Πανεπιστήμιο Ιωαννίνων

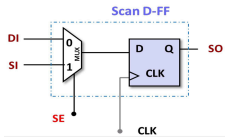
Τμήμα Μηχανικών Ηλεκτρονικών
Υπολογιστών & Πληροφορικής

Δοκιμή και Αξιοπιστία Ηλεκτρονικών Συστημάτων

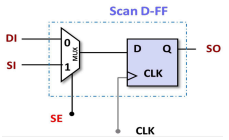
Εαρινό Εξάμηνο 2024

Scan Testing Circuit

Χρήστος Δημητρέσης 4351 cs04351@uoi.gr



Περιεχόμενα



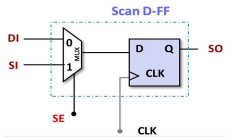
Ερώτημα 1.1 :
Υλοποίηση Testable-Ready Circuit Under Test

Δημιουργία Scan D- Flip Flop

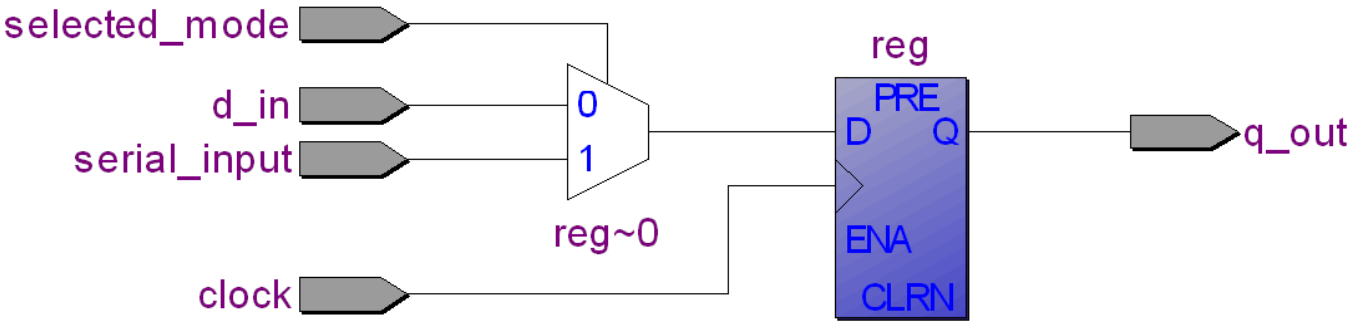
Ο κώδικας code_snippet_01 δημιουργεί ένα Scan D Flip Flop σε γλώσσα VHDL με την ζητούμενη λειτουργικότητα για να χρησιμοποιηθεί παρακάτω στον ελέγχο του κυκλώματος με την μέθοδο του Scan Testing.

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

entity 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 entity;

architecture behavior of scanDFF is

   signal reg : std_logic := 'X';

begin

   process(clock)
   begin
       if rising_edge(clock) then
           case selected_mode is
               when '0'    => reg <= d_in;
               when '1'    => reg <= serial_input;
               when others => reg <= 'X';
           end case;
       end if;
   end process;

   q_out <= reg;

end architecture;
```

code_snippet_01



Συνθέτοντας το κύκλωμα code_snippet_01 στο quartus προκύπτει το παρακάτω κύκλωμα (screen_01) σε επίπεδο RTL πράγμα που επιβεβαιώνει την ορθότητα της σχεδίασης του.

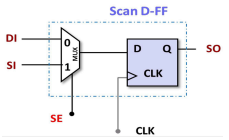


screen_01

Δημιουργία Circuit Under Test (CUT)

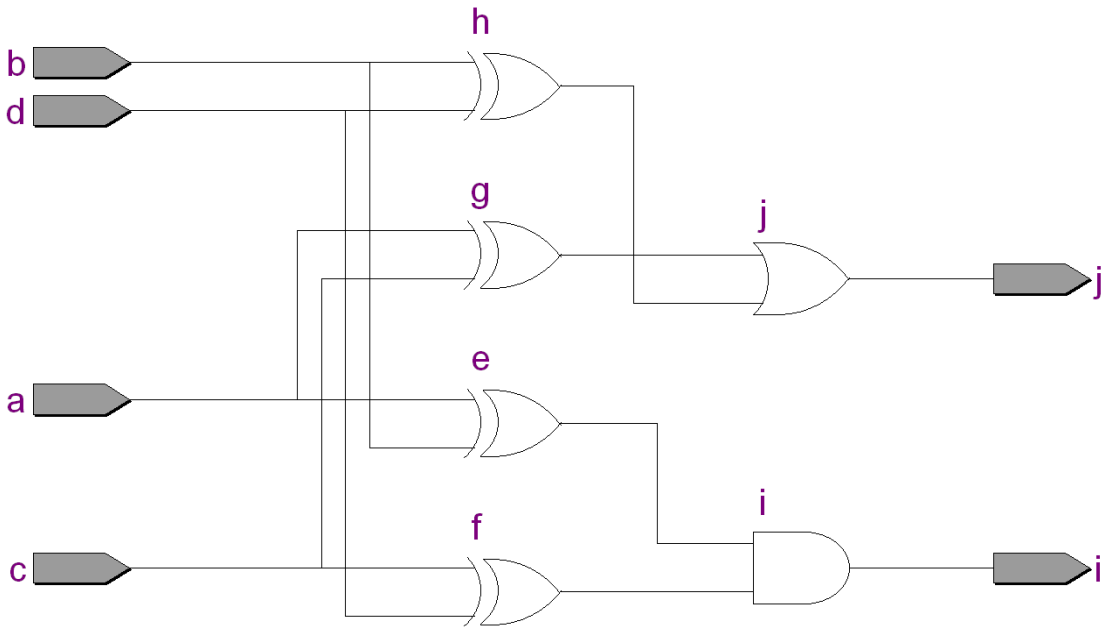
Ο κώδικας code_snippet_02 δημιουργεί το ζητούμενο κυκλωμα CUT

Συνθέτοντας το κύκλωμα code_snippet_02 στο quartus προκύπτει η παρακάτω σχεδίαση (screen_02) σε επίπεδο RTL. Η ταύτιση της σχεδίασης σε επίπεδο RTL του quartus με την εικόνα της εκφώνησης επιβεβαιώνει την ορθότητα του κυκλώματος.

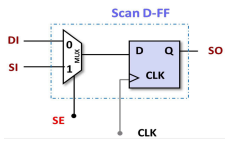



```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

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 dataFlow 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 dataFlow;
```

code_snippet_02

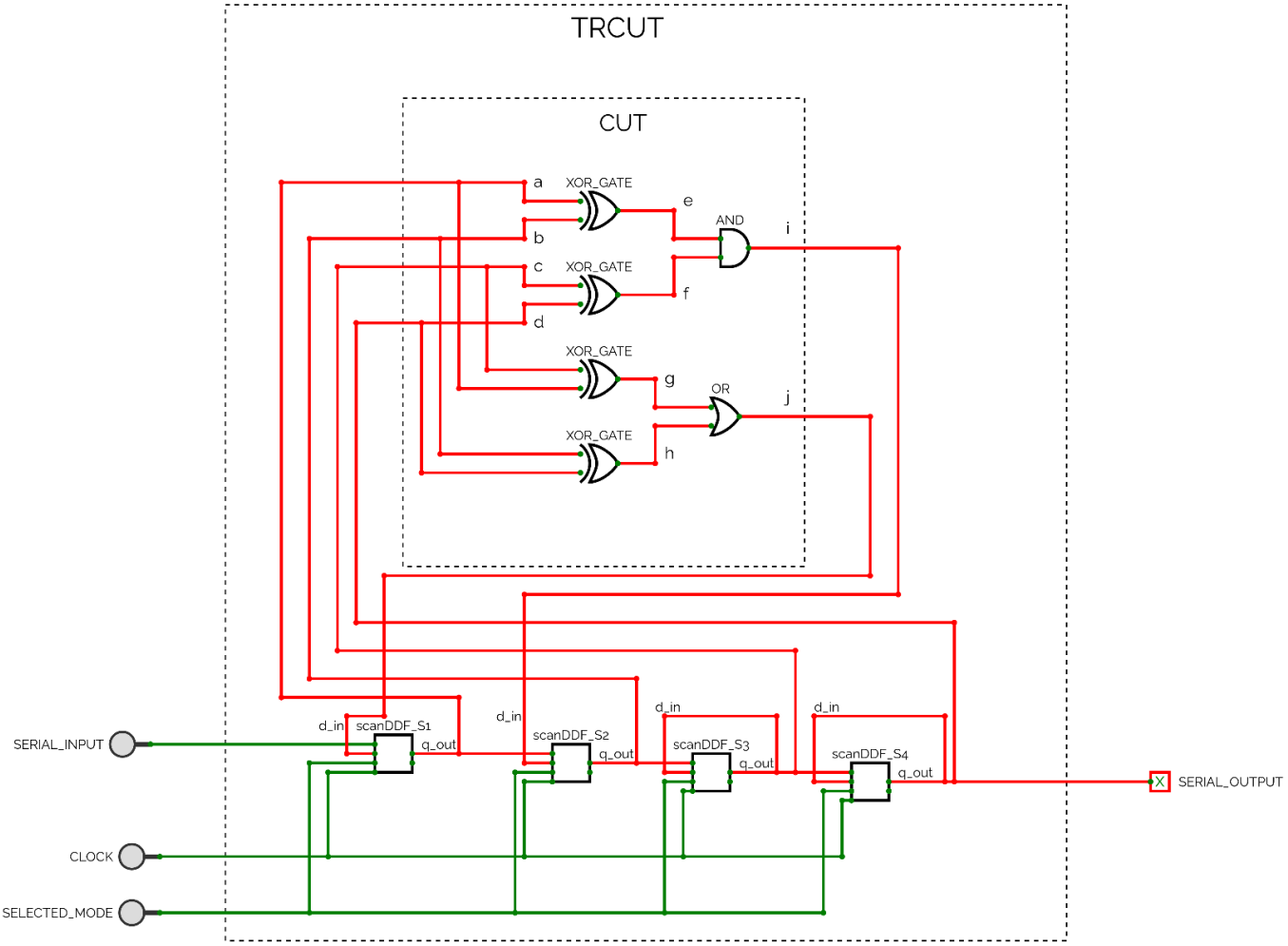


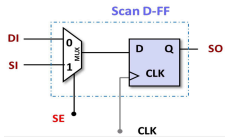
screen_02



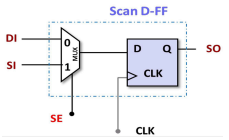
Δημιουργία Testable-Ready Circuit Under Test (TRCUT)

Ο κώδικας code_snippet_03 δημιουργεί το κυκλωμα TRCUT σε γλώσσα VHDL με την λειτουργικότητα της εκφώνησης. Το παρακάτω σχηματικό επεξηγεί την τοπολογία του. Ουσιαστικά συνδέουμε το CUT που δημιουργήσαμε προηγουμένως με 4 scanDFFs της μορφής που δημιουργήσαμε προηγουμένως. Δηλαδή το TRCUT αποτελείται από το CUT συνδεδεμένο κατάλληλα με 4 scanDFFs τα οποία αποτελούν την αλυσίδα σάρωσης.



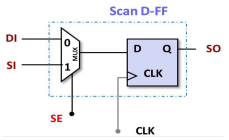



```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

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 dataFlow 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

   Circuit_Under_Test_inst : Circuit_Under_Test
       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
       );
```

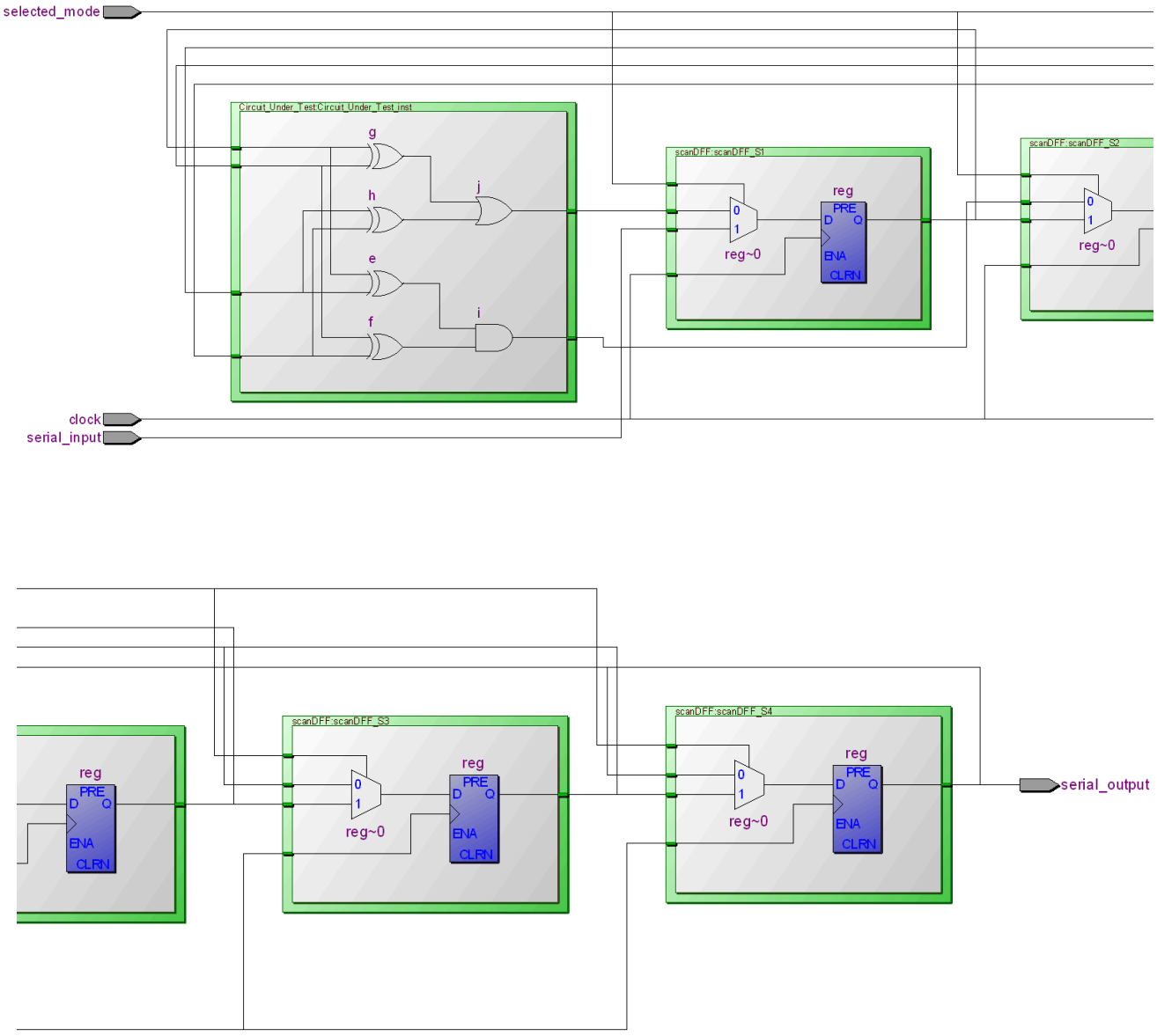



```
   scanDFF_S1 : scanDFF
       port map (
           clock => clock,
           d_in => j_signal,
           serial_input => serial_input,
           q_out => dff_q_out_1,
           selected_mode => selected_mode
       );

   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;
```

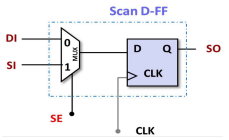
code_snippet_03



Συνθέτοντας το κύκλωμα code_snippet_03 στο quartus προκυπτει η σχεδίαση screen_03 σε επίπεδο RTL. **Λόγω της απλότητας του κυκλώματος με ευκολία** εποπτικά μπορούμε να συμπεράνουμε οτι ταυτίζεται με την σχεδίαση που καναμε εμεις στο αρχικό στάδιο.



screen_03

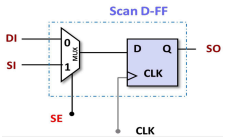


Ερώτημα 1.2 :
Υλοποίηση Testbench για το TRCUT

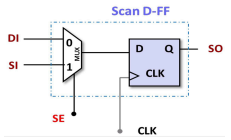
Ο παρακάτω κώδικας (code_snippet_04) δημιουργεί ένα testbench το οποίο φορτώνει σειρικά τα scanDFF με τον πίνακα αλήθειας του CUT και ακολουθόντας την μέθοδο του scan testing εξετάζει την ορθή λειτουργία του κυκλώματος.

Λεπτομερής εξήγηση των σταδίων της εξομοίωσης θα γίνει παραθέτοντας στιγμιότυπα αποκρίσης σημάτων μετά την εκτέλεση testbench

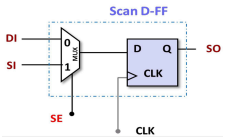
```
LIBRARY ieee  ;
LIBRARY std  ;
USE ieee.std_logic_1164.all  ;
USE ieee.std_logic_arith.all  ;
USE ieee.std_logic_textio.all  ;
USE ieee.STD_LOGIC_UNSIGNED.all  ;
USE ieee.std_logic_unsigned.all  ;
USE std.textio.all  ;

entity scan_testing_testbench is
end entity;


architecture testbench of scan_testing_testbench is

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

   type std_logic_vector_array is array (0 to 15) of std_logic_vector(3 downto 0);
   signal truth_table : std_logic_vector_array;

   type std_logic_vector_array1 is array (0 to 15) of std_logic_vector(3 downto
0);
   signal serial_output_log : std_logic_vector_array1;

   signal tb_line: std_logic_vector(3 downto 0) := "0000";
```




```
   signal first_execution : boolean := true;

   signal correct_circuit_output : std_logic := 'X';

   signal test_bit : std_logic;

   signal a,b,c,d,e,f,g,h,ii,j : std_logic := 'X';

begin

   DUT : Test_Ready_Circuit_Under_Test
       port map (
           clock           => clock,
           serial_input    => serial_input,
           selected_mode   => selected_mode,
           serial_output   => serial_output);

   clock <= not clock after 2.5 ns;



   process is
   begin


       if first_execution then first_execution <= false;


           -- truth table initialization
           for i in 0 to 15 loop
               truth_table(i) <= tb_line;
               tb_line <= tb_line + '1';
               wait for 5 ns;
           end loop;


           -- Checking serial is working
           wait for 2.5 ns;
           serial_input <= '0';
           selected_mode <= '1';
           wait for 5 ns;


           serial_input <= not serial_input;
           wait for 5 ns;

           serial_input <= not serial_input;
           wait for 5 ns;

           serial_input <= not serial_input;
           wait for 5 ns;
```

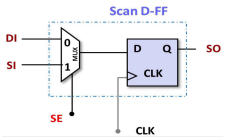



```
        serial_input <= 'X';
        wait for 5 ns;

        -- Scan  Testing
        for i in 0 to 15 loop
            for j in 0 to 3 loop
                serial_input <= truth_table(i)(j);
                wait for 5 ns;
            end loop;
            selected_mode <= '0';
            wait for 5 ns;
            selected_mode <= '1';
        end loop;

        serial_input <= 'X';


    else
        wait for 5 ns;
    end if;
    wait;
end process;

process is
begin

    wait for 135 ns;

    -- serial_output_log_fill
    for i in 0 to 15 loop
        for j in 0 to 3 loop
            serial_output_log(i)(j) <= serial_output;
            wait for 5 ns;
        end loop;
        --ignore last one bit from serial
        wait for 5 ns;
    end loop;


    --- results confirm

    for i in 0 to 15 loop
        a <= truth_table(i)(3);
        b <= truth_table(i)(2);
        c <= truth_table(i)(1);
        d <= truth_table(i)(0);

        wait for 5 ns;

        e <= a xor b;
        f <= c xor d;
```




```
            g <= a xor c;
            h <= b xor d;

            wait for 5 ns;

            ii <= e and f;
            j <= g or h;

            wait for 5 ns;


            if (serial_output_log(i)(3) = j) and (serial_output_log(i)(2) = ii)
then
                correct_circuit_output <= '1';
            else
                correct_circuit_output <= '0';
            end if;

            wait for 5 ns;
        end loop;

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

end architecture;
```

code_snippet_04



Στιγμιότυπα Testbench και επεξήγηση

Τα σήματα που μας ενδιαφέρουν στην εξομοίωση μας είναι τα εξής :

- **clock** : Το σήμα ρολογιου του για το TRCUT. Έχει επιλεγεί περίοδος ρολογιού 5 ns και το ρολόι ξεκινάει αρχικοποιημένο στο λογικό ‘1’ . Άρα στην εξομοίωση μας ξεκινώντας από χρόνο t = 5 ns και άνα 5 ns στην συνέχεια, αντιστοιχείται θετική ακμή του ρολογιού.

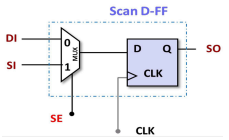
- **selected_mode :** Το σήμα το οποίο καθορίζει την λειτουργία της scan chain. Οταν το σήμα έχει τεθεί στο λογικό ‘0’ και βρισκόμαστε σε θετική ακμή του ρολογιού , τα DFF που αποτελούν την αλυσίδα σάρωσης είτε παίρνουν ως είσοδο D τις εξόδους του κυκλώματος (CUT) είτε ανακυκλώνουν την τιμή που είχα ήδη. Συγκεκριμένα δεδομένου ότι το κυκλωμά μας έχει 2 εξόδους και η αλυσίδα σάρωσης έχει 4 scanDFFs, **όταν το selected_mode βρίσκεται στο λογικό ‘0’ έχει επιλεγει το πρώτο DFF στην αλυσίδα να δέχεται ως είσοδο την τιμή του σήματος j , το δεύτερο scanDFF να δέχεται ως είσοδο την τιμή του σήματος i , ενώ τα υπόλοιπα δύο να ανακυκλώνουν την τιμή τους.** Οταν το σήμα έχει τεθεί στο λογικό ‘1’ , εκτελείται δεξιά ολίσθηση των λογικών τιμών που βρίσκονται αποθηκευμένες μέσα στο εκάστοτε flip-flop.

- **serial_input :** Το σήμα του οποίου η λογική τιμή είναι συνδεδεμένη στην σειριακή είσοδο του πρώτου scanDFF, δηλαδή στην είσοδο της αλυσίδας scan chain. Μέσω αυτού το σήματος φορτώνουμε την αλυσίδα σάρωσης με τις επιθυμητές λογικές τιμές.

- **serial_output :** Το σήμα του οποίου η λογική τιμή αντιστοιχεί στην σειριακή έξοδο της αλυσίδας scan chain , δηλαδή στην έξοδο Q του τελευταίου scanDFF της σειριακής μας αλυσιδας. Μέσω αυτου του σήματος παίρνουμε τις αποκρίσεις του κυκλώματος μας τις οποίες θα τις χρησιμοποιήσουμε για επιβεβαίωση της λειτουργίας του σε δεύτερο χρόνο.

- **truth_table :** Αυτό το σήμα είναι ουσιαστικά ένας πίνακας 16 θέσεων από σήματα των 4ων bit. Όπως προδίδει το όνομα του χρησιμοποιείται για να δημιουργουν όλοι οι πιθανοί συνδυασμοί εισόδου για το κυκλωμά μας και να φορτωθούν μέσω αυτού στην αλυσίδα σάρωσης για να εξετάσουμε την ορθή λειτουργία του CUT.

- **scanDFFregs:** Σε αυτό το σήμα ομαδοποιούνται όλα τα scanDFFs της αλυσίδας σάρωσης μας με την σειρα που βρίσκονται στην αλυσίδα σάρωσης, έτσι ώστε να τα βλέπουμε εποπτικά σαν έναν καταχωρητή των 4ων bit και να μπορούμε εύκολα να βλέπουμε ποιο διάνυσμα εισόδου για το κυκλωμα μας ειναι φορτωμένο στην αλυσίδα σάρωσης κάθε δεδομένη στιγμή αλλά και το εάν οι αποκρίσεις του cut αποθηκεύτηκαν την επιθυμητη χρονική στιγμή επιτυχώς μέσα στην αλυσιδα σάρωσης.

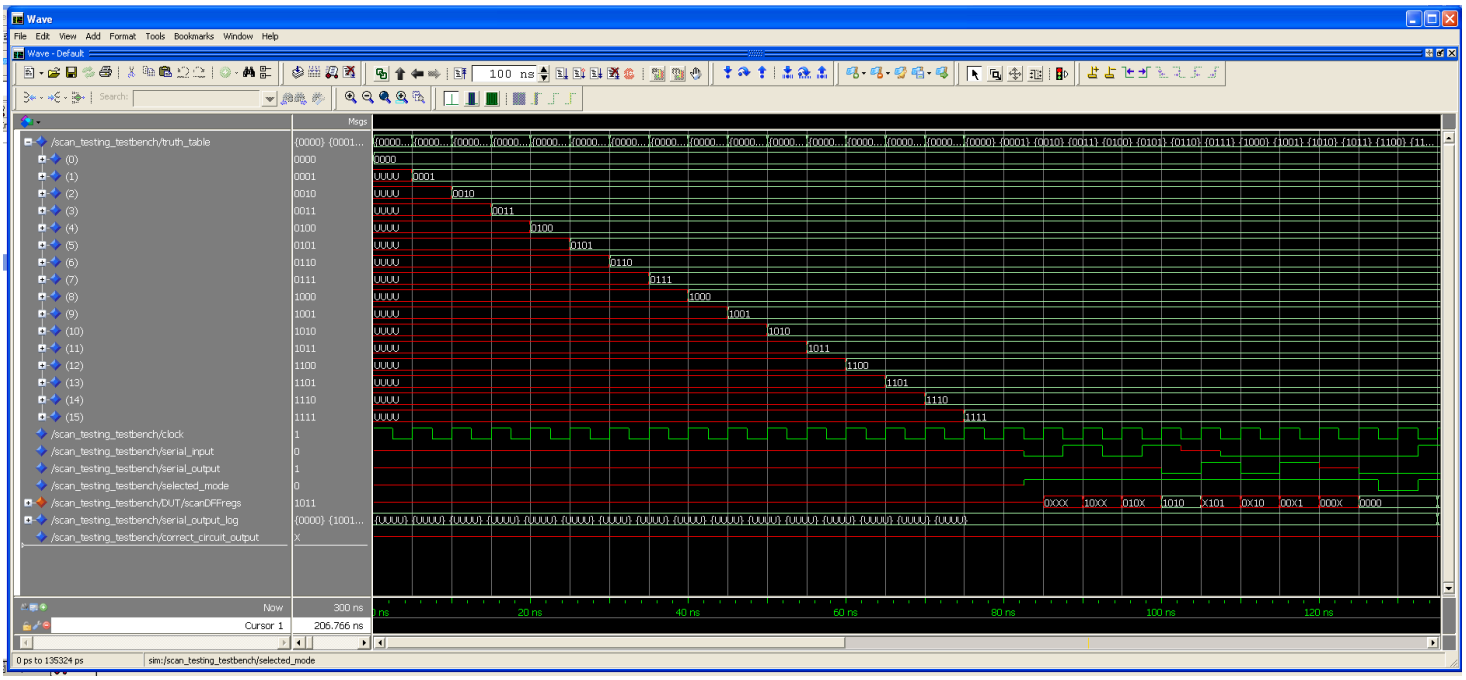


- **serial_output_log :** Αυτο το σήμα είναι ουσιαστικά ένας πίνακας 16 θεσεων ο οποίος αποθηκεύει τις λογικές τιμές της σειριακής εξόδου της αλυσίδας σάρωσης προκειμένου να μπορέσει μετά να γίνει επαλήθευση των αποκρίσεων του κυκλώματος μας. Το σε τι αντιστοιχούν οι τιμές που αποθηκεύει αλλά και το πως συλλέγονται θα εξηγηθεί στην συνέχεια.
- **correct_circuit_output :** Το σήμα αυτό χρησιμοποιείται για να δείξει ότι οι τιμές που συλλέχθηκαν από την αλυσίδα σάρωσης ως αποτέλεσμα της φόρτωσης της με τον πίνακα αλήθειας του CUT είναι σωστές. Συγκεκριμένα όταν έχει τελειώσει το scan testing και έχουν συλλεχθεί οι αποκρίσεις του CUT από την αλυσίδα σάρωσης, συγκρίνουμε τα αποτελέσματα που συλλέξαμε απο την αλυσίδα σάρωσης με τις εξόδους του CUT δεχομένο ως είσοδο τα ιδια διανυσματα που φορτώσαμε στην αλυσιδα σάρωσης. Αν οι αποκρίσεις του CUT για το ίδιο διάνυσμα ως είσοδο ταυτίζονται θέτουμε το εν λογο σήμα στο λογικό ‘1’ αλλιώς το θέτουμε στο λογικό ‘0’. Ζητούμενο προφανώς είναι το συγκεκριμένο σήμα για ολες τις συγκρίσεις που θα γίνουν να βρίσκεται στο λογικό ‘1’.

Στην συνέχεια θα παρουσιαστούν στιγμιότυπα από την εξομοίωση του κυκλώματος με την χρήση testbench.

Όλα τα στιγμιότυπα προέρχονται από την ίδια εξομοίωση και το καθένα είναι η συνέχεια στο χρόνο του προηγούμενου.

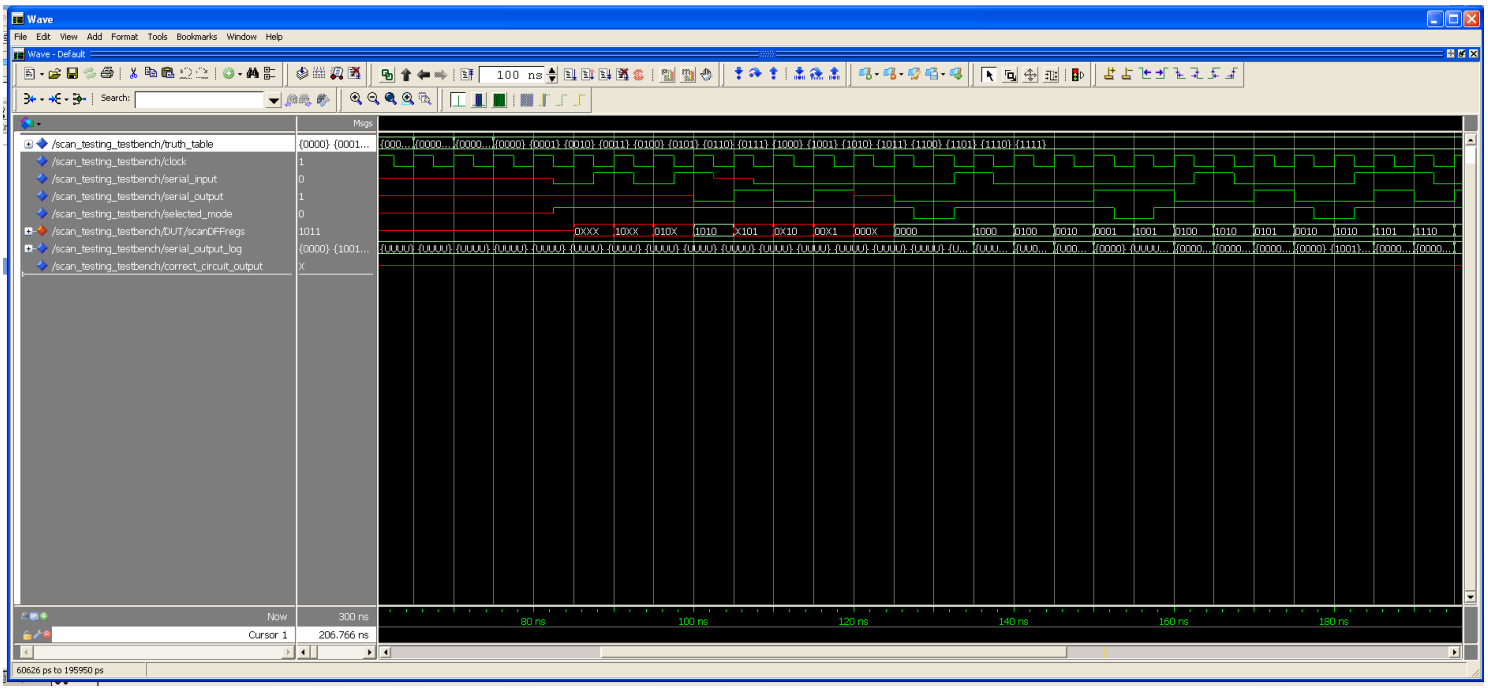
Το scale της προσομοίωσης έχει επιλεχθεί να έχει βήμα τα 5 ns (κάθε άσπρη γραμμή) για να ταυτίζεται και με την περίοδο λειτουργίας του ρολογιού και να είναι εύκολη η οπτική κατανόηση των πεπραγμένων.



Στιγμιότυπο 1

Στο παραπάνω στιγμιότυπο εμφανίζεται η αρχή της εξομοίωσης μας.
Απο την χρονική στιγμή t0 = 0ns εώς την χρονική στιγμή t1 = 75 ns δημιουργείται ο πίνακας αλήθειας του κυκλώματος μας και αποθηκεύεται στο σήμα truth table ως ένας πίνακας από std_logic_vectors(3 downto 0).

Στο code snippet 04 στο σημείο κάτω από το σχόλιο #truth table initialization βρίσκεται ο κωδικας που αντιστοιχεί στην παραπάνω πράξη.



Στιγμιότυπο 2

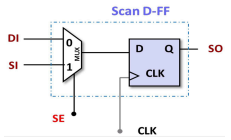
Στο παραπάνω στιγμιότυπο την χρονική στιγμή t = 80 ns έχει ολοκληρωθεί η δημιουργία του πίνακα αλήθειας. Επόμενο βήμα είναι η εξέταση του αν η δεξιά ολίσθηση της αλυσίδας σάρωσης μας λειτουργεί όρθα πριν φορτώσουμε τον πίνακα αλήθειας και αρχίσουμε το scan testing.

Αρα :

Την χρονική στιγμή t = 82.5 ns θέτουμε το selected_mode <= ‘1’ και το σήμα που συνδέεται με την σειριακή είσοδο της αλυσίδας σάρωσης στο serial_input <= ‘0’.

Αρα στην επόμενη θετική ακμή του ρολογιού την χρονική στιγμή t = 85 ns θα αρχισουν να φορτώνονται στην αλυσίδα σάρωσης οι λογικές τιμές του σήματος serial_input.

Με κατάλληλες αλλαγές του σήματος serial_input (με διαφορά φάσης 2.5 ns απο την θετική ακμή του ρολογιού) φορτώνουμε το λογικό διάνυσμα “1010” στην αλυσίδα σάρωσης πράγμα που γίνεται εμφανές παρατηρώντας τις λογικές τιμές που αποθηκεύονται λόγω της δεξιάς ολίσθησης μέσα στο σήμα scanDFFregs μέτα απο 4 κύκλους ρολογιού.

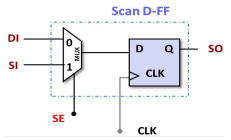


Αφου έχει φορτωθεί το διάνυσμα αυτό των 4ων bit θέτουμε serial_input<='X' **για 5 ns (δηλαδή για μια θετική ακμή ρολογιού)** προκειμένου το ξεχωρισουμε και να το δούμε σειριακά να βγαίνει από την αλυσίδα σάρωσης.
Στο code_snippet_04 στο σημείο κάτω από το σχόλιο #Checking serial is working βρίσκεται ο κωδικας που αντιστοιχεί στην παραπάνω πράξη.

Μετα απο τα 5 ns όπου το serial input <= 'X' **(την χρονική στιγμή t= 107.5 ns δηλαδη)** και ενώ παραμένει στο selected mode <= '1' (δεξιά ολίσθηση) και αφου επιβεβαιώσαμε οπτικά οτι η αλυσίδα σάρωσης όσον αφορά την σειριακή μετατόπιση λειτουργει οπως αναμενόταν, αρχίζουμε να φορτώνουμε στην αλυσίδα σάρωσης bit ανα bit γραμμές διανυσμάτων του πίνακα αληθείας που φτιάξαμε προηγουμένως.

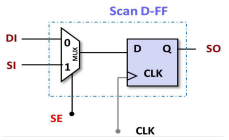
Για κάθε διάνυσμα του πίνακα αλήθειας των 4ων bit ανα 5ns και έχοντας διαφορά φάσης με το ρολόι 2.5 ns θέτουμε το serial_input να ισούται με το εκάστοτε bit του διανυσματος αυτου **ξεκινωντας απο το λιγότερο σημαντικο bit** , δηλαδή φορτώνουμε σειριακά απο τα δεξιά προς τα αριστερα το κάθε διάνυσμα του πίνακα αλήθειας.

Προκειμένου να γίνει κατανοητή η λειτουργία της αλυσίδας σάρωσης θα εξηγηθεί βήμα-βήμα η φόρτωση του λογικου διανύσματος '0000' στην αλυσίδα σάρωσης , το πως γίνεται η καταγραφή της απόκρισης του CUT αλλά και το πως βγαίνει το αποτέλεσμα προς τα εξω για επεξεργασία του σε δεύτερο χρόνο.



Επεξήγηση λειτουργίας της αλυσίδας σάρωσης για το πρώτο διάνυσμα εισόδου -> “0000”

1. Θετική ακμή ρολογιού την χρονική στιγμή t = 110 ns
    - Φορτώνεται το πρώτο bit του διανύσματος “000**0**” με δεξιά ολίσθηση στην αλυσίδα σάρωσης.
2. Θετική ακμή ρολογιού την χρονική στιγμή t = 115 ns
    - Φορτώνεται το δεύτερο bit του διανύσματος “00**0**0” με δεξιά ολίσθηση στην αλυσίδα σάρωσης.
3. Θετική ακμή ρολογιού την χρονική στιγμή t = 120 ns
    - Φορτώνεται το τρίτο bit του διανύσματος “0**0**00” με δεξιά ολίσθηση στην αλυσίδα σάρωσης.
4. Θετική ακμή ρολογιού την χρονική στιγμή t = 125 ns
    - Φορτώνεται το τέταρτο bit του διανύσματος “**0**000” με δεξιά ολίσθηση στην αλυσίδα σάρωσης.
5. Χρονική στιγμή τ = 127,5 ns
    - Αλλάζουμε το σήμα selected_mode στο λογικό ‘0’ έτσι ώστε στην επόμενη θετική ακμή του ρολογιού να συλλέξουμε και να αποθηκεύσουμε την απόκριση του CUT στα πρώτα 2 scanF της αλυσίδας σάρωσης.
6. Θετική ακμή ρολογιού την χρονική στιγμή t = 130 ns
    - Η αλυσιδα σάρωσης ενημερώνεται με το αποτέλεσμα της απόκρισης του κυκλώματος. **Στο πρώτο scanDFF (έχοντας ως σημείο αναφοράς την είσοδο της αλυσίδας σάρωσης) αποθηκεύεται η τιμή της εξόδου j του CUT ενώ στο δεύτερο η τιμή της εξόδου i.** Στα υπόλοιπα 2 scanDFFs απλά έχει ανακυκλωθεί η τιμή τους.
    - Επειδή όπως θα φανεί αργότερα **η έξοδος του CUT οταν δέχεται ώς είσοδο το λογικο διάνυσμα “0000”(δηλαδη όταν a=b=c=d= ‘0’) είναι το j = 0 και j = 0** , δεν φαίνεται κάποια οπτική αλλαγή στις τιμές που ειναι αποθηκευμένες στο σήμα scanDFFregs, όμως τα 2 πρώτα μηδενικά αντιστοιχούν στην απόκριση του CUT και δεν είναι η τιμές του διανύσματος εισόδου !
7. Χρονική στιγμή 132.5 ns
    - Αλλάζουμε το σήμα selected_mode στο λογικό ‘1’ έτσι ώστε στην επόμενη θετική ακμή του ρολογιού να αρχίσει να φορτώνεται η επομενη γραμμη του πίνακα αλήθειας ενώ παράλληλα θα αρχίσει να βγαινει εξω απο την αλυσίδα σάρωσης με δεξιά ολίσθηση το λιγότερο σημαντικό bit του πρώτου διανύσματος του πίνακα αλήθειας “κουβαλώντας” μέσα του στα 2 πρώτα bits την αποκριση του κυκλωματος σε αυτό. **Δηλαδή κάθε 4 bits που βγαινουν απο την αλυσίδα σάρωσης , τα πρώτα 2 στο διάνυσμα εξόδου είναι η απόκριση του CUT σε κάποιο διάνυσμα που δέχτηκε ως είσοδο.**

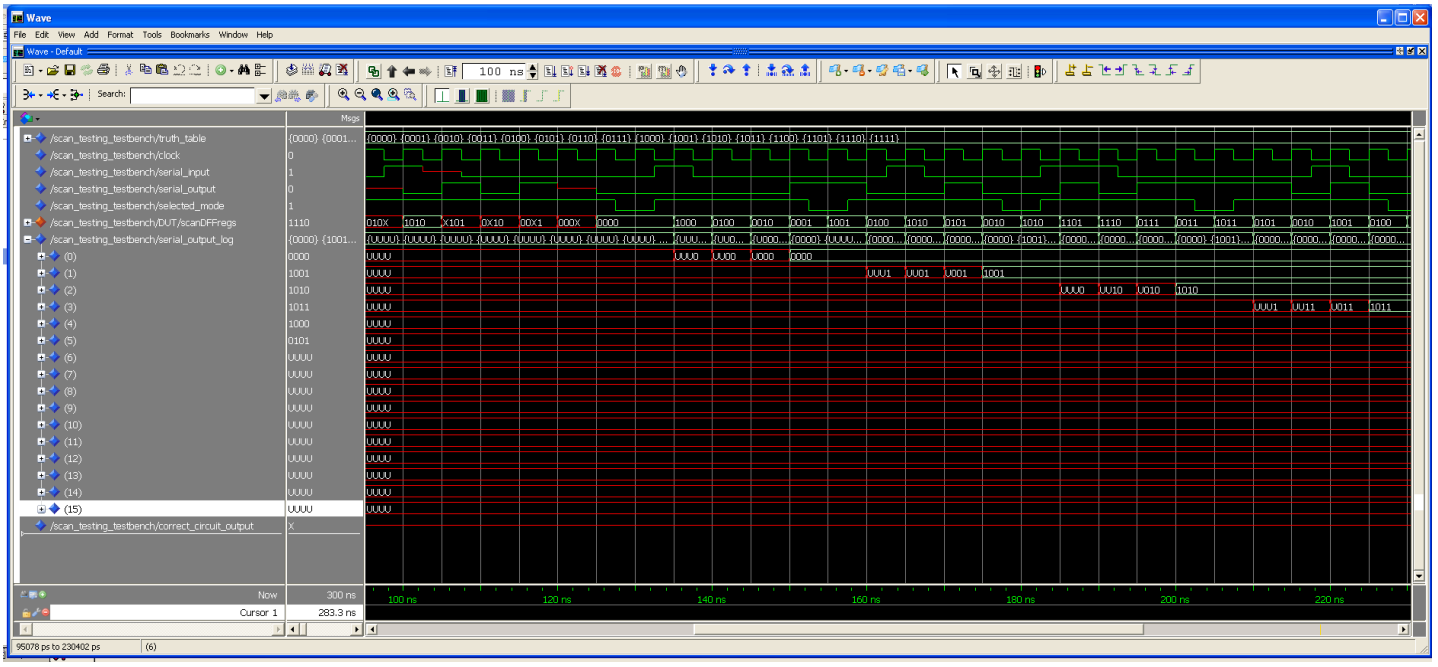


8. Χρονική στιγμή t = 135 ns
    - Σε αυτό το σημείο επαναλαμβάνεται η διαδικασία που ακολουθήθηκε προηγουμένως, ξεκινώντας πάλι από το βήμα 1, με ώς είσοδο στην αλυσίδα το επόμενο διάνυσμα του πίνακα αλήθειας, δηλαδή το “0001” εώς ότου φορτώσουμε όλα τα διανύσματα του πίνακα αλήθειας που απαιτουνται για ενα κυκλωμα των 4ων εισοδων.

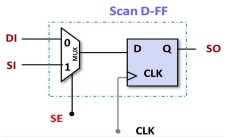
Με κατάλληλο συγχρονισμο στο test bench μας έχουμε φροντίσει απο την θετική ακμή του ρολογιού που βρισκεται την χρονική στιγμή t = 135 ns και μετα , να αποθηκεύουμε στο σήμα serial_output_log την σειριακη εξοδο της αλυσίδας σάρωσης σε κάθε θετική ακμή του ρολογιού **εκτος απο την θετική ακμή του ρολογιού κάθε χρονική στιγμή που η αλυσίδα βρίσκεται σε capture_mode και αρα δεν γινεται δεξιά ολίσθηση αλλά αλλά παραμένει σταθερή η προηγούμενη τιμή που είχε στο τελευταιο της scanDFF ως έξοδος πάλι.** Δηλαδή πρακτικά ,μόνο όταν γίνεται δεξιά ολίσθηση αποθηκεύουμε την έξοδο της αλυσίδας σάρωσης.

Στο code_snippet_04 στο σημείο μετά το σχόλιο #serial_output_log_fill βρίσκεται ο κωδικας που αντιστοιχεί στην παραπάνω πράξη.

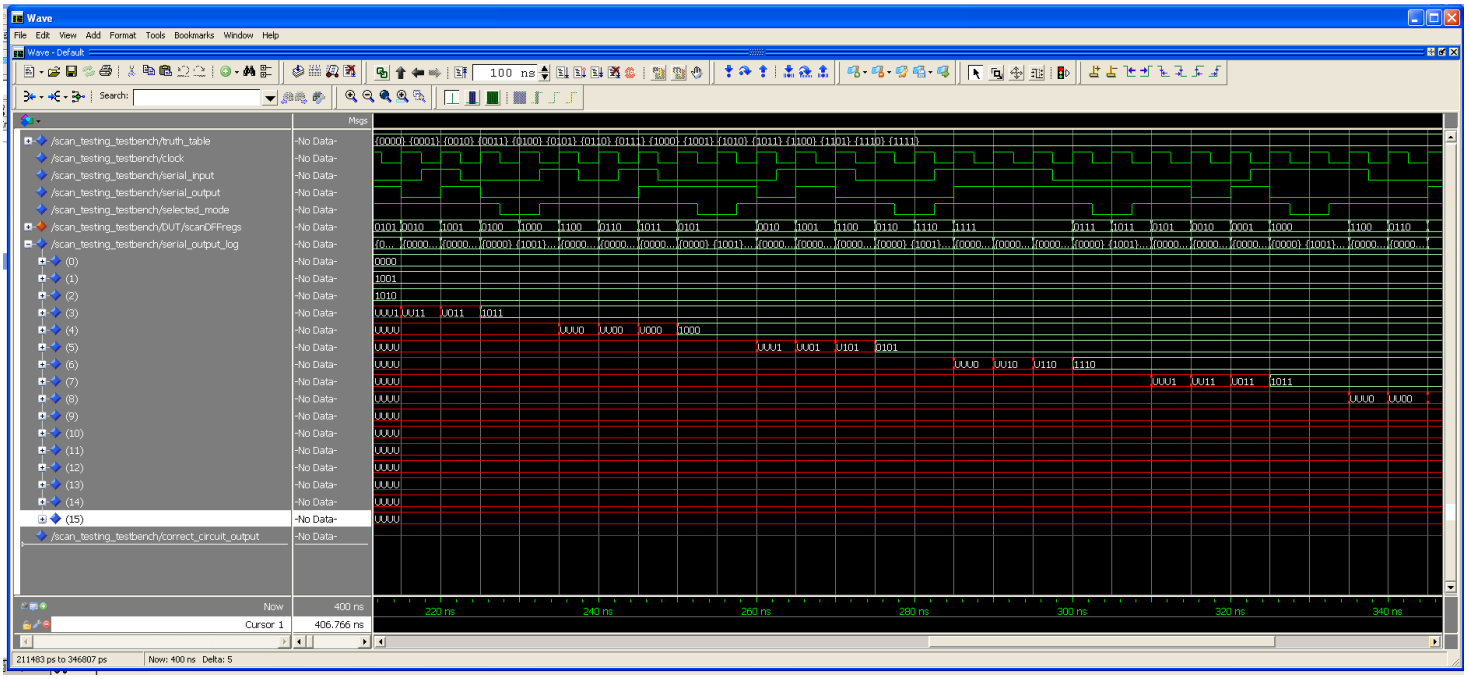
Στην παρακάτω εικόνα φαίνεται λεπτομερως το πως αποθηκεύεται η σειριακή έξοδος της αλυσίδας σάρωσης στο διάνυσμα serial_output_log αλλά και το πως αγνοείται για έναν κύκλο ρολογιού το σήμα στην έξοδο αυτης, όταν είμαστε σε capture mode.



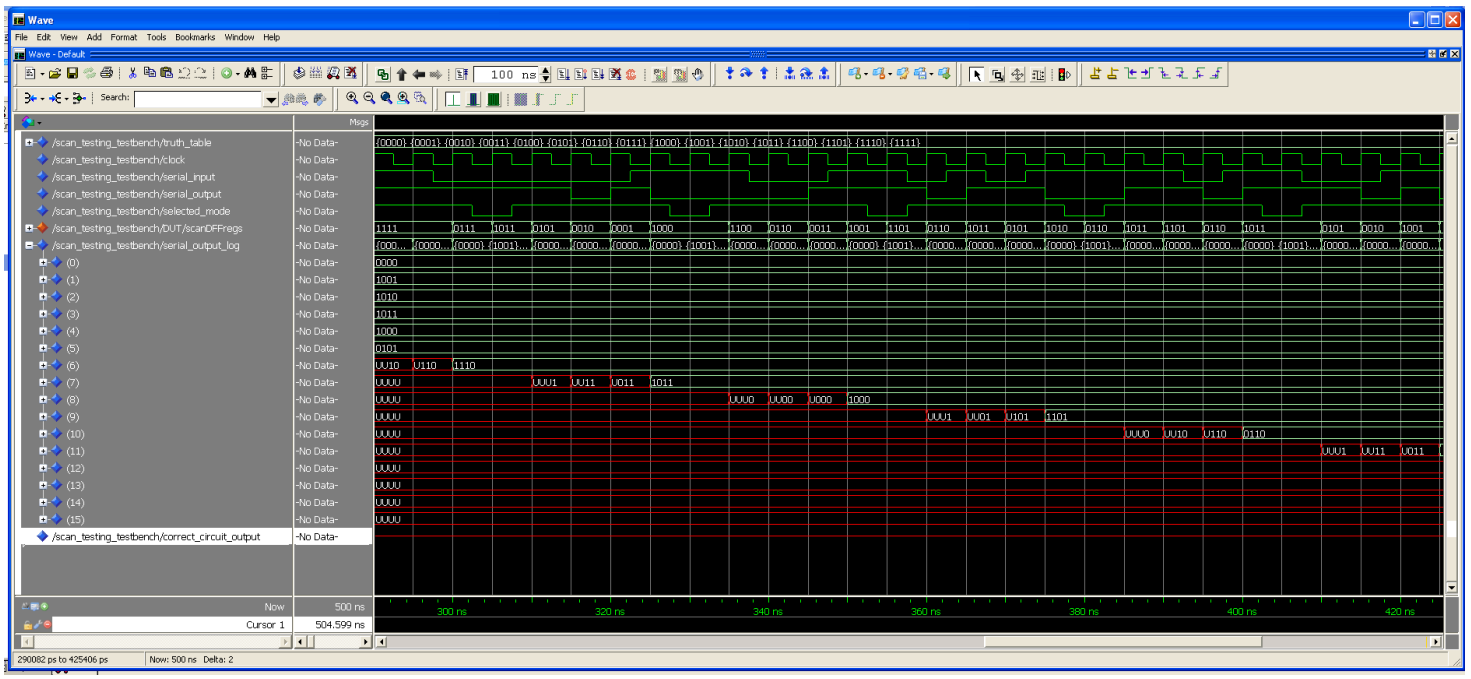
Στιγμιότυπο 4



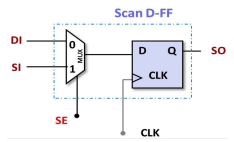
Στα παρακάτω στιγμιότυπα φαίνεται η λειτουργία της αλυσίδας σάρωσης, επιτελώντας όλη την λειτουργικότητα που αναφέρθηκε προηγουμένως , για όλα τα διανύσματα του πίνακα αλήθειας που απομένουν.

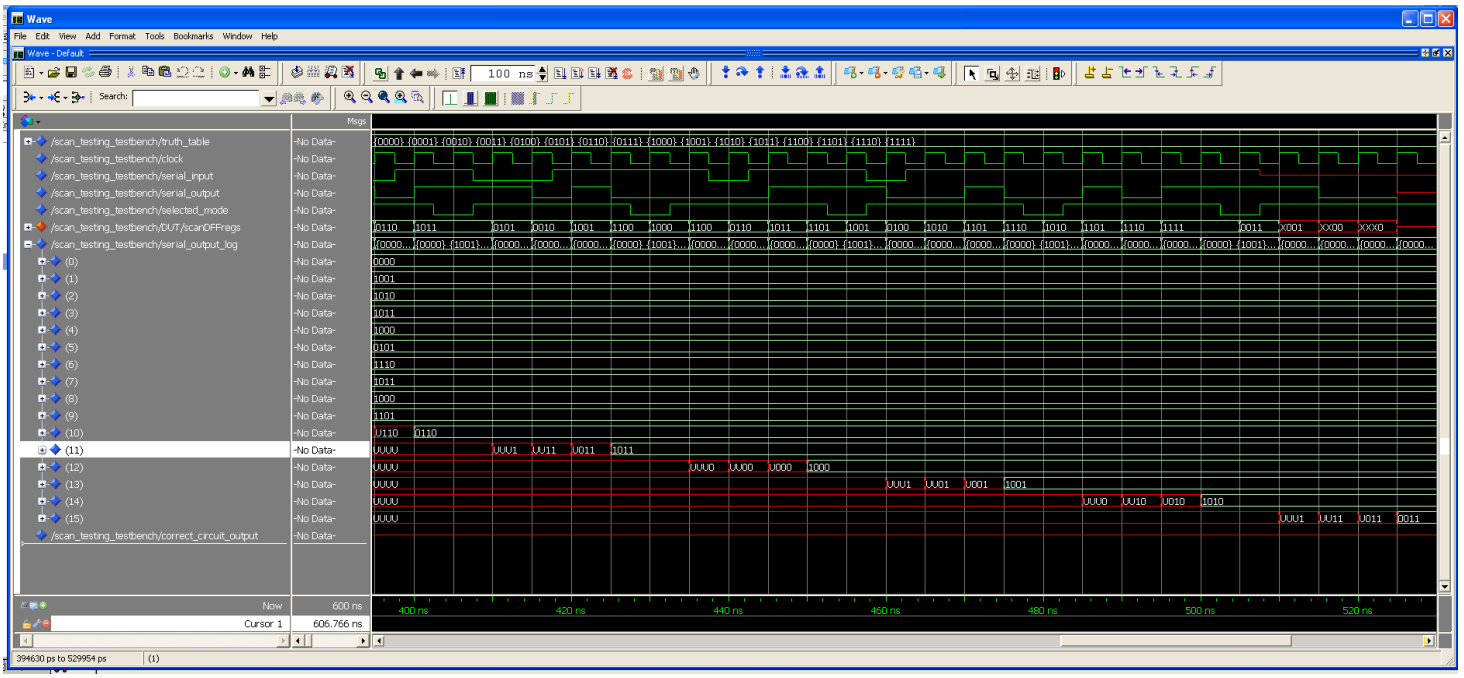


Στιγμιότυπο 5

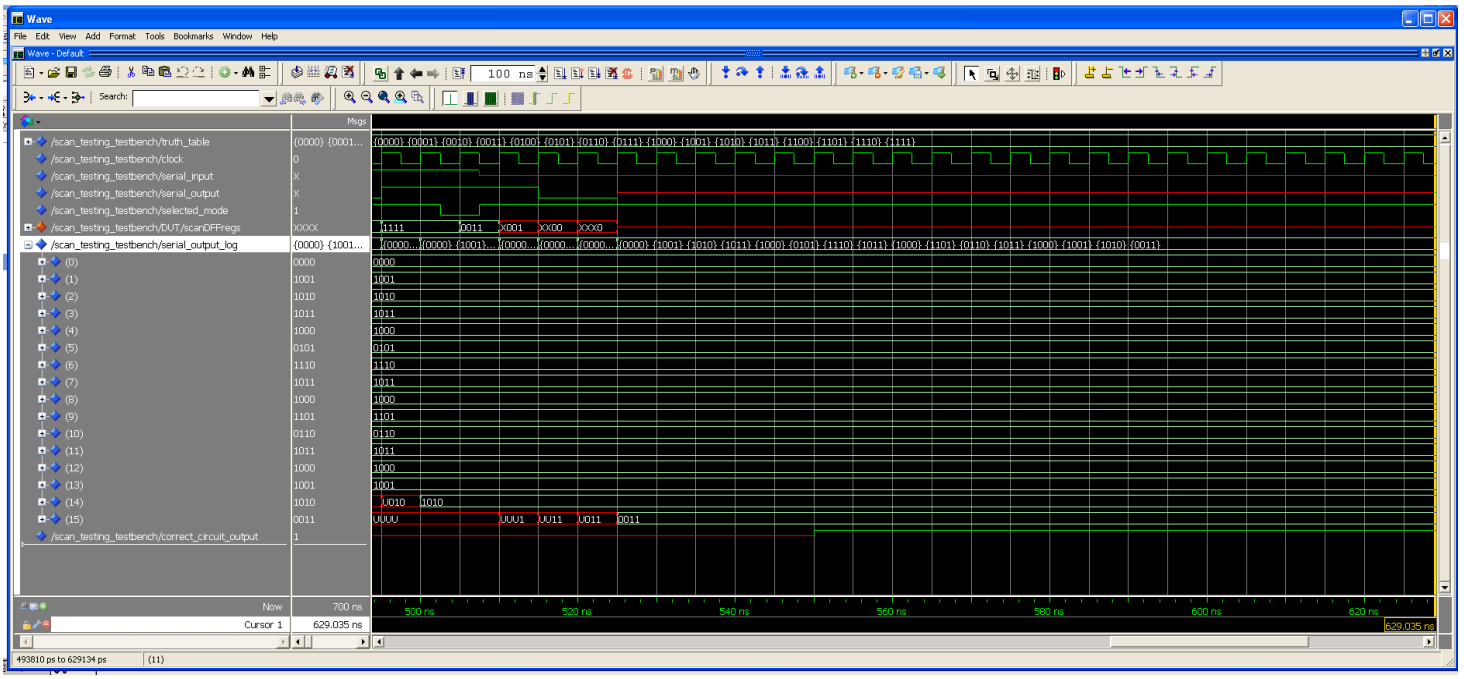


Στιγμιότυπο 6

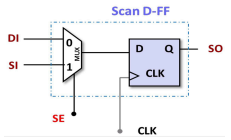




Στιγμιότυπο 7



Στιγμιότυπο 8



Την χρονική στιγμή t = 500 ns παρατηρούμε ότι έχει φορτωθεί στην αλυσίδα σάρωσης και το τελευταίο διάνυσμα του πίνακα αλήθειας ενώ την χρονική στιγμή τ = 525 ns έχουν “βγει” και συλλεχθεί από την αλυσίδα σάρωσης όλα τα διανυσματα που μας ενδιαφέρουν.

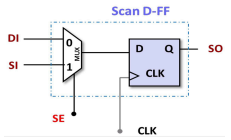
Για να είναι πιο εμφανές το σημείο που τελειώνει να δέχεται ως είσοδο η αλυσίδα σάρωσης διανύσματα του πίνακα αλήθειας, μετά το τελευταίο διάνυσμα “1111” θέτουμε την σειριακή είσοδο της αλυσίδας σάρωσης να δέχεται ως είσοδο το σήμα ‘X’.

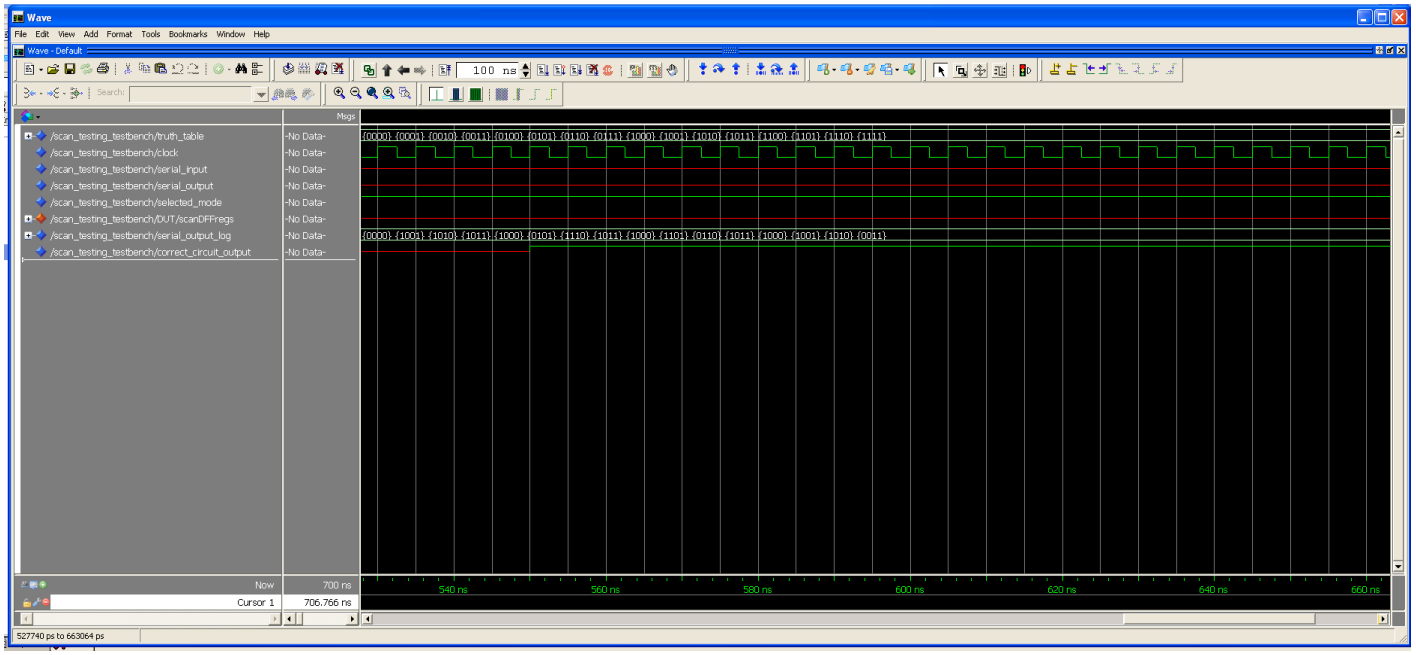
Τέλος με κώδικα στο testbench συγκρίνουμε τις αποκρίσεις της αλυσίδας σάρωσης με τις αναμενομενες αποκρίσεις του κυκλώματος και αν το αποτελεσμα ειναι σωστό θέτουμε το σήμα correct_circuit_output στο λογικό ‘1’.

Στο code_snippet_04 στο σημείο κάτω από το σχόλιο #results confirmation βρίσκεται ο κωδικας που αντιστοιχεί στην παραπάνω πράξη.

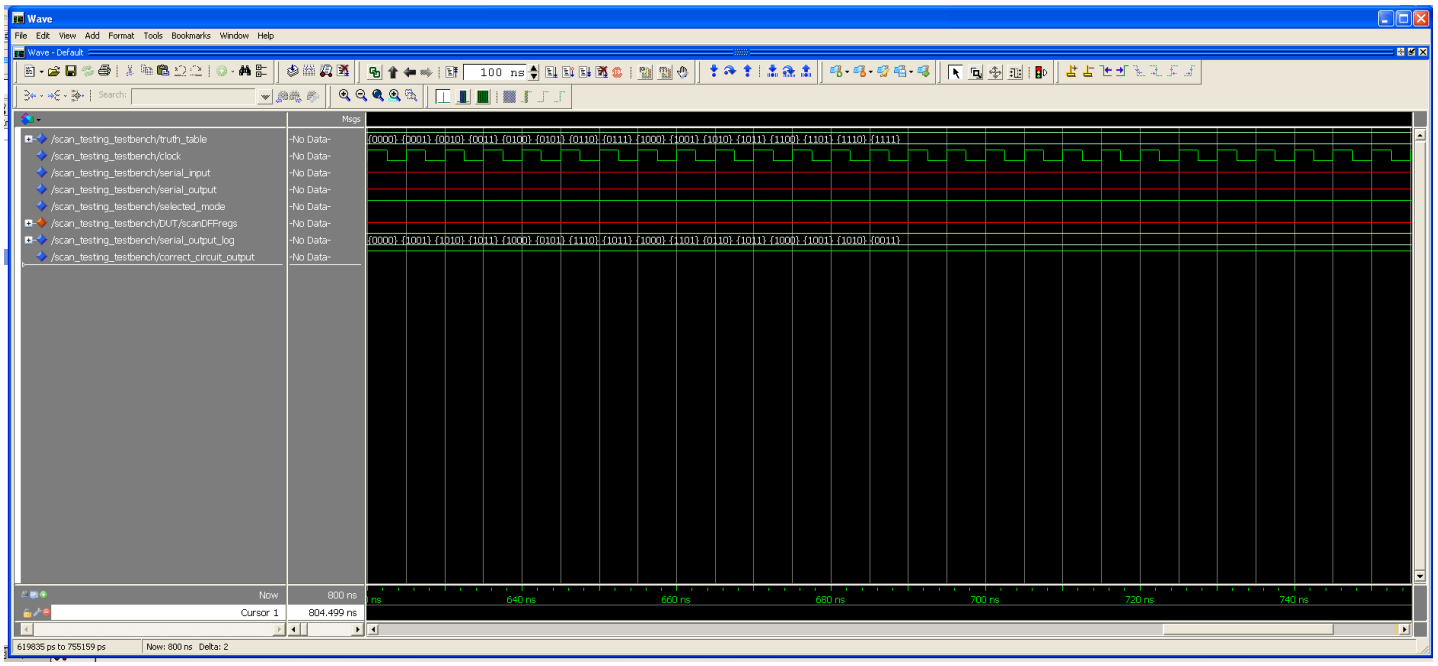
Απο την χρονική στιγμή t = 550 ns εως και την χρονική στιγμή t = 855 ns **(χρόνο δηλαδη ίσο με το χρονο που χρειαστηκε για να επαληθευτουν ολες οι λογικες τιμες του πίνακα αλήθειας με τις αποκρίσεις του κυκλώματος από τον πίνακα seria_output_log)** το σήμα correct_circuit_output παραμένει συνεχώς στο λογικό 1 , γεγονός που δηλώνει την ορθότητα της αλυσίδας σάρωσης μας αλλα και της εξομοιωσης.

Για οπτικούς λόγους , μετά το πέρας της επαλήθευσης το σήμα correct_circuit_output τίθεται στο ‘X’

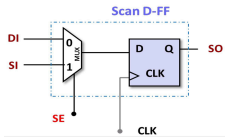


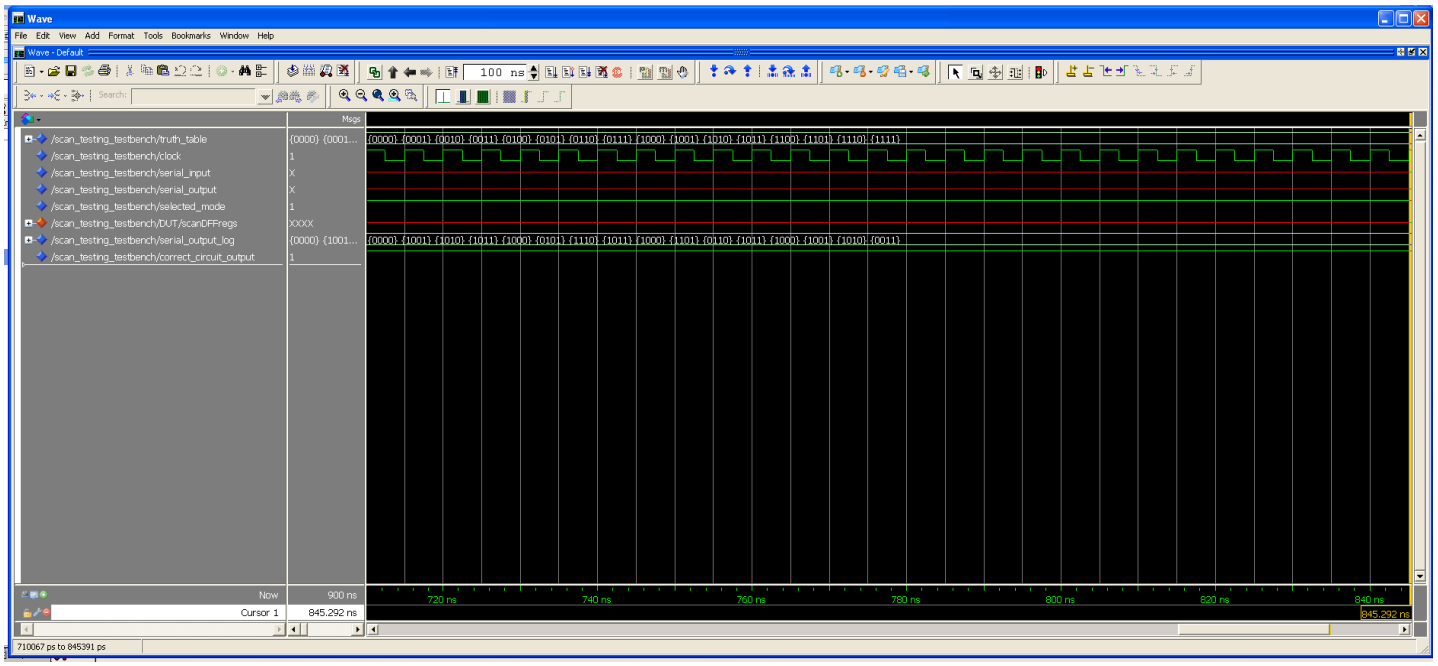


Στιγμιότυπο 9

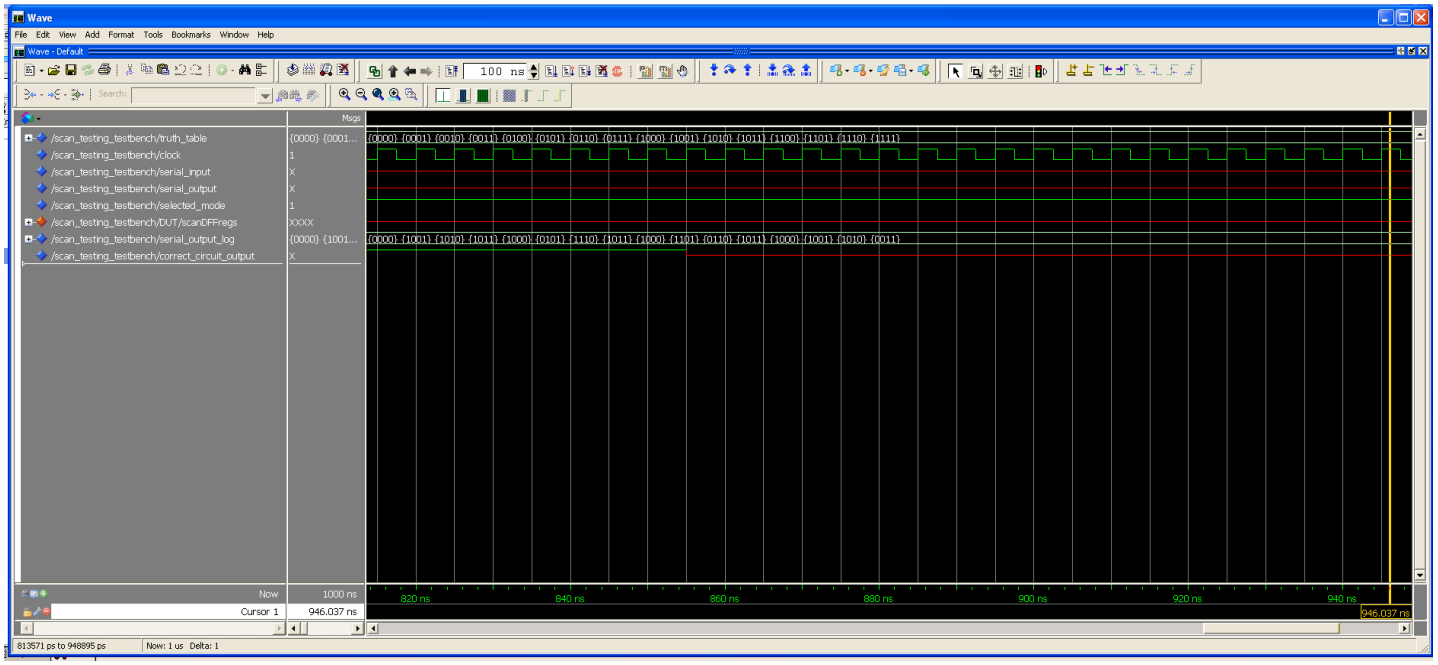


Στιγμιότυπο 10

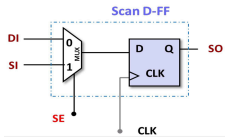




Στιγμιότυπο 11



Στιγμιότυπο 12



Ερώτημα 1.3 : Χρόνος Scan Testing

Δεδομένου των προηγούμενων ερωτημάτων μπορούμε να συμπεράνουμε οτι μια αλυσίδα σάρωσης για να διεξάγει έλεγχο σωστής λειτουργίας ενός κυκλώματος χρησιμοποιώντας τον πίνακα αλήθειας του χρειάζεται χρόνο :

t = t1*t2 + t3

Όπου

t1 = 2^(Αριθμός_εισόδων_CUT)

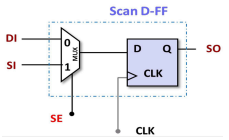
Αριθμός στηλών του πίνακα αλήθειας.

t2 = clokc_cycle*(Αριθμός_εισόδων_CUT + 1)

Κύκλοι ρολογιού που χρειάζονται για να φορτωθεί μια γραμμή του πίνακα αλήθειας στην αλυσίδα σάρωσης συν ενα κύκλο ρολογιού που αντιστοιχεί στο capture της γραμμής.

t3 = clock_cycle*Αριθμός_εισόδων_CUT

Κύκλοι ρολογιου που απαιτούνται για να βγει η τελευταία γραμμή του πίνακα αλήθειας (η οποία θα περιέχει και την απόκριση του κυκλώματος μέσα της) απο την αλυσιδα σάρωσης.



Για συχνότητα ρολογιού = 10 Mhz = $10 * 10^6$ Hz έχουμε περίοδο ρολογιού t = $1/10 * 10^6$ second = 10^{-7} seconds = 100 ns περίοδος ρολογιού.

Χρησιμοποιώντας τον παραπάνω τύπο εύκολα προκύπτει αν N = αριθμός εισόδων κυκλώματος ότι :

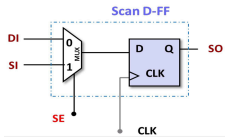
N=10 => Χρόνος scan testing = $2^{10} * 10^{-7} * (10 + 1) + 10^{-7}*10$ = 0.0011274 s

N=20 => Χρόνος scan testing = $2^{20} * 10^{-7} * (20 + 1) + 10^{-7}*20$ = 2.2020116 s

N=30 => Χρόνος scan testing = $2^{30} * 10^{-7} * (30 + 1) + 10^{-7}*30$ = 3328.59965 s

N=40 => Χρόνος scan testing
= $2^{40} * 10^{-7} * (40 + 1) + 10^{-7}*40$ = 4507997.6738856 s

Είναι εμφανές ότι για N > 20 εισόδων σε κάποιο κυκλώμα ή εξέταση όλων των διανυσμάτων του πίνακα αλήθειας του με την μέθοδο scan testing καταλήγει να είναι λόγω χρόνου μη πρακτικά εφαρμόσιμη.


Scan D-FF
DI
SI
SE
SO
CLK
D
Q
CLK
MUX
0
1