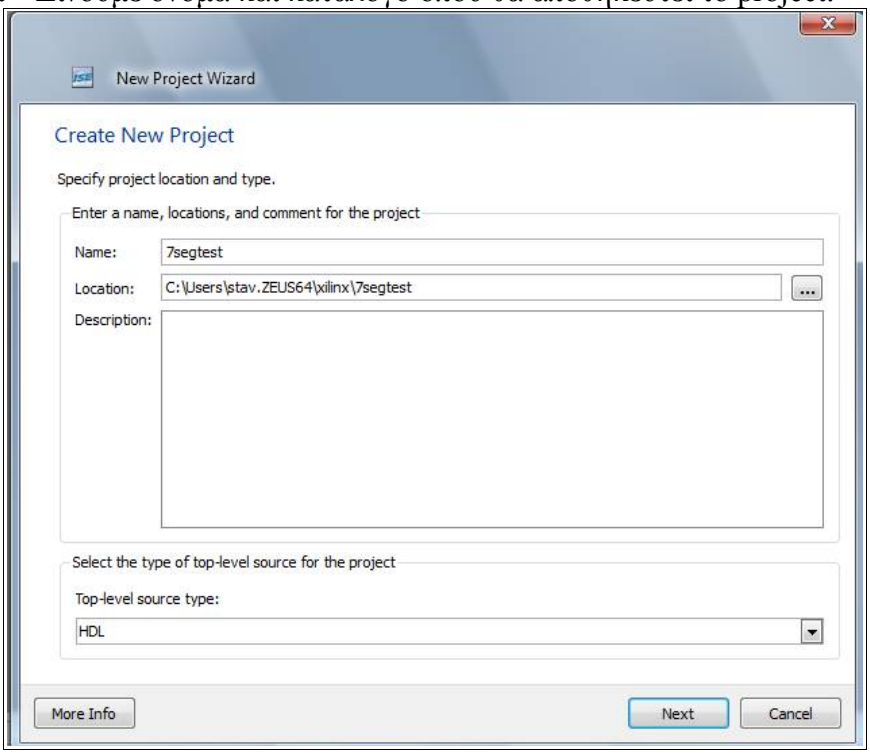
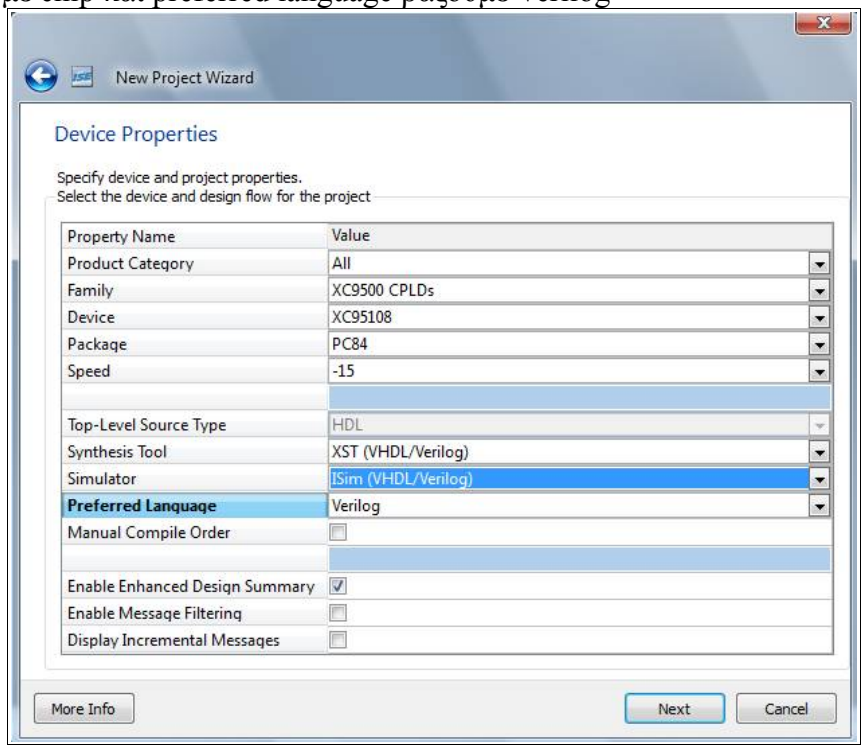
Εισαγωγή στη Verilog με το ISE

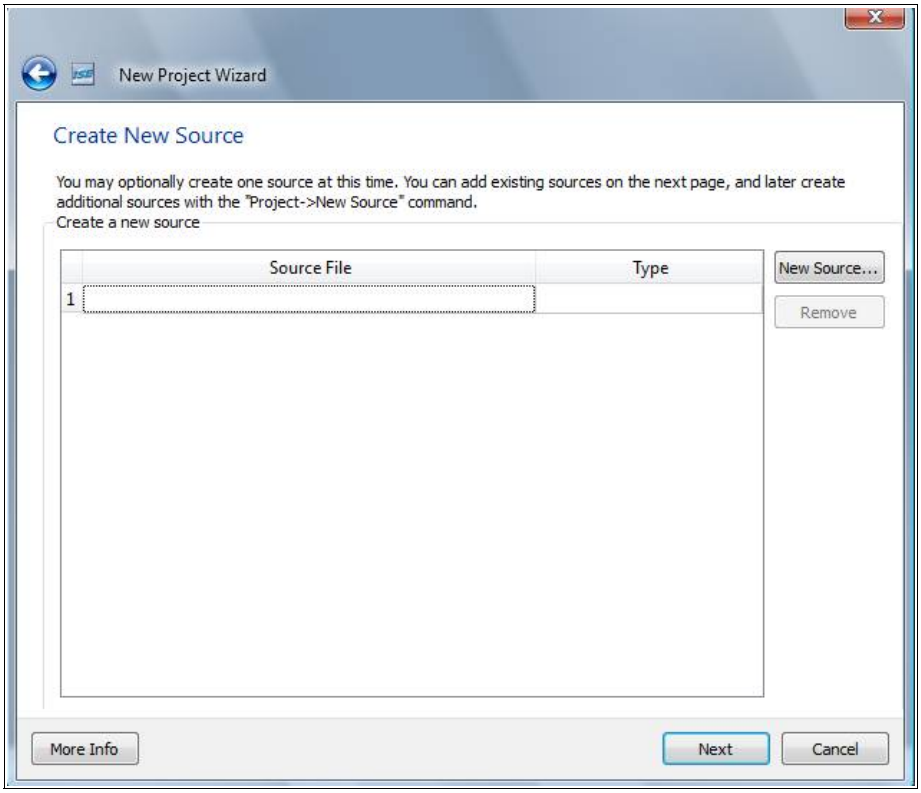
Πατάμε new project – Δίνουμε όνομα και κατάλογο όπου θα αποθηκευτεί το project.



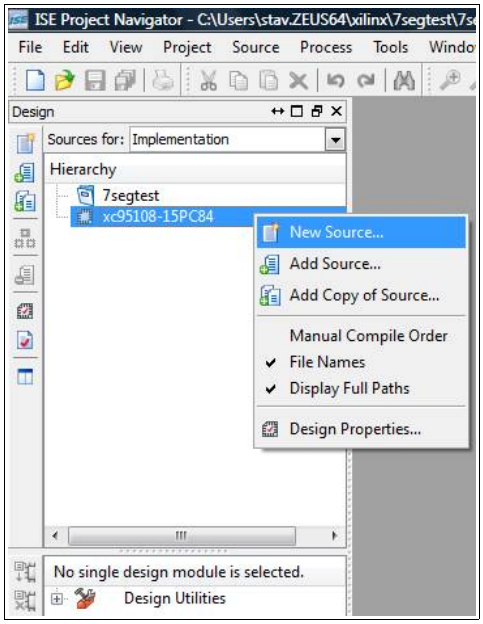
Next όπου επιλέγουμε chip και preferred language βάζουμε Verilog



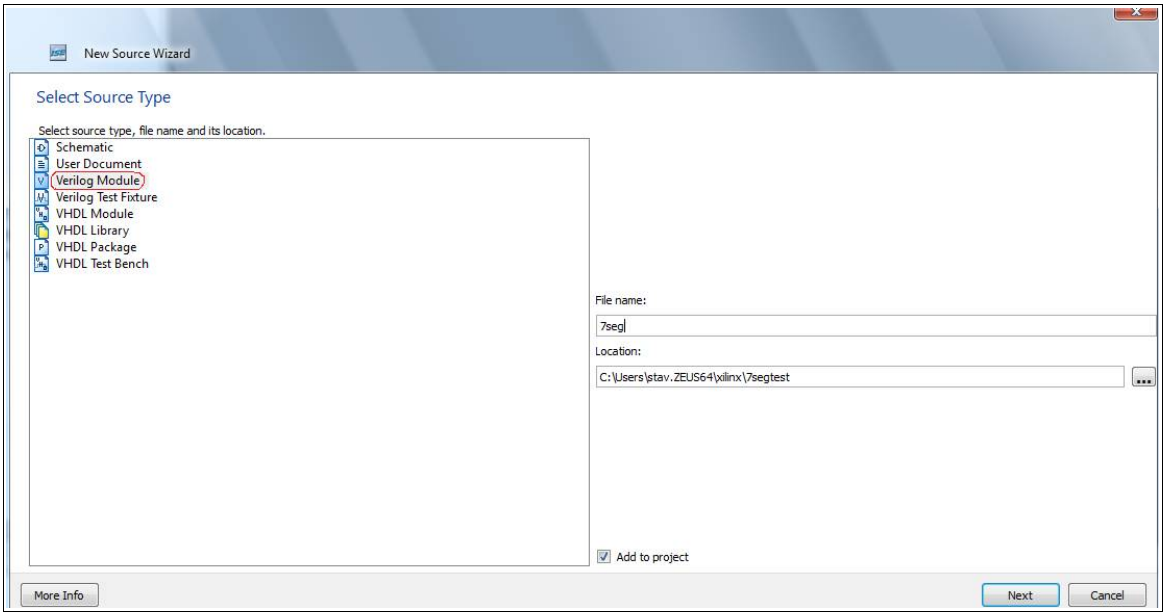
Next και στο Create new source το παραλείπουμε πατώντας και πάλι next.



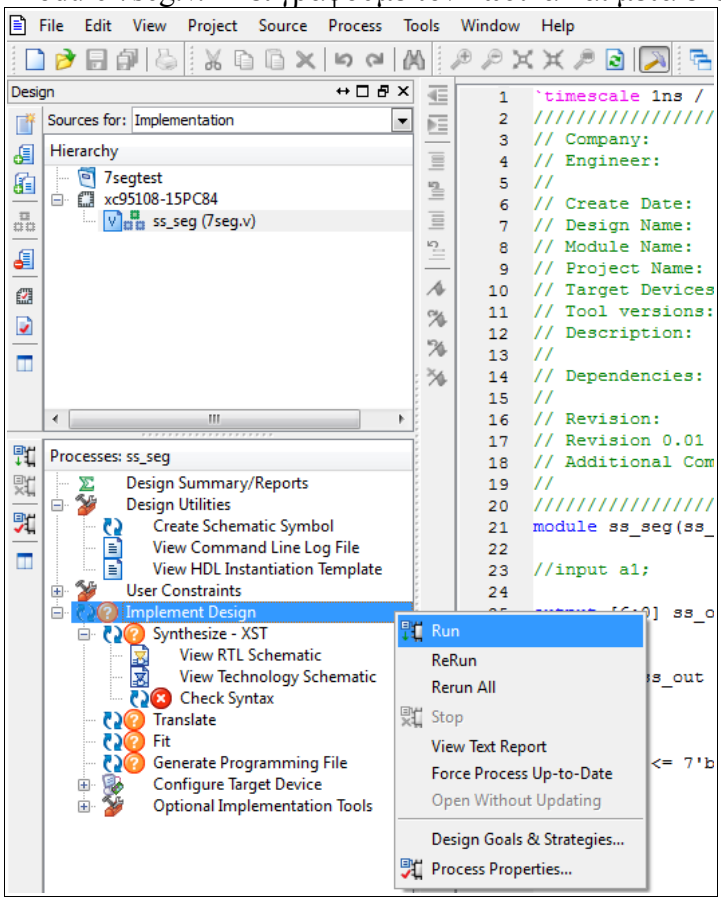
Και πάλι next και μετά finish.



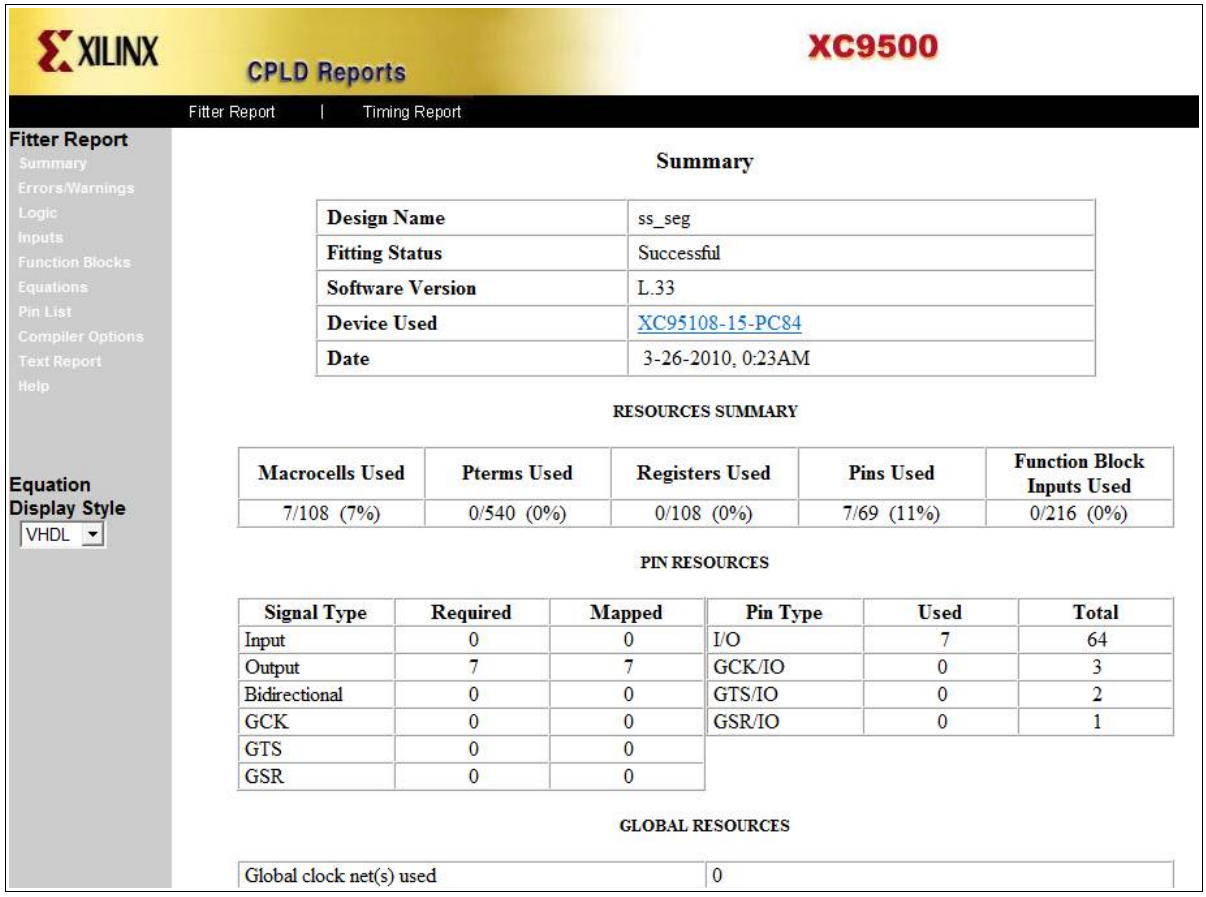
Πατάμε verilog module και δίνουμε και ένα όνομα.



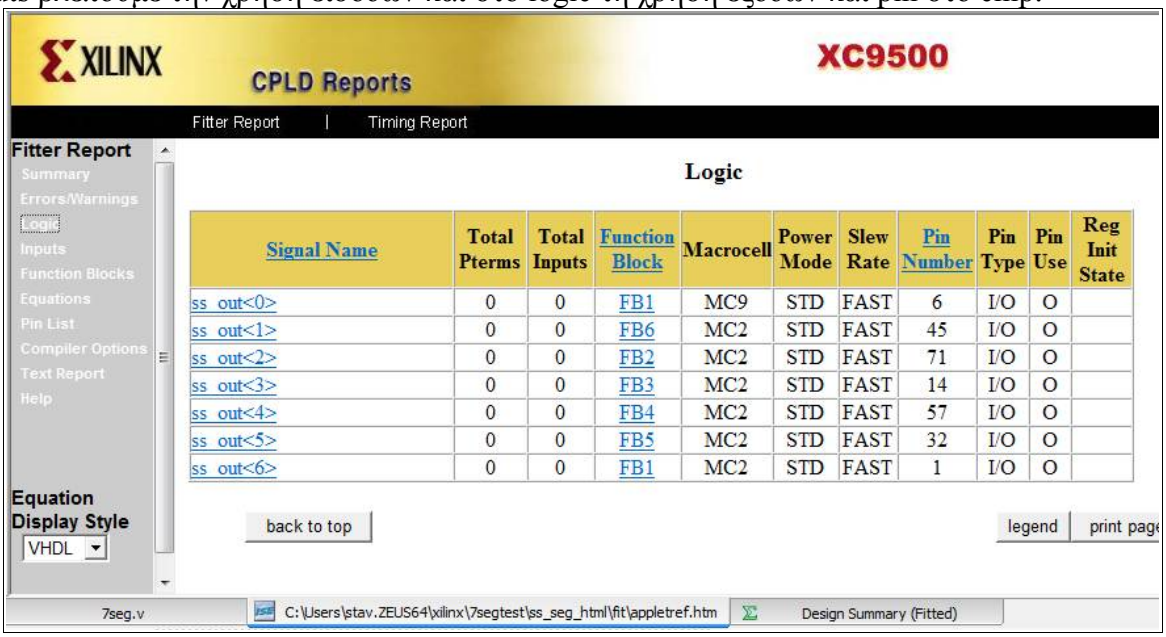
Next και στο define module πάλι δίνουμε Next και μετά finish. Τώρα στο κυρίως παράθυρο ανοίγει ο editor με τα αρχικά για το module 7seg.v. Εκεί γράφουμε τον κώδικα και μετά ελέγχουμε την ορθότητα.



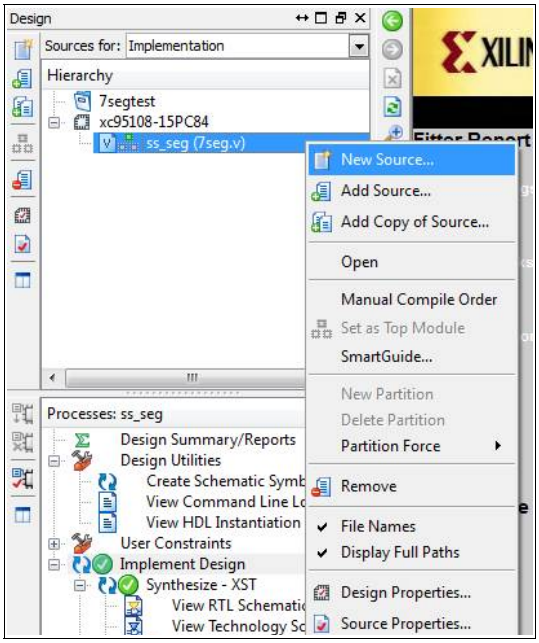
Αν όλα πάνε καλά δημιουργείται ένα tab με πληροφορίες για το project και τη χρήση πόρων.



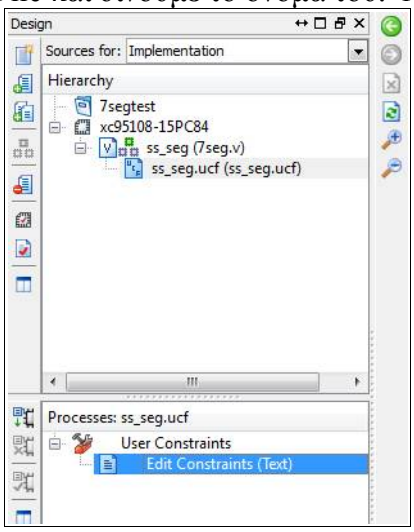
Στο Inputs βλέπουμε την χρήση εισόδων και στο logic τη χρήση εξόδων και pin στο chip.



Επειδή ορίζει με δική του πρωτοβουλία θα φτιάξουμε αρχείο ucf που θα λέμε ποια pin να συνδεθούν στο κάθε σήμα. Πάμε στο Design στο αρχείο του module και new source.



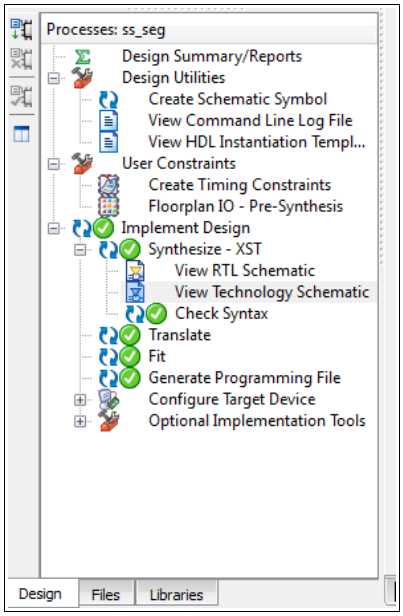
Πατάμε Implementation Constraints File και δίνουμε το όνομά του. Μετά Next και Finish.



Διπλό κλικ στο Edit Constraints (Text) και ανοίγει κενό tab όπου γράφουμε τα παρακάτω :

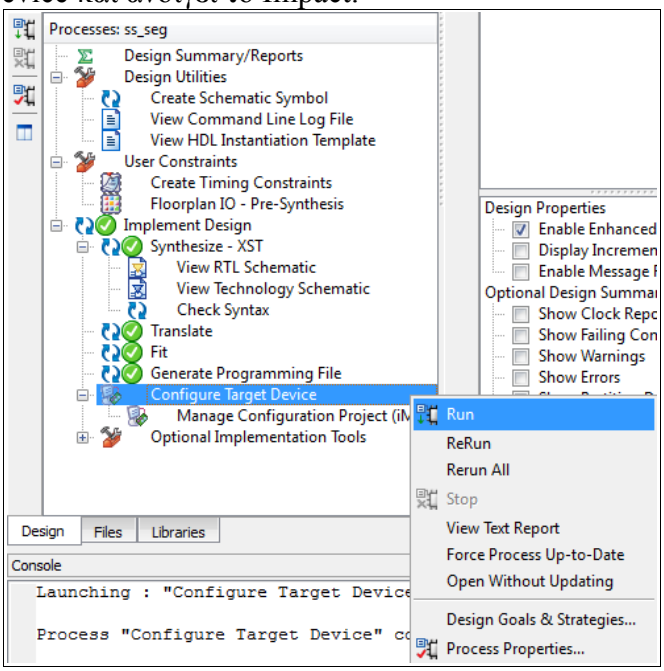
```
NET "ss_out<0>"          LOC = "P14" ;
NET "ss_out<1>"          LOC = "P15" ;
NET "ss_out<2>"          LOC = "P17" ;
NET "ss_out<3>"          LOC = "P18" ;
NET "ss_out<4>"          LOC = "P19" ;
NET "ss_out<5>"          LOC = "P20" ;
NET "ss_out<6>"          LOC = "P21" ;
```

Αφού το αποθηκεύσουμε πατάμε και πάλι Implement Design. Αν όλα πάνε καλά βλέπουμε το εξής :

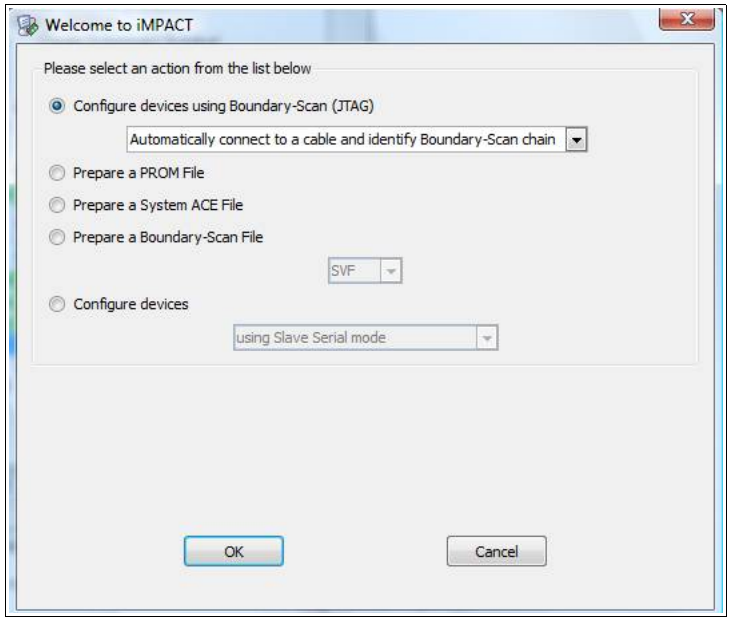


Εγγραφή στο chip

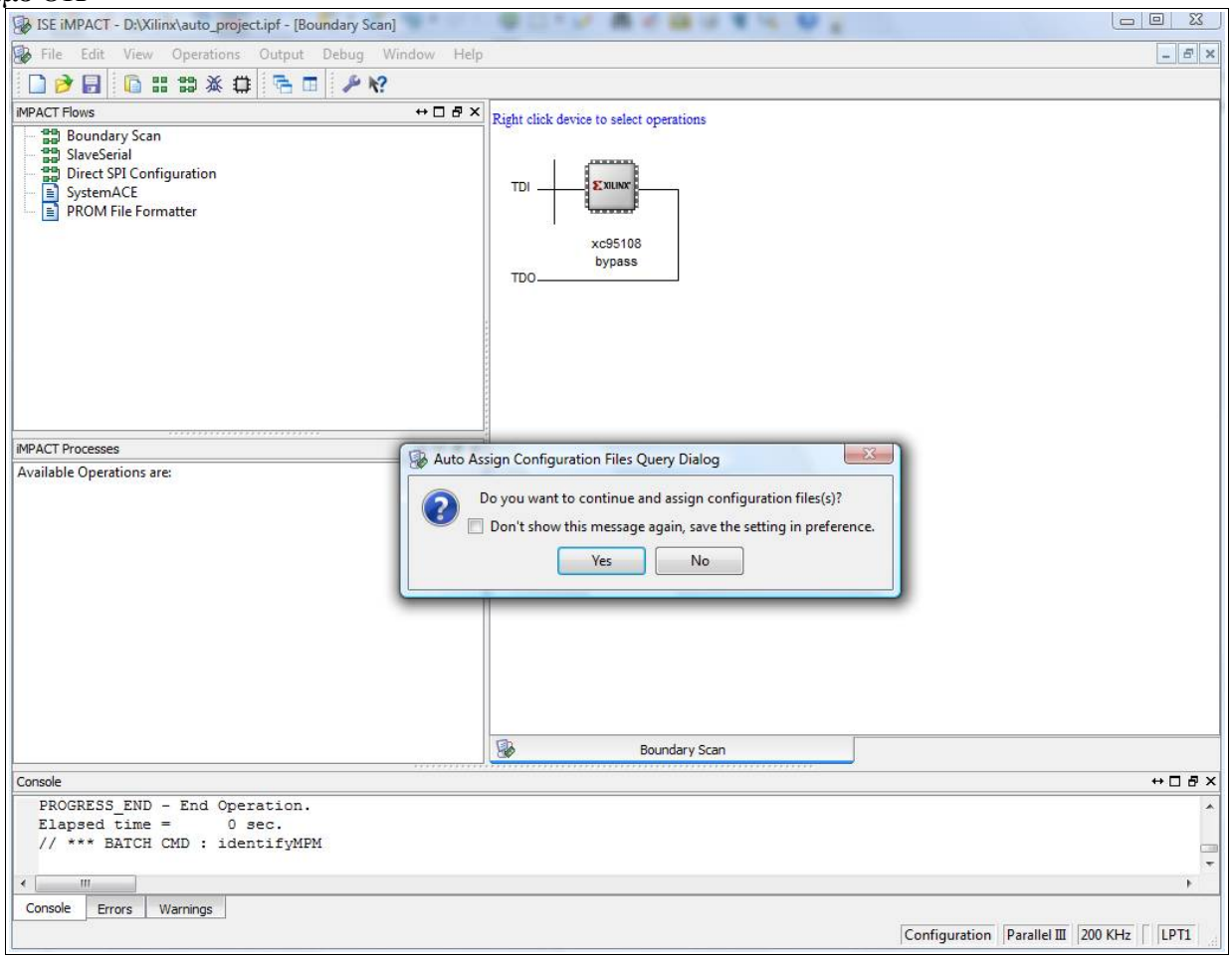
Πατάμε Configure Target Device και ανοίγει το Impact.



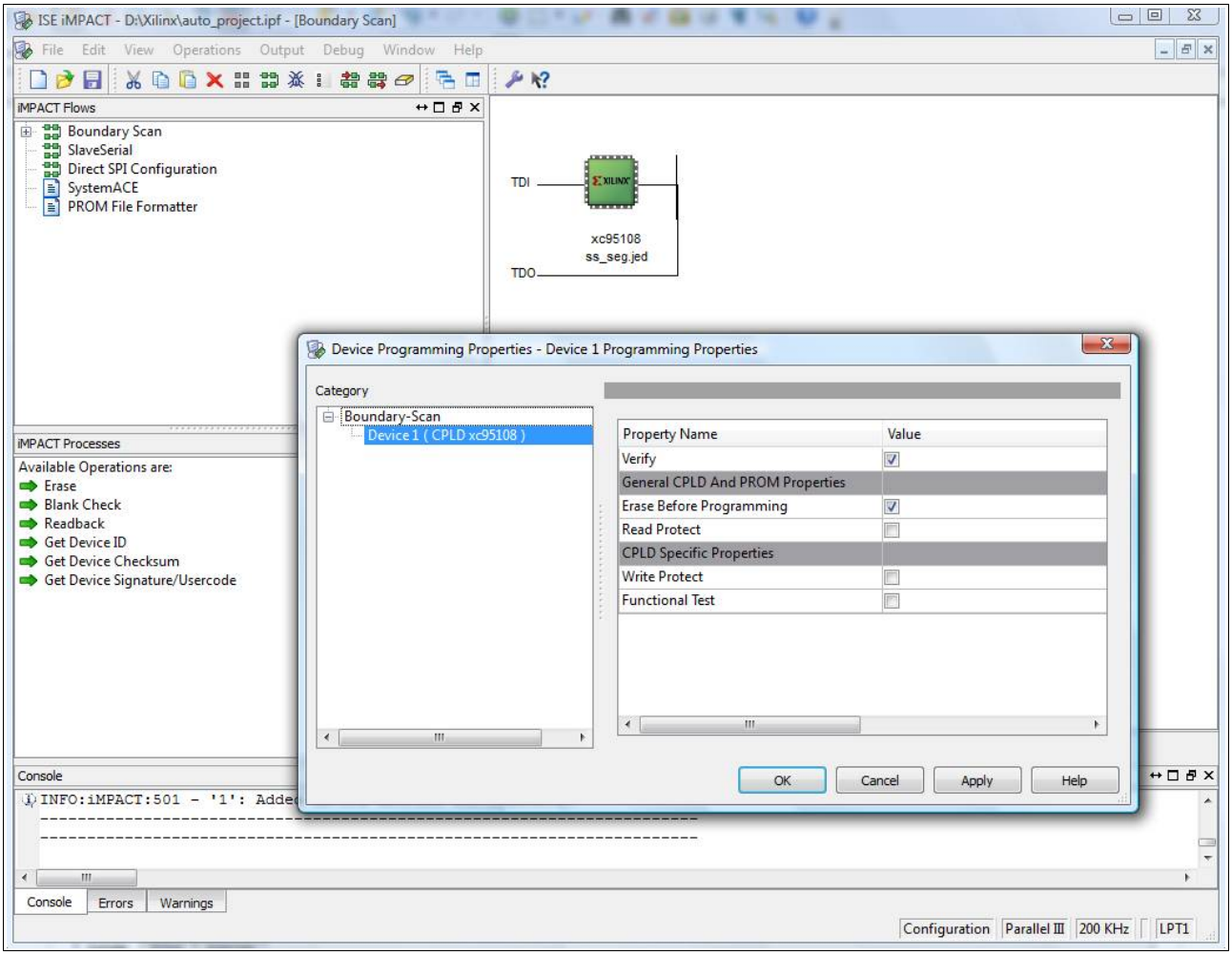
File – New Project. Στην ερώτηση Yes



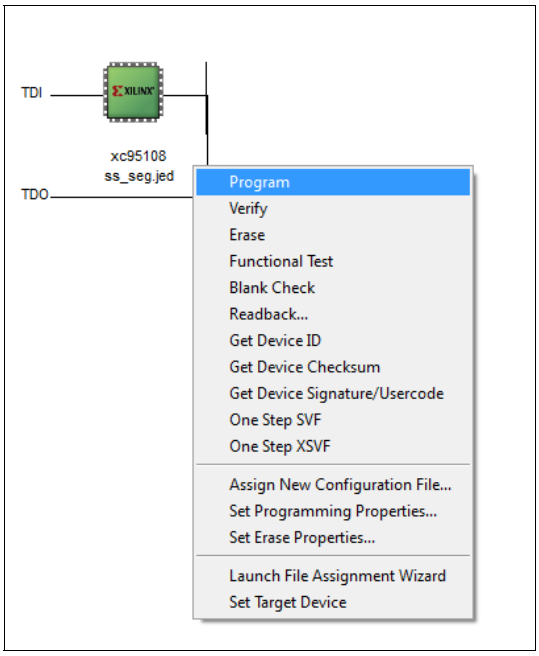
Πατάμε OK



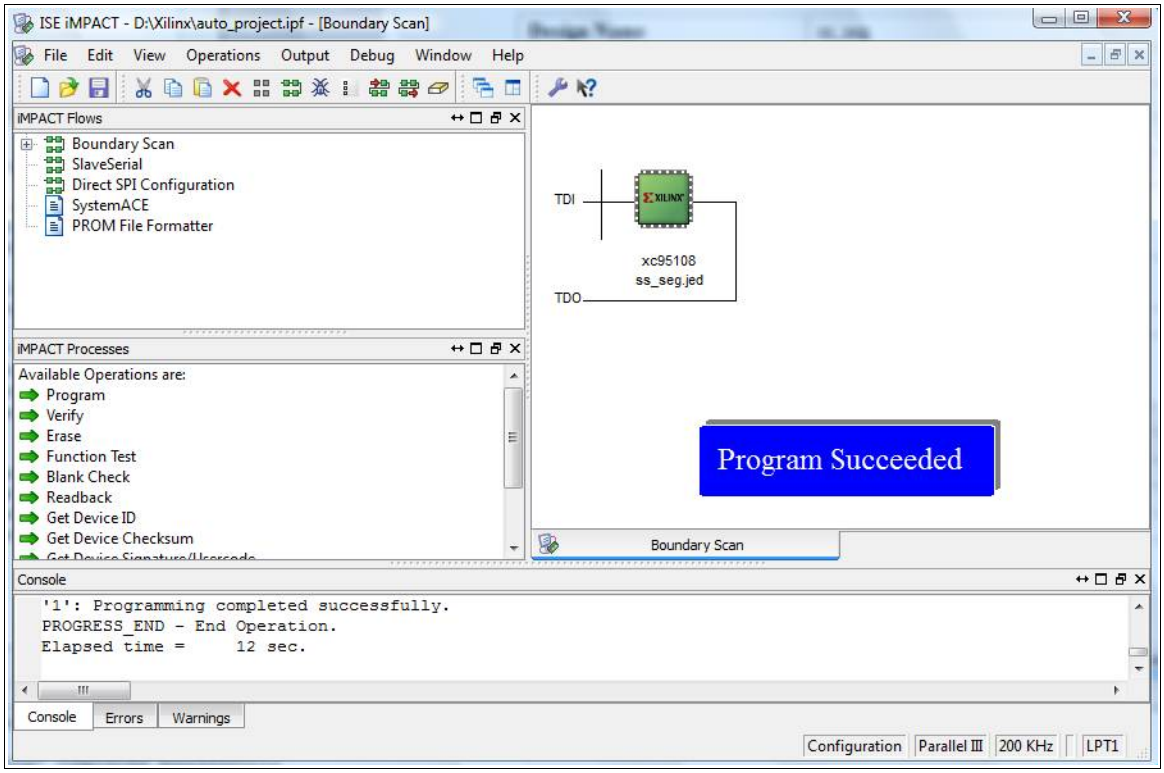
Πατάμε Yes και ψάχνουμε για το αρχείο .jed που βρίσκεται στον φάκελο του project.



Πατάμε OK.



Μετά δεξί click πάνω στο chip και Program. Αν όλα πάνε καλά θα βγάλει :



Μερικά βασικά για την verilog

Για συνδυαστικά κυκλώματα μπορώ να χρησιμοποιώ δομές assign ή always. Για ακολουθιακά μόνο always. Καλό να αποφεύγουμε τις always αν αυτό υλοποιείται και με assign. Ο παρακάτω κώδικας εμφανίζει το γράμμα a ή b σε ένα 7 segment display ανάλογα με το bit a1. Το bit a2 απλώς ανάβει ή όχι το decimal point.

```
module ss_seg(a1, a2, ss_out, ss_dp);

input a1, a2;

output [6:0] ss_out;
output ss_dp;

wire a1, a2, ss_dp;
reg [6:0] ss_out;
assign ss_dp = a2;

always @ (a1)
begin
    //ss_out = 0;
    if (a1 == 1'b0)
        ss_out = 7'b1111101; //a char
    else
         ss_out = 7'b1011110; //b char
end

endmodule
```

Ακολουθεί το ίδιο κύκλωμα φτιαγμένο μόνο με εντολές assign.

```
module ss_seg(a1, a2, ss_out, ss_dp);

input a1, a2;

output [6:0] ss_out;
output ss_dp;

wire a1, a2, ss_dp;

assign ss_dp = a2;
assign ss_out = (a1 == 1'b1) ? 7'b1111101 : 7'b1011110;
```

```
endmodule
```

Ακολουθεί κώδικας για binary to 7 segment decoder. Με τα dip switches 1-4 δίνω είσοδο στο δυαδικό και βλέπω στο display τα ψηφία 0-9. Αν η είσοδος είναι > 9 τότε εμφανίζει το μείον (-).
module ss_seg(inp, dp, ss_out, ss_dp);

```
input [3:0] inp;
input dp;

output [6:0] ss_out;
output ss_dp;

wire [3:0] inp;
wire dp, ss_dp;
reg [6:0] ss_out;

assign ss_dp = dp;

always @ (inp)
begin
    ss_out = 0;
    case (inp)
                4'b0000: ss_out = 7'b0111111; //0
                4'b0001: ss_out = 7'b0110000; //1
                4'b0010: ss_out = 7'b1101101; //2
                4'b0011: ss_out = 7'b1111001; //3
                4'b0100: ss_out = 7'b1110010; //4
                4'b0101: ss_out = 7'b1011011; //5
                4'b0110: ss_out = 7'b1011111; //6
                4'b0111: ss_out = 7'b0110001; //7
                4'b1000: ss_out = 7'b1111111; //8
                4'b1001: ss_out = 7'b1111011; //9
                default: ss_out = 7'b1000000; //-
    endcase
end

endmodule
```

Και το αρχείο ucf.

```
NET "ss_out<0>"                         LOC = "P14";
NET "ss_out<1>"                         LOC = "P15";
NET "ss_out<2>"                         LOC = "P17";
NET "ss_out<3>"                         LOC = "P18";
NET "ss_out<4>"                         LOC = "P19";
NET "ss_out<5>"                         LOC = "P20";
NET "ss_out<6>"                         LOC = "P21";
NET "ss_dp"                             LOC = "P23";
NET "inp<0>"                            LOC = "P1";
NET "inp<1>"                            LOC = "P2";
NET "inp<2>"                            LOC = "P3";
NET "inp<3>"                            LOC = "P4";
NET "dp"                                LOC = "P5";
```

Αν θέλω δεκαδικό ή δεκαεξαδικό σύστημα το τροποποιώ ως εξής:

```
       case (inp)
              4'd0: ss_out = 7'h3f;      //0
              4'd1: ss_out = 7'b0110000; //1
              4'd2: ss_out = 7'b1101101; //2
              4'd3: ss_out = 7'b1111001; //3
              4'd4: ss_out = 7'b1110010; //4
              4'd5: ss_out = 7'b1011011; //5
              4'd6: ss_out = 7'b1011111; //6
              4'd7: ss_out = 7'b0110001; //7
              4'd8: ss_out = 7'b1111111; //8
              4'd9: ss_out = 7'b1111011; //9
              default: ss_out = 7'b1000000; //-
```

Το ίδιο αλλά μόνο με assign:

```
module ss_seg(inp, dp, ss_out, ss_dp);

input [3:0] inp;
input dp;
```

```
output [6:0] ss_out;
output ss_dp;

wire [3:0] inp;
wire dp, ss_dp;
wire [6:0] ss_out;

assign ss_dp = dp;
assign ss_out = (inp == 4'd0) ? 7'b0111111 :  //0
                (inp == 4'd1) ? 7'b0110000 :  //1
                (inp == 4'd2) ? 7'b1101101 :  //2
                (inp == 4'd3) ? 7'b1111001 :  //3
                (inp == 4'd4) ? 7'b1110010 :  //4
                (inp == 4'd5) ? 7'b1011011 :  //5
                (inp == 4'd6) ? 7'b1011111 :  //6
                (inp == 4'd7) ? 7'b0110001 :  //7
                (inp == 4'd8) ? 7'b1111111 :  //8
                (inp == 4'd9) ? 7'b1111011 : 7'b1000000; //9 : default

endmodule
```

Διαιρέτης / απαριθμητής

Δέχεται στην είσοδο ρολόι συχνότητας 244,140625 Hz από το HC4060 και βγάζει στην έξοδο παλμό 0,953 Hz. **Προσοχή** το = μέσα σε δομή always είναι blocking εντολή, πράγμα που σημαίνει ότι οι εντολές εκτελούνται με τη σειρά. Το <= είναι no blocking που αφήνει να εκτελεστούν παράλληλα.

```
module counter(clk, tc);
input  clk;
output tc;

wire clk;
reg [7:0] cnt;
reg tc;

always @ (posedge clk)
begin
  tc = 0;
      if (cnt == 8'hff)
         begin
           tc = 1'b1;
           cnt = 8'h00; //ή cnt <= 8'h00;
         end
      else
         cnt = cnt + 1; //ή cnt <= cnt + 1;
end

endmodule

================================== ucf ================================
NET "clk"                          LOC = "P9";
NET "tc"                           LOC = "P71";
```

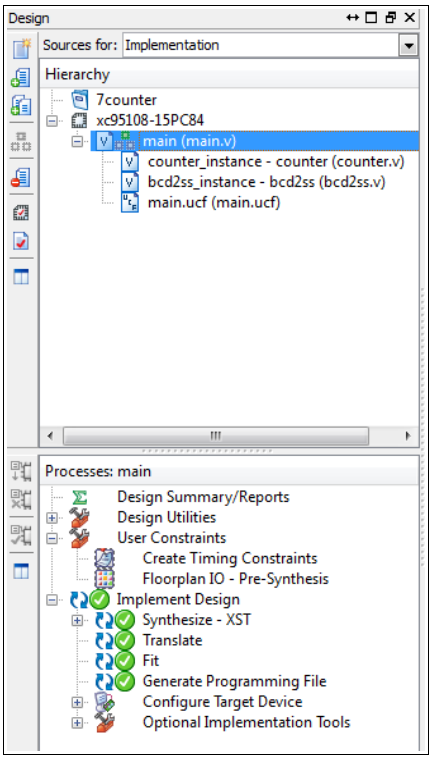
Μετρητής 12 bit που μόνο τα 4 σημαντικά ψηφία συνδέονται σε pin εξόδου
module counter(clk, out);

```
input  clk;
output out;

wire clk;
reg [11:0] cnt;
reg [3:0] out;

always @ (posedge clk)
begin
  out = 0;
       if (cnt == 12'h9ff)
         cnt = 12'h000;
       else
         cnt = cnt + 1;
  out = {cnt[11],cnt[10],cnt[9],cnt[8]};
end

endmodule

====================================== ucf ============================
```

```
NET "clk"                       LOC = "P9";
NET "out<0>"                    LOC = "P14";
NET "out<1>"                    LOC = "P15";
NET "out<2>"                    LOC = "P17";
NET "out<3>"                    LOC = "P18";
```

Δουλεύοντας με submodules

Καλό είναι να σπάμε το project σε κομμάτια τα οποία επικοινωνούν μεταξύ τους. Θα δούμε πως θα βάλουμε τον προηγούμενο counter να οδηγεί το bcd to 7 segment και να μετράει από 0 έως 9. Στο design βλέπουμε το συνολικό module main και τα δύο submodules.



Και ο κώδικας είναι ο εξής :

Submodule counter

```
module counter(clk, out);
input  clk;
output out;

wire clk;
reg [11:0] cnt;
reg [3:0] out;

always @ (posedge clk)
begin
  out = 0;
       if (cnt == 12'h9ff)
         cnt = 12'h000;
       else
         cnt = cnt + 1;
  out = {cnt[11],cnt[10],cnt[9],cnt[8]};
end

endmodule
```

Submodule bcd2ss

```
module bcd2ss(inp, ss_out);

input [3:0] inp;
output [6:0] ss_out;
wire [3:0] inp;
wire [6:0] ss_out;

assign ss_out = (inp == 4'd0) ? 7'b0111111 :  //0
                (inp == 4'd1) ? 7'b0110000 :  //1
                (inp == 4'd2) ? 7'b1101101 :  //2
```

```
                (inp == 4'd3) ? 7'b1111001 :  //3
                (inp == 4'd4) ? 7'b1110010 :  //4
                (inp == 4'd5) ? 7'b1011011 :  //5
                (inp == 4'd6) ? 7'b1011111 :  //6
                (inp == 4'd7) ? 7'b0110001 :  //7
                (inp == 4'd8) ? 7'b1111111 :  //8
                (inp == 4'd9) ? 7'b1111011 : 7'b1000000; //9 : default

endmodule
```

Κύριο module

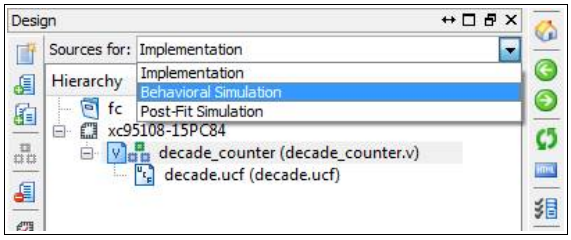
```
module main(clk, ss_out);
input clk;
output [6:0] ss_out;

wire [3:0] out;

counter counter_instance (clk, out);
bcd2ss bcd2ss_instance (out, ss_out);

endmodule
```

Αρχείο ucf

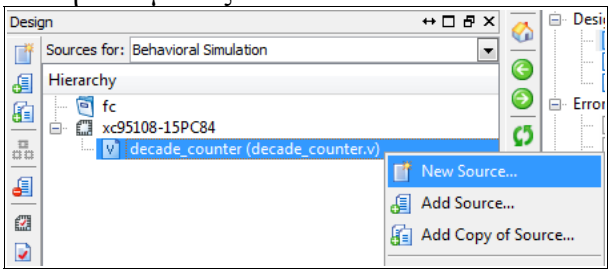
```
NET "clk"                       LOC = "P9";
NET "ss_out<0>"                 LOC = "P14";
NET "ss_out<1>"                 LOC = "P15";
NET "ss_out<2>"                 LOC = "P17";
NET "ss_out<3>"                 LOC = "P18";
NET "ss_out<4>"                 LOC = "P19";
NET "ss_out<5>"                 LOC = "P20";
NET "ss_out<6>"                 LOC = "P21";
```

Simulation

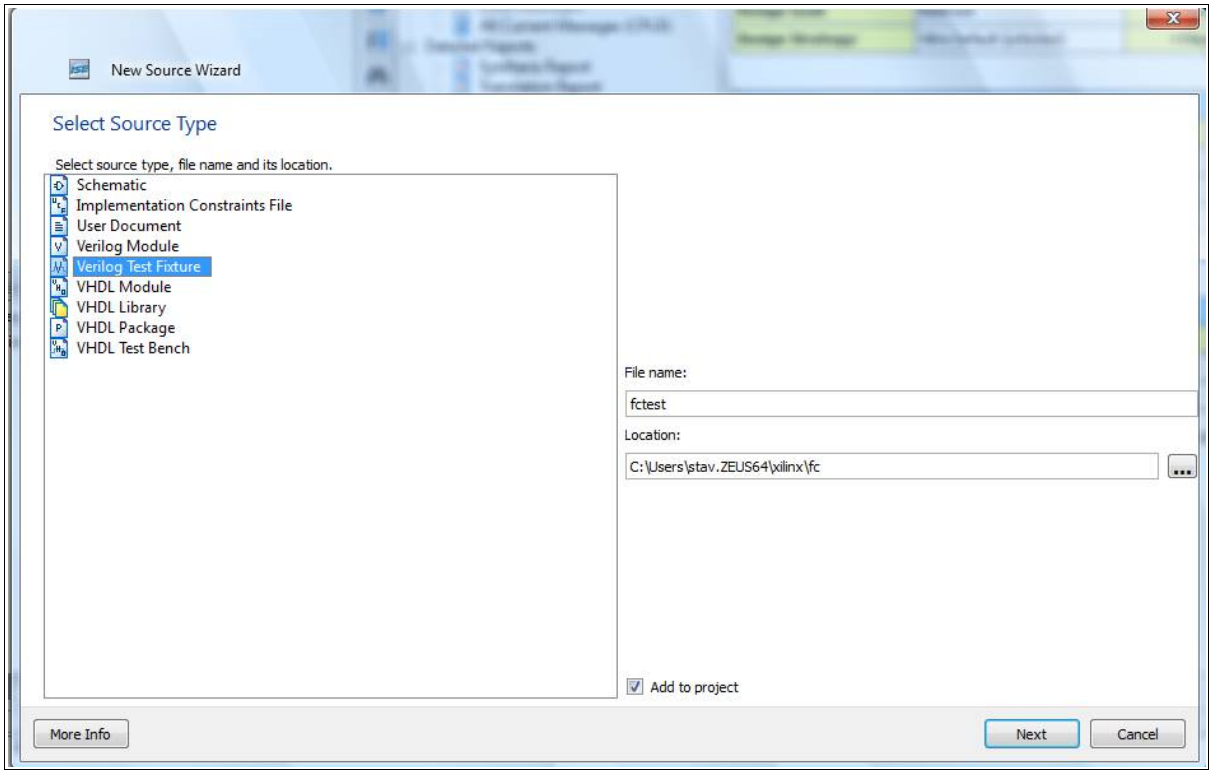
Στο Design πατάμε το listbox sources for και επιλέγουμε Behavioral Simulation. Πριν που γράφαμε τον κώδικα ήταν Implementation.



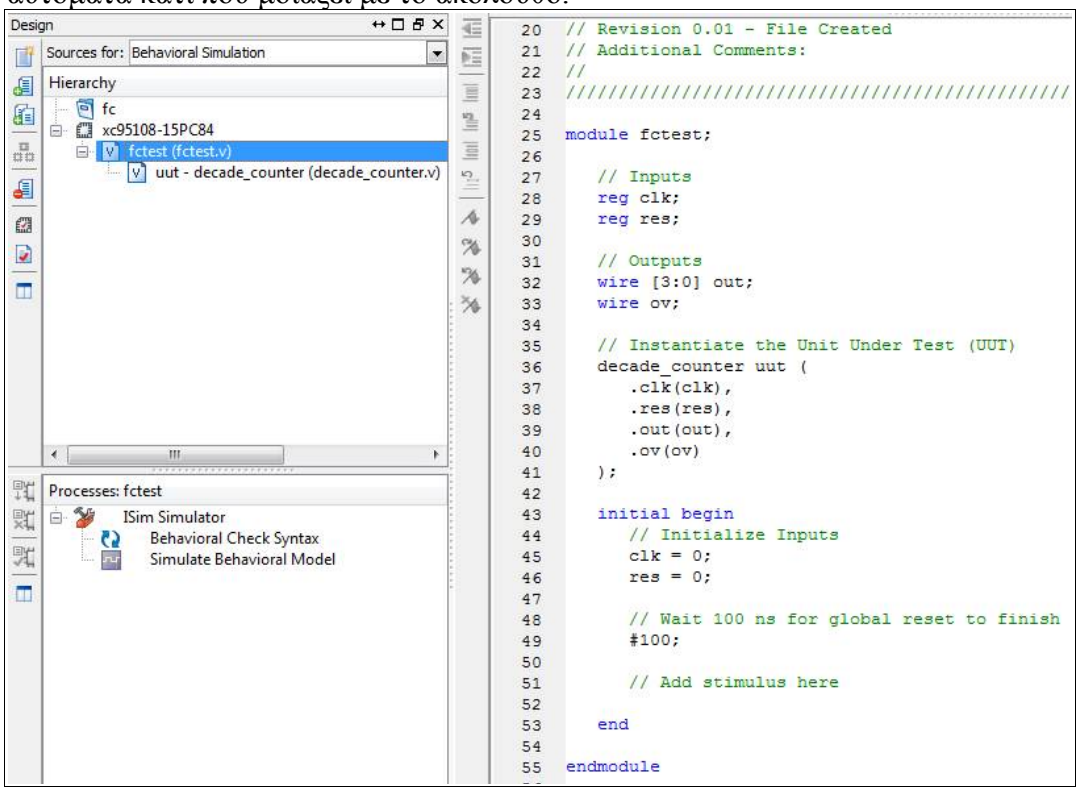
Στο module που θα γίνει εξομοίωση πατάμε δεξί κλικ και New Source.



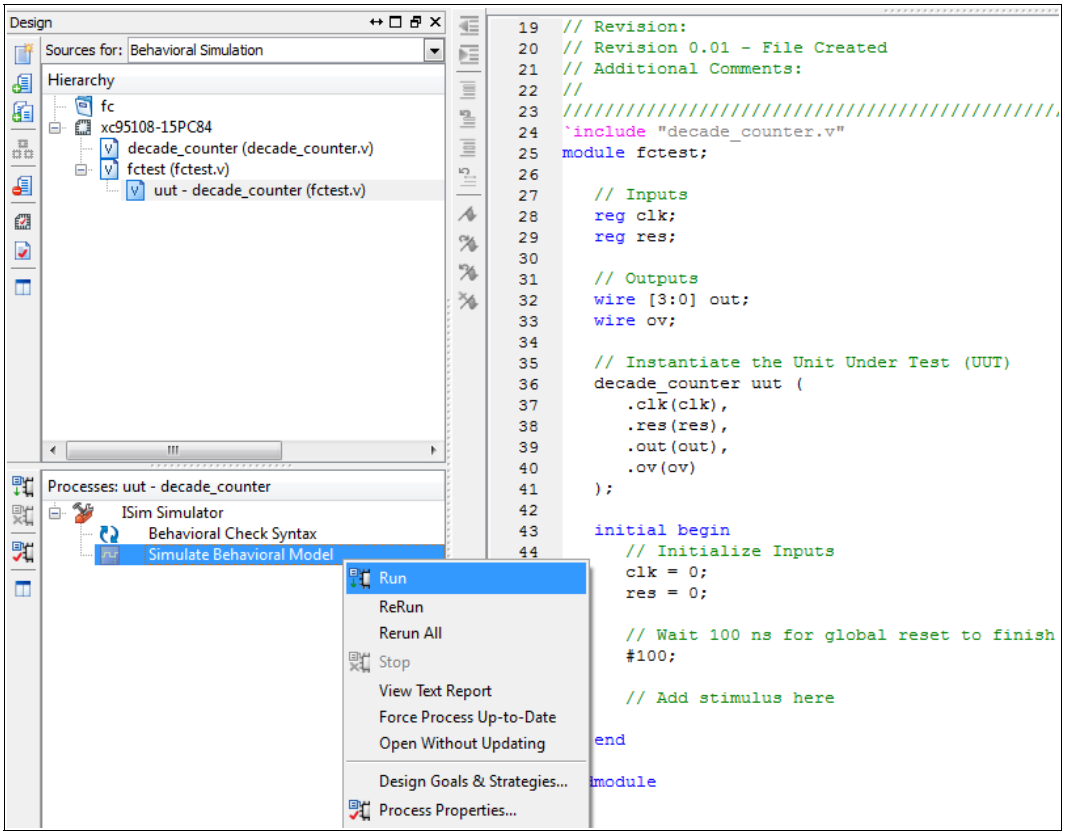
Από το παράθυρο που ανοίγει επιλέγουμε Verilog Test Fixture και δίνουμε ένα όνομα και Next. Μετά ανοίγει ένα νέο παράθυρο που προτείνει Associate Source και έχει επιλεγμένο το decade_counter. Πατάμε Next και στο επόμενο finish.



Δημιουργεί αυτόματα κάτι που μοιάζει με το ακόλουθο.



Αν τρέξουμε το Simulate Behavioral Model βγάζει διάφορα λάθη. Γι' αυτό πριν το module fctest; βάζουμε **`include "decade_counter.v"** και τώρα γίνεται όπως παρακάτω.

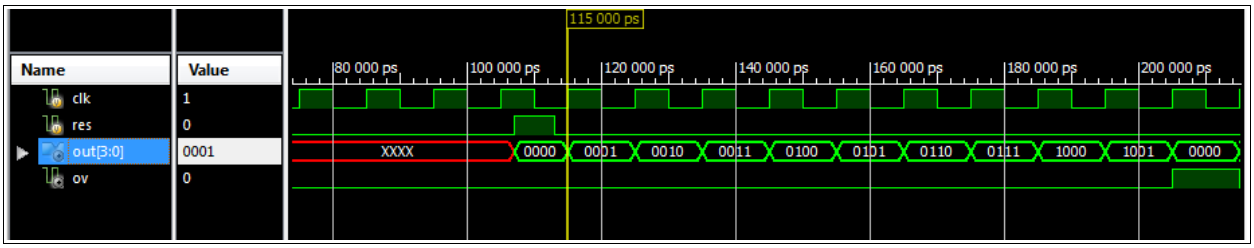



```
Πριν το τέλος του module προσθέτουμε :
// Clock generator
always begin
  #5  clk = ~clk; // Toggle clock every 5 ticks
end
```

Για να δουλέψει σωστά το simulation χωρίς λάθη, στο hierarchy επιλέγω το fctest (fctest.v) και μετά κάτω στο Processes τρέχω το Simulate Behavioral Model.
Μετά προσθέτω λίγο κώδικα στο initial begin για να δώσω αρχικές τιμές και να κάνω reset. Ο τελικός κώδικας είναι ο παρακάτω :

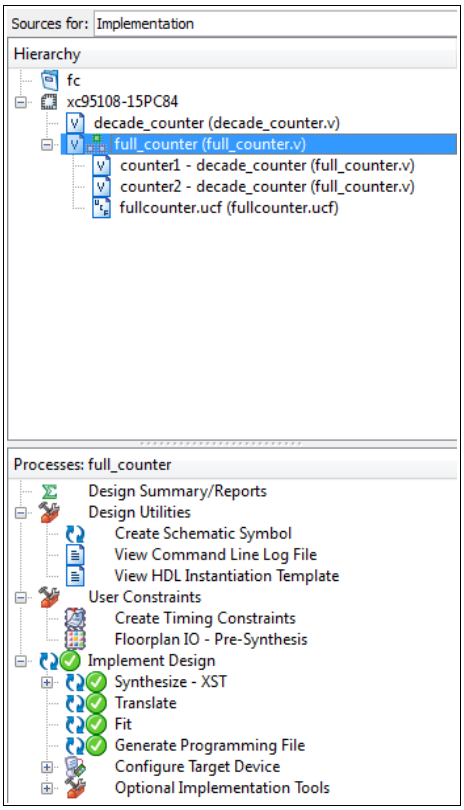
```
`include "decade_counter.v"
module fctest;

	// Inputs
	reg clk;
	reg res;

	// Outputs
	wire [3:0] out;
	wire ov;

	// Instantiate the Unit Under Test (UUT)
	decade_counter uut (
		.clk(clk),
		.res(res),
		.out(out),
		.ov(ov)
	);

	initial begin
		// Initialize Inputs
		clk = 0;
		res = 0;

		// Wait 100 ns for global reset to finish
		#100;

		// Add stimulus here
		#7  res = 1;    // Assert the reset
		#6  res = 0;   // De-assert the reset
```

```
        end

// Clock generator
always begin
  #5  clk = ~clk; // Toggle clock every 5 ticks
end

endmodule
```



Εδώ χρησιμοποίησα το ISIM που τρέχει μόνο στην 32 bit έκδοση. Αν έχω και το ModelSim το δηλώνω στο design properties και χρησιμοποιώ αυτό αντί του ISIM.

Χρήση module



Το κύριο module

```
`include "decade_counter.v"
module full_counter(mclk, mrst, out_cnt, mov);
input mclk, mrst;
output [7:0] out_cnt;
output mov;

//wire [3:0] out1;
//wire [3:0] out2;
wire ov1;

decade_counter counter1 (mclk, mrst, out_cnt[3:0], ov1);
decade_counter counter2 (ov1, mrst, out_cnt[7:4], mov);
//assign out_cnt = {out2, out1};
endmodule
```

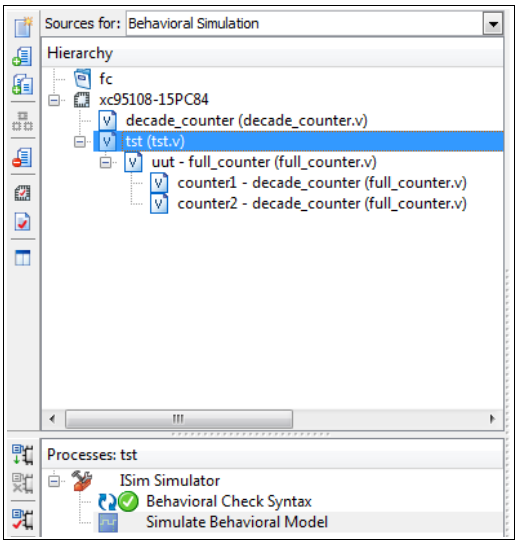
To submodule

```
module decade_counter(clk, res, out, ov);
input  clk, res;
output [3:0] out;
output ov;

wire clk, res;
reg [3:0] out;
reg [3:0] cnt;
reg ov;

always @ ( posedge clk or posedge res)
begin
  out = 0;
  ov = 0;
  if (res)
    begin
           out = 4'b0000;
               ov = 1'b0;
               cnt = 4'b0000;
        end
  else
    begin
      cnt = cnt + 1;
               ov = 1'b0;
               if (cnt == 4'd10)
       begin
         cnt = 4'd0;
              ov = 1'b1;
        end
      out = cnt;
    end
end
endmodule
```

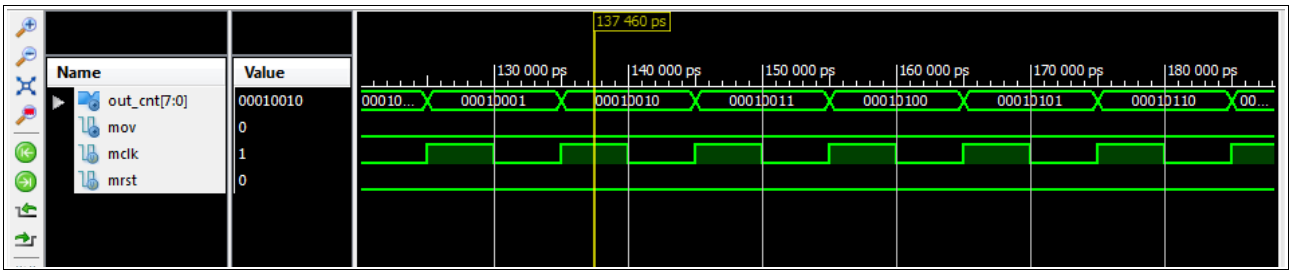
Το ucf ανάβει τα τμήματα ενός 7 segment ως εξής : μονάδες 1-a, 2-b, 4-g, 8-f και δεκάδες 1-c, 2-d, 4-e, 8-decimal point και mov – ένα led.

```
NET "mclk"                      LOC = "P10";
NET "mrst"                      LOC = "P74";
NET "out_cnt<0>"                LOC = "P14";
NET "out_cnt<1>"                LOC = "P15";
NET "out_cnt<2>"                LOC = "P21";
NET "out_cnt<3>"                LOC = "P20";
NET "out_cnt<4>"                LOC = "P17";
NET "out_cnt<5>"                LOC = "P18";
NET "out_cnt<6>"                LOC = "P19";
NET "out_cnt<7>"                LOC = "P23";
NET "mov"                       LOC = "P71";
```

και για simulation

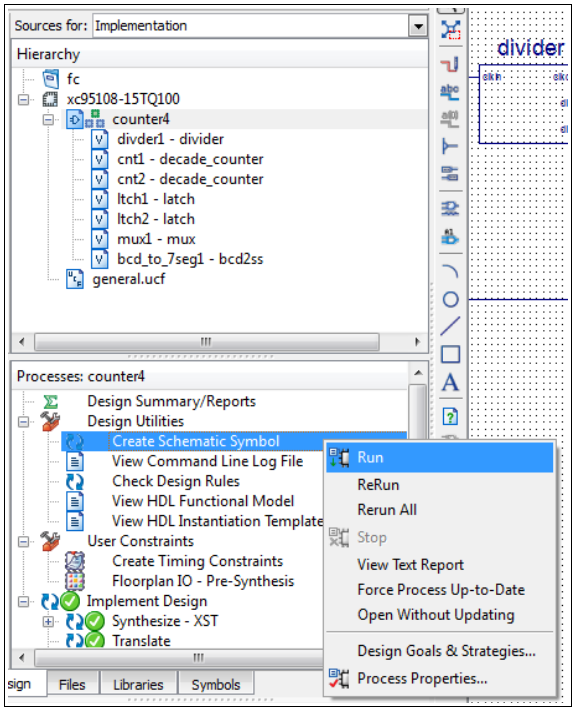


```
module tst;

	// Inputs
	reg mclk;
	reg mrst;

	// Outputs
	wire [7:0] out_cnt;
	wire mov;

	// Instantiate the Unit Under Test (UUT)
	full_counter uut (
		.mclk(mclk),
		.mrst(mrst),
		.out_cnt(out_cnt),
		.mov(mov)
	);

	initial begin
		// Initialize Inputs
		mclk = 0;
		mrst = 0;

		// Wait 100 ns for global reset to finish
		#10;


		// Add stimulus here
		#7 mrst = 1;    // Assert the reset
      #6 mrst = 0;   // De-assert the reset
	  end

	  always begin
   #5  mclk = ~mclk; // Toggle clock every 5 ticks
end

endmodule
```

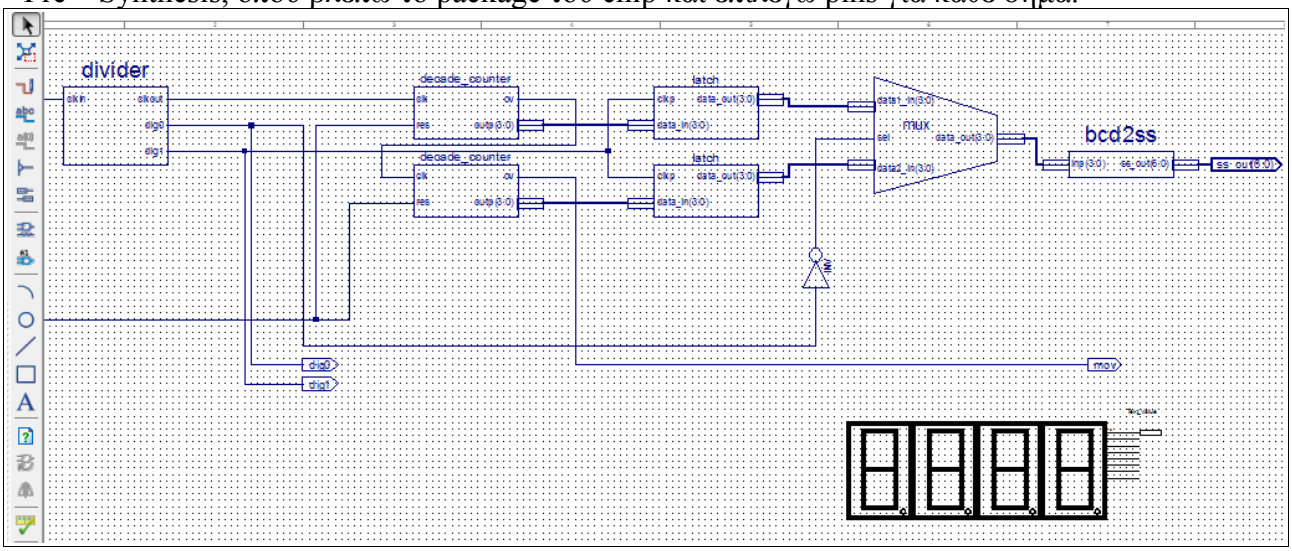


Σύνθεση με χρήση σχημάτων

Ένας πολύ ωραίος τρόπος για σύνθεση με χρήση πολλών modules είναι τα σχέδια. Αρχικά εισάγω τον κάθε module ξεχωριστά σε πηγαίο κώδικα verilog. Μετά το κάνω σχήμα από Processes – Design Utilities – Create Schematic Symbol.



Μετά την δημιουργία όλων των σχημάτων, στο design properties – Top level source type βάζω schematic. Δημιουργώ νέο source – schematic όπου εισάγω και συνδέω τα modules. Το κάνω top module και τελικά δημιουργώ το ucf. Αν θέλω το ucf να γίνει με γραφικό τρόπο πάω Processes – User constraints – Floorplan IO – Pre – Synthesis, όπου βλέπω το package του chip και επιλέγω pins για κάθε σήμα.



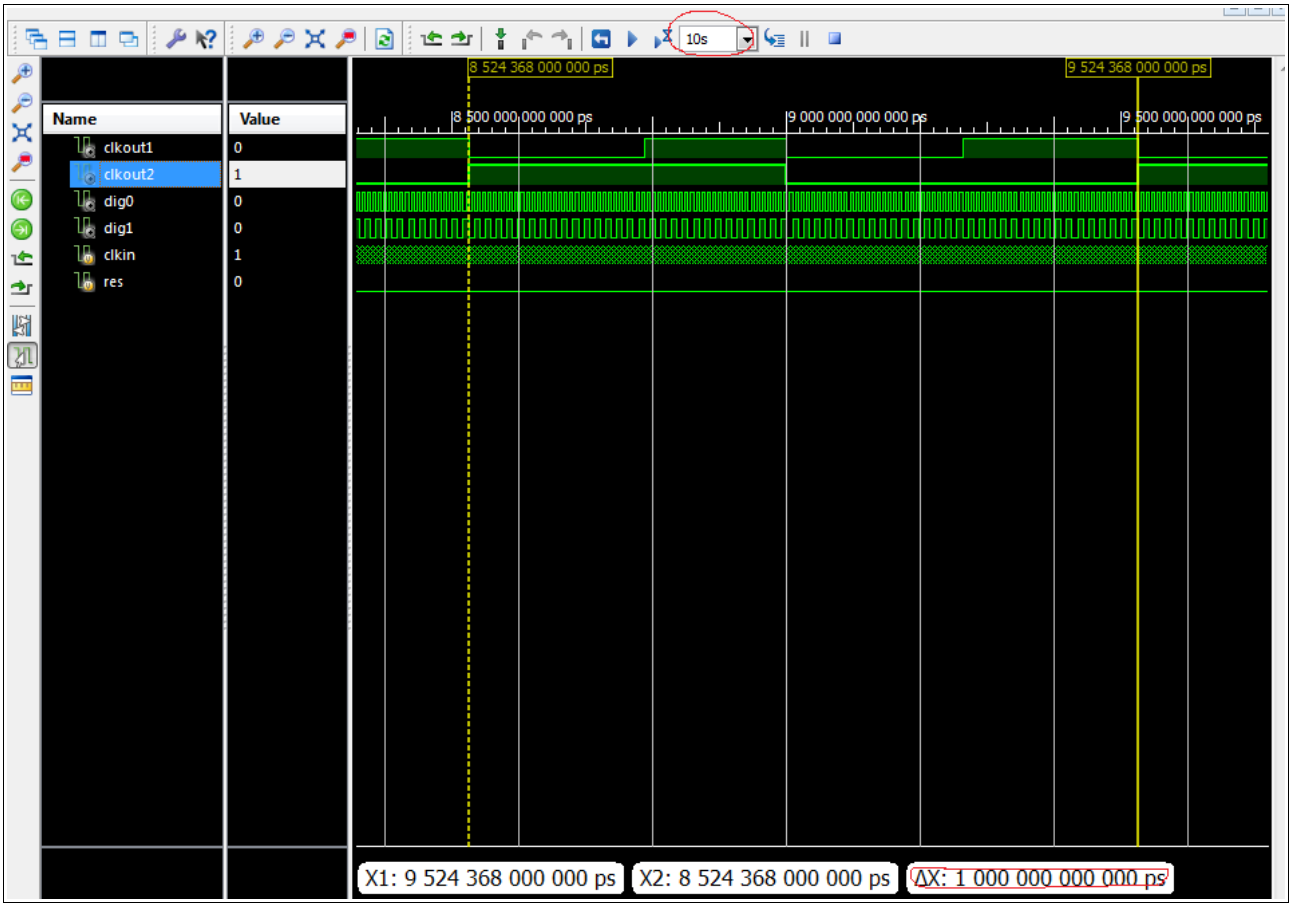
Πατάω implement design και αν όλα πάνε καλά μπορώ να γράψω την λογική στο πυρίτιο.

Εξομοίωση και μέτρηση χρόνου

Για δοκιμή θα φτιάξω ένα διαιρέτη που θα δέχεται ρολόι 31250 Hz και θα βγάζει παλμό 1 Hz. Αρχικά βάζω `timescale 1us / 1ns δηλαδή το τικ είναι 1μsec και έχω ακρίβεια στην εξομοίωση 1ns.
Στο τέλος για το ρολόι γράφω :

```
always begin
   #16  clkin = ~clkin; // Toggle clock every 16 ticks -> 16+16=32us -> 31250 Hz
end
```

Δηλαδή η ημιπερίοδος του παλμού θα διαρκεί 16 μsec.



Θέτω τον χρόνο εξομοίωσης 1sec και επιλέγω την κυματομορφή που θέλω να μετρήσω. Αυτή γίνεται έντονη. Αν κάνω αριστερό κλικ ο κέρσορας θα πάει αυτόματα στην πλησιέστερη αλλαγή κατάστασης. Μετά πάω λίγο δεξιά και πατάω κλικ και σέρνω μέχρι το σημείο που θέλω να μετρήσω. Τότε εμφανίζεται και δεύτερος κέρσορας και κάτω βλέπω την διαφορά ΔΧ, όπου στο παράδειγμά μας είναι 1 sec.

Post Fit Simulation
Εδώ βλέπω και τους χρόνους καθυστέρησης (propagation delay) για το συγκεκριμένο chip.

Σταύρος Φώτογλου