Circuit Reliability and Testing

Built-In-Self-Test Implementation

Name: Patsaoglou Panteleimon

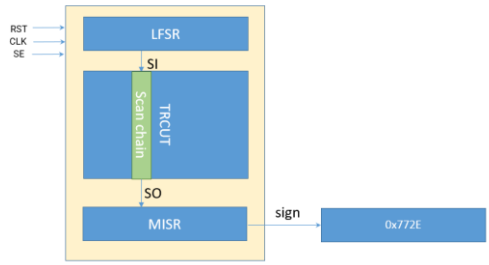
AM: 5102

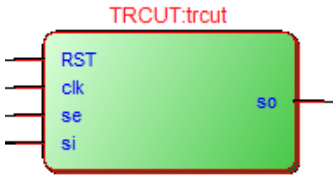
- ## Assignment

In the previous assignment a traditional scan chain was implemented, and it was integrated into a basic circuit under test **(CUT)**. From the test benches performed, the test vectors generated from the truth table of the CUT were fed into the scan chain, a capture was performed to get the observation points, and in the next 4 cycles we had the response of the CUT shifted out from SO.

While this test method was feasible for our test circuit which had 4 inputs, we concluded that the method becomes largely impractical as input number increases. The issues that arise are that it also becomes impractical and expensive to store the large number of test vectors on-chip, while feeding each test vector and checking if the response is the expected one is time-consuming.

To address the limitations of the previous testing methods, Build-In-Self-Test (BIST) architectures can be introduced to make testing scalable and optimal. Therefore, **Linear Feedback Shift Registers (LFSRs)** responsible for test pattern generation and **Multiple Input Signature Registers (MISRs)** responsible for generating a signature bit-stream based on the CUT's response must be implemented and integrated as modules into the existing TRCUT made in the previous assignment.





- ## BIST Flow

The test flow begins by setting the asynchronous reset of the TRCUTwithMISR to logic 1. This will reset all the combinational logic to a known state. More precisely, the reset will set the random seed of the LFSR to 0xA5, and the registers of the scan chain and the MISR to 0. In the next clock cycles (4 Cycles) SE is set 1 to shift the test vectors generated randomly by the LFSR into the TRCUT. In the 5th cycle the capture is performed by setting SE to logic 0. In the next clock cycles the response from the SO of the TRCUT is fed into the tap of the MISR so the generation of the signature can begin.

This process, in our test benches made, repeats for 32 captures-Pseudo-Random-Test-Vectors generated by the LFSR and after the last capture we additionally wait for 16 cycles to get the signature produced by the MISR.

The logic behind this process is that if there are no issues with the IC block tested, and the signature produced during testing matches the known-good one, then it is a Test-Passed. If the signature produced differs due to issues like stuck-at-faults or Bit-Errors caused by cosmic radiation during the testing phase, then the circuit block is a Test-Failed and therefore additional testing must be conducted to confirm that the errors were systematic and not random.

- **TRCUT modification**

A modification was made to the TRCUT so the reset can be performed in the scan chain registers. The modification was important because at the first clock cycles undefined values were fed in the MISR, which was locked in an undefined state for the hole test process.

```
module TRCUT(clk,RST,si,se,so);

input clk,si,se, RST; // Addition – added rst pin to reset dff of scan chain to 0
output so;

wire a,b,c,d,i,j;

assign so = d;

SDFF Raj(clk, RST, j, si, se, a);
SDFF Rbi(clk, RST, i, a, se, b);
SDFF Rc(clk, RST, c, b, se, c);
SDFF Rd(clk, RST, d, c, se, d);

CUT cut (.a(a), .b(b), .c(c), .d(d), .i(i), .j(j));
endmodule
```

- **TRCUTwithLFSR Implementation** (2.1)

For the LFSR implementation the **LFSR Testbench 1.30** was used to generate the 8bit LFSR. The software also calculated the optimal number and position of the XOR taps to generate the maximum sequence length. In the given code some modifications were performed like the addition of asynchronous Reset to set the seed value to the LFSR.

In the TRCUTwithLFSR module the necessary ports and wires were initialized to wire up the LFSR with the TRCUT. The SI of the TRCUT is wired with the bit[0] of the LFSR and the SO is the SO of the TRCUT.

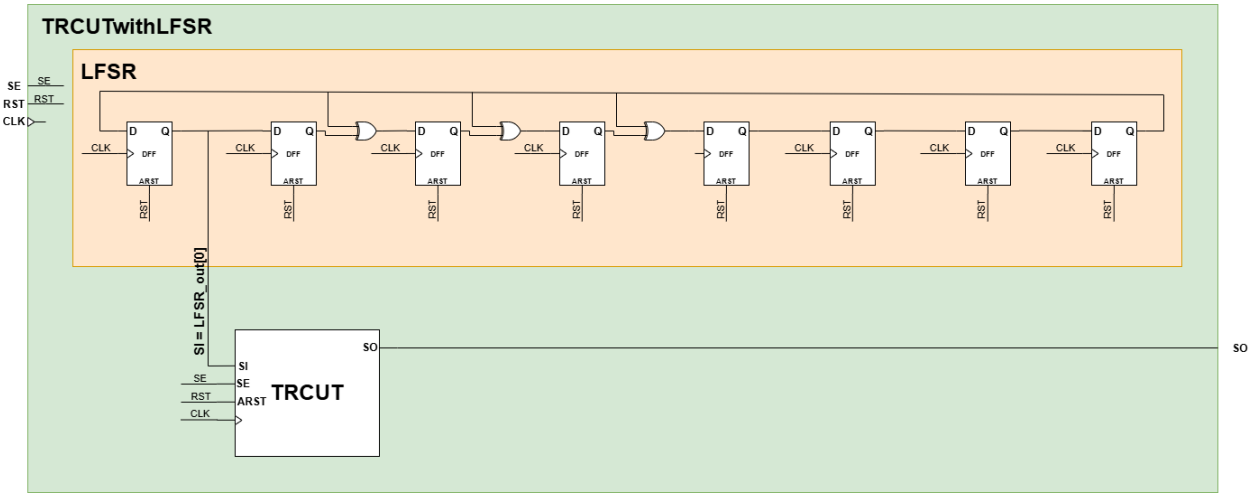
- **LFSR Verilog**

```
module LFSR(
    input clk,
    input rst,              // ADDITION - reset signal to initialize LFSR with
the seed value
    output reg [7:0] LFSR = 255
);

wire feedback = LFSR[7];

parameter SEED = 8'hA5; // ADDITION - passing seed parameter

always @(posedge clk or posedge rst)
            begin
            if (rst)
                        LFSR <= SEED; // ADDITION - set initial seed
            else begin
                LFSR[0] <= feedback;
                LFSR[1] <= LFSR[0];
                LFSR[2] <= LFSR[1] ^ feedback;
                LFSR[3] <= LFSR[2] ^ feedback;
                LFSR[4] <= LFSR[3] ^ feedback;
                LFSR[5] <= LFSR[4];
                LFSR[6] <= LFSR[5];
                LFSR[7] <= LFSR[6];
            end
end
endmodule
```

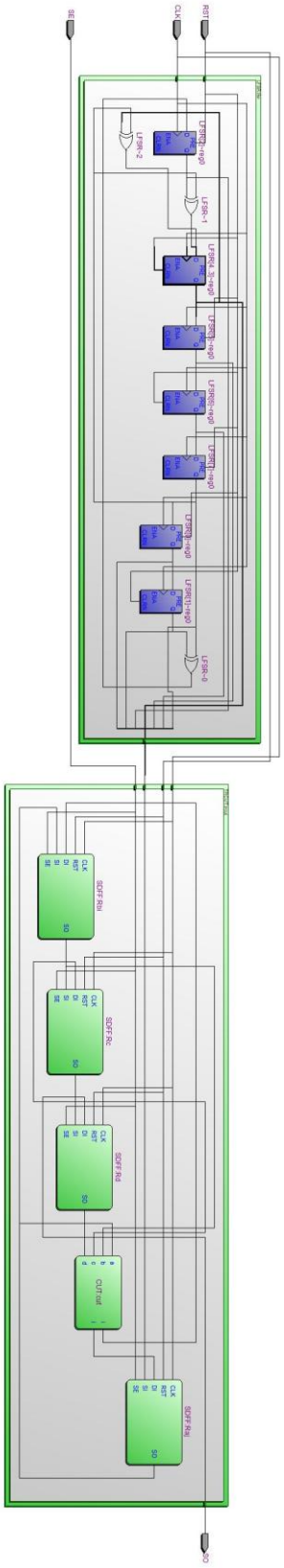
- **TRCUTwithLFSR Verilog**

```
module TRCUTwithLFSR(SE,RST,CLK,SO);

input SE,RST,CLK;
output SO;

wire SE,RST,CLK,SI,SO;          // wires to connect modules
wire [7:0] LFSR_out;            // bus array for the lfsr output

assign SI = LFSR_out[0];        // connecting one bit of the LFSR with
TRCUT(SI)

//          ADDITION - added reset to trcut chain to initialize dffs
TRCUT trcut (.clk(CLK),.si(SI),.se(SE),.RST(RST),.so(SO));
LFSR lfsr (.clk(CLK),.rst(RST),.LFSR(LFSR_out));
endmodule
```

- **TRCUTwithLFSR expected diagram**



- **TRCUTwithLFSR Synthesized**



- **TRCUTwithLFSR Testbench** (2.2)

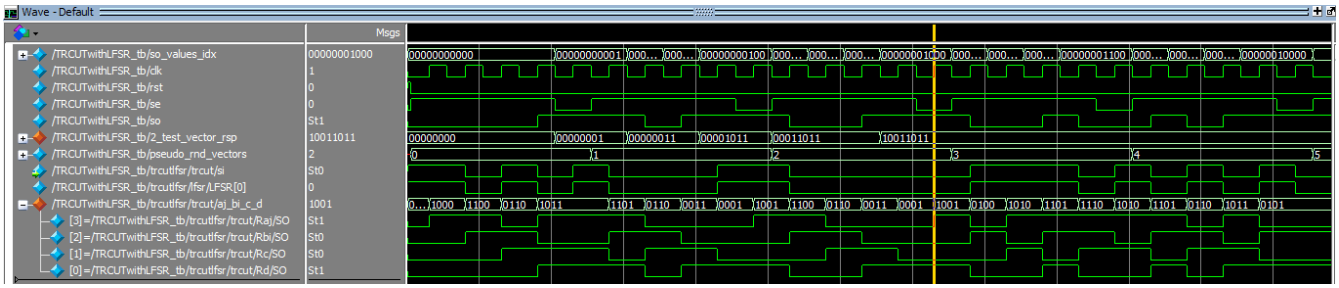
The TRCUTwithLFSR_tb testbench initializes and verifies the TRCUTwithLFSR module by applying 32 pseudo-random test vectors generated internally. It begins with a reset (rst = 1) and initializes all necessary registers, including the so_values register to store scan outputs. A continuous clock is generated with a 10-time unit period.

The testbench applies each vector by enabling the scan chain (se = 1) for 4 clock cycles (#40) and then disabling scan enable (se = 0) for 1 cycle (#10) to simulate a capture phase. During the scan enable phase, the output bit SO is stored into so_values sequentially using an index (so_values_idx). This process is repeated for 32 vectors, and the total scanned output is stored for later analysis.

- **TRCUTwithLFSR Verilog**

```
module TRCUTwithLFSR_tb();

reg clk,rst,se;          // regs to drive trcutlfsr
wire so;                 // wire to connect so with the tb so_values

reg [255:0] so_values;           // reg to store bits from so
reg [10:0] so_values_idx;        // reg to have idx of so_values

integer pseudo_rnd_vectors; // integer for vectors for-loop
// wiring up TRCUTwithLFSR
TRCUTwithLFSR trcutlfsr (.SE(se), .RST(rst),.CLK(clk), .SO(so));

initial begin
        rst = 1; // reset regs of lfsr to seed and trcut regs to 0

        //          initilize array stuff
        so_values = 0;
        so_values_idx = 0;
        se = 0;                  // initilize scan-en
        clk = 0;                 // initilize clk
        #1;

        rst = 0;                 // setting rst to 0 to begin

        forever begin
                #5 clk =! clk;
        end
end
initial begin
        #1; //      delay so there is sync with first initial block

        // for-loop to for the 32 pseudo_rnd_vectors from the lfsr
        for (pseudo_rnd_vectors = 0; pseudo_rnd_vectors<32; pseudo_rnd_vectors = pseudo_rnd_vectors + 1) begin
                se = 1;
                #40;        // 4 cycles for shifting the vector into the scan chain
                se = 0;
                #10;        // 1 cycles for capturing the observation points
        end
end
initial begin
        #41         // wait so the first vector inside the scan chain to begin saving response
        forever begin
                if (se) begin           // if not in capture save the result of so
                        so_values[so_values_idx] = so;          // save si respose to array
                        so_values_idx = so_values_idx + 1;      // increase idx
                end
                #10;
        end
end
endmodule
```

- **TRCUTwithLFSR Waveform**



- **TRCUTwithLFSR Waveform description**

From the waveform generated, we can observe the reset at the beginning, initializing the combinational logic before the first clock cycle ticks. Observing the SI that is wired with the bit[0] of the LFSR, we see from the first 4 cycles that bits from the LFSR are shifted into the scan chain. In the 5th cycle capture occurs and we get the observation points of the TRCUT. We see that for (a,b,c,d)=(1,0,1,1) we get the (j,i,c,d)=(1,0,1,0) response that is the expected one. In the next 4 cycles, we see the response shifting out of SO, and saved into the 2_test_vector_array while the generated bitstream of the LFSR is shifted into the scan chain for the next capture.

- **TRCUTwithMISR Implementation** (2.3)

For the MISR implementation the **LFSR Testbench 1.30** was also used to generate the 16bit MISR. The software also calculated the optimal number and position of the XOR taps to generate the maximum sequence length. In the given code some modifications were performed like the addition of a synchronous Reset to set the seed value to the MISR to initialize to 0. An SO_TAP port was also defined to feed the response of the TRCUT into the MISR.

In the TRCUTwithMISR module the necessary ports and wires were initialized to wire up the MISR with the TRCUT. Also, a SIGN port was defined and wired with the 16th bit of the MISR, to get the signature of the test.

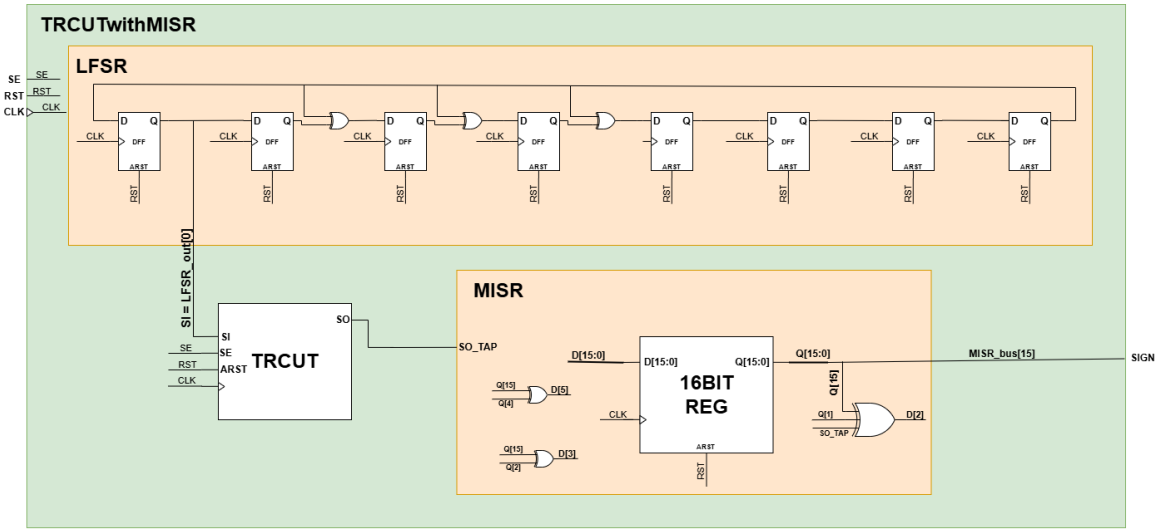
- **MISR Verilog**

```
module MISR(
    input clk,
    input rst, // ADDITION - reset signal to initialize LFSR with the seed value
    input so_tap, // ADDITION - input for the so of the trcut
    output reg [15:0] LFSR = 65535
);

wire feedback = LFSR[15];

parameter SEED = 16'h0000; // ADDITION - passing seed parameter

always @(posedge clk or posedge rst)
begin
            if (rst)
                        LFSR <= SEED; // ADDITION - set initial seed
            else begin
                LFSR[0] <= feedback;
                LFSR[1] <= LFSR[0];
                LFSR[2] <= LFSR[1] ^ feedback ^ so_tap;
                LFSR[3] <= LFSR[2] ^ feedback;
                LFSR[4] <= LFSR[3];
                LFSR[5] <= LFSR[4] ^ feedback;
                LFSR[6] <= LFSR[5];
                LFSR[7] <= LFSR[6];
                LFSR[8] <= LFSR[7];
                LFSR[9] <= LFSR[8];
                LFSR[10] <= LFSR[9];
                LFSR[11] <= LFSR[10];
                LFSR[12] <= LFSR[11];
                LFSR[13] <= LFSR[12];
                LFSR[14] <= LFSR[13];
                LFSR[15] <= LFSR[14];
            end
end
endmodule
```

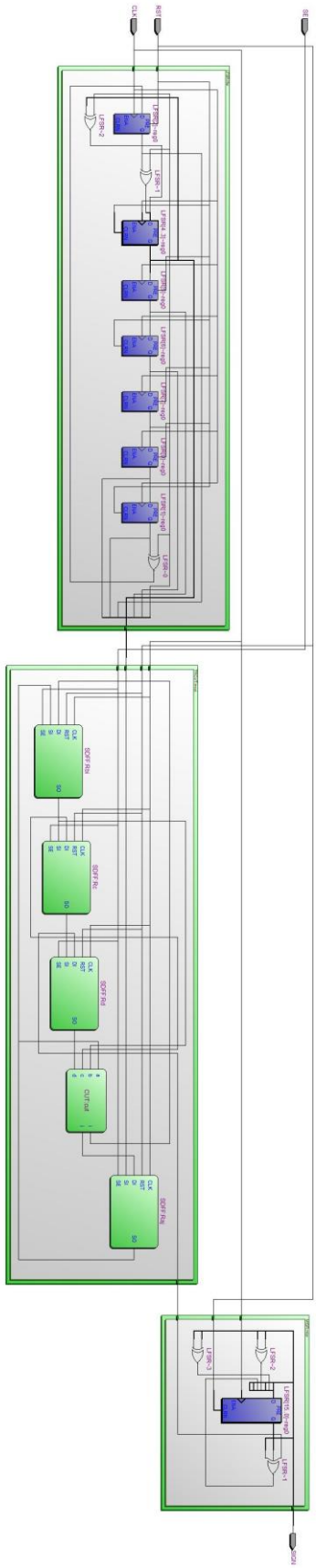
- **TRCUTwithMISR Verilog**

```
module TRCUTwithMISR(SE,RST,CLK,SIGN);

input SE,RST,CLK;
output SIGN;

wire SE,RST,CLK,SI,SIGN,SO; // wires to connect modules
wire [7:0] LFSR_out;              // bus array for the lfsr output
wire [15:0] MISR_bus;             // for the misr output

assign SI = LFSR_out[0];          // connecting one bit of the LFSR with TRCUT(SI)
assign SIGN = MISR_bus[15]; // assign SIGN to the last bit of the MISR

TRCUT trcut (.clk(CLK),.si(SI),.se(SE),.RST(RST),.so(SO));
LFSR lfsr (.clk(CLK),.rst(RST),.LFSR(LFSR_out));
MISR misr (.clk(CLK),.rst(RST),.so_tap(SO),.LFSR(MISR_bus));
endmodule
```

- **TRCUTwithMISR Expected Diagram**



- **TRCUTwithMISR Synthesized**



- **TRCUTwithMISR Testbench** (2.4)

The TRCUTwithMISR_tb testbench verifies the TRCUTwithMISR module using a BIST structure with pseudo-random test vectors. Initially, the system is reset (rst = 1) and all registers, including the sign_values array for storing the MISR signature, are initialized. A clock signal is generated with a period of 10 time units.

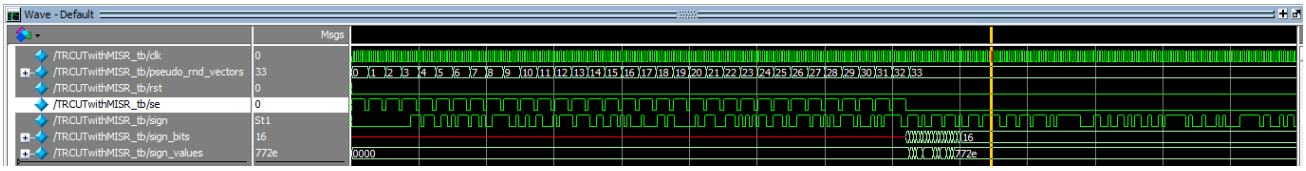
The testbench then runs a loop for 33 iterations (32 vectors + 1 to shift the last vector response out). In each iteration, scan enable (se) is set high for 4 clock cycles (#40) to shift a test vector into the scan chain, then low for 1 clock cycle (#10) to capture the response.

Optional fault injection lines are commented on in the code for simulating stuck-at and cosmic radiation bit errors. After all vectors are applied, the final 16-bit MISR output is got over 16 clock cycles and stored into the sign_values array for later analysis of the signature.

- **TRCUTwithMISR Verilog**

```
module TRCUTwithMISR_tb();

reg clk,rst,se;		// regs to drive trcutmisr
wire sign;			// wire to get signature out

reg [15:0] sign_values;		// reg array to hold signature bits

integer pseudo_rnd_vectors; // for-loop counter
integer sign_bits;

//		wiring up TRCUTwithMISR
TRCUTwithMISR trcutmisr (.SE(se),.RST(rst),.CLK(clk),.SIGN(sign));

initial begin
	rst = 1;	// reset regs of lfsr to seed and trcut regs to 0

	//		initilize array stuff
	sign_values = 0;
	se = 0;			// initilize scan-en
	clk = 0;		// initilize clk
	#1;

	// Forcing signals to simulate stuck-at faults
	// force trcutmisr.trcut.b = 1;

	rst = 0;		// setting rst to 0 to begin

	forever begin
		#5 clk =! clk;
	end
end

initial begin
	#1;		//		delay so there is sync with first initial block

	// for-loop to for the 32 pseudo_rnd_vectors coming from the misr
	for (pseudo_rnd_vectors = 0; pseudo_rnd_vectors<32 + 1; pseudo_rnd_vectors = pseudo_rnd_vectors + 1) begin
		se = 1;
		#40;	// 4 cycles for shifting the vector into the scan chain
		se = 0;
		#10;	// 1 cycles for capturing the observation points

		// Forcing signals to simulate cosmic radiation bit errors
		//if (pseudo_rnd_vectors == 21) begin
		//		force trcutmisr.trcut.Rbi.dffinstance.Q = 1;
		//		release trcutmisr.trcut.Rbi.dffinstance.Q;
		//end
	end
end

initial begin
	#1641	// wait for the 32nd vector to get into the chain and save the response
	for (sign_bits = 0; sign_bits<16; sign_bits = sign_bits + 1) begin
		sign_values[sign_bits] = sign;
		#10;
	end
end

endmodule
```

- **TRCUTwithMISR Waveform**



- **TRCUTwithMISR Waveform description**

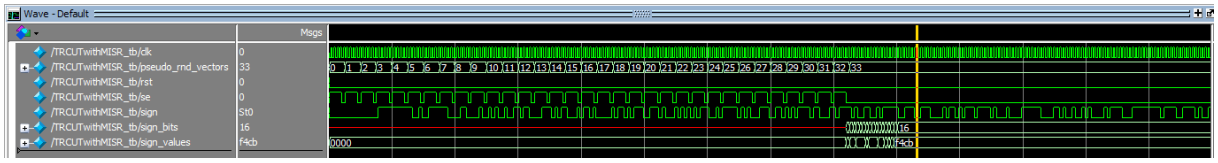
The shifting of the test vectors, generated from the LFSR, into the scan chain happens as previously described in the TRCUTwithLFSR waveform description. The testbench is configured to run for 32 pseudo-random vectors as seen from the 32 captures that happen in the waveform while the TRCUT response from each capture is fed into the so_tap MISR input for the generation of the test signature. After the 32 captures, we wait for additionally 16 clock cycles so the signature of the 16bit MISR is got at the sign output and saved in the sign_values array. At the end, we observe the value of the signature when the circuit under test has no faults **(0x772e).**

- **TRCUTwithMISR Simulating Stuck-At faults**

Stuck at faults can be simulated by forcing internal wires in the test bench at a certain stuck value and then checking that the produced signature differs from the correct one. On the waveform below, we can clearly see that the signature **(0xf4cb)** produced by the MISR, differs from the correct one, meaning there is an issue with the circuit under test.

```
// Forcing signals to simulate stuck-at faults
        force trcutmisr.trcut.b = 1;
```

- **TRCUTwithMISR Simulating Stuck-At faults – Wave response**



- **TRCUTwithMISR Simulating Cosmic Radiation Bit Errors**

Bit errors caused by cosmic radiation during the testing phase can be simulated by forcing random values of internal flip flops while the test vectors are applied, and then releasing them. Here again the signature should differ from the correct one. Similarly with the stuck-at fault wave, we can again see on the waveform below that the signature is not the correct one.

```
initial begin
        #1;         //          delay so there is sync with first initial block

        // for-loop to for the 32 pseudo_rnd_vectors coming from the misr
        for (pseudo_rnd_vectors = 0; pseudo_rnd_vectors<32 + 1; pseudo_rnd_vectors = pseudo_rnd_vectors + 1) begin
                se = 1;
                #40;        // 4 cycles for shifting the vector into the scan chain
                se = 0;
                #10;        // 1 cycles for capturing the observation points

                // Forcing signals to simulate cosmic radiation bit errors
                if (pseudo_rnd_vectors == 21) begin
                        force trcutmisr.trcut.Rbi.dffinstance.Q = 1;
                        release trcutmisr.trcut.Rbi.dffinstance.Q;
                end
        end
end
```

- **TRCUTwithLFSR Simulating Cosmic Radiation Bit Errors - Wave response**