Δοκιμή και Αξιοπιστία Ηλεκτρονικών Κυκλωμάτων

Άσκηση 2:

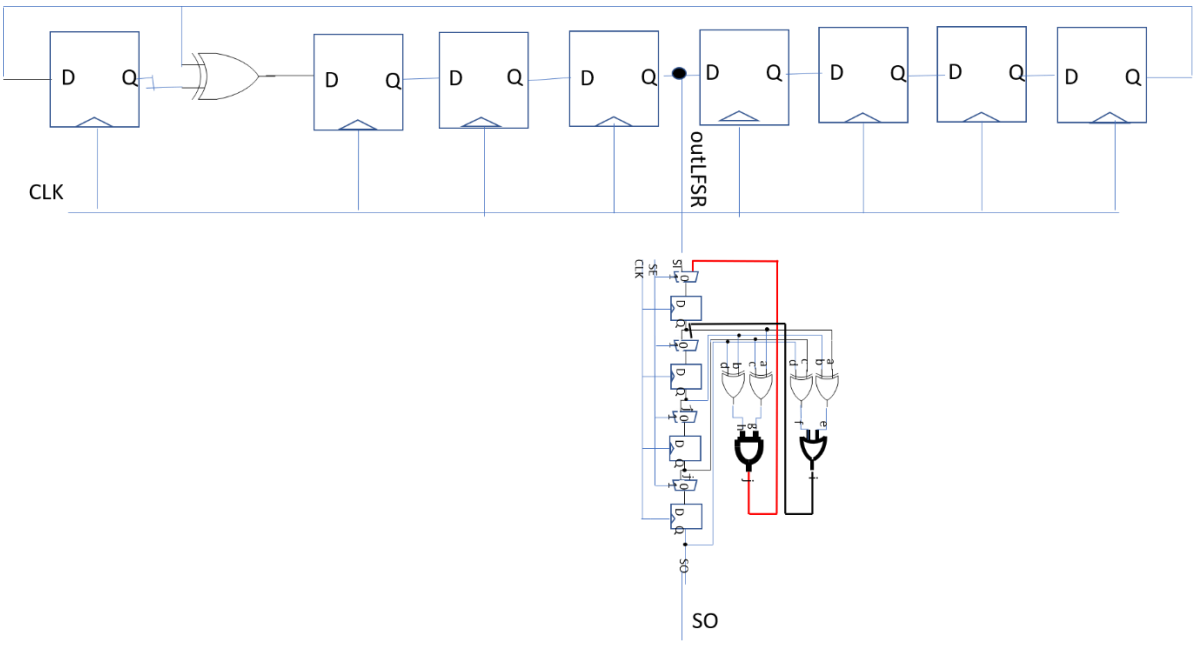
Υλοποίηση Ελέγξιμου Κυκλώματος με Pseudorandom Testing

Θεοδώρα Τζιάστα

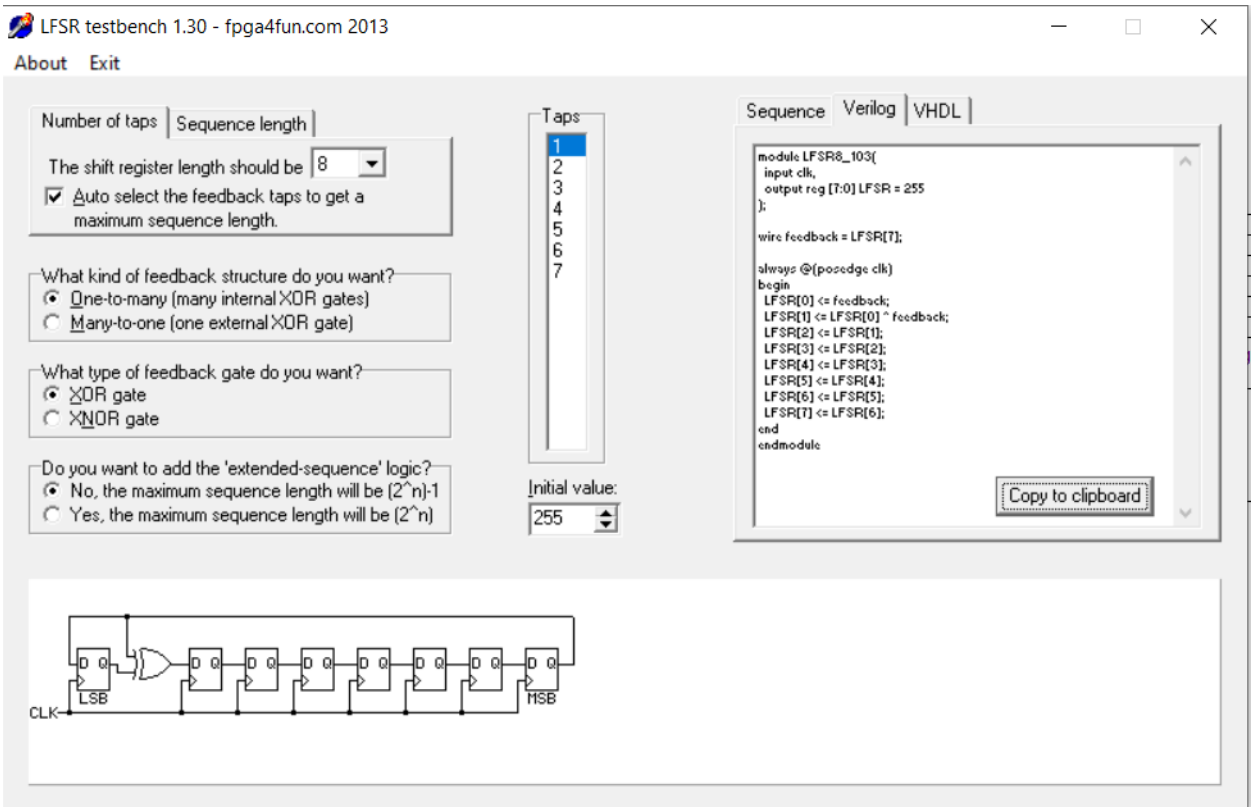
AM: 4178

Άσκηση 2.1:

Για αυτό το ερώτημα της άσκησης 2 δημιουργήθηκε το παρακάτω προσχέδιο:



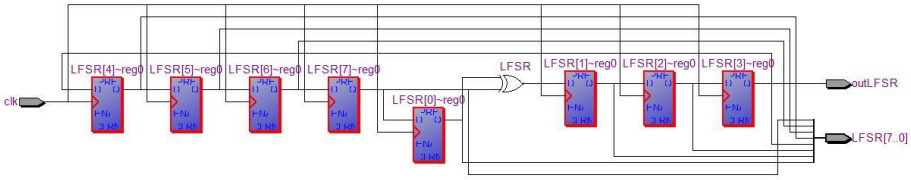
Έπειτα για την υλοπίηση του χρειάστηκε αρχικά με την βοήθεια του λογισμικού LFSRTestbench να παραχθεί ένα LFSR 8^{ου} βαθμού το οποίο φαίνεται στην παρακάτω εικόνα:



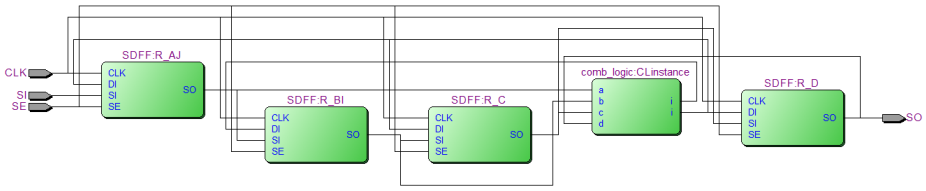
Έπειτα για να υλοποιηθεί το κύκλωμα που μας ζητήθηκε ήταν απαραίτητη η δημιουργία ενός επιπλέον port εξόδου το outLFSR το οποίο οδηγείται από την έξοδο του 3ου DFF του LFSR. Το LFSR τροφοδοτείται με ένα τυχαίο default seed, το διάνυσμα 255.

```
module LFSR_8(
  input clk,
  output outLFSR,
  output reg [7:0] LFSR = 255

);
wire feedback = LFSR[7];
assign outLFSR=LFSR[3];
always @(posedge clk)
begin
  LFSR[0] <= feedback;
  LFSR[1] <= LFSR[0] ^ feedback;
  LFSR[2] <= LFSR[1];
  LFSR[3] <= LFSR[2];
  LFSR[4] <= LFSR[3];
  LFSR[5] <= LFSR[4];
  LFSR[6] <= LFSR[5];
  LFSR[7] <= LFSR[6];
end

endmodule
```

Το LFSR φαίνεται κάπως έτσι:

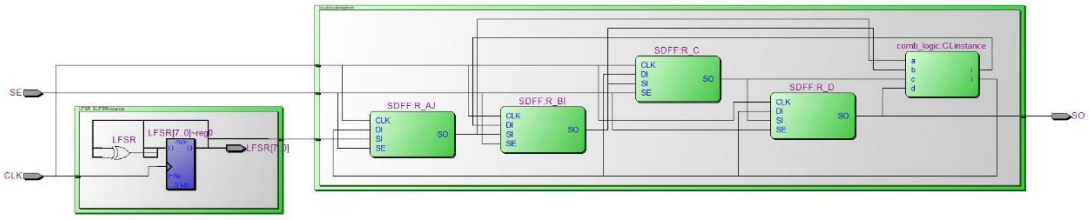


Εν συνεχεία χρησιμοποιήθηκε το TRCUT που είχε δημιουργηθεί στην άσκηση 1. Όμως για να μπορέσει να λειτουργήσει σωστά στο νέο κύκλωμα ήταν αναπόφευκτες κάποιες αλλαγές. Συγκεκριμένα τα observability points των δυο τελευταίων scan cells έπρεπε να συνδεθούν και σε κάποιο υπάρχον σήμα του κυκλώματος, ώστε να μην παράγουν άγνωστες τιμές κατά την λειτουργία του. Συνδέθηκαν στην έξοδο j του κυκλώματος της συνδυαστικής λογικής. Οι αλλαγές είναι εμφανής στο παρακάτω σχήμα που προκύπτει από το Quartus:



Τέλος όλα τα παραπάνω modules συντέθηκαν στο TRCUTwithLFSR και ακολουθεί ο κώδικα σε Verilog και η εικόνα που προκύπτει από το Quartus:

```
module TRCUTwithLFSR(CLK,SE,SO);
        input CLK,SE;
        output SO;
        wire [7:0] out;
        LFSR_8 LFSRinstance(CLK,lfsrout3,out);

        trcut trcutinstance(CLK,SE,lfsrout3,SO);

endmodule
```



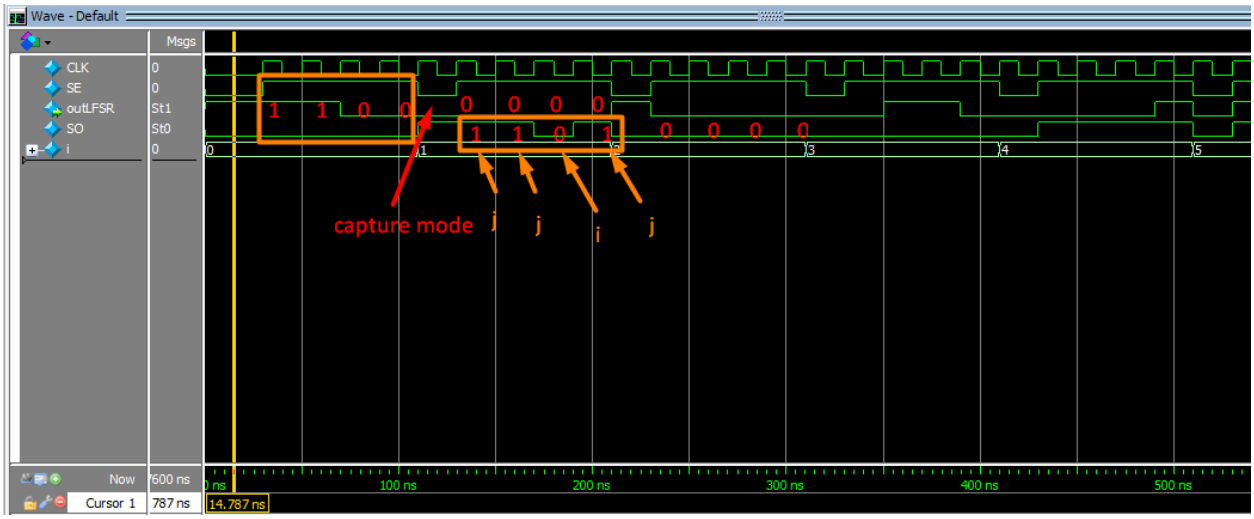
Άσκηση 2.2:

Στο δεύτερο ερώτημα υλοποιήθηκε testbench στο οποίο ελέγχεται η λειτουργία του TRCUTwithLFSR. Με την βοήθεια του LFSR παράγονται 32 ψευδοτυχαία διανύσματα τα οποία εισάγονται στο TRCUT μέσω τις εξόδου outLFSR. Για κάθε ένα διάνυσμα ενεργοποιείται το scan mode με SE=1 για 4 κύκλους ρολογιού εισάγοντας σειριακά στο κύκλωμα τα bit που εφαρμόζονται στην είσοδο SI. Έπειτα, αφού φορτωθεί το διάνυσμα ενεργοποιείται το capture mode με SE=0 για 1 κύκλο ρολογιού φορτώνοντας τις εξόδους του CUT στην αλυσίδα ώστε να παρατηρηθούν οι αποκρίσεις στην έξοδο SO. Στη δική μας περίπτωση παρατηρώντας τα διανύσματα που εισάγονται στο CUT και έχοντας τον πίνακα αληθείας ήδη από την προηγούμενη άσκηση μπορούμε να επιβεβαιώσουμε αν οι αποκρίσεις που προκύπτουν είναι οι αναμενόμενες ή όχι. Αυτό είναι εφικτό επειδή τα κυκλώματα μας είναι σχετικά μικρά. Κλιμακώνοντας το πρόβλημα χρησιμοποιώντας μεγαλύτερα LFSRs, μεγαλύτερες και περισσότερες scan chains τα πράγματα θα γίνουν πολύ περίπλοκα και το να επιβεβαιώσει κανείς την ορθότητα των αποκρίσεων θα αποβεί σε ένα πολύ δύσκολο και χρονοβόρο πρόβλημα. Το testbench ακολουθεί παρακάτω:

```
`timescale 1ns/1ps
module TRCUTwithLFSR_tb();
        reg CLK,SE;
        wire SO;
        integer i;

        TRCUTwithLFSR myinstance(CLK,SE,SO);

        //Block for clock generation
        initial begin
                CLK=0;
                #20
                forever begin
                        #10 CLK=!CLK;
                end
        end
```

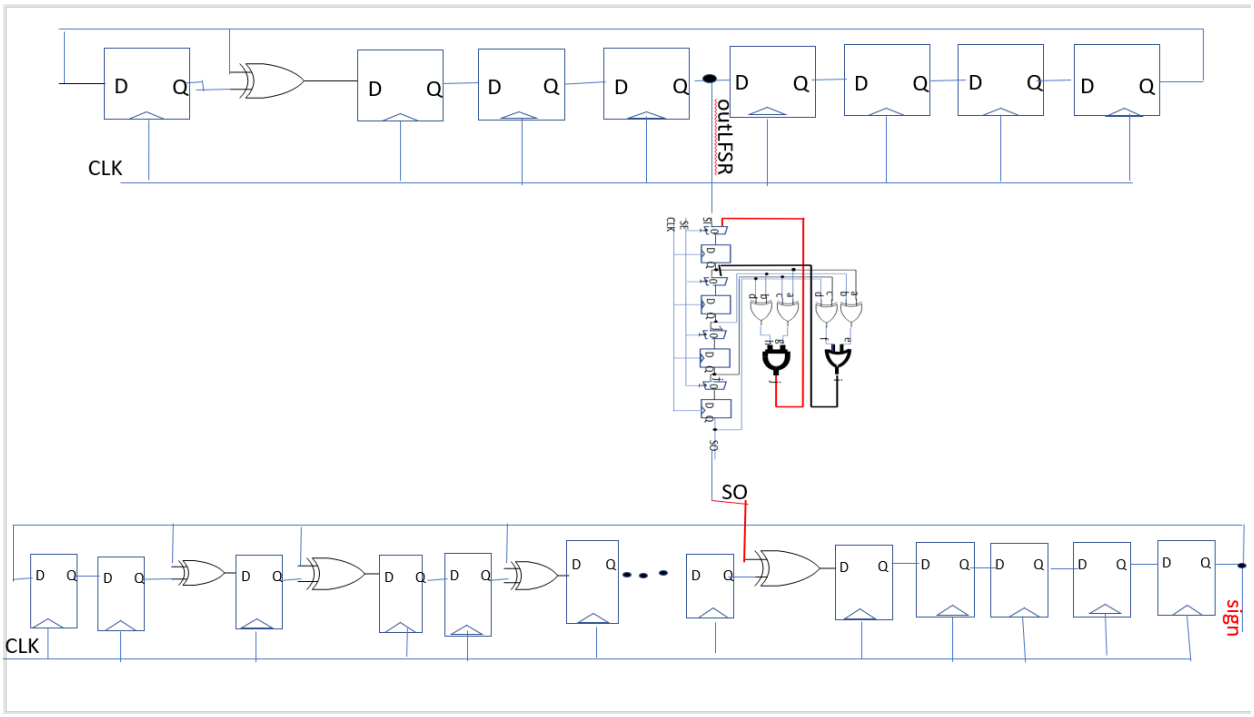
```
initial  begin
        i=0;
        SE=0;
        for(i=0;i<32;i=i+1)begin
                repeat(4) @(posedge CLK)
                        SE=1;//scan mode

                repeat (1)@(posedge CLK)
                        SE=0;//capture mode

        end
```

Ακολουθεί εικόνα με τις κυματομορφές του κυκλώματος που φαίνεται να εξάγονται οι σωστές αποκρίσεις για τα τυχαία διανύσματα που εφαρμόστηκαν στην είσοδο:



Άσκηση 2.3:

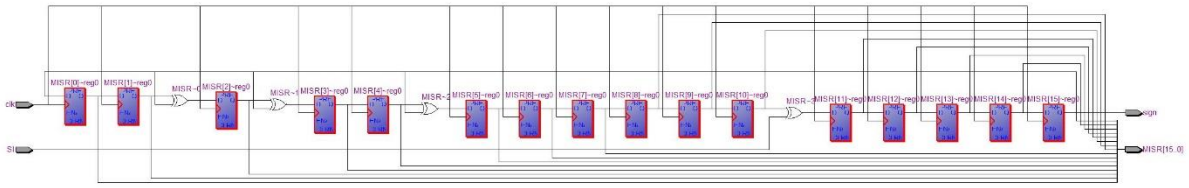
Το προσχέδιο αυτού του ερωτήματος είναι το παρακάτω:



Για την υλοποίηση του παραπάνω προσχεδίου παράχθηκε μέσω του λογισμικού LFSRTestbench ένα LFSR 16ου βαθμού το οποίο θα αποτελεί το MISR στο νέο κύκλωμα που δημιουργείται. Έπειτα προστέθηκε η είσοδος SI, η οποία λαμβάνει τα bits εισόδου και με ένα xor tap μεταξύ του 10ου DFF και τις SI, τα ενσωματώνει στο MISR. Η έξοδος του νέου κυκλώματος θα είναι το sign η οποία οδηγείται από το 15ο DFF. Παρακάτω φαίνεται ο κώδικας και το σχήμα που παράχθηκε από το Quartus:

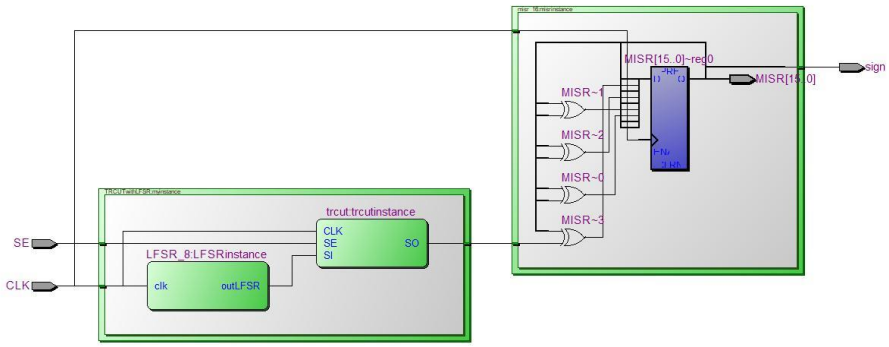
```
module misr_16(
  input clk,
  input SI,
  output sign,
  output reg [15:0] MISR = 65535

);

wire feedback = MISR[15];
assign sign=MISR[15];
always @(posedge clk)
begin
  MISR[0] <= feedback;
  MISR[1] <= MISR[0];
  MISR[2] <= MISR[1] ^ feedback;
  MISR[3] <= MISR[2] ^ feedback;
  MISR[4] <= MISR[3];
  MISR[5] <= MISR[4] ^ feedback;
  MISR[6] <= MISR[5];
  MISR[7] <= MISR[6];
```

```
  MISR[8] <= MISR[7];
  MISR[9] <= MISR[8];
  MISR[10] <= MISR[9];
  MISR[11] <= MISR[10]^SI;
  MISR[12] <= MISR[11];
  MISR[13] <= MISR[12];
  MISR[14] <= MISR[13];
  MISR[15] <= MISR[14];
end
endmodule
```



Έπειτα το MISR ενσωματώθηκε στο νέο κύκλωμα TRCUTwithMISR του οποίου ο κώδικας και το σχήμα φαίνονται παρακάτω:

```
module TRCUTwithMISR(CLK,SE,sign);
        input CLK,SE;
        output sign;


        TRCUTwithLFSR myinstance(CLK,SE,out);

        misr_16 misrinstance(CLK,out,sign,out_misr);

endmodule
```



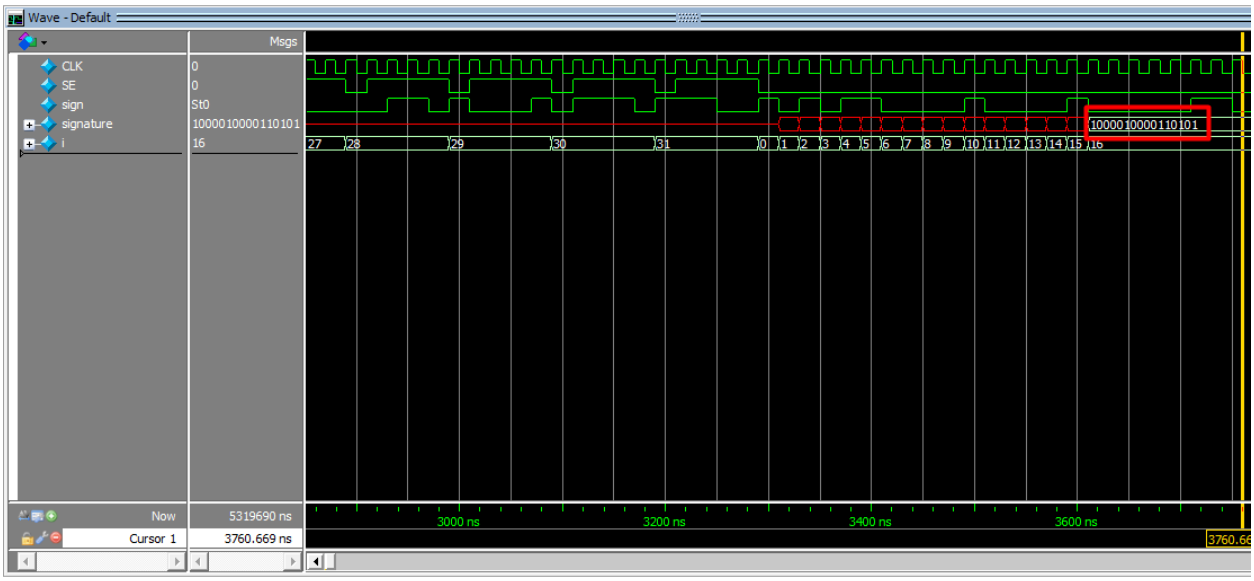
Άσκηση 2.4:

Έπειτα δημιουργήθηκε το testbench που μας ζητείται. Η λογική είναι παρόμοια με αυτήν του testbench του προηγούμενου ερωτήματος μόνο που τώρα για τους 16 κύκλους μετά την εισαγωγή των 32 διανυσμάτων λαμβάνουμε την υπογραφή των 32 διανυσμάτων που εφαρμόστηκαν. Γνωρίζοντας την υπογραφή είναι πολύ πιο εύκολο να ανιχνευτεί οποιαδήποτε αλλαγή/σφάλμα που θα προκύψει στα διανύσματα που εφαρμόζονται στο κύκλωμά μας. Το testbench είναι το ακόλουθο:

```
`timescale 1ns/1ps
module TRCUTwithMISR_tb();
        reg CLK,SE;
        wire sign;
        integer i;
        reg [15:0] signature;
        TRCUTwithMISR  myinstance(CLK,SE,sign);

        //Block for clock generation
        initial begin
                CLK=0;
                #100
                forever begin
                        #10 CLK=!CLK;
                end
        end
        initial begin
                i=0;
                SE=1;
                for(i=0;i<32;i=i+1)begin
```

```
                        repeat(4) @(posedge CLK)
                                SE=1;//scan mode

                        repeat (1)@(posedge CLK)
                                SE=0;//capture mode
                end
                for(i=0;i<16;i=i+1)@(posedge CLK)begin
                        signature[i]<=sign;
                end
        end


endmodule
```

Ακολοθούν οι κυματομορφές, στις οποίες φαίνεται η σωστή υπογραφή του κυκλώματος, η οποία είναι **1000010000110101** . Η υπογραφή παρέμεινε η ίδια όσες φορές και αν ελέγχθηκε το κύκλωμα με τις συγκεκριμένες συνθήκες. Πρέπει να σημειωθεί ότι αριχικοποιήθηκαν οι έξοδοι των DFF στο 0 για να μην εισάγονται αρχικά στο MISR τιμές σκουπίδια που προϋπήρχαν μέσα στο TRCUT.



Για να επιβεβαιωθεί ότι το κύκλωμά μας αντιδρά σε σφάλματα, έγινε fault injection μετατρέποντας την RTL του CUT κατάλληλα ώστε το σήμα h να είναι πάντα 1 όπως φαίνεται παρακάτω:

```
module comb_logic(a,b,c,d,i,j);
  input a,b,c,d;
  output i,j;

  wire e,f,g,h;

  xor(e,a,b);
```

```
  xor(f,c,d);
  xor(g,a,c);
  xor(h,b,d);

  and(i,e,f);
  or(j,g,1);


endmodule
```

Τρέχοντας ξανά το testbench του TRCUTwithMISR η υπογραφή άλλαξε και έγινε **0110101110001001** όπως φαίνεται και από τις κυματομορφές που προέκυψαν: