. Examples of pattern generators are ROM with stored patterns, a counter, a linear feedback shift register (LFSR). An output response analyzer (ORA) typically is a comparator with stored responses or an LFSR used as a signature analyzer. ORA compacts and analyzes the output responses to check the correctness of the circuit. The Normal input are not tested by the BIST.

*Block Diagram of BIST:*

LFSR

BIST Controller

MISR

CUT

Here, we have used a linear feedback shift register to generate pseudo-random patterns for testing. The CUT for the BIST is an Arithmetic Unit which includes addition and multiplication operations. Multiple input shift register is used as an ORA.

The BIST controller operates 2 mode in CUT. During normal operation, primary inputs are executed in the AU and outputs are obtained. In BIST mode, CUT is fed by LFSR and the responses are sent to MISR where the responses are compacted and compared to Golden signature. If the output of MISR matched golden signature, it implies that the circuit is fault free.

*Golden Signature:*

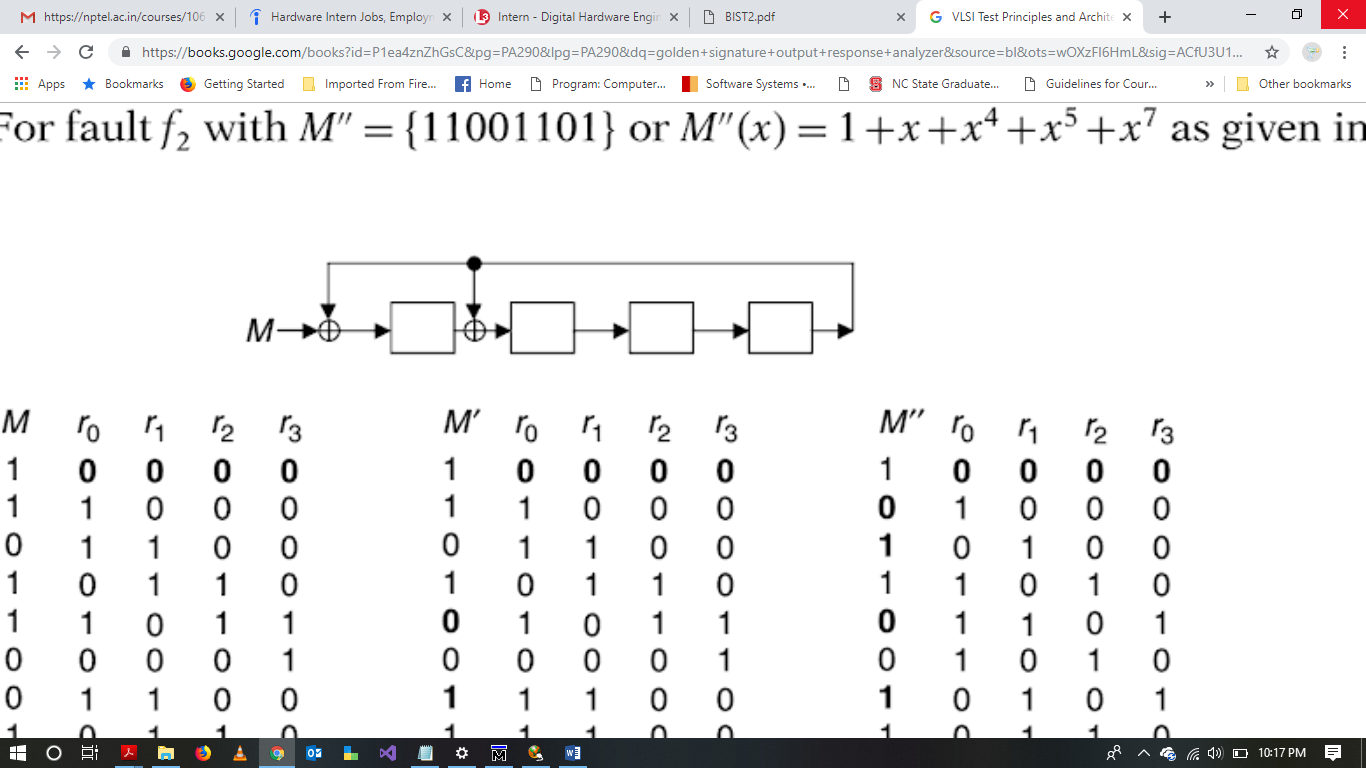
A golden signature is obtained from LFSR to compare with the compacted responses to find faults.

Input polynomial: Input polynomial is obtained by the input stream we feed to the input of LFSR.

Characteristics polynomial: This is obtained by the XOR structure of LFSR.

To get the golden signature, we need to obtain both the polynomials and divide the input polynomial with characteristic polynomial. The remainder obtained is our golden signature.

Considering the characteristics polynomial to be f(x) = X4+X+1. The circuit is as shown below:



Assuming input sequence for this circuit is: M = {10101101}

Input polynomial obtained is: M(x) = X7 + X5 + X4 + X2 + 1

LFSR is first initialized to a starting 0000 and then we feed the input pattern. The signature obtained is {1101}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| M | X1 | X2 | X3 | X4 |
| 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| R | 1 | 1 | 0 | 1 |

Now, we can obtain the remainder of polynomial by dividing M(x) with f(x)

X7 + X5 + X4 + X2 + 1 / X4 + X + 1 which gives us X3 + X + 1 = *{1101}* which matches the signature obtained above.

**2. VHDL/Verilog implementation of each of the three major BIST components.**

**3. Test benches for each component and simulation waveforms.**

**BIST Implementation**

BIST was implemented using Verilog. Given below are the source code, testbench and waveforms for LFSR, MISR, and controller.

1. Linear feedback shift register (LFSR)

*Source Code:*

// Source Code for 9-bit linear feedback shift register (LFSR)

//

// This module provide pseudo random test patterns for the CUT

// Complete provides the status until LFSR will generate pattern

// Once all the patterns are generated and counter = 8, Complete

// status changes to 1.

module ML\_lfsr(data\_out,complete,reset,clock);

input reset;

input clock;

output [8:0] data\_out; output complete;

reg complete; reg [8:0] lfsr\_reg; // reg to store complete status and lfsr to store patterns

reg [8:0] counter; reg tap;

always@(posedge clock or posedge reset)

begin

if(reset == 1)

begin

lfsr\_reg <= 9'b101101010;

counter <= 9'b000000000;

end

else

begin

tap = lfsr\_reg[8] ^ lfsr\_reg[4]; // Tapping is done at 5th and 9th bit

lfsr\_reg[8:0] <= { lfsr\_reg[7:0], tap };

counter = counter + 1;

if (counter < 9'b000001001) //counter runs for 8 times

complete = 0;

else

complete = 1;

end

end

assign data\_out = lfsr\_reg; // Register data is assigned to output after computation

endmodule

*Testbench for LFSR*

// Testbench for LFSR

module tb\_ML\_lfsr;

reg clock,reset;

wire [8:0] dt\_out;

ML\_lfsr lfsr1(dt\_out, complete, reset, clock);

initial

begin

clock = 1'b0;

end

always

#5 clock = ~clock;

initial

begin

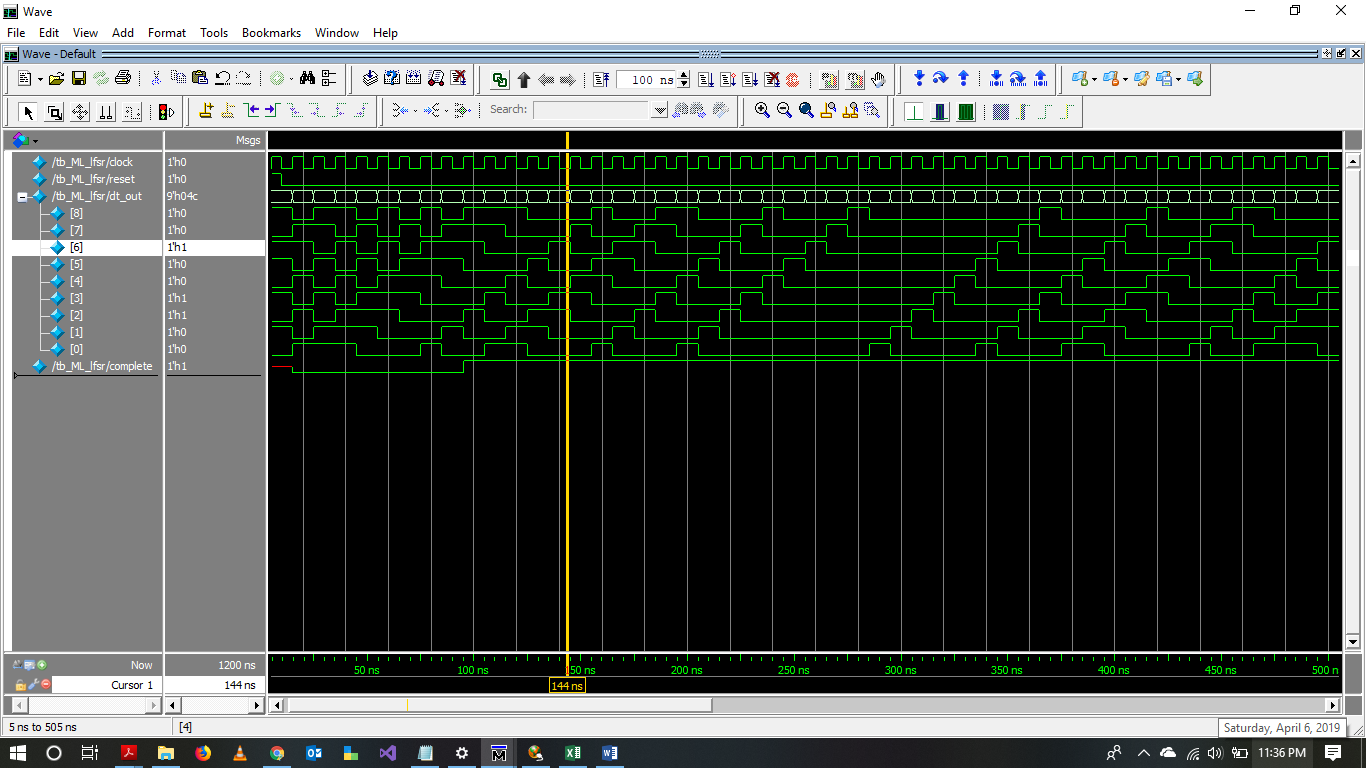
reset = 1'b1;

#10 reset = 1'b0;

end

endmodule

*Waveform of LFSR testbench:*



1. Multiple Input Shift Register (MISR)

*Source Code:*

//Source Code for 4-bit Multiple Input Shift Register (MISR)

module MISR\_4bit(dataIn,reset,clock,dataOut);

input [3:0] dataIn;

input reset,clock;

output reg [3:0] dataOut;

//reg [3:0] dataOut;

reg [3:0] misr\_temp;

always@(posedge clock or posedge reset)

begin

if(reset == 1)

dataOut <= 4'b0000;

else

begin

misr\_temp = dataOut;

dataOut[0] = misr\_temp[3] ^ dataIn[0]; //XORing bit 3 and data in gives 1st bit

dataOut[1] = misr\_temp[3] ^ misr\_temp[0] ^ dataIn[1];

dataOut[2] = misr\_temp[1] ^ dataIn[2];

dataOut[3] = misr\_temp[2] ^ dataIn[3];

end

end

endmodule

*Testbench*

// Testbench for MISR

module tb\_MISR4bit;

reg clock,reset;

wire [3:0] dataOut;

reg [3:0] dataIn;

MISR\_4bit misr4bit1(dataIn,reset,clock,dataOut);

initial

begin

clock = 1'b0;

end

always

#5 clock = ~clock;

initial

begin

reset = 1'b1;

#10 reset = 1'b0;

dataIn = 4'b0111;

#10 dataIn = 4'b0000;

#10 dataIn = 4'b1100;

#10 dataIn = 4'b1010;

#10 dataIn = 4'b0111;

#10 dataIn = 4'b0001;

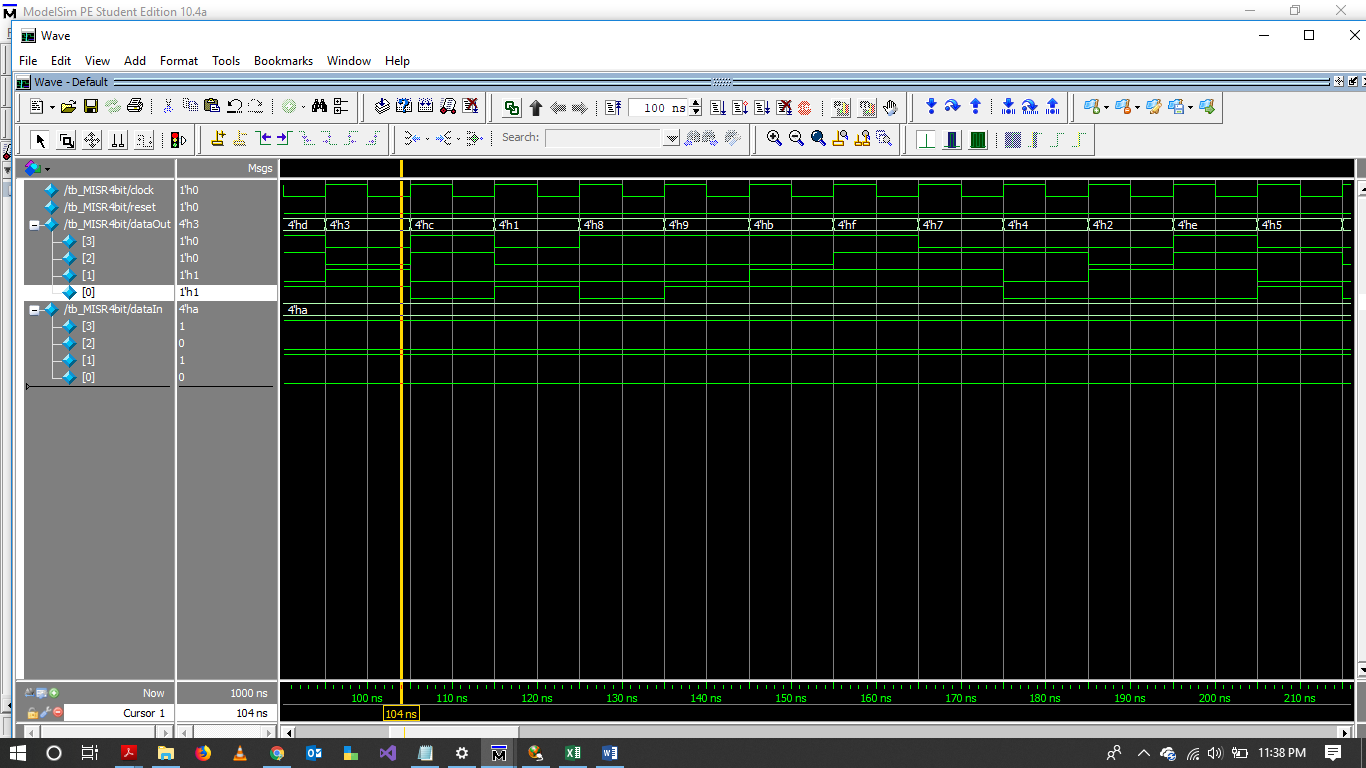
#10 dataIn = 4'b1101;

#10 dataIn = 4'b1010;

end

endmodule

*Waveform for Testbench:*



1. BIST Controller

*Source Code:*

// Controller for BIST

// Controller runs in 2 modes. 1 for BIST mode and 0 for Normal mode

// In BIST mode LFSR output is fed to CUT (Arithmetic unit) and the output is given to MISR

// where it is compared to golden signature to detect faults

module Controller (faultfound, Mode, BIST\_in, Complete, rst, clk, MISR\_out);

input Mode; // BIST mode

input [8:0] BIST\_in;

input clk, rst, Complete;

wire [3:0] ALU\_out;

wire[3:0] a, b;

wire carry;

wire select;

reg [3:0] ALU\_A, ALU\_B; // ALU input registers

output reg faultfound; //Sets to 1 when fault is detected

output [3:0] MISR\_out;

parameter golden\_signature = 4'b1101; // Golden signature obtained to be compared with MISR output

ML\_lfsr lfsr1 (.data\_out(BIST\_in), .complete(Complete), .reset(rst), .clock(clk));

AU CUT1 (.A(ALU\_A), .B(ALU\_B), .Sel(select), .Result(ALU\_out), .Output(carry));

MISR\_4bit misr1 (.dataIn(ALU\_out), .reset(rst), .clock(clk), .dataOut(MISR\_out));

assign select = BIST\_in [8];

assign a = BIST\_in [3:0];

assign b = BIST\_in [7:4];

always @ (posedge clk or posedge rst)

begin

if (rst == 1)

faultfound = 0;

else

begin

if (Mode == 1) //BIST Mode

begin

assign ALU\_A = a;

assign ALU\_B = b;

if (Complete == 1)

begin

if (golden\_signature != MISR\_out) //If the golden signature does not match MISR output then fault is detected

faultfound = 1;

else

faultfound = 0;

end

end

else // Normal Mode

faultfound = 0;

end

end

endmodule

*Testbench:*

// Testbench for BIST Controller

`timescale 1ns/100ps

module Controllernew\_tb;

reg [8:0] BIST\_in;

reg clk;

reg rst;

reg Complete, Mode;

reg [3:0] a, b;

reg [3:0] golden\_signature = 4'b1101;

wire [3:0] MISR\_out;

Controller c (faultfound, Mode, BIST\_in, Complete, rst, clk, MISR\_out);

initial

begin

clk = 0;

rst = 1;

forever #5 clk = ~clk;

end

initial

begin

forever #8 rst = ~rst;

end

always@(posedge clk)

begin

Mode = 0; //Normal Mode

a = 4'b0010;

b = 4'b0101;

BIST\_in = 9'b110001010; // Feeding data to ALU circuit under test when Mode = 1

#10 Complete = 1;

#15 Mode = 1; //BIST Mode

#20 BIST\_in = 9'b110000001;

#25 BIST\_in = 9'b111111010;

#28 BIST\_in = 9'b000111001;

#30 Mode = 0;

a = 4'b1111;

b = 4'b1100;

a = 4'b101;

b = 4'b1111;

#35 Mode = 1;

BIST\_in = 9'b110001010;

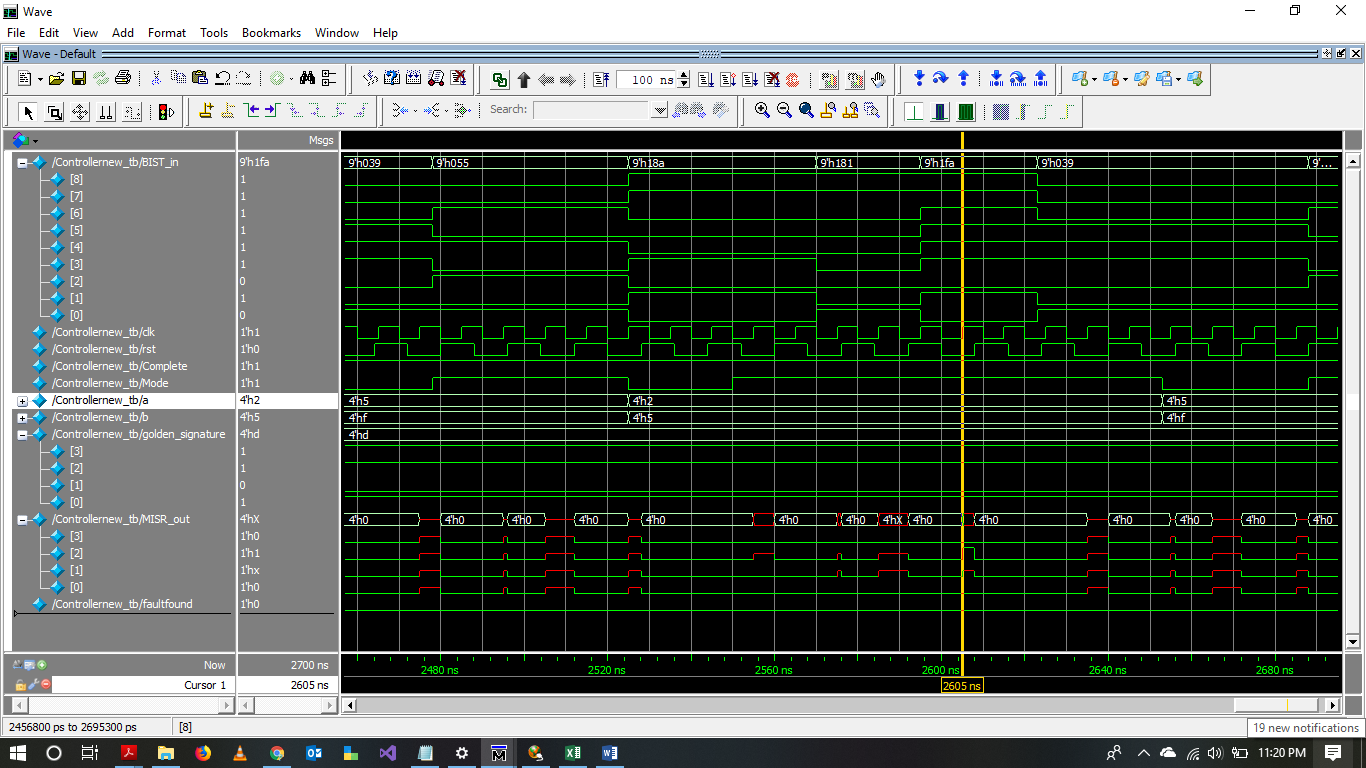
BIST\_in = 9'b001010101;

#40 Complete = 1;

end

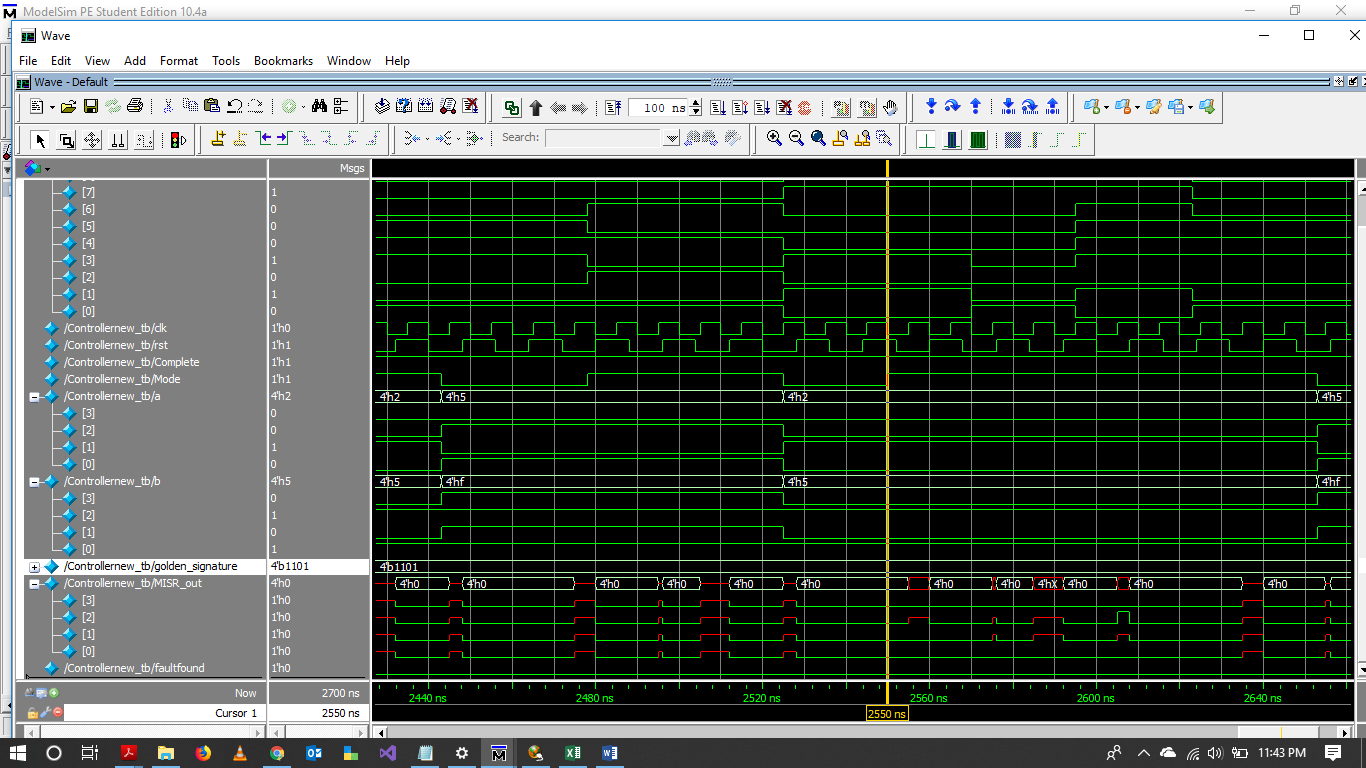
endmodule

*Waveform for BIST Controller:*

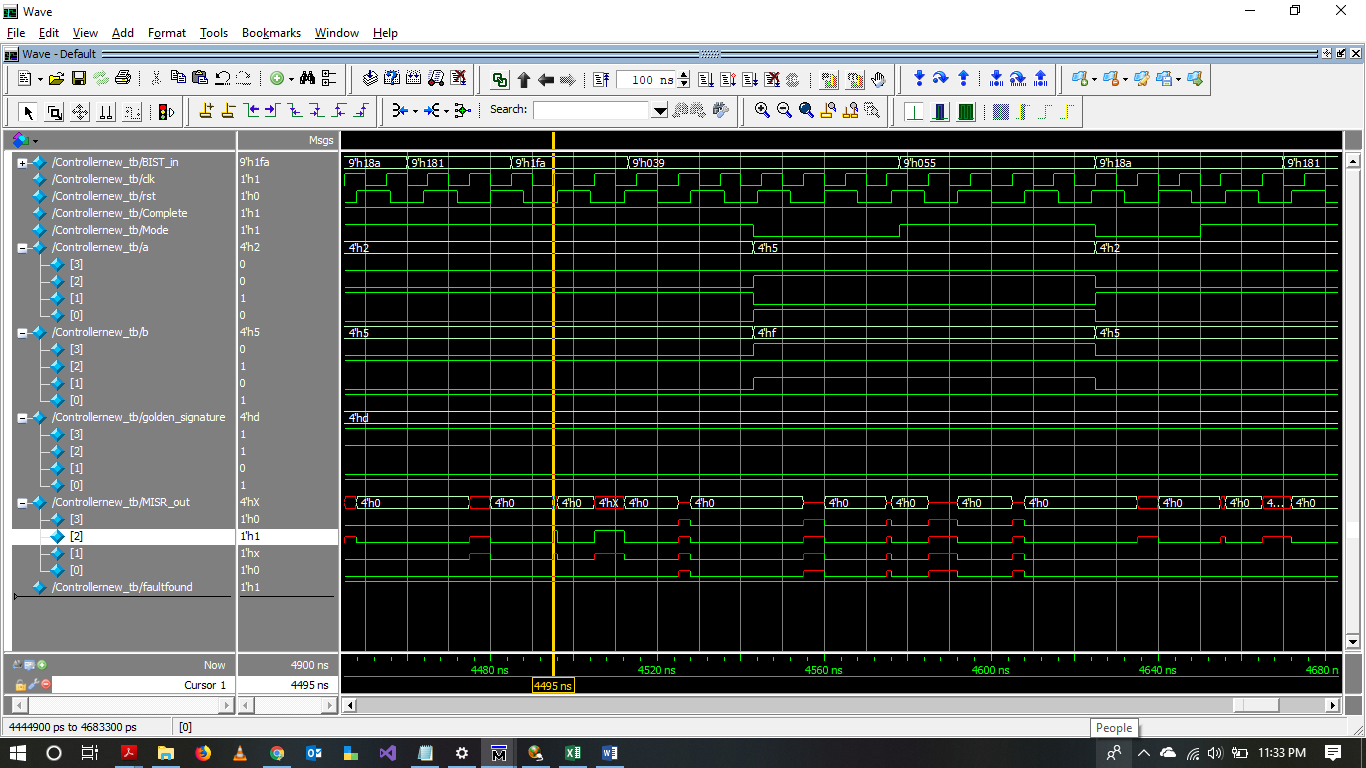


1. **Simulation waveforms for test mode with FAULT-FREE (PASS) and FAULTY (FAIL) cases, and normal mode with some operations.**

Shown below is the waveform of the fault free circuit. It can see from the waveform that the fault detected is 0 and hence the circuit is fault-free.



Later, faults were induced in the Arithmetic unit which is our CUT for BIST. The waveform shown below is the faulty circuit. The fault detected is hence the signal fault found is 1.



1. **DC compiler script for synthesis and synthesis results.**

I tried to compile BIST through DC compile, however, there were few errors and hence was not able to synthesis it successfully.

**6. Please elaborate also on what you’ve learnt and how your approach can be extended to**

**create a complete DFT EDA software.**

* BIST is a test procedure which facilitates testing of circuits before every time they start their operations
* The main components of BIST are TPG, ORA and Test Controller
* BIST is an alternative for software tests as it eliminates the disadvantages of software test such as low hardware fault coverage, poor diagnostic results and consumption of time
* BIST is highly reliable and it reduces cost as the complexity of the test is reduced as it uses pseudo random pattern generation
* BIST applications are widespread and are used for testing in industries. Few examples are Exhaustive test in Intel 80386, Pseudorandom Test in the IBM RISC/6000, Embedded Cache Memories BIST of MC68060 etc.
* BIST doesn’t require ROM to store test patterns as the TPG generates pattern in minimal amount of time

***REMARKS***

* In this lab, we designed a BIST in Verilog using ModelSim
* LFSR, MISR and Controller were implemented to check for any faults present in the designed circuit
* Synthesis could not be performed successfully due to multiple errors generated in Design Compiler
* Golden vector is obtained in the report using polynomial division, however, I was not able to implement any code for it
* Faults were induced in the circuit under test to check if the BIST detects faults