Πανεπιστήμιο Ιωαννίνων

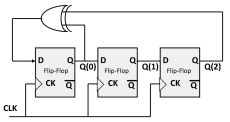
Τμήμα Μηχανικών Ηλεκτρονικών
Υπολογιστών & Πληροφορικής

Δοκιμή και Αξιοπιστία Ηλεκτρονικών Συστημάτων

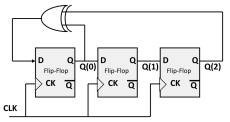
Εαρινό Εξάμηνο 2024

LFSRs - MISRs Testing

Χρήστος Δημητρέσης 4351 cs04351@uoi.gr



Περιεχόμενα



1 : TRCUT με χρήση LFSR.

1.1 : Υλοποίηση LFSR 8ου βαθμού.

Βασιζόμενοι στην υλοποίηση του TRUCT της άσκησης 1 , προσθέτουμε στην υλοποίηση μας εναν Linear-feedback shift register (LFSR) προκειμένου να δημιουργήσουμε ψευδοτυχαία διανυσματα εισόδου και εφαρμόζοντας τα στο TRCUT να ελέγξουμε την ορθή λειτουργία του.

Θα χρησιμοποιήσουμε ένα LFSR 8ου βαθμού , δηλαδή μηκούς 8 bit. Προκειμένου να εξασφαλίσουμε οτι το LFSR μας υλοποιεί πολυώνυμο πρώτου βαθμού και άρα παράγει την μεγίστη ακολουθία διανυσμάτων που μπορεί να παραχθεί από 8 bit (πλην ενός -> του “00000000”) χρησιμοποιούμε την εφαρμογή LFSR_TESTBENCH. Επιλέχθηκε μια τυχαία αρχική κατάσταση διάφορη του μηδενός η οποία αποτελεί και το seed του LFSR .

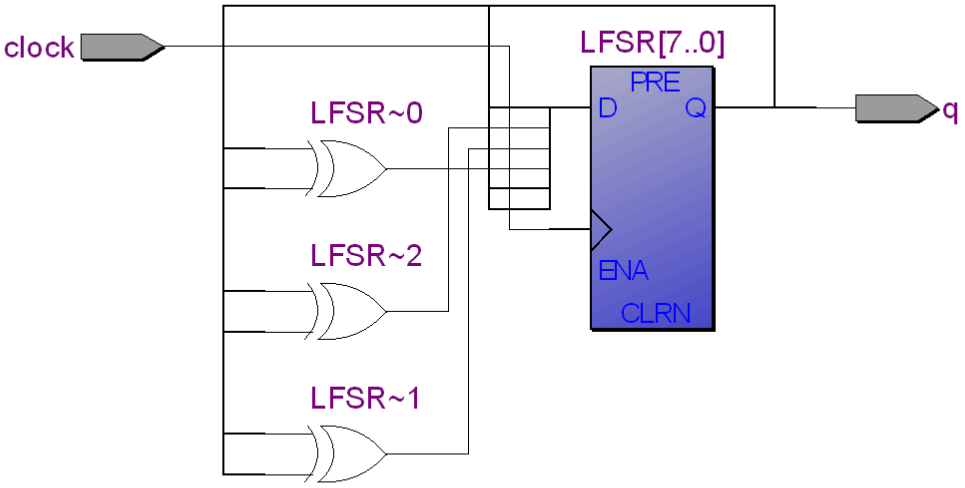
Τονίζεται ότι το LFSR επιτελεί την λειτουργία του κάνοντας **αριστερή ολίσθηση** με βάση την υλοποίηση μας και άρα η έξοδος του είναι το περισσότερο σημαντικό bit του διανύσματος που περιέχει.

Ο κώδικας code_01 υλοποιει το περιγραφεν LFSR σε γλώσσα VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;

entity LFSR_8_BITS is
   port(clock:in std_logic;   q:out std_logic);
end;

architecture RTL of LFSR_8_BITS is
   signal LFSR: std_logic_vector(7 downto 0) := "01011100"; --random seed
begin
   process(clock)
       variable feedback: std_logic;
   begin
       if clock'EVENT and clock='1' then
           feedback := LFSR(7);
           LFSR(0) <= feedback;
           LFSR(1) <= LFSR(0);
           LFSR(2) <= LFSR(1) xor feedback;
           LFSR(3) <= LFSR(2) xor feedback;
           LFSR(4) <= LFSR(3) xor feedback;
           LFSR(5) <= LFSR(4);
           LFSR(6) <= LFSR(5);
           LFSR(7) <= LFSR(6);
       end if;
       q <= LFSR(7);
   end process;
end;
```

code_01

Συνθέτοντας το παραπάνω κύκλωμα με την χρήση του quartus προκύπτει η παρακάτω υλοποίηση σε επιπεδο RTL screen_01 , πράγμα που επιβεβαιωνει οτι δημιουργήθηκε η ζητούμενη λειτουργικότητα .



screen_01

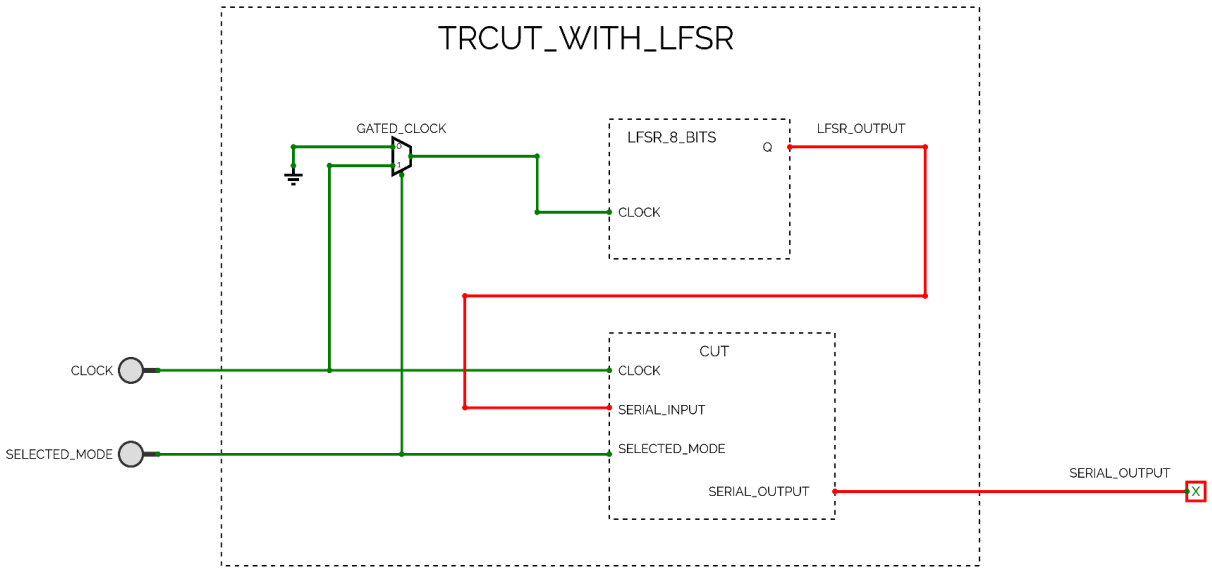
1.2 : Υλοποίηση TRCUT_WITH_LFSR

Ο κώδικας code_02 υλοποιεί ένα κύκλωμα ονόματι TRCUT_WITH_LFSR το οποίο περιέχει ως component το TRCUT της άσκησης 1 συνδεδεμένο κατάλληλα με το LFSR που δημιουργήσαμε προηγουμένως.

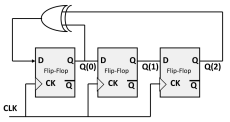
Συγκεκριμένα η σειριακή έξοδος του LFSR συνδέεται με την σειριακή είσοδο του TRCUT. Λόγω του ότι στον κύκλο ρολογιού που το TRCUT βρίσκεται σε λειτουργία capture mode το LFSR θα συνέχιζε την παραγωγή διανυσμάτων και θα χανόταν ενα παραχθέν bit, επιλέχθηκε κάθε φορά που βρισκόμαστε σε capture_mode στο TRCUT να κόβεται το ρολόι στο LFSR με την χρήση ενος πολυπλέκτη και άρα να διατηρείται η τιμή που περιέχει για έναν ακόμα κυκλό.

Και χωρίς αυτήν την πρόνοια δεν θα υπάρχει κάποιο λογικό λάθος στην διαδικασία ελέγχου μας , όμως με την προσθήκη αυτή εξασφαλίζουμε ότι το κάθε διάνυσμα που δινεται ως είσοδος στο TRCUT είναι αποτέλεσμα της ακολουθίας του πολυωνύμου του LFSR και όχι κάποια μίξη λόγω του ενός bit που θα χανόταν κάθε φορά στον κύκλο του capture.

Το desing_01 αποτελεί την λογική σχεδίαση του κυκλώματος απο εμας σε αρχικό στάδιο.

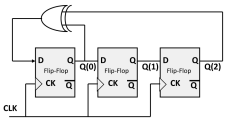


desing_01

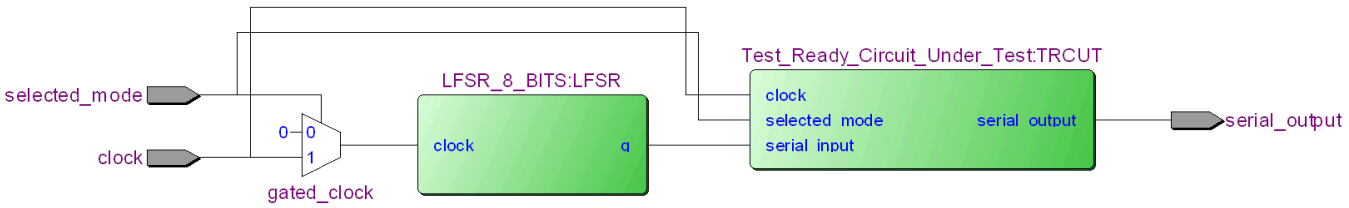



```
library IEEE;
use IEEE.std_logic_1164.all;

entity TRCUT_with_LFSR is port (       clock : in std_logic;
                                       selected_mode : in std_logic;
                                       serial_output : out std_logic);
end entity;

architecture noFault of TRCUT_with_LFSR is

   component LFSR_8_BITS is
       port(
           clock : in std_logic;
           q     : out std_logic
       );
   end component;


   component Test_Ready_Circuit_Under_Test is
       port (
           clock           : in std_logic;
           serial_input    : in std_logic;
           selected_mode   : in std_logic;
           serial_output   : out std_logic
       );
   end component;

   signal lfsr_output : std_logic := 'X';
   signal gated_clock : std_logic ;

begin


   LFSR : LFSR_8_BITS
       port map(
           clock => gated_clock,
           q     => lfsr_output
       );

   TRCUT: entity work.Test_Ready_Circuit_Under_Test(noFault)
       port map (
           clock           => clock,
           serial_input    => lfsr_output,
           selected_mode   => selected_mode,
           serial_output   => serial_output
       );


   process(clock, selected_mode) is
   begin
       if selected_mode = '1' then gated_clock <= clock; else gated_clock <=
'0'; end if;

   end process;
end architecture noFault;
```

code_02



Συνθέτοντας το παραπάνω κύκλωμα με την χρήση του quartus προκύπτει η παρακάτω υλοποίηση σε επιπεδο RTL screen_02 η οποία ταυτίζεται με την αρχιτεκτονική που σχεδιάσαμε εμείς σε αρχικό στάδιο (desing_01) και αρα επιβεβαιώνει την ορθότητα του κώδικα VHDL.



screen_02

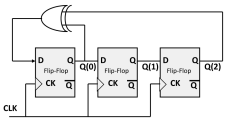
***Αρχιτεκτονικές του TRCUT και του CUT

Επισημαίνεται οτι προκειμενου να προσομοιώσουμε τα faut injections και τα error injections στα ερωτήματα στην συνέχεια χρειάζεται να επέμβουμε στην αρχιτεκτονική του CUT διότι αλλάζουμε την συμπεριφορά του.
Άρα δημιουργήθηκαν διαφορετικές αρχιτεκτονικές για το CUT και ανάλογα το σενάριο που θέλουμε να εξετάσουμε επιλέγουμε την αντίστοιχη.
Στο τρέχον ερώτημα στο οποίο δεν προσομοιώνουμε κάποιο σφάλμα του CUT χρησιμοποιείται η αρχιτεκτονική του CUT (noFalut) η οποία ταυτίζεται με την αρχιτεκτονική της άσκησης 1 .
Οι υπόλοιπες αρχιτεκτονικές στα εξηγηθούν στην συνέχεια.

Πρακτικά μόνο στο κύκλωμα CUT υπάρχουν αλλαγές αλλά δεδομένου ότι το TRCUT έχει ως component του το CUT είμαστε αναγκασμένοι να δημιουργήσουμε διαφορετικές αρχιτεκτονικές για τα σενάρια που θέλουμε να προσομοιώσουμε και για το TRCUT.

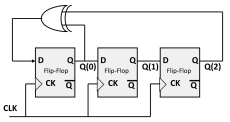
Παρατίθενται παρακάτω οι αρχιτεκτονικές του CUT αλλά και του TRCUT_LFSR αντίστοιχα.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Circuit_Under_Test is port(  a : in std_logic;
                                    b : in std_logic;
                                    c : in std_logic;
                                    d : in std_logic;
                                    i : out std_logic;
                                    j : out std_logic);
end entity;

architecture noFault of Circuit_Under_Test is

   signal e, f, g, h : std_logic;

begin

   -- First Level --
   e <= a xor b;
   f <= c xor d;
   g <= a xor c;
   h <= b xor d;

   -- Second Level --
   i <= e and f;
   j <= g or h;

end architecture noFault;


architecture stuckAtOne of Circuit_Under_Test is

   signal e, f, g, h : std_logic;

begin
```

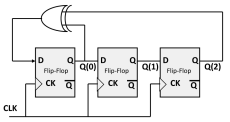



```
    -- First Level --
    e <= a xor b;
    f <= c xor '1';
    g <= a xor c;
    h <= b xor d;

    -- Second Level --
    i <= e and f;
    j <= g or h;

end architecture stuckAtOne;


architecture stuckAtZero of Circuit_Under_Test is

    signal e, f, g, h : std_logic;

begin

    -- First Level --
    e <= a xor b;
    f <= c xor d;
    g <= a xor c;
    h <= b xor d;

    -- Second Level --
    i <= e and f;
    j <= g or '0';

end architecture stuckAtZero;
```

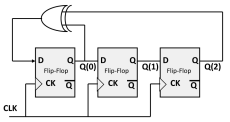
cut



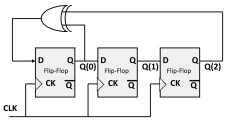

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Test_Ready_Circuit_Under_Test is port(   clock         : in std_logic;
                                                serial_input    : in std_logic;
                                                selected_mode   : in std_logic;
                                                serial_output   : out std_logic);
end entity;


architecture noFault of Test_Ready_Circuit_Under_Test is

   component scanDFF is
       port (
           clock : in std_logic;
           d_in : in std_logic;
           serial_input : in std_logic;
           q_out : out std_logic;
           selected_mode : in std_logic
       );
   end component;

   component Circuit_Under_Test is
       port (
           a, b, c, d : in std_logic;
           i, j : out std_logic
       );
   end component;


   signal i_signal, j_signal, dff_q_out_1, dff_q_out_2, dff_q_out_3, dff_q_out_4
: std_logic := 'X';


begin

   CUT : entity work.Circuit_Under_Test(noFault)
       port map (
           a => dff_q_out_1,
           b => dff_q_out_2,
           c => dff_q_out_3,
           d => dff_q_out_4,
           i => i_signal,
           j => j_signal
       );

   scanDFF_S1 : scanDFF
       port map (
           clock => clock,
           d_in => j_signal,
           serial_input => serial_input,
           q_out => dff_q_out_1,
           selected_mode => selected_mode
       );

   scanDFF_S2 : scanDFF
       port map (
```



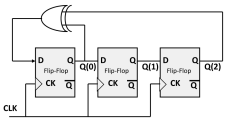

```
            clock => clock,
            d_in => i_signal,
            serial_input => dff_q_out_1,
            q_out => dff_q_out_2,
            selected_mode => selected_mode
        );

    scanDFF_S3 : scanDFF
        port map (
            clock => clock,
            d_in => dff_q_out_3,
            serial_input => dff_q_out_2,
            q_out => dff_q_out_3,
            selected_mode => selected_mode
        );

    scanDFF_S4 : scanDFF
        port map (
            clock => clock,
            d_in => dff_q_out_4,
            serial_input => dff_q_out_3,
            q_out => dff_q_out_4,
            selected_mode => selected_mode
        );

    serial_output <= dff_q_out_4;

end architecture noFault;

---------------------------
---------------------------
---------------------------

architecture stuckAtOne of Test_Ready_Circuit_Under_Test is

    component scanDFF is
        port (
            clock : in std_logic;
            d_in : in std_logic;
            serial_input : in std_logic;
            q_out : out std_logic;
            selected_mode : in std_logic
        );
    end component;

    component Circuit_Under_Test is
        port (
            a, b, c, d : in std_logic;
            i, j : out std_logic
        );
    end component;


    signal i_signal, j_signal, dff_q_out_1, dff_q_out_2, dff_q_out_3, dff_q_out_4
: std_logic := 'X';
```




```
begin

   CUT : entity work.Circuit_Under_Test(stuckAtOne)
       port map (
           a => dff_q_out_1,
           b => dff_q_out_2,
           c => dff_q_out_3,
           d => dff_q_out_4,
           i => i_signal,
           j => j_signal
       );

   scanDFF_S1 : scanDFF
       port map (
           clock => clock,
           d_in => j_signal,
           serial_input => serial_input,
           q_out => dff_q_out_1,
           selected_mode => selected_mode
       );

   scanDFF_S2 : scanDFF
       port map (
           clock => clock,
           d_in => i_signal,
           serial_input => dff_q_out_1,
           q_out => dff_q_out_2,
           selected_mode => selected_mode
       );

   scanDFF_S3 : scanDFF
       port map (
           clock => clock,
           d_in => dff_q_out_3,
           serial_input => dff_q_out_2,
           q_out => dff_q_out_3,
           selected_mode => selected_mode
       );

   scanDFF_S4 : scanDFF
       port map (
           clock => clock,
           d_in => dff_q_out_4,
           serial_input => dff_q_out_3,
           q_out => dff_q_out_4,
           selected_mode => selected_mode
       );

   serial_output <= dff_q_out_4;

end architecture stuckAtOne;


--------------------------
--------------------------
--------------------------
architecture stuckAtZero of Test_Ready_Circuit_Under_Test is
```

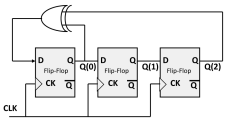



```
    component scanDFF is
        port (
            clock : in std_logic;
            d_in : in std_logic;
            serial_input : in std_logic;
            q_out : out std_logic;
            selected_mode : in std_logic
        );
    end component;

    component Circuit_Under_Test is
        port (
            a, b, c, d : in std_logic;
            i, j : out std_logic
        );
    end component;


    signal i_signal, j_signal, dff_q_out_1, dff_q_out_2, dff_q_out_3, dff_q_out_4
: std_logic := 'X';


begin

    CUT : entity work.Circuit_Under_Test(stuckAtZero)
        port map (
            a => dff_q_out_1,
            b => dff_q_out_2,
            c => dff_q_out_3,
            d => dff_q_out_4,
            i => i_signal,
            j => j_signal
        );

    scanDFF_S1 : scanDFF
        port map (
            clock => clock,
            d_in => j_signal,
            serial_input => serial_input,
            q_out => dff_q_out_1,
            selected_mode => selected_mode
        );

    scanDFF_S2 : scanDFF
        port map (
            clock => clock,
            d_in => i_signal,
            serial_input => dff_q_out_1,
            q_out => dff_q_out_2,
            selected_mode => selected_mode
        );

    scanDFF_S3 : scanDFF
        port map (
            clock => clock,
            d_in => dff_q_out_3,
```




```
            serial_input => dff_q_out_2,
            q_out => dff_q_out_3,
            selected_mode => selected_mode
        );

    scanDFF_S4 : scanDFF
        port map (
            clock => clock,
            d_in => dff_q_out_4,
            serial_input => dff_q_out_3,
            q_out => dff_q_out_4,
            selected_mode => selected_mode
        );

    serial_output <= dff_q_out_4;

end architecture stuckAtZero;
```

trcut_with_LFSR

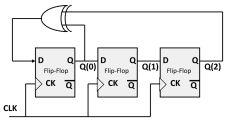


1.2 : Υλοποίηση TRCUT_WITH_LFSR_Testbench

Ο κώδικας code_03 δημιουργεί ένα testbench σε γλώσσα VHDL προκειμένου να ελεγχθεί η ορθή υλοποίηση του κυκλώματος TRCUT_WITH_LFSR

Συγκεκριμένα φορτώνουμε σειριακά 32 ψευδοτυχαία διανυσματα των 4ων bit δημιουργημένα απο το LFSR ως είσοδο στο TRCUT και αποθηκευουμε τις αποκρίσεις σε ένα σήμα ονόματι TRCUT_RESPONSES.

```
LIBRARY ieee  ;
LIBRARY std  ;
USE ieee.std_logic_1164.all  ;
USE ieee.std_logic_arith.all  ;
USE ieee.std_logic_textio.all  ;
USE ieee.STD_LOGIC_UNSIGNED.all  ;
USE ieee.std_logic_unsigned.all  ;
USE std.textio.all  ;

entity TRCUT_WITH_LFSR_testbench is
end entity;


architecture testbench of TRCUT_WITH_LFSR_testbench is
   component TRCUT_with_LFSR is
       port (
           clock          : in std_logic;
           selected_mode  : in std_logic;
           serial_output  : out std_logic
       );
   end component;




   signal clock  : std_logic := '1';
   signal serial_output : std_logic := 'X';
   signal selected_mode : std_logic  := '1';


   type std_logic_vector_array is array (0 to 31) of std_logic_vector(3 downto
0);
   signal TRCUT_RESPONSES: std_logic_vector_array;



begin

   UUT : entity work.TRCUT_with_LFSR(noFault)
       port map (
           clock          => clock,
           selected_mode  => selected_mode,
           serial_output  => serial_output
```




```
    );


clock <= not clock after 2.5 ns;



-- scanTesting
process is begin

    wait for 5.0 ns;

    for i in 0 to 31 loop

        --load s1
        wait for 5 ns;

        --load s2;
        wait for 5 ns;

        -- load s3;
        wait for 5 ns;

        --load s4
        wait for 2.5 ns;

        selected_mode <= '0';
        wait for 5 ns;

        selected_mode <= '1';
        wait for 2.5 ns;


    end loop;
    wait;
end process;
-- Responses Capture
process is begin
    wait for 30 ns;


    for i in 0 to 31 loop
        for j in 0 to 3 loop
            TRCUT_RESPONSES(i)(j) <= serial_output;
            wait for 5 ns;
        end loop;
        --ignore last one bit from serial
        wait for 5 ns;
    end loop;

    wait;
end process;

end architecture;
```

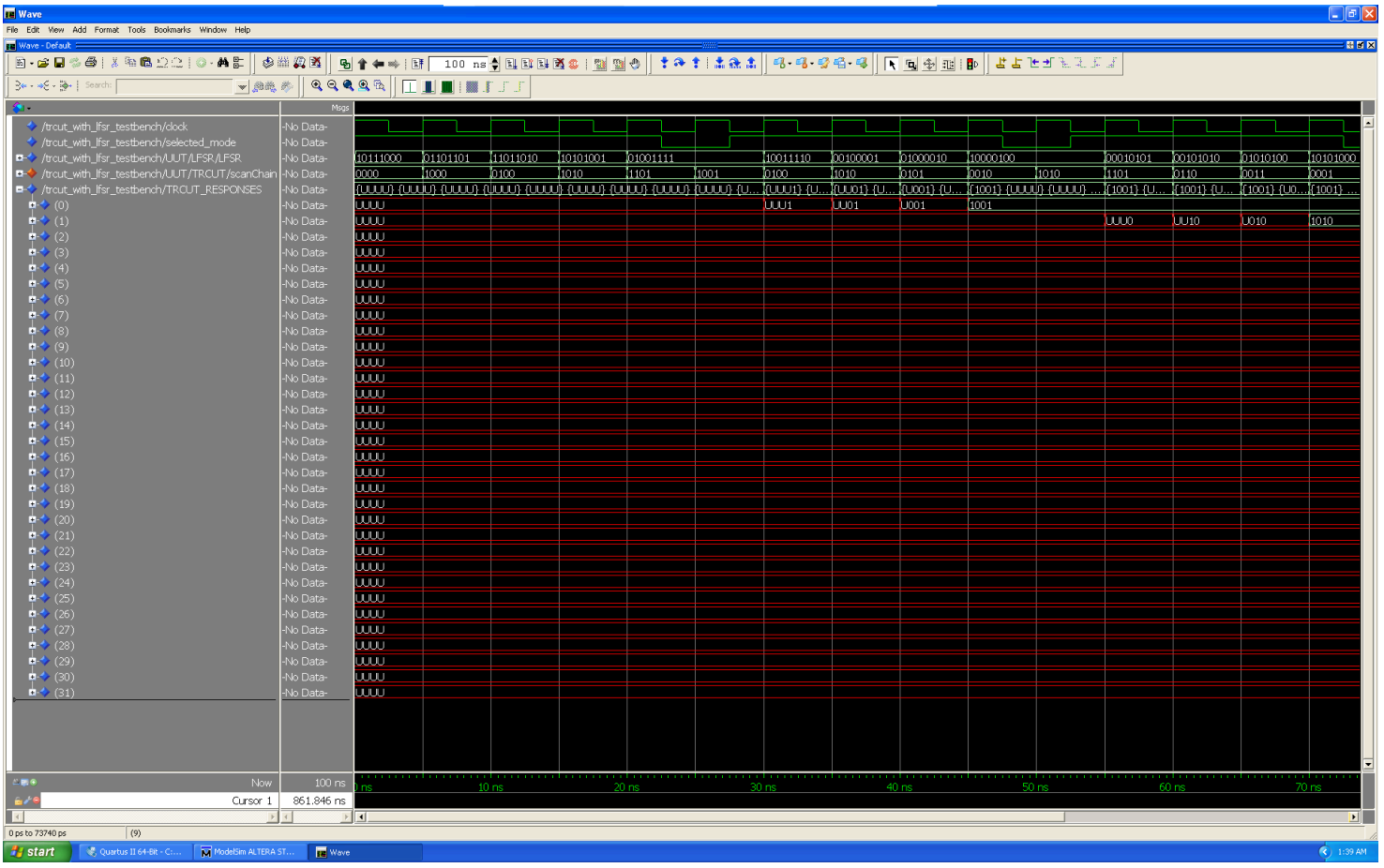
code_03

Στιγμιότυπα και επεξήγηση του Testbench_WITH_LFSR

Σήματα της προσομοίωσης :

- **Clock** : Το ρολόι του κυκλώματος μας το οποίο τίθεται σε περιόδο t = 5 ns
- **Selected_mode** : Σήμα το οποίο καθορίζει την εσωτερική λειτουργία του TRCUT. Όταν τίθεται στο λογικό ‘1’ η αλυσίδα σάρωσης (Scan Chain) του TRCUT επιτελεί την λειτουργία της δεξιάς ολίσθησης. Όταν τίθεται στο λογικό ‘0’ το TRCUT κάνει capture τις τιμές του CUT και τις αποθηκεύει στα Flip-Flop της αλυσίδας σάρωσης.
- **Serial_Output** : Το σήμα το οποίο αντιστοιχεί στην έξοδο του TRCUT. Δηλαδή η τιμή του δεξιότερου Flip-Flop της αλυσίδας σάρωσης.
- **TRCUT_RESPONES** : Πίνακας σημάτων στον όποιον αποθηκεύονται οι αποκρίσεις του TRCUT στα διανυσματα εισόδου που δέχτηκε από το LFSR. Δίνουμε 32 διανύσματα ως είσοδο για αυτο και ο πίνακας αποτελείται από 32 σήματα των 4ων bit. Στα πρώτα 2 bit αντιστοιχεί η απόκριση του TRCUT και τα τελευταία 2 είναι τα bit του διανύσματος εισόδου που παρέμειναν απαράλλαχτα μέσα στην αλυσίδα δεδομένου ότι οι έξοδοι του TRCUT είναι μόνο 2 bit. .

Παρατίθενται στιγμιότυπα της εξομοίωσης με αναφορά στο αντιστοιχό σημείου του code_03



Στιγμιότυπο 1

Στο στιγμιότυπο_1 φαίνεται η αρχή της εξομοίωσης μας. Η εξομοίωση ξεκινάει με το LFSR αρχικοποιημένο στο default seed που του έχουμε δώσει και με την αλυσίδα σάρωσης να έχει όλα τα Flip-Flop της αρχικοποιημένα στο λογικό '0'. Μετα απο χρόνο t = 5 ns ξεκινάμε να φορτώνουμε ως είσοδο στο TRCUT τιμές από την έξοδο του LFSR λειτουργώντας ακριβώς με την ίδια λογική όπως στην άσκηση 1.

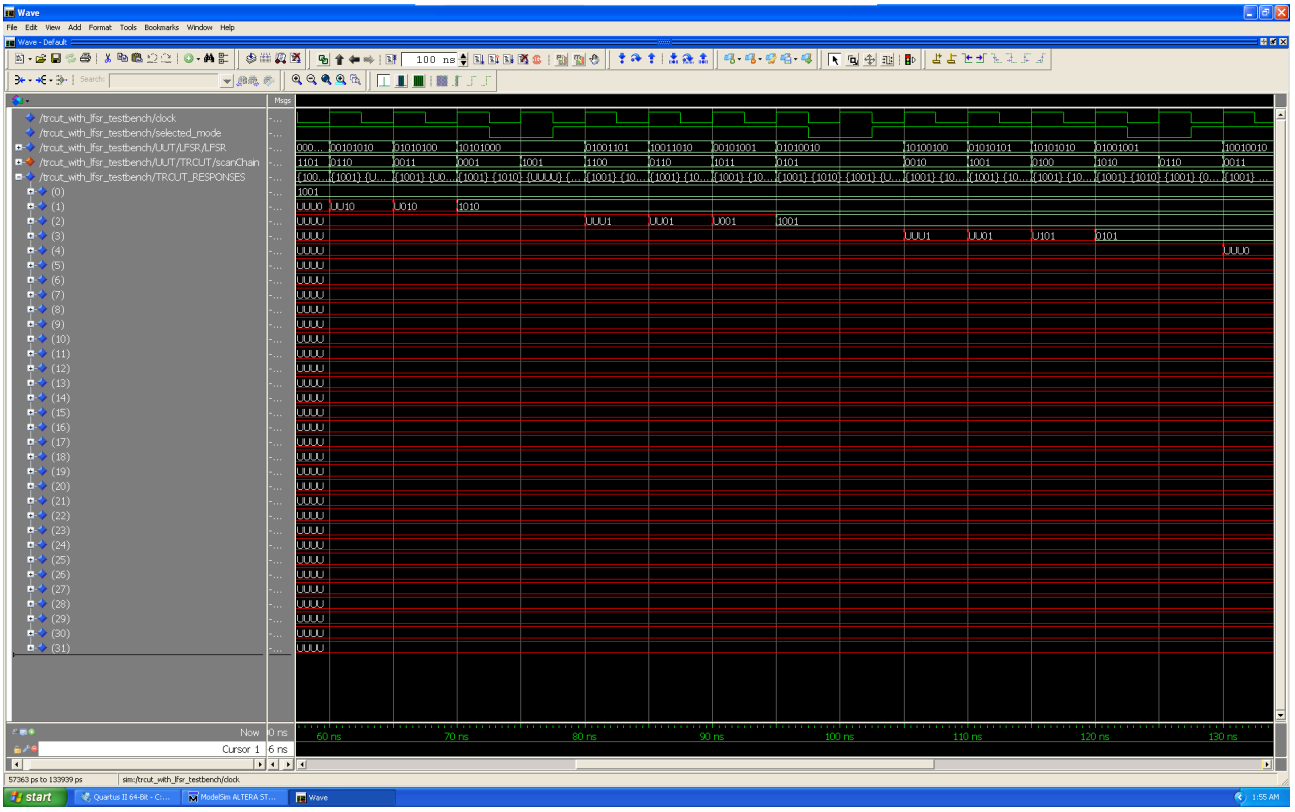
Στον κώδικα code_03 στο σημειο που υπάρχει το σχολιο - - scanTesting ειναι το σημειο που στην εξομοίωση μας ξεκινάνε να φορτώνονται τιμές στο TRCUT την χρονική στιγμή t = 5 ns.

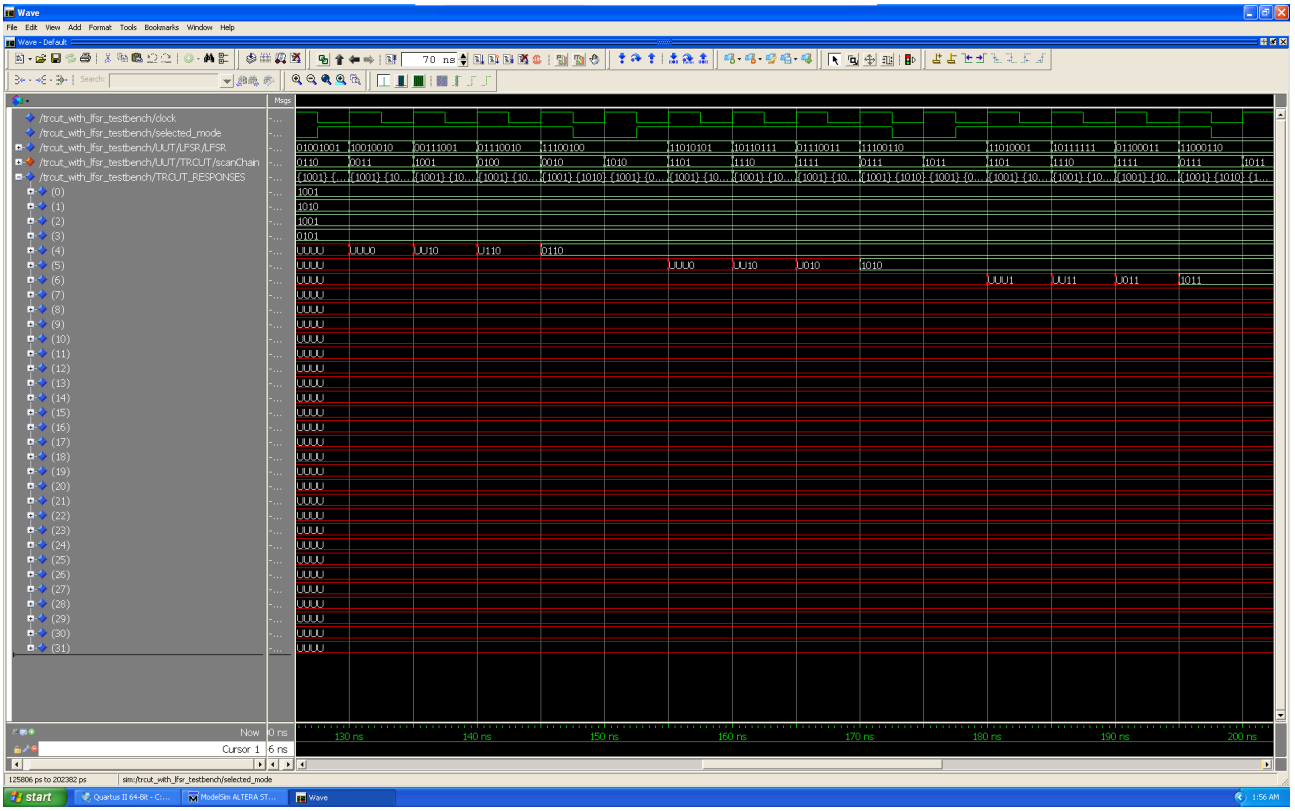
Οι αποκρίσεις του TRCUT ως αποτέλεσμα των διανυσμάτων που δέχεται ως είσοδο από την έξοδο του LFSR αποθηκεύονται στον πίνακα σημάτων TRCT_RESPONSES με κατάλληλο χρονισμό ακολουθώντας την λογική της άσκησης 1.

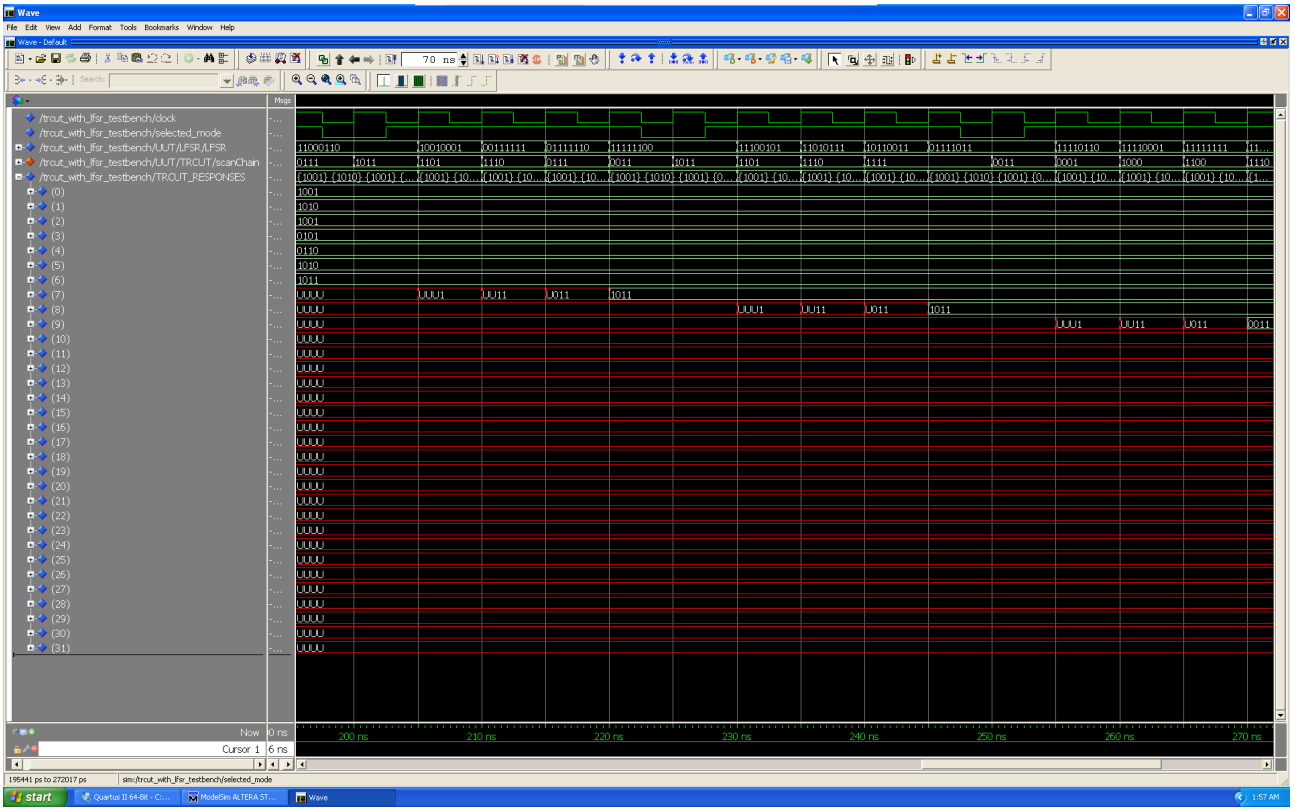
Στον κώδικα code_03 στο σημείο που υπάρχει το σχολιο - - responses Capture βρισκεται ο κώδικας που επιτελεί την συγκεκριμένη λειτουργία.

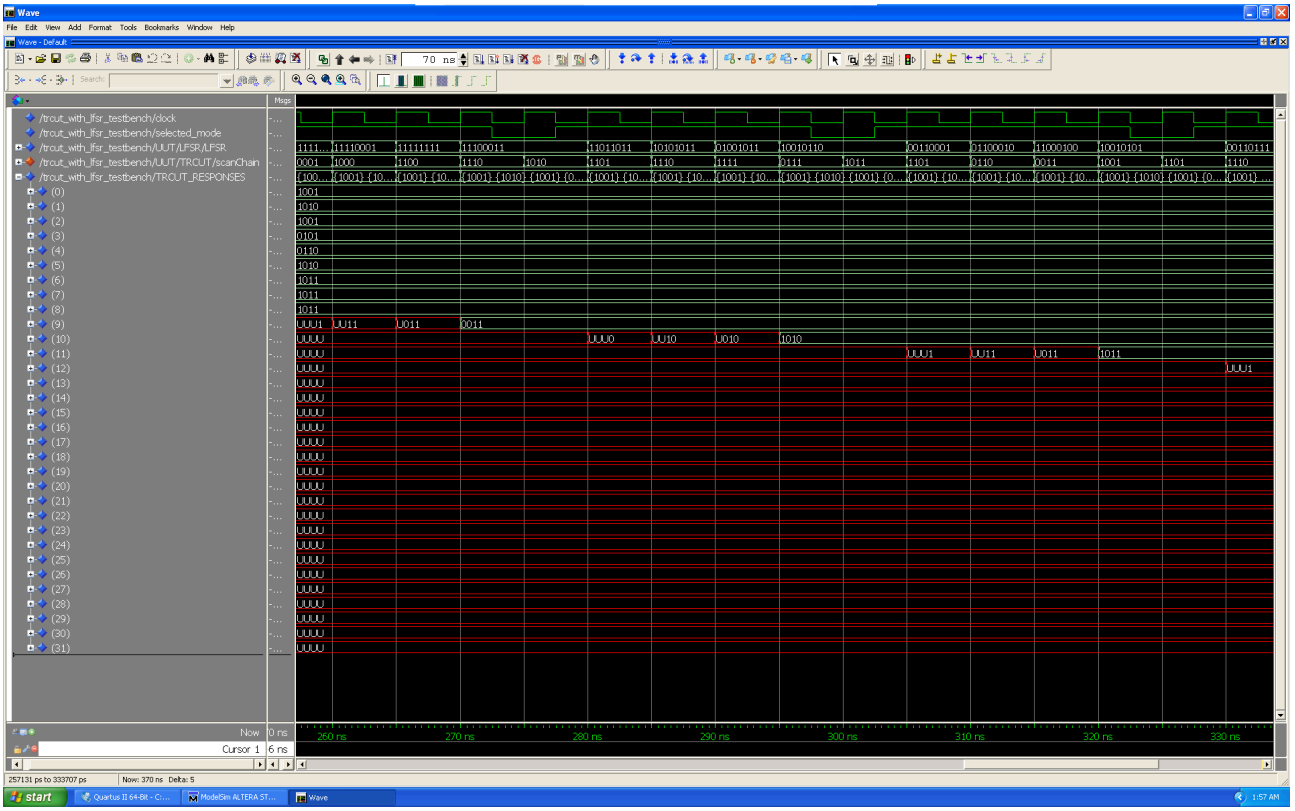
Υπενθυμίζεται πως επιλέχθηκε κάθε φορά που βρισκόμαστε στην λειτουργία Capture το ρόλοι να κόβεται στο LFSR και διατηρεί την τιμή του , πράγμα που είναι εμφανές και στην εξομοίωση.

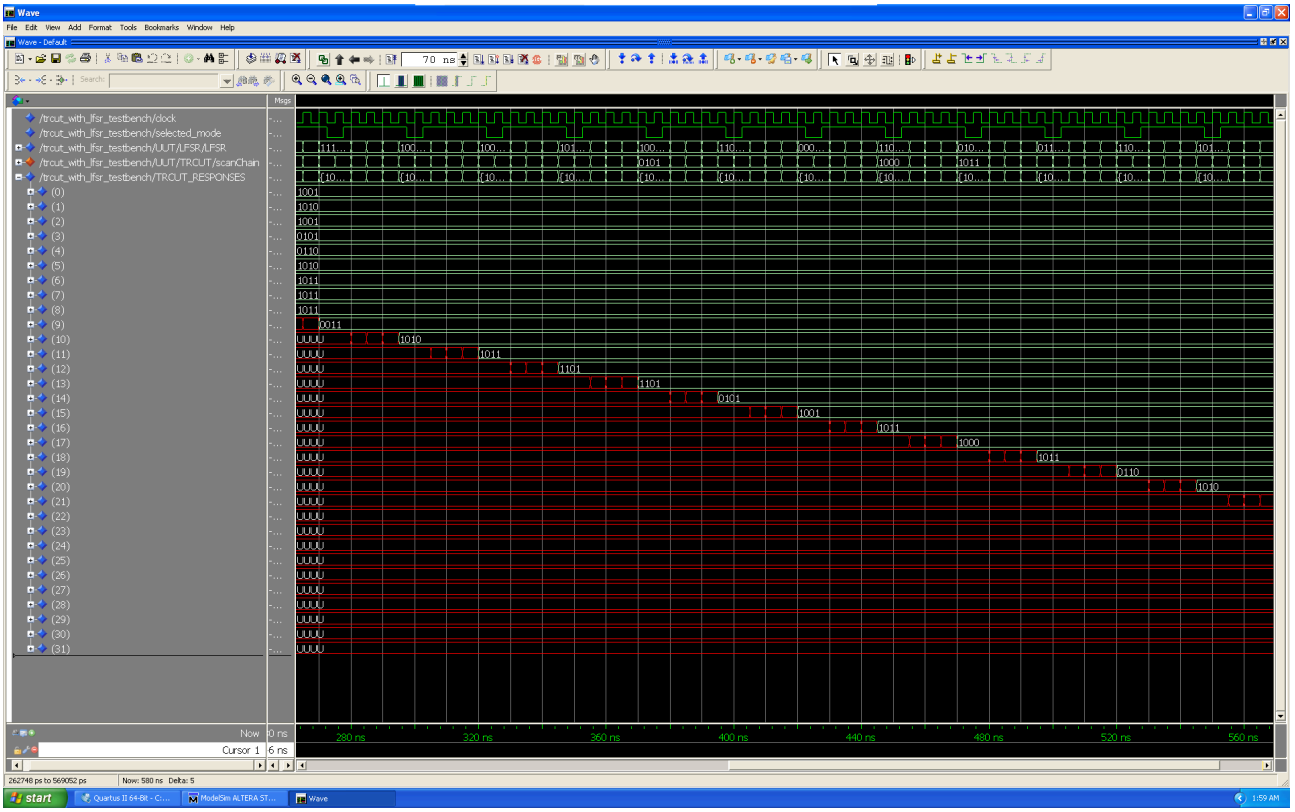
Στην συνέχεια παρατίθενται στιγμιοτυπα της εξομοίωσης με την σειρά (το ένα η χρονική συνέχεια του άλλου) μέχρι να φορτωθούν και τα 32 διανύσματα εισόδου στο TRCUT και να φτάσει στο τέλος της.

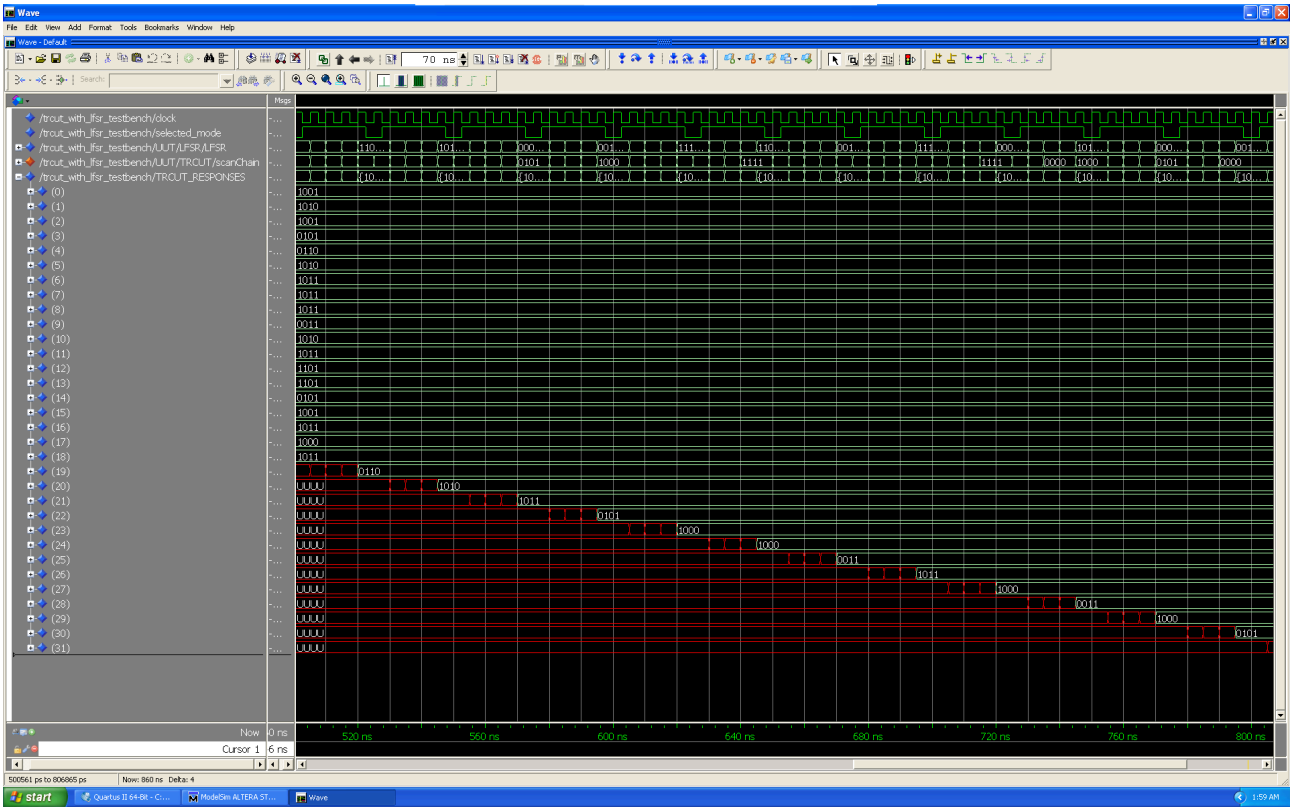


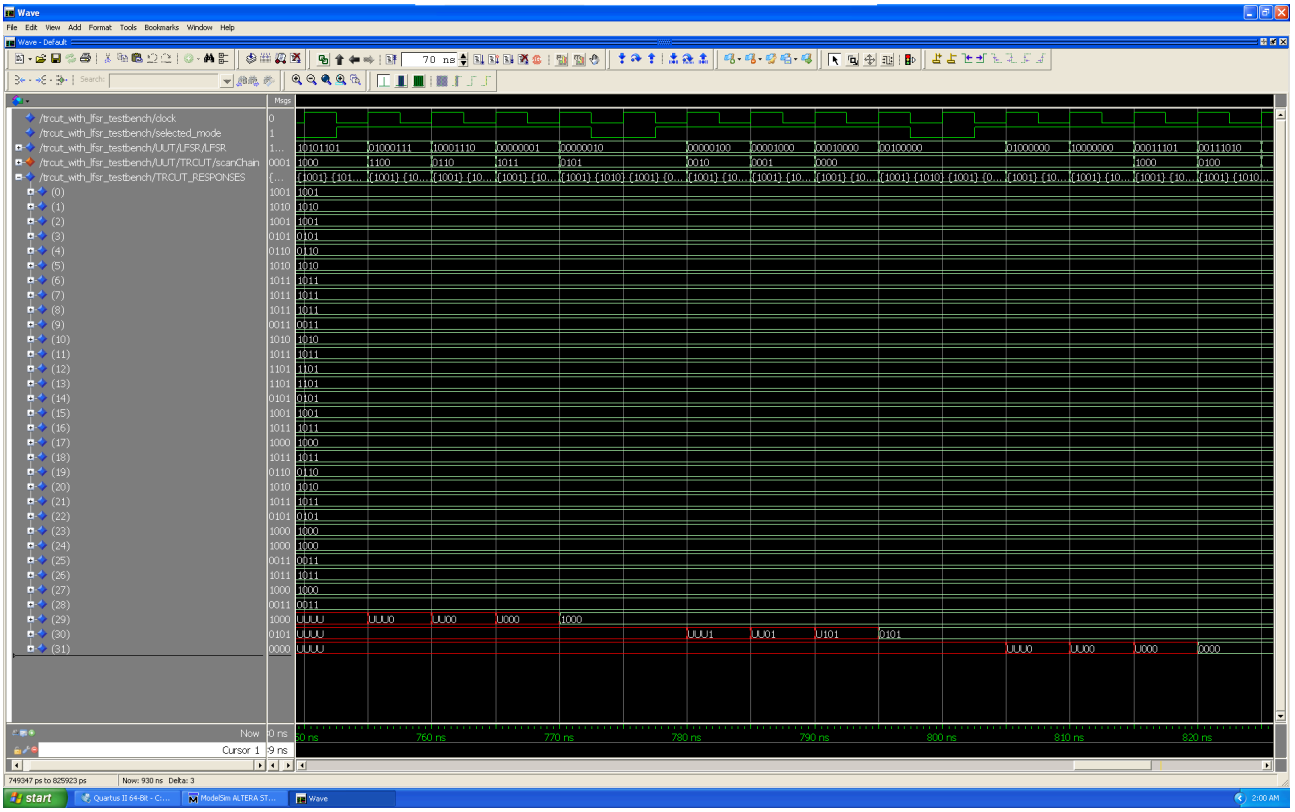

Wave
/trcut_with_lfsr_testbench/clock
/trcut_with_lfsr_testbench/selected_mode
/trcut_with_lfsr_testbench/UUT/LFSR/LFSR
/trcut_with_lfsr_testbench/UUT/TRCUT/scanChain
/trcut_with_lfsr_testbench/TRCUT_RESPONSES
Now
Cursor 1
130 ns
140 ns
150 ns
160 ns
170 ns
180 ns
190 ns
200 ns

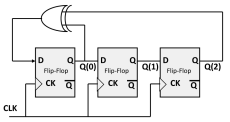

Wave
/trcut_with_lfsr_testbench/clock
/trcut_with_lfsr_testbench/selected_mode
/trcut_with_lfsr_testbench/UUT/LFSR/LFSR
/trcut_with_lfsr_testbench/UUT/TRCUT/scanChain
/trcut_with_lfsr_testbench/TRCUT_RESPONSES
Now
Cursor 1
200 ns
210 ns
220 ns
230 ns
240 ns
250 ns
260 ns
270 ns


Wave
/trcut_with_lfsr_testbench/clock
/trcut_with_lfsr_testbench/selected_mode
/trcut_with_lfsr_testbench/UUT/LFSR/LFSR
/trcut_with_lfsr_testbench/UUT/TRCUT/scanChain
/trcut_with_lfsr_testbench/TRCUT_RESPONSES
Now
Cursor 1


Wave
/trcut_with_lfsr_testbench/clock
/trcut_with_lfsr_testbench/selected_mode
/trcut_with_lfsr_testbench/UUT/LFSR/LFSR
/trcut_with_lfsr_testbench/UUT/TRCUT/scanChain
/trcut_with_lfsr_testbench/TRCUT_RESPONSES
Now
Cursor 1







Συμπέρασμα

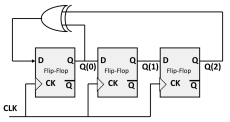
Γίνεται εμφανές ότι χωρίς πρόσβαση στα διανυσματα εισόδου που δέχεται ως είσοδο το TRCUT δεν είναι δυνατόν να γνωρίζουμε αν οι αποκρίσεις του CUT είναι σωστές.

Δηλαδή, αν θέλαμε να κάνουμε επαλήθευση της ορθής λειτουργίας του CUT χρησιμοποιώντας τις αποκρίσεις που αποθηκεύτηκαν στο σήμα **TRCUT_RESPONSES** θα έπρεπε να αποθηκεύουμε σε ένα άλλο σήμα αντίστοιχα και τα διανυσματα εισόδου του TRCUT , δηλαδή την έξοδο του LFSR.

Στην συνέχεια θα προβαίναμε σε επαλήθευση εξετάζοντας brute force τις αποκρίσεις σε σχέση με τα διανύσματα εισοδου ακολουθόντας ακριβώς την ίδια λογική με την άσκηση 1.

Η παραπάνω πρακτική σε κυκλώματα που δέχονται εκατομμύρια διανύσματα ελέγχου και όχι απλα έναν διαχειρίσιμο αριθμό όπως στην περίπτωση μας που είναι 32 , είναι μη εφαρμόσιμη.

Προβλήματα αποθηκευτικού χώρου αλλά και χρόνου επαλήθευσης μας οδηγούν σε διαφορετικές τεχνικές όπως αυτή της δημιουργίας μιας υπογραφής (golden signature) του κυκλώματος TRCUT μέσω της χρήσης ενός MISR και ελέγχο αυτής ως κριτήριο κάθε φόρα προκειμένου να καθοριστεί η ορθότητα ή οχι του κυκλώματος.



2 : TRCUT με χρήση LFSR και MISR

Επεκτείνοντας ένα βήμα ακόμα την αρχιτεκτονική του TRCUT και για να λύσουμε το πρόβλημα που περιγράφηκε στο προηγούμενο ερώτημα σχετικά με την αδυναμία επιβεβαίωσης της ορθής λειτουργίας του TRCUT χωρις να αποθηκεύουμε τα διανύσματα εισόδου του, θα προσθέσουμε ενα MISR στην αρχιτεκτονική μας προκειμένου να δημιουργείται μια υπογραφη του κυκλώματος (golden signature) και βάση αυτής να κρίνεται η ορθότητα του.

2.1 : Υλοποίηση MISR

Θα υλοποιήσουμε ενα MISR 16^{ου} βαθμού , δηλαδή ένα LFSR 16 bit το οποίο μέσω ενός xorTap θα δέχεται ως είσοδο της αποκρίσεις του TRCUT. Επιλέγουμε το xorTap να μην είναι μέρος της αρχιτεκτονικής του MISR αλλά να είναι εξωτερικό στοιχείο που θα βρίσκεται μέσα στην αρχιτεκτονική του TRCUT.
Ουσιαστικά σε σχέση με ένα LFSR έχουμε παρέμβει και εχουμε προσθέση μια σειριακή είσοδο και μια σειριακή έξοδο προκειμένου μέσα στο TRUCT_WITH_MISR να γίνεται XOR η έξοδος του MISR με την έξοδο του TRCUT και να μπαίνει ξανά μέσα ως σειριακή είσοδος στο MISR.

Επιπλέον έχουμε προσθέσει και ένα σήμα ονόματι MISR_SET. Το σήμα αυτό όπως προδίδει το όνομά του χρησιμοποιείται προκειμένου με την χρήση του να μπορούμε ασύγχρονα να θέσουμε όλα τα FLIP_FLOP του MISR στο λογικό ‘1’.

Δηλαδή το seed του MISR στην υλοποίηση μας είναι το “1111111111111111”

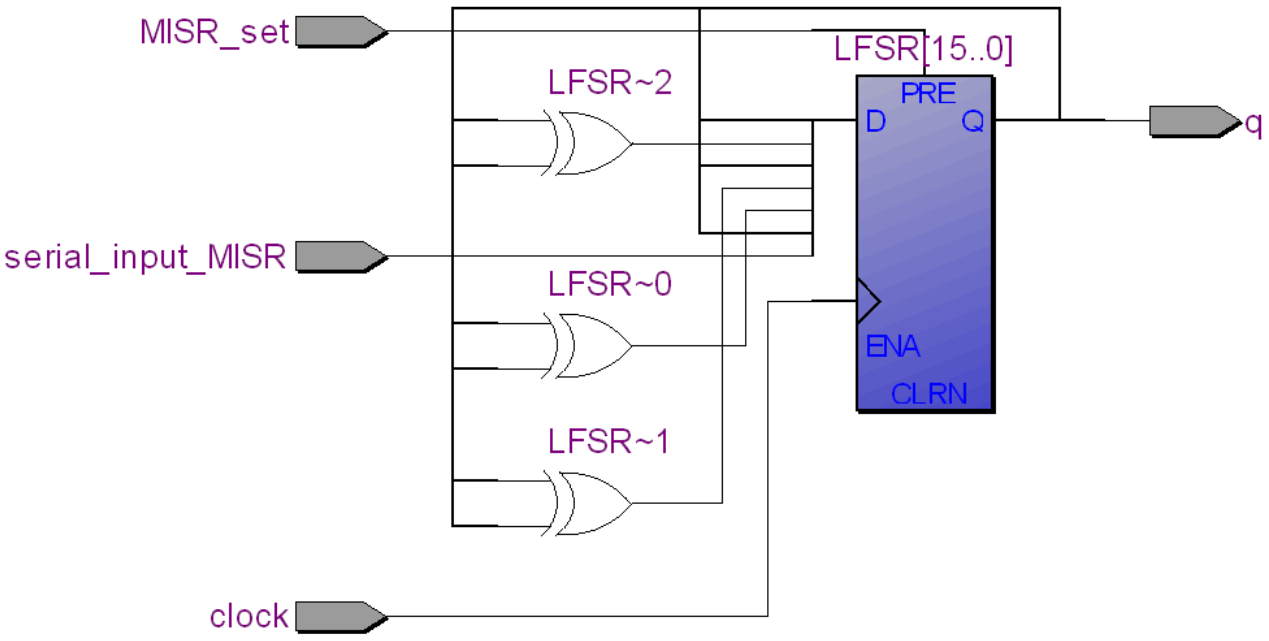
Ο κώδικας code_04 περιγράφει την αρχιτεκτονική του MISR.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity MISR_16BITS is
   port(clock:in std_logic; MISR_set: in std_logic; serial_input_MISR:
std_logic;  q:out std_logic);
end;

architecture RTL of MISR_16BITS is
   signal LFSR: std_logic_vector(15 downto 0) := (others => '1');
begin
   process(clock)
       variable feedback: std_logic;
   begin
       if MISR_set = '0' then
           if clock'EVENT and clock='1' then
               feedback := LFSR(15);
               LFSR(0) <= serial_input_MISR;
```

```
                LFSR(1) <= LFSR(0);
                LFSR(2) <= LFSR(1) xor feedback;
                LFSR(3) <= LFSR(2) xor feedback;
                LFSR(4) <= LFSR(3);
                LFSR(5) <= LFSR(4) xor 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 if;
        else LFSR <= (others => '1'); end if;
        q <= LFSR(15);
    end process;
end;
```

code_04

Μεταγλωτίζοντας τον παραπάνω κώδικα με την χρήση του quartus προκύπτει η απεικόνιση screen_03 σε επίπεδο RTL , γεγονός που επιβεβαιώνει ότι δημιουργήθηκε η ζητούμενη λειτουργικότητα.



screen_03

2.2 : Υλοποίηση TRCUT με MISR

Το MISR που υλοποιήσαμε προηγούμενος θα το προσθέσουμε στην αρχιτεκτονική του TRCUT του ερωτήματος 1 , προσθέτοντας και επιπλέον λογική πριν την είσοδο του για να πετύχουμε την ζητούμενη λειτουργικότητα.

Συγκεκριμένα πριν απο την σειριακή είσοδο του MISR θα υπάρχει μια πύλη XOR ή οποία θα κάνει XOR το σήμα της σειριακής εξόδου του MISR και της σειριακής εξόδου του TRCUT και θα αποτελεί το xorTAP του MISR.

Προκειμένου σε επόμενο στάδιο να μπορούμε να προσομοιώσουμε την περίπτωση του ERROR_INJECTION στην προσομοίωση μας , προσθέτουμε μια ακόμα είσοδο σε αυτην την πυλη XOR (στο xorTAP δηλαδή) , το σήμα ERROR_INJECTION_MODE.

Οταν το σήμα ERROR_INJECTION_MODE τίθεται στο λογικό μηδέν , με βάση τον πίνακα αλήθειας μιας πύλης XOR ειναι εμφανες οτι δεν επηρεάζει την έξοδο της πύλης και απλά θα περάσει μέσα στο MISR το αποτέλεσμα του XOR των άλλων 2 εισόδων.

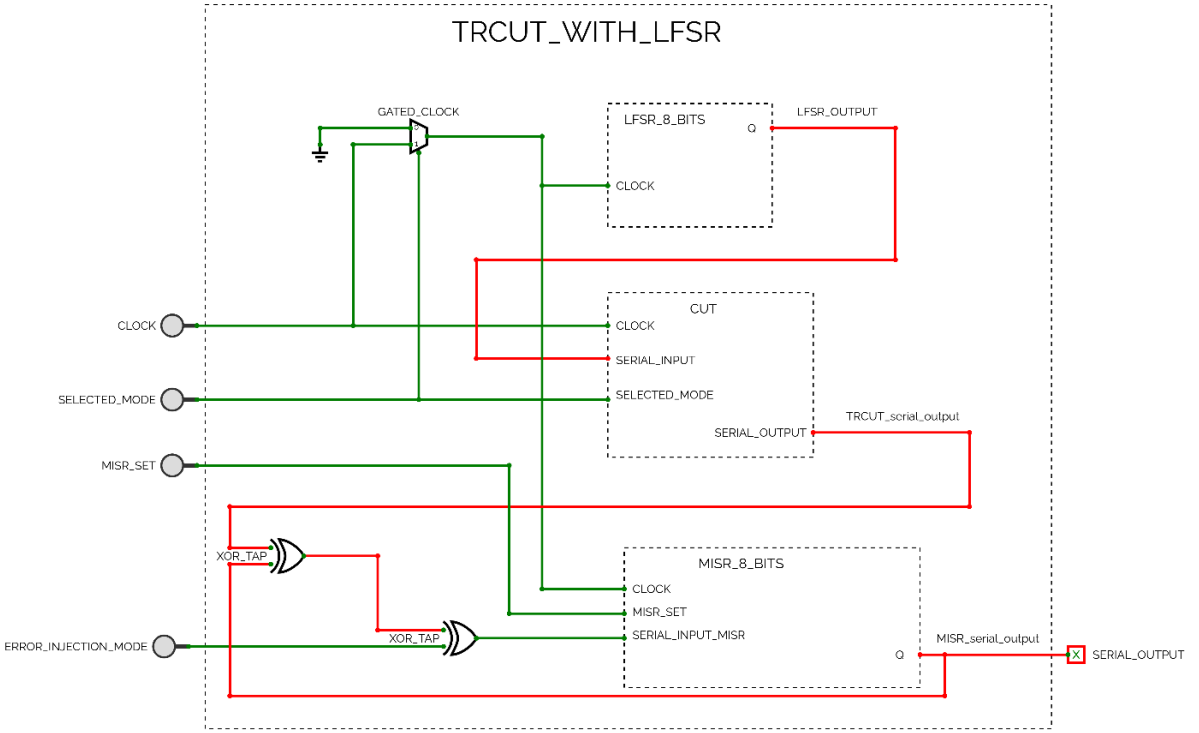
Οταν το σήμα ERROR_INJECTION_MODE τίθεται στο λογικό ‘1’ το αποτέλεσμα του XOR των άλλων 2 εισόδων θα αντιστράφει.

Άρα φροντίζοντας να θέσουμε το σήμα αυτο στο λογικό ΄1΄για έναν μόνο κύκλο ρολογιού κάποια τυχαία στιγμή κατά την διάρκεια της προσομοίωσης μας , μπορούμε να προσομοιώσουμε το ενδεχόμενο ένα τυχαίο και μη επαναλαμβανόμενο σφάλμα να συμβεί κατά την διάρκεια του ελέγχου του κυκλώματος μας , παρατηρόντας παράλληλα και το εάν η παρουσία αυτού του σφάλματος θα έχει αντίκτυπο στην τελική υπογραφή του κυκλώματος μας.

Η εικόνα desing_02 αποτελεί την λογική σχεδίαση του κυκλώματος μας σε αρχικό στάδιο

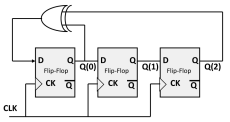
Ο κώδικας code_05 αποτελεί την περιγραφή του κυκλώματος σε γλώσσα VHDL

Επισημαίνεται ότι όπως και στο προηγούμενο ερώτημα υπάρχουν διαφορετικές αρχιτεκτονικές υλοποίησης του TRCUT_WITH_MISR προκειμένου να μπορούμε να προσομοιώσουμε στα testbench διαφορετικά σενάρια σφαλμάτων ανάλογα με την επιλεχθείσα αρχιτεκτονική.

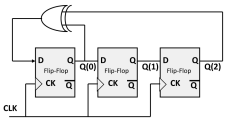


desing_02

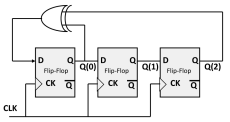
```
library IEEE;
use IEEE.std_logic_1164.all;

entity TRCUT_with_MISR is port (        selected_mode : in std_logic;
                                        clock : in std_logic;
                                        MISR_set: in std_logic;
                                        serial_output : out std_logic;
                                        error_Injection_mode : in
std_logic);
end entity;

architecture noFault of TRCUT_with_MISR is
   component LFSR_8_BITS is
       port(
           clock : in std_logic;
           q     : out std_logic
       );
   end component;
```



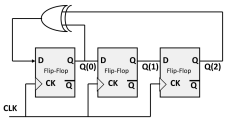

```
   component Test_Ready_Circuit_Under_Test is
       port (
           clock           : in std_logic;
           serial_input    : in std_logic;
           selected_mode   : in std_logic;
           serial_output   : out std_logic
       );
   end component;


   component MISR_16BITS
       port (
           clock : in std_logic;
           q : out std_logic;
           MISR_set: in std_logic;
           serial_input_MISR: in std_logic
       );
   end component;




   signal LFSR_serial_output ,TRCUT_serial_output , xorTAP ,
MISR_serial_output: std_logic := '1';
   signal gated_clock : std_logic ;


begin

   LFSR : LFSR_8_BITS
       port map(
           clock => gated_clock,
           q     => LFSR_serial_output
       );

   TRCUT: entity work.Test_Ready_Circuit_Under_Test(noFault)
       port map (
           clock           => clock,
           serial_input    => LFSR_serial_output,
           selected_mode   => selected_mode,
           serial_output   => TRCUT_serial_output
       );

   MISR : MISR_16BITS
       port map (
           clock => gated_clock,
           q => MISR_serial_output,
           MISR_set => MISR_set,
           serial_input_MISR => xorTAP
       );
```



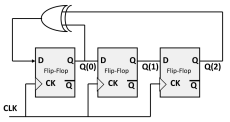

```
   process(clock, TRCUT_serial_output, MISR_serial_output) is
   begin
       if selected_mode = '1' then gated_clock <= clock;
       else gated_clock <= '0';
       end if;
       xorTAP <= TRCUT_serial_output xor error_Injection_mode xor
MISR_serial_output;
   end process;

   serial_output <= MISR_serial_output;

end architecture noFault;


-------------
---------------
---------------

architecture stuckAtOne of TRCUT_with_MISR is
   component LFSR_8_BITS is
       port(
           clock : in std_logic;
           q     : out std_logic
       );
   end component;


   component Test_Ready_Circuit_Under_Test is
       port (
           clock           : in std_logic;
           serial_input    : in std_logic;
           selected_mode   : in std_logic;
           serial_output   : out std_logic
       );
   end component;


   component MISR_16BITS
       port (
           clock : in std_logic;
           q : out std_logic;
           MISR_set: in std_logic;
           serial_input_MISR: in std_logic
       );
   end component;




   signal LFSR_serial_output ,TRCUT_serial_output , xorTAP ,
MISR_serial_output: std_logic := '1';
```




```
    signal gated_clock : std_logic ;


begin

    LFSR : LFSR_8_BITS
        port map(
            clock => gated_clock,
            q     => LFSR_serial_output
        );

    TRCUT: entity work.Test_Ready_Circuit_Under_Test(stuckAtOne)
        port map (
            clock           => clock,
            serial_input    => LFSR_serial_output,
            selected_mode   => selected_mode,
            serial_output   => TRCUT_serial_output
        );

    MISR : MISR_16BITS
        port map (
            clock => gated_clock,
            q => MISR_serial_output,
            MISR_set => MISR_set,
            serial_input_MISR => xorTAP
        );


    process(clock, TRCUT_serial_output, MISR_serial_output) is
    begin
        if selected_mode = '1' then gated_clock <= clock;
        else gated_clock <= '0';
        end if;
        xorTAP <= TRCUT_serial_output xor error_Injection_mode xor
MISR_serial_output;
    end process;

    serial_output <= MISR_serial_output;

end architecture stuckAtOne;

------------
------------
architecture stuckAtZero of TRCUT_with_MISR is
    component LFSR_8_BITS is
        port(
            clock : in std_logic;
            q     : out std_logic
        );
    end component;
```



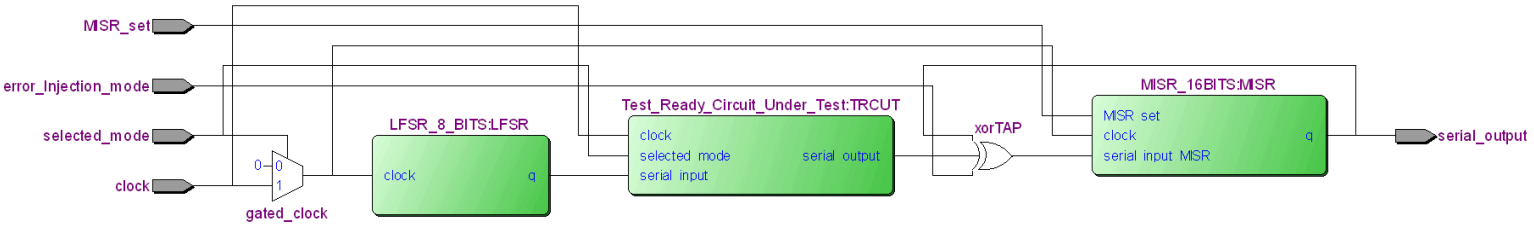

```
    component Test_Ready_Circuit_Under_Test is
        port (
            clock           : in std_logic;
            serial_input    : in std_logic;
            selected_mode   : in std_logic;
            serial_output   : out std_logic
        );
    end component;


    component MISR_16BITS
        port (
            clock : in std_logic;
            q : out std_logic;
            MISR_set: in std_logic;
            serial_input_MISR: in std_logic
        );
    end component;




    signal LFSR_serial_output ,TRCUT_serial_output , xorTAP ,
MISR_serial_output: std_logic := '1';
    signal gated_clock : std_logic ;


begin

    LFSR : LFSR_8_BITS
        port map(
            clock => gated_clock,
            q     => LFSR_serial_output
        );

    TRCUT: entity work.Test_Ready_Circuit_Under_Test(stuckAtZero)
        port map (
            clock           => clock,
            serial_input    => LFSR_serial_output,
            selected_mode   => selected_mode,
            serial_output   => TRCUT_serial_output
        );

    MISR : MISR_16BITS
        port map (
            clock => gated_clock,
            q => MISR_serial_output,
            MISR_set => MISR_set,
            serial_input_MISR => xorTAP
        );
```



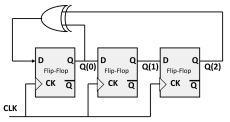
```
   process(clock, TRCUT_serial_output, MISR_serial_output) is
   begin
       if selected_mode = '1' then gated_clock <= clock;
       else gated_clock <= '0';
       end if;
       xorTAP <= TRCUT_serial_output xor error_Injection_mode xor
MISR_serial_output;
   end process;

   serial_output <= MISR_serial_output;

end architecture stuckAtZero;
```

code_05

Μετά από μεταγγλώτιση του κυκλώματος στο Quartus προκύπτει το κύκλωμα στην εικόνα screen_4 σε επίπεδο RTL , αρχιτεκτονική που ταυτίζεται με την αρχιτεκτονική στην αρχική μας σχεδίαση desing_02 και αρα επιβεβαιώνει ότι δημιουργήθηκε η ζητούμενη λειτουργικότητα.



screen_04



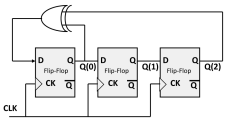
2.3 : TRCUT_WITH_MISR_TESTBENCH_NO_FAULT

Δημιουργούμε ένα testbench προκειμένου να ελέγξουμε την ορθη λειτουργία του κυκλώματος μας.
Στο συγκεκριμένο testbench θα χρησιμοποιήσουμε τη αρχιτεκτονική noFault σε ολά τα modules που αποτελούν το TRCUT. Δηλαδή θα προσομοιώσουμε την λειτουργία του κυκλώματος οταν δεν υπάρχει κανένα σφάλμα στην λειτουργία του και θα αποθηκεύσουμε το golden signature του σε ένα σήμα.

Ο παρακάτω κώδικας code_05 υλοποιεί το testbench σε γλώσσα VHDL.

Η επιμέρους εξήγηση του θα γίνει σε αντιπαραβολή με στιγμιότυπα της προσομοίωσης.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity TRCUT_WITH_MISR_TESTBENCH_NO_FAULT is
end entity;


architecture testbench of TRCUT_WITH_MISR_TESTBENCH_NO_FAULT is
   component TRCUT_WITH_MISR
   is port (selected_mode : in std_logic;
            clock : in std_logic;
            MISR_set: in std_logic;
            serial_output : out std_logic;
            error_Injection_mode : in std_logic);
   end component;

   signal clock, selected_mode , MISR_set , serial_output : std_logic
:= '1';
   signal error_Injection_mode : std_logic := '0';
   signal signature : std_logic_vector(15 downto 0);



begin

   DUT : entity work.TRCUT_WITH_MISR(NoFault)
       port map (
           selected_mode => selected_mode,
           clock => clock,
           MISR_set => MISR_set,
           serial_output => serial_output,
           error_Injection_mode => error_Injection_mode
       );

   clock <= not clock after 2.5 ns;
```




```
    -- scanTesting
    process is begin

        wait for 2.5 ns;
        MISR_set <= '0';
        wait for 2.5 ns;

        for i in 0 to 31 loop

            --load s1
            wait for 5 ns;

            --load s2;
            wait for 5 ns;

            -- load s3;
            wait for 5 ns;

            --load s4
            wait for 2.5 ns;

            selected_mode <= '0';
            wait for 5 ns;

            selected_mode <= '1';
            wait for 2.5 ns;


        end loop;

        -- scan chain last capture exit time
        wait for 4 * 5 ns ;


        -- signature capture
        for i in 0 to 15 loop
            signature(i) <= serial_output;
            wait for 5 ns;
        end loop;



        error_Injection_mode <= 'X';
        wait;
    end process;

end architecture;
```

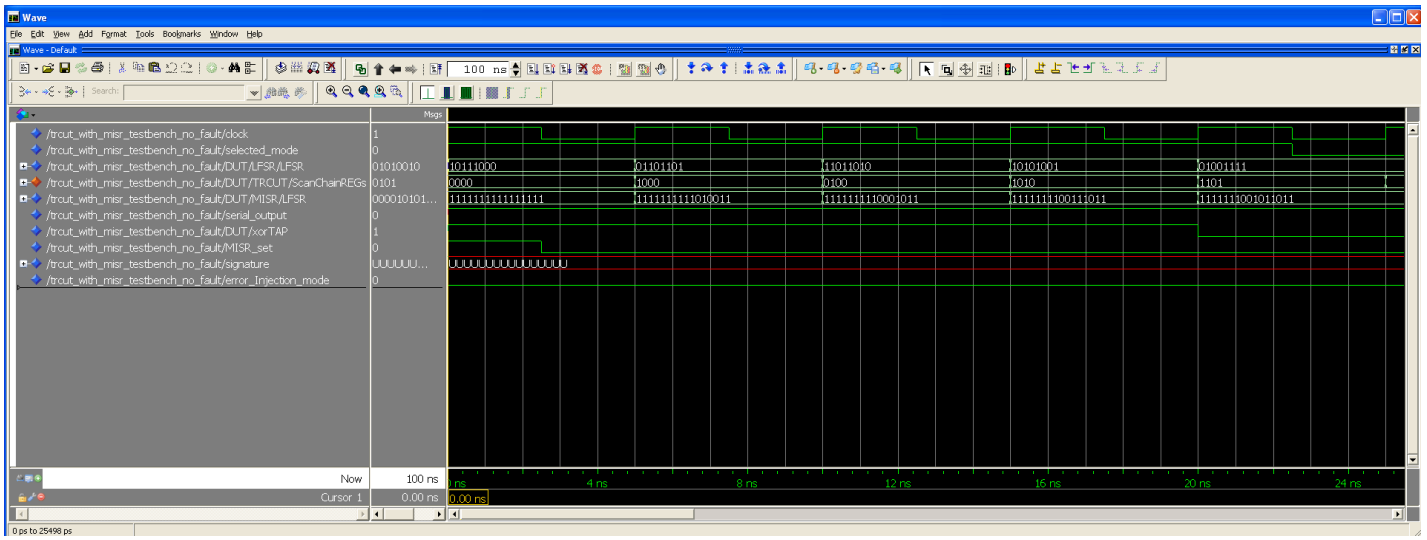
code_05

Στιγμιότυπα και επεξήγηση του Testbench_WITH_MISR_NO_FAULT

Σήματα της προσομοίωσης :

- **Clock** : Το ρολόι του κυκλώματος μας το οποίο τίθεται σε περιόδο t = 5 ns
- **Selected_mode** : Σήμα το οποίο καθορίζει την εσωτερική λειτουργία του TRCUT. Όταν τίθεται στο λογικό ‘1’ η αλυσίδα σάρωσης (Scan Chain) του TRCUT επιτελεί την λειτουργία της δεξιάς ολίσθησης. Όταν τίθεται στο λογικό ‘0’ το TRCUT κάνει capture τις τιμές του CUT και τις αποθηκεύει στα Flip-Flop της αλυσίδας σάρωσης.
- **Serial_Output** : Το σήμα το οποίο αντιστοιχεί στην έξοδο του TRCUT. Δηλαδή το bit εξόδου του MISR
- **MISR_SET**: Το σήμα το οποίο κάνει set τα flip-flop που περιέχονται μέσα στο MISR στην λογική τιμή ‘1’ και μας βοηθάει να αρχικοποιήσουμε το MISR στο default Seed.
- **ERROR_INJECTION_MODE** : Σήμα το οποίο μας βοηθάει να προσομοιώσουμε την ύπαρξη ενός τυχαίου μη επαναλαμβανόμενου σφάλματος. Ο τρόπος λειτουργίας του εξηγήθηκε εδώ.
- **Signature** : Σήμα στο οποίο θα αποθηκεύσουμε την υπογραφή του TRCUT μετά το πέρας της εφαρμογής 32 διανυσμάτων εισόδου απο το LFSR

Παρατίθενται στιγμιότυπα προσομοίωσης με αναφορα στο αντίστοιχο σημείο του κώδικα code_05

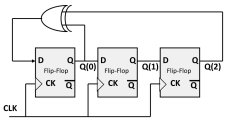


Στο παραπάνω στιγμιότυπο απεικονίζεται η αρχή της προσομοίωσης μας.
Το σήμα MISR_SET αρχικοποιεί το MISR στο default seed πριν την πρώτη ακμή του ρολογιού την χρονική στιγμή t = 5 ns ενώ το LFSR είναι και αυτό αρχικοποιημένο στο default seed που έχουμε επιλέξει αλλά μέσω του κώδικα VHDL χωρις να απαιτείται σήμα αρχικοποίησης.

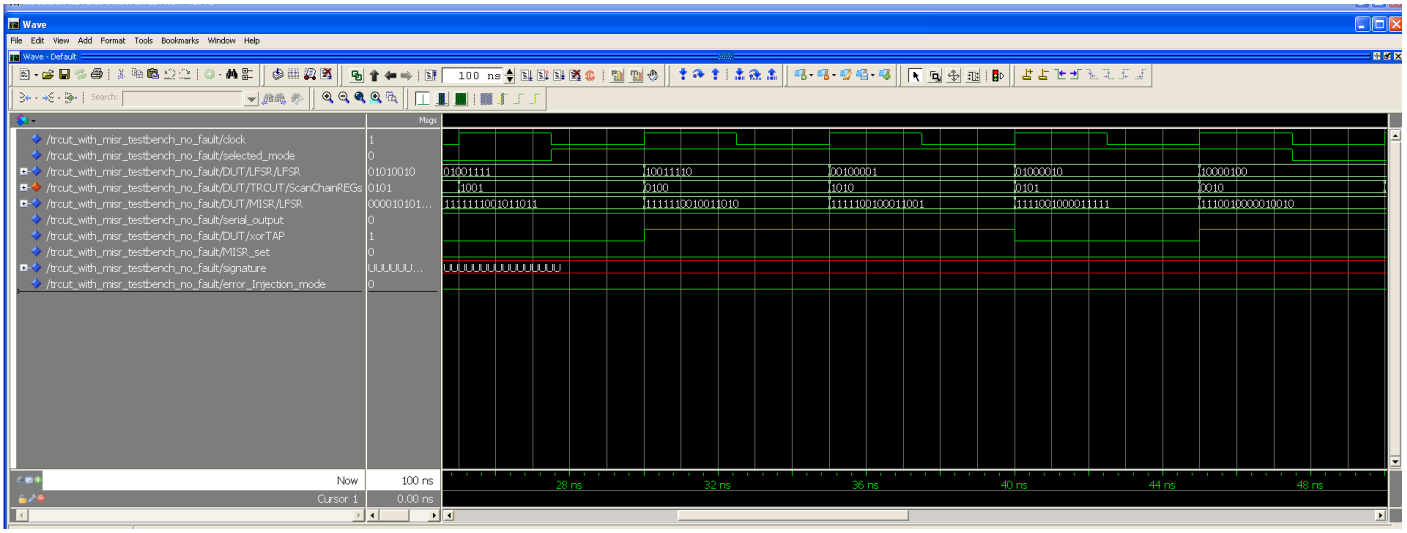
Στις πρώτες 4εις θετικές ακμές του ρολογιού το πιο σημαντικο bit του διανύσματος που έχει μέσα του το LFSR φορτώνεται στην αλυσίδα σάρωσης.

Την χρονική στιγμή t = 25 ns έχουμε το πρώτο capture μπαίνει σε λειτουργία capture

Τα παραπάνω αντιστοιχούν στον κώδικα κάτω από το σχόλιο - - scanTesting στο αρχείο code_05



Απο την χρονική στιγμή τ = 30 ns και για 4εις θετικές ακμές του ρολογιού , βλέπουμε τα περιεχόμενα της αλυσίδας σάρωσης να περνάνε μέσα από το XOR_TAP πριν το MISR και να μπαινουν σειριακα μέσα στο MISR



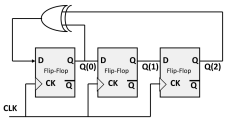
Συγκεκριμένα :

- T = 29 ns **- Λίγο πριν την θετική ακμή του ρολογιού –**
  - Το λιγότερο σημαντικό bit της scanChain που είναι το bit που θα βγει απο την αλυσίδα σάρωσης, είναι στο λογικό ‘1’
  - Το περισσότερο σημαντικό bit του MISR που είναι το bit που βγαινει απο την σειριακή έξοδο του MISR, είναι στο λογικό ‘1’
  - Το xorTap που βρίσκεται μέσα στο TRCUT και είναι ασύγχρονο να έχει κάνει XOR τις τιμές των 2 αυτών σημάτων (μαζί με το error_injectio_mode signal που όπως έχει αναφερθεί στην συγκεκριμένη προσομοίωση δεν επηρεάζει διότι είναι μονίμως στο λογικό ‘0’) και να έχει βγάλει έξοδο το λογικό ‘0’

- T = 30 ns **– Ακριβώς πάνω στην θετική ακμή του ρολογιού -**
  - Η έξοδος του xorTap μπαίνει ως σειριακή είσοδος στο MISR και για αυτό στην προσομοίωση φαινεται μετα την θετική ακμή του ρολογιού το λιγότερο σημαντικό bit του MISR ΝΑ έχει τεθεί στο λογικό ‘0’ .

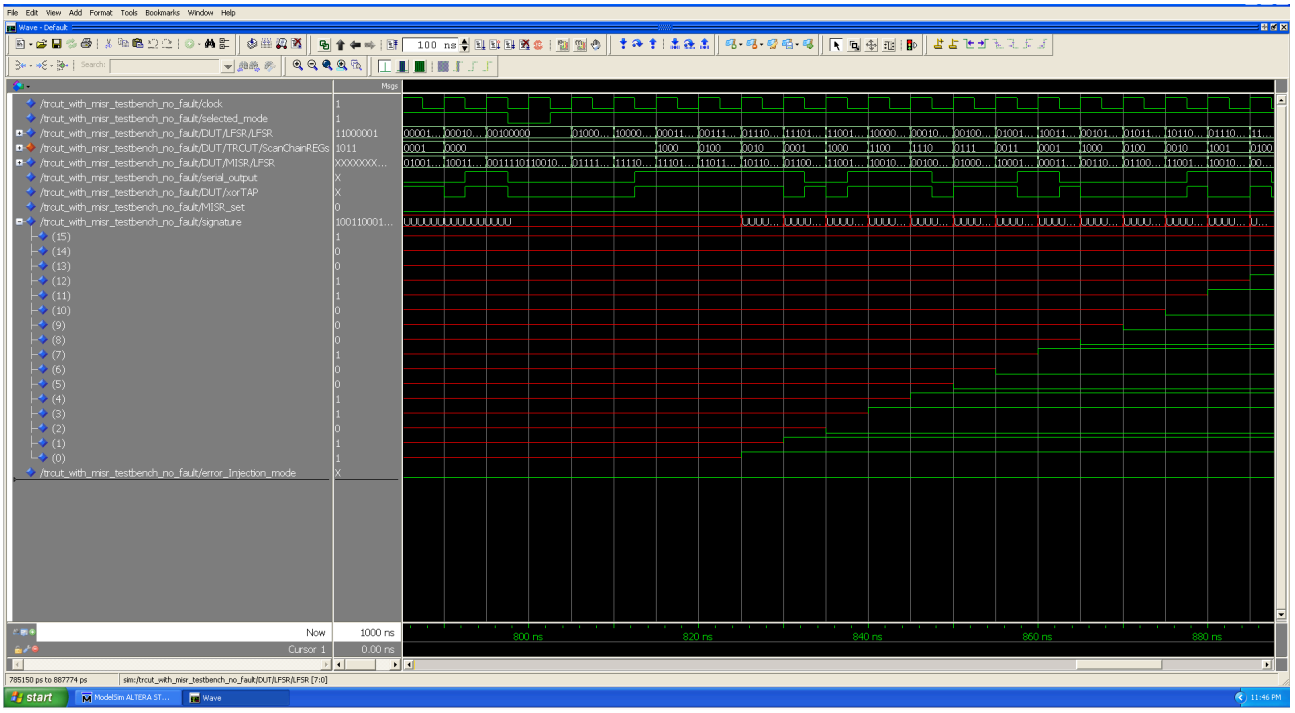
Ακριβώς η ίδια λογική συμβαίνει και για τους επόμενους 3εις εναπομείναντες κυκλους που η αλυσίδα σάρωσης εκτελεί δεξιά ολίσθηση αλλά και γενικότερα όταν το TRCUT_WITH_MISR βρίσκεται σε λειτουργία σειριακής ολίσθησης.

Οπως έχει αναφερθεί όταν βρισκόμαστε σε caputre mode κόβουμε το ρολοι στο LFSR και το MISR , πράγμα που είναι εμφανές και στην προσομοίωση , οπως πχ την χρονική στιγμή t = 25 ns οπου τα περιεχόμενα του LFSR και του MISR μένουν αναλλοίωτα για έναν ακόμα κύκλο.



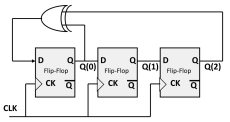
*
*
*
*

Στιγμιότυπα για τα ενδιάμεσα στάδια της προσομοίωσης παραλείπονται



Στο παραπάνω στιγμιότυπο βλέπουμε την στιγμή που ολοκληρώνεται η διαδικασία του scanTesting στο κύκλωμα μας.

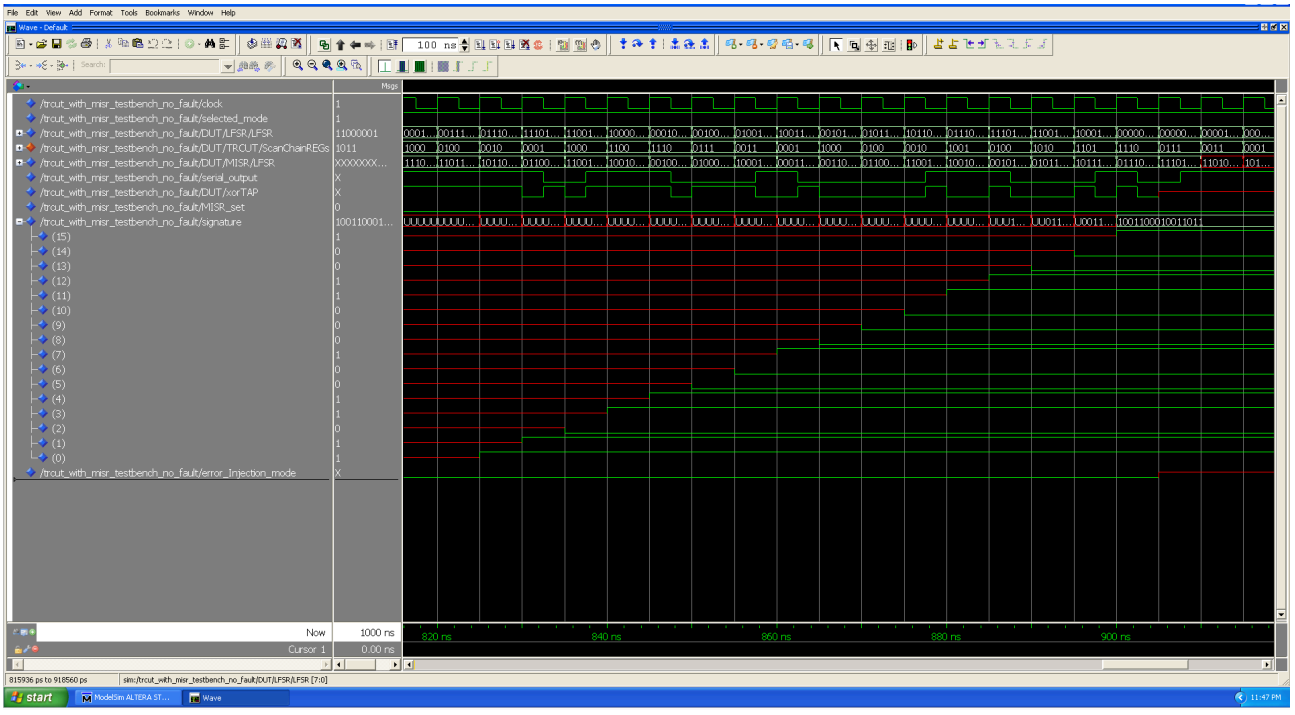
Συγκεκριμένα την χρονική στιγμή t = 800 ns γίνεται το capture στην αλυσιδα σάρωσης η απόκριση του CUT στο τελευταίο διάνυσμα εισόδου του ελέγχου μας. Περιμένουμε 4εις κύκλους ρολογιού έτσι ώστε η απόκριση αυτή να προλάβει να βγει απο την αλυσίδα σάρωσης και να φορτωθεί μέσα στο MISR

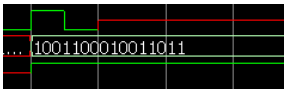


Την χρονική στιγμή t = 825 ns , ακριβώς ένα κύκλο μετά από τη είσοδο του τελευταίου bit της αλυσίδας σάρωσης μέσα στο MISR , έχει δημιουργηθεί η υπογραφή μας και ξεκινάμε αποθηκεύουμε για 16 κύκλους ρολογιού την σειριακή έξοδο του MISR μέσα σε ένα σήμα ονόματι signature , σήμα το οποίοι η τιμή του όταν φορτωθεί τελείως θα αντιπροσωπεύει την golden signature του κυκλώματος μας.

Όταν έχουμε αποθηκεύσει την υπογραφή έχει ολοκληρωθεί η προσομοίωση μας και για λόγους έμφασης θέτουμε το σήμα error_injection_mode στο "X" προκειμένου να είναι εύκολα οπτικά εντοπίσιμο αυτό το χρονικό σημείο στην προμοίωση μας .

Η παραπάνω διαδικασία που αφορά την δημιουργία της υπογραφής βρίσκεται κάτω από το σχόλιο - - signature capture στο κώδικα [code_05]



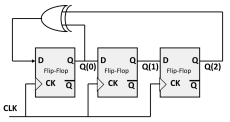
Υπογραφή που προκύπτει -> 

2.4 : TRCUT_WITH_MISR_TESTBENCH_STACK_AT_ONE

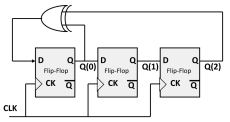
Χρησιμοποιώντας την αρχιτεκτονική stackAtOne του CUT προσομοιώνουμε την περίπτωση που κάποιο λογικό σήμα έχει κολλήσει στο λογικό 1 στο CUT

Ο κώδικας του TestBench είναι ακριβώς ίδιος με πριν απλά έχουν αλλάξει οι δηλώσεις που επιλέγουν ποια αρχιτεκτονική του CUT - TRCUT χρησιμοποιούμε.

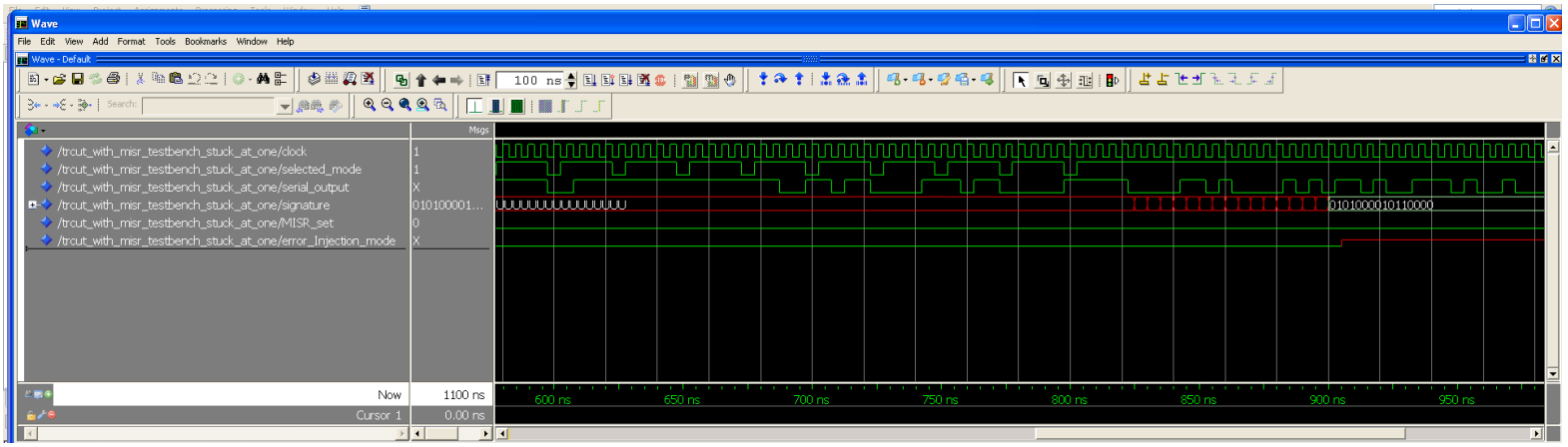
```
library IEEE;
use IEEE.std_logic_1164.all;

entity TRCUT_WITH_MISR_TESTBENCH_STUCK_AT_ONE is
end entity;


architecture testbench of TRCUT_WITH_MISR_TESTBENCH_STUCK_AT_ONE is
   component TRCUT_WITH_MISR
   is port (selected_mode : in std_logic;
            clock : in std_logic;
            MISR_set: in std_logic;
            serial_output : out std_logic;
            error_Injection_mode : in std_logic);
   end component;

   signal clock, selected_mode , MISR_set , serial_output : std_logic
:= '1';
   signal error_Injection_mode : std_logic := '0';
   signal signature : std_logic_vector(15 downto 0);



begin

   DUT : entity work.TRCUT_WITH_MISR(stuckAtOne)
       port map (
           selected_mode => selected_mode,
           clock => clock,
           MISR_set => MISR_set,
           serial_output => serial_output,
           error_Injection_mode => error_Injection_mode
       );

   clock <= not clock after 2.5 ns;


   -- scanTesting
   process is begin

       wait for 2.5 ns;
```

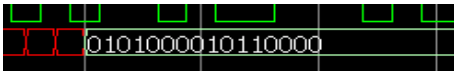



```
        MISR_set <= '0';
        wait for 2.5 ns;

        for i in 0 to 31 loop

            --load s1
            wait for 5 ns;

            --load s2;
            wait for 5 ns;

            -- load s3;
            wait for 5 ns;

            --load s4
            wait for 2.5 ns;

            selected_mode <= '0';
            wait for 5 ns;

            selected_mode <= '1';
            wait for 2.5 ns;


        end loop;

        -- scan chain last capture exit time
        wait for 4 * 5 ns ;


        -- signature capture

        for i in 0 to 15 loop
            signature(i) <= serial_output;
            wait for 5 ns;
        end loop;




        error_Injection_mode <= 'X';
        wait;
    end process;
end architecture;
```



Στιγμιότυπο προσομοίωσης :



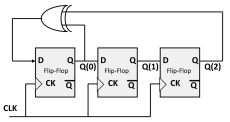
Υπογραφή που προκύπτει -> 

2.5 : TRCUT_WITH_MISR_TESTBENCH_STACK_AT_ZERO

Χρησιμοποιώντας την αρχιτεκτονική stackAtZero του CUT προσομοιώνουμε την περίπτωση που κάποιο λογικό σήμα έχει κολλήσει στο λογικό 0 στο CUT

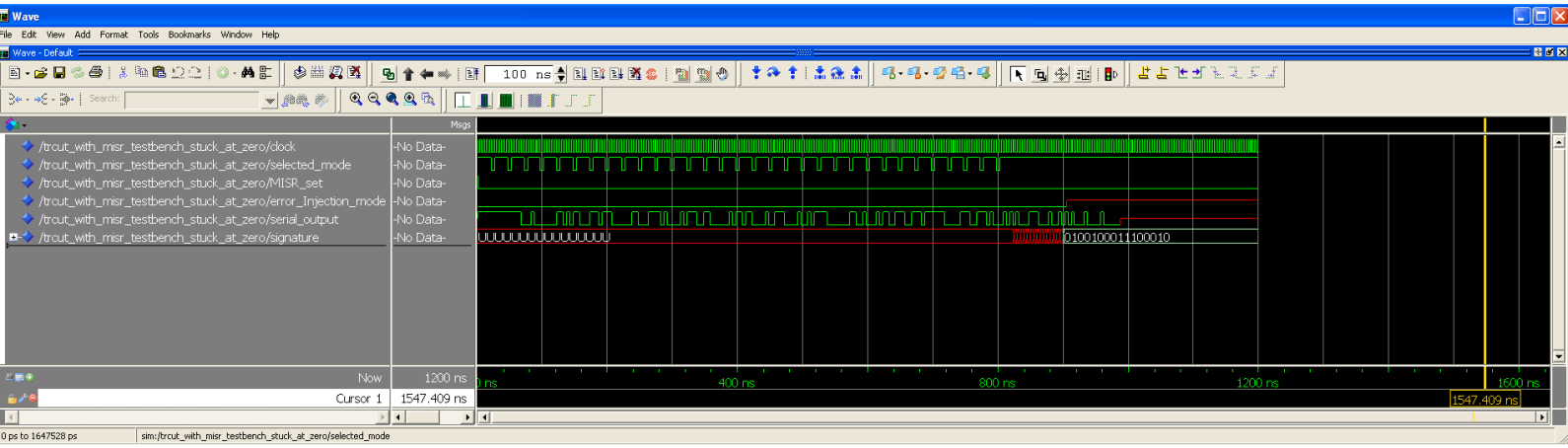
Ο κωδικάς του TestBench είναι ακριβώς ίδιος με πριν απλά έχουν αλλάξει οι δηλώσεις που επιλέγουν ποια αρχιτεκτονική του CUT - TRCUT χρησιμοποιούμε.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity TRCUT_WITH_MISR_TESTBENCH_STUCK_AT_ZERO is
end entity;


architecture testbench of TRCUT_WITH_MISR_TESTBENCH_STUCK_AT_ZERO is
   component TRCUT_WITH_MISR
   is port (selected_mode : in std_logic;
            clock : in std_logic;
            MISR_set: in std_logic;
            serial_output : out std_logic;
            error_Injection_mode : in std_logic);
   end component;

   signal clock, selected_mode , MISR_set , serial_output : std_logic
:= '1';
   signal error_Injection_mode : std_logic := '0';
   signal signature : std_logic_vector(15 downto 0);



begin

   DUT : entity work.TRCUT_WITH_MISR(stuckAtZero)
       port map (
           selected_mode => selected_mode,
           clock => clock,
           MISR_set => MISR_set,
           serial_output => serial_output,
           error_Injection_mode => error_Injection_mode
       );

   clock <= not clock after 2.5 ns;


   -- scanTesting
   process is begin
```

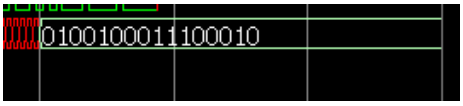



```
        wait for 2.5 ns;
        MISR_set <= '0';
        wait for 2.5 ns;

        for i in 0 to 31 loop

            --load s1
            wait for 5 ns;

            --load s2;
            wait for 5 ns;

            -- load s3;
            wait for 5 ns;

            --load s4
            wait for 2.5 ns;

            selected_mode <= '0';
            wait for 5 ns;

            selected_mode <= '1';
            wait for 2.5 ns;


        end loop;

        -- scan chain last capture exit time
        wait for 4 * 5 ns ;


        -- signature capture

        for i in 0 to 15 loop
            signature(i) <= serial_output;
            wait for 5 ns;
        end loop;




        error_Injection_mode <= 'X';
        wait;
    end process;
end architecture;
```

Στιγμιότυπο Προσομοίωσης ->



Υπογραφή που προκύπτει ->



2.6 :TRCUT_WITH_MISR_TESTBENCH_ERROR_INJECTION

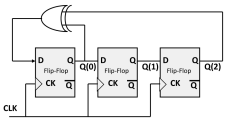
Χρησιμοποιώντας την αρχιτεκτονική noFault του CUT προσομοιώνουμε την περίπτωση που κάποιο τυχαίο μη επαναλαμβανόμενο σφάλμα αλλάξει κάποιο bit απο την αποκριση του TRCUT

Ο κώδικας του TestBench είναι ακριβώς ίδιος με πριν απλά έχουν αλλάξει οι δηλώσεις που επιλέγουν ποια αρχιτεκτονική του CUT - TRCUT χρησιμοποιούμε.

Επιπλέον προκειμένου να προσομοιωθεί το error_injection έχει προσθεθέι το process : error_injection_mode

Η λειτουργία του συγκεκριμένου process είναι την χρονική στιγμή t = 345 ns να ενεργοποιήσει για μια και μονο θετική ακμή του ρολογιού το σήμα error_injection_mode και να αναγκάσει να αντιστραφεί η τιμή που θα εμπαινε μέσα στο MISR

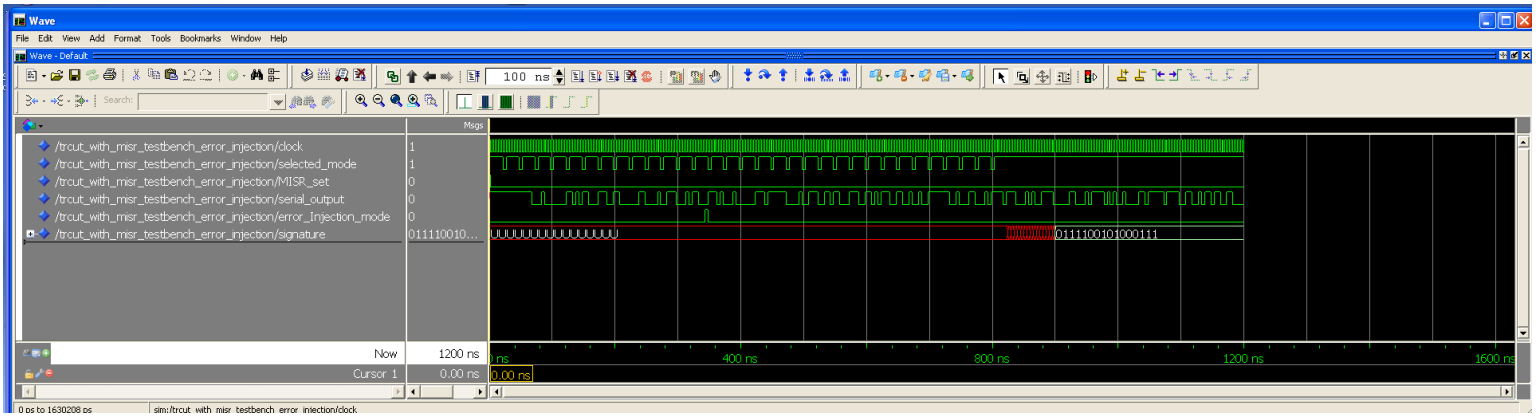
```
library IEEE;
use IEEE.std_logic_1164.all;

entity TRCUT_WITH_MISR_TESTBENCH_ERROR_INJECTION is
end entity;


architecture testbench of TRCUT_WITH_MISR_TESTBENCH_ERROR_INJECTION
is
   component TRCUT_WITH_MISR
   is port (selected_mode : in std_logic;
            clock : in std_logic;
            MISR_set: in std_logic;
            serial_output : out std_logic;
            error_Injection_mode : in std_logic);
   end component;

   signal clock, selected_mode , MISR_set , serial_output : std_logic
:= '1';
   signal error_Injection_mode : std_logic := '0';
   signal signature : std_logic_vector(15 downto 0);



begin

   DUT : entity work.TRCUT_WITH_MISR(NoFault)
       port map (
           selected_mode => selected_mode,
           clock => clock,
```



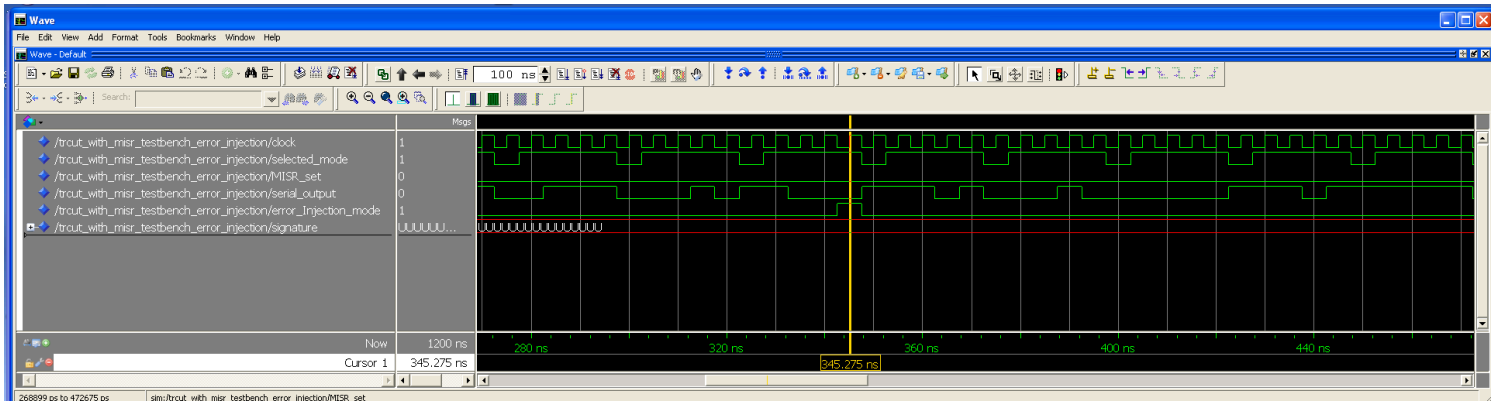

```
        MISR_set => MISR_set,
        serial_output => serial_output,
        error_Injection_mode => error_Injection_mode
    );

clock <= not clock after 2.5 ns;


-- scanTesting
process is begin

    wait for 2.5 ns;
    MISR_set <= '0';
    wait for 2.5 ns;

    for i in 0 to 31 loop

        --load s1
        wait for 5 ns;

        --load s2;
        wait for 5 ns;

        -- load s3;
        wait for 5 ns;

        --load s4
        wait for 2.5 ns;

        selected_mode <= '0';
        wait for 5 ns;

        selected_mode <= '1';
        wait for 2.5 ns;


    end loop;

    -- scan chain last capture exit time
    wait for 4 * 5 ns ;


    -- signature capture

    for i in 0 to 15 loop
        signature(i) <= serial_output;
        wait for 5 ns;
    end loop;



    wait;
```

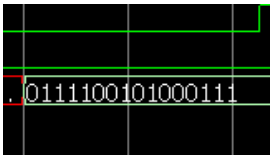
```
    end process;


    error_injection : process is begin
        wait for 342.5 ns ;

        error_Injection_mode <= '1';
        wait for 5 ns;
        error_Injection_mode <= '0';
        wait for 5 ns;

        wait;

    end process;

end architecture;
```

Στιγμιότυπο προσομοίωσης :



Χρονική Στιγμή του error injection με λεπτομέρεια



Υπογραφή που προκύπτει -> 

Συμπέρασμα

Παρατηρούμε ότι και στις 3εις περιπτώσεις σφαλμάτων που εξετάσαμε η υπογραφή που προκύπτει είναι διαφορετική από την golden signature , ακόμα και στην περίπτωση του error injection που το σφάλμα συμβαίνει μόνο για έναν κύκλο ρολογιού.