30

MATLAB & ψηφιακή λογική με FPGA

30.1 MATLAB και FPGA

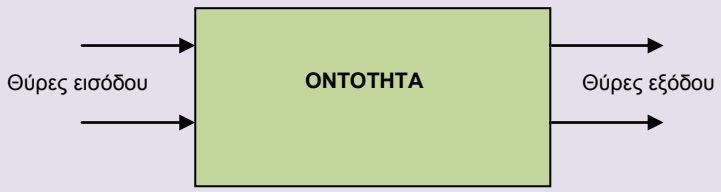
Η οντότητα (entity) και η αρχιτεκτονική (architecture) αποτελούν τις βασικές προγραμματιστικές δομές στη γλώσσα περιγραφής υλικού (hardware description language, HDL). Ενώ οι κώδικες των testbenches χρησιμοποιούνται για την εξομοίωση της σχεδίασης και την εξφαλμάτωσή της.

Οντότητα (entity): Κάθε οντότητα ορίζει τις εισόδους και τις εξόδους του κυκλώματος. Κάθε κύκλωμα πρέπει να έχει μια μοναδική οντότητα.

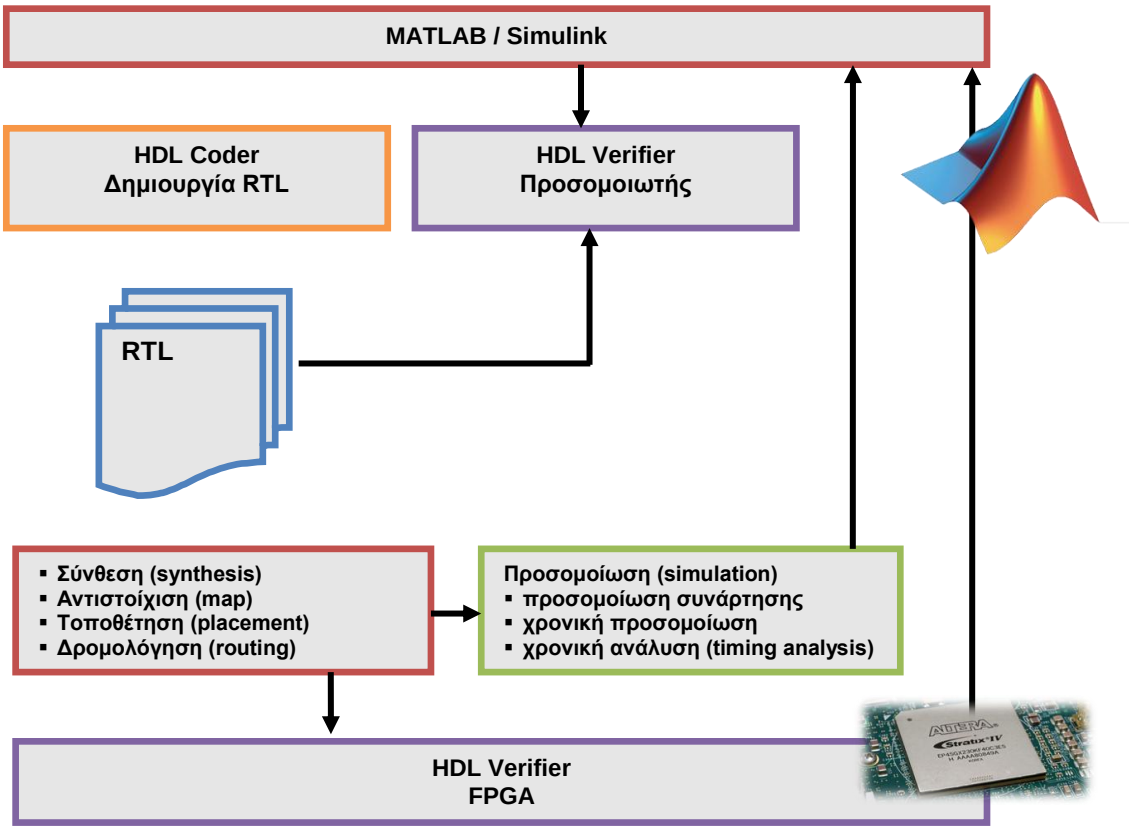
Αρχιτεκτονική (architecture): Περιγράφει το κύκλωμα το οποίο ορίζεται από την οντότητα. Για την περιγραφή του συστήματος ή της οντότητας μπορούμε να χρησιμοποιήσουμε μοντέλα ροής δεδομένων (data flow), συμπεριφοράς (behavioral) ή δομής (structural).

Παράδειγμα κώδικα VHDL για υλοποίηση πύλης AND

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

-- this is the entity
entity ANDGATE is
  port (
    I1 : in std_logic;
    I2 : in std_logic;
    O  : out std_logic);
end entity ANDGATE;
                    -- this is the architecture
architecture RTL of ANDGATE is
begin
  O <= I1 and I2;
end architecture RTL;
```



Η δυνατότητα που μας δίνει το εργαλείο του HDL Coder είναι ότι μέσω των εντολών MATLAB μπορούμε να δημιουργήσουμε κώδικα για την ανάπτυξη ενσωματωμένων εφαρμογών. Επίσης μέσω του HDL Verifier μπορούμε να επαληθεύσουμε τη λειτουργία της εφαρμογής μας. Για την υλοποίηση της εφαρμογής σε μονάδα FPGA (field programmable gate array) μπορούμε να χρησιμοποιήσουμε ενσωματωμένη λειτουργία του MATLAB η οποία συνεργάζεται με λογισμικά προγραμματισμού της Xilinx και της Altera. Στη ενότητα αυτή θα παρουσιάσουμε τα βήματα σχεδίασης και υλοποίησης ενός λογικού κυκλώματος και της σύνθεσης του στη μονάδα DE0-nano της Altera.



Τα τεχνικά χαρακτηριστικά της μονάδας DE0-nano που θα χρησιμοποιήσουμε είναι:

- FPGA

Altera Cyclone IV EP4CE22F17C6N: περιλαμβάνει 22,320 λογικά στοιχεία (logic elements - LEs), 594 Kbits ενσωματωμένη μνήμη, 66 18x18 πολλαπλασιαστές (οι οποίοι μπορούν να χρησιμοποιηθούν και ως 132 9x9 πολλαπλασιαστές), καθώς και 4 γενικού σκοπού PLLs

- Στοιχεία εγκατάστασης και σειριακή συσκευή διαμόρφωσης (αντίστοιχα)

Εγκατεστημένο στη πλακέτα USB-Blaster, Spansion EPCS63

- Υποδοχές επέκτασης

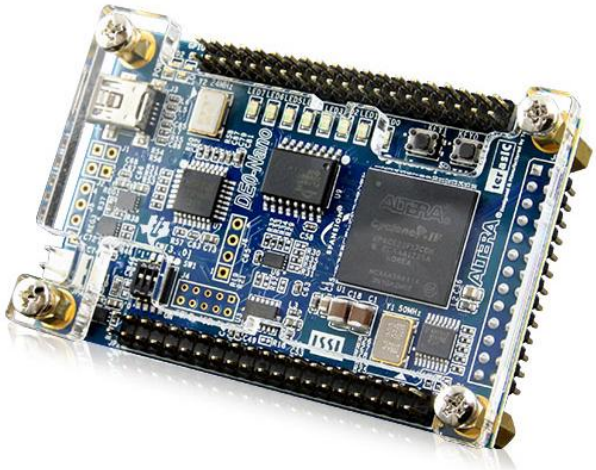
Δύο 40-pin Headers (υποδοχές εισόδου/εξόδου γενικού σκοπού - GPIOs), οι οποίες παρέχουν 72 pins εισόδου εξόδου, 1 pin 5V τροφοδοσίας, 2 pins 3.3V τροφοδοσίας και 4 pins γείωσης

- Μνήμη

32 MB SDRAM
2 Kb I2C EEPROM

- Γενική διεπαφή εισόδου/εξόδου χρήστη

8 πράσινα LEDs
2 debounced κουμπιά
4 διακόπτες DIP



- G-sensor

ADI ADXL345, 3D επιταχυσνιόμετρο με υψηλής ακρίβειας ανάλυση (13-bit)

- Μετατροπέας από αναλογικό σε ψηφιακό σήμα (A/D converter)

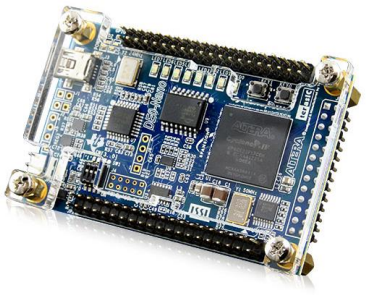
NS ADC128S022, 8 σημάτων, 12 bit, 50 Ksps to 200 Ksps

- Σύστημα ρολογιού

Ενσωματωμένο 50 MHz ρολόι oscillator

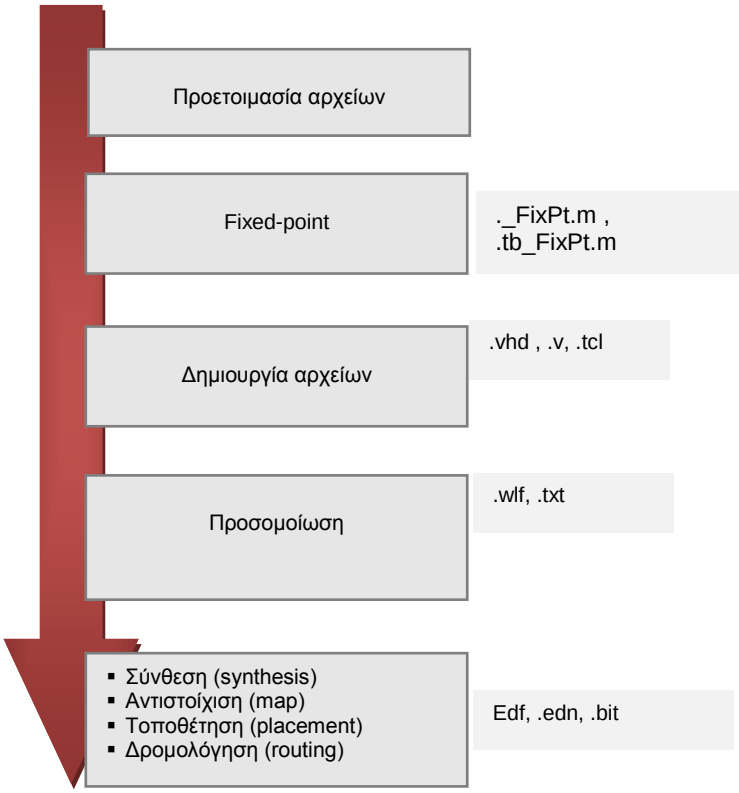
- Τροφοδοσία

USB τύπου mini-AB 5V
DC 5V pin για κάθε GPIO υποδοχή επέκτασης
2 pin εξωτερικής τροφοδοσίας 3.6V – 5.7V



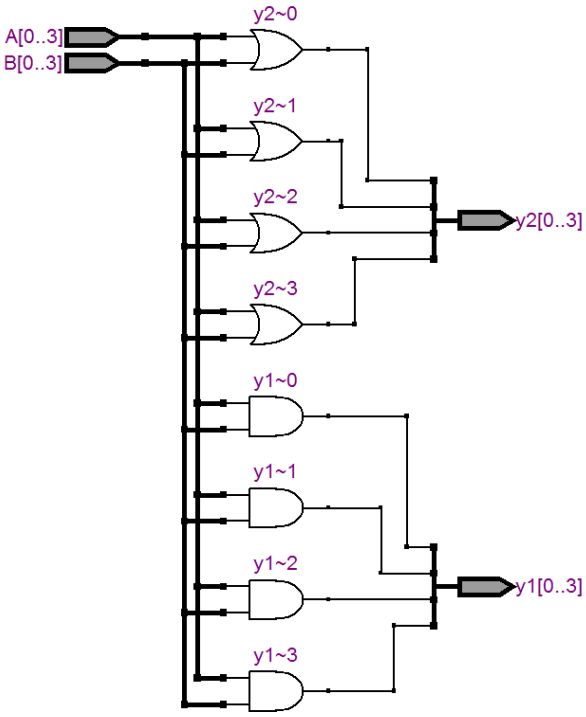
Τα στάδια που ακλουθούμε παρουσιάζονται στο διπλανό διάγραμμα.

- ✓ Δημιουργία αυτόματου τύπου δεδομένων Fixed-Point MATLAB με χρήση αρχείου testbench.
- ✓ Διαφοροποίηση του τύπου των δεδομένων ανάλογα της εφαρμογής.
- ✓ Έλεγχος του κώδικα Fixed-Point μέσω τον τύπο δεδομένων Floating-Point του κώδικα MATLAB.
- ✓ Δημιουργία κώδικα .vhd
- ✓ Έλεγχος του κώδικα μέσω προσομοίωσης.
- ✓ Σύνθεση και προγραμματισμός της μονάδας FPGA.



Δημιουργία 4bit λογικής πράξης AND και OR

Στο παράδειγμα θα παρουσιάσουμε όλα τα στάδια υλοποίησης λογικών πυλών 4 bit AND και OR στη μονάδα FPGA DE0-nano όπως απεικονίζεται στην διπλανή ψηφιακή σχεδίαση.

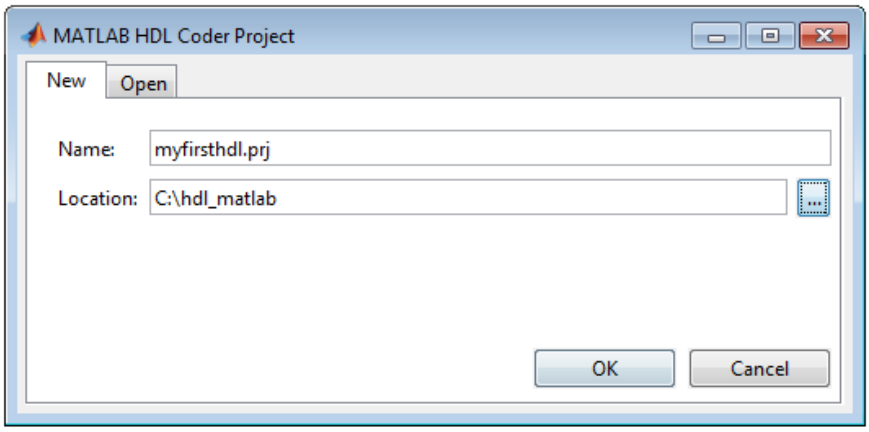


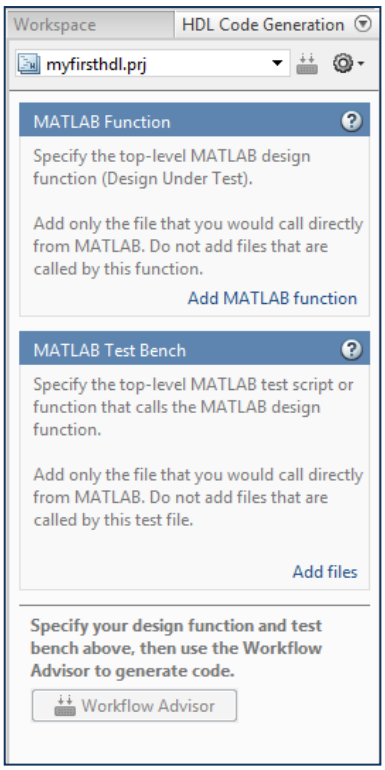
Βήματα προγραμματισμού οντότητα & αρχιτεκτονικής

Βήμα 1: Εκκινούμε το εργαλείο HDL Coder. Το εργαλείο HDL Coder βρίσκεται στη παλέτα APPS στο μενού του MATLAB.

Βήμα 2: Στο παράθυρο που ανοίγεται δίνουμε όνομα στο έργο (project).

Για παράδειγμα myfirsthdl.prj, και ορίζουμε τη διαδρομή στην οποία θα εκτελούνται όλες οι εργασίες του έργου και πατάμε OK.



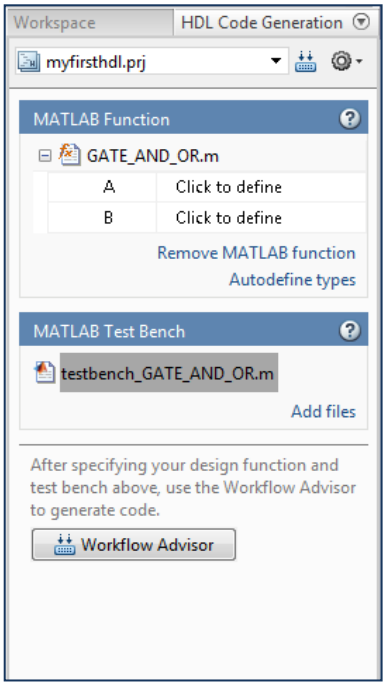


Στο σημείο αυτό το MATLAB ανοίγει στο περιβάλλον εργασίας του, το πεδίο HDL Code Generation.

Βήμα 3: Δημιουργούμε το αρχείο της οντότητας και του testbech σε διαφορετικά αρχεία MATLAB.

Προσοχή! Θα πρέπει τα παραπάνω αρχεία να βρίσκονται στον ίδιο φάκελο το αρχείο του έργου (project).

Δημιουργούμε τη συνάρτηση που ορίζει την οντότητα των πυλών. Θα πρέπει το όνομα του αρχείου να φέρει το ίδιο όνομα με τη συνάρτηση.



Προσδιορισμός αρχικών τιμών μεγέθους 4ibt

Λογική έκφραση AND

Λογική έκφραση OR

```
function [y1, y2] =
GATE_AND_OR( A, B)
 y1= false (1,4);
 y2= false (1,4);

 y1 = A & B ;

 y2 = A | B ;

end
```

Δημιουργούμε τη συνάρτηση που ορίζει το αρχείο testbench

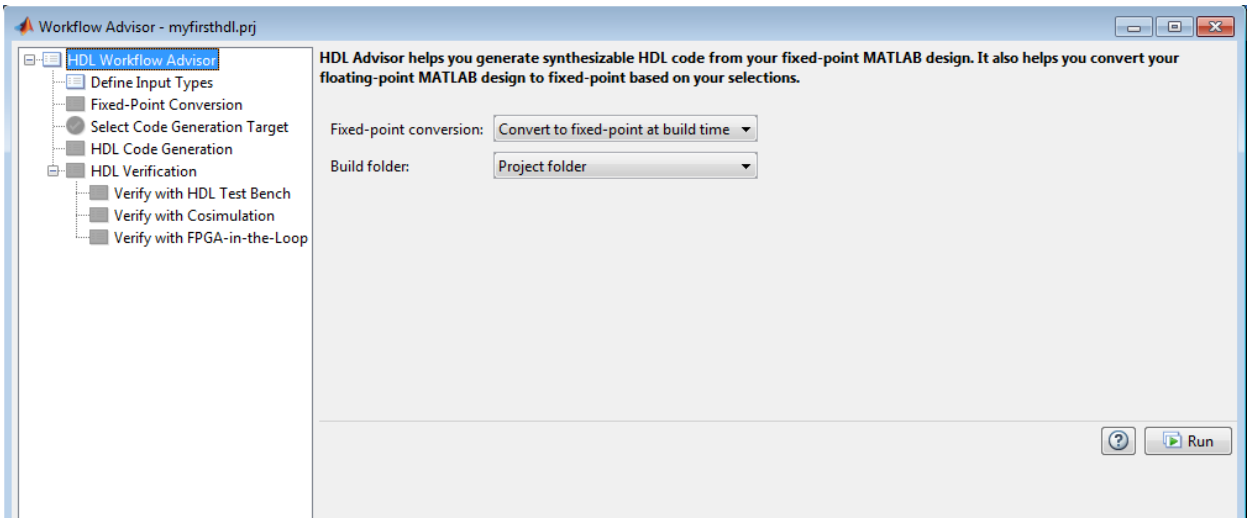
Προσδιορισμός λογικών τιμών προς έλεγχο της λειτουργίας

```
clear all;

A = 0001;
B = 0010;

[Y1, Y2] = GATE AND OR( A, B)
```

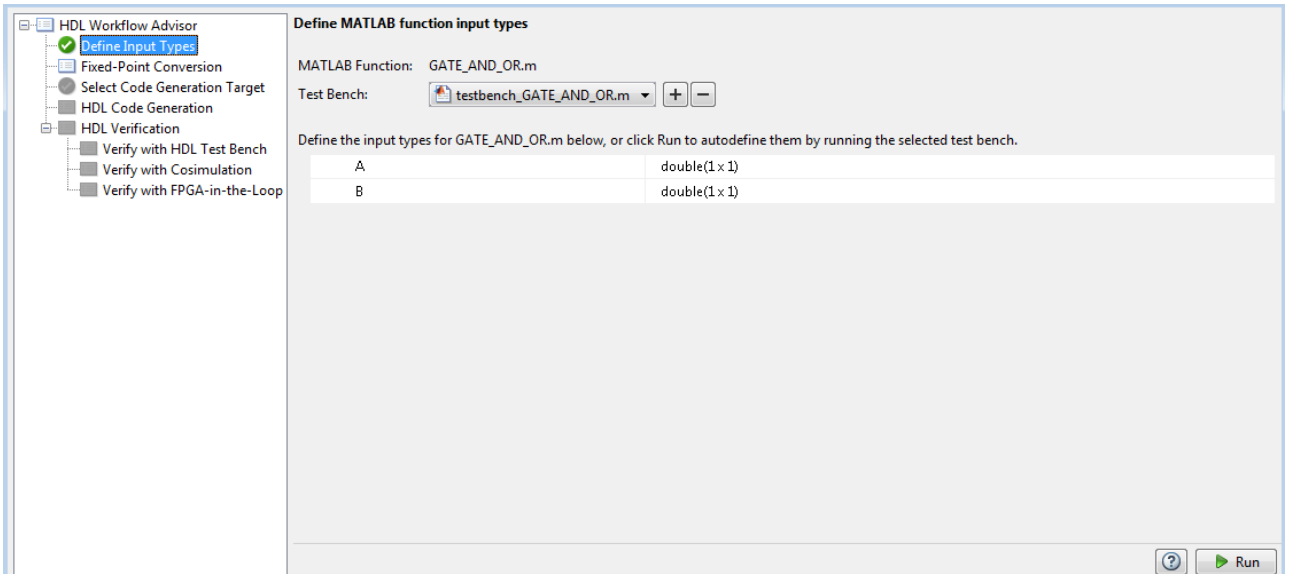
Βήμα 4: Εισάγουμε τον κώδικα της οντότητας και τον κώδικα του testbench επιλέγοντας Add MATLAB function και Add MATLAB test bench αντίστοιχα. Κάνοντας αυτές τις ενέργειες, στο πεδίο του HDL Code Generation εμφανίζονται οι είσοδοι και έξοδοι της οντότητας και το αρχείο του testbench.

Βήμα 5: Πατάμε το πλήκτρο Workflow Advisor.

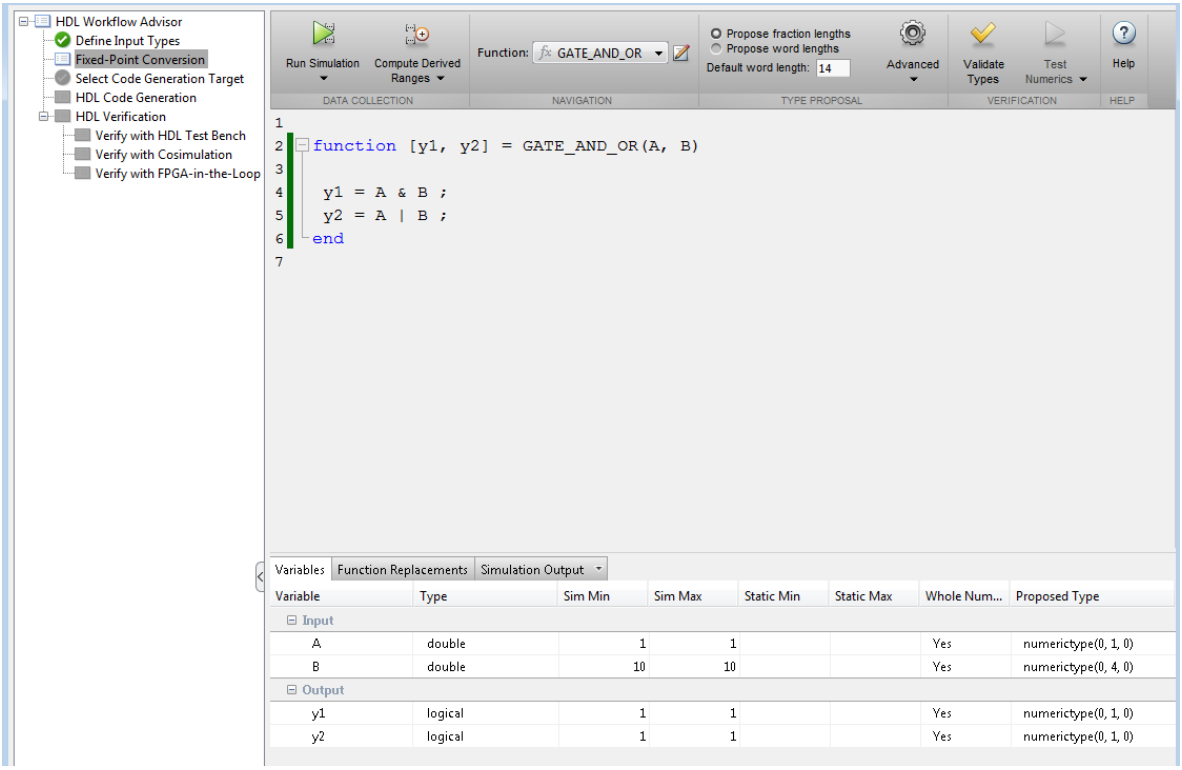
Στο περιβάλλον του Workflow Advisor οδηγούμαστε μέσα από ένα σύνολο βημάτων και επεξεργασίας δεδομένων στη δημιουργία του κώδικα HDL μέσα από κώδικα εντολών fixed-point και floating-point του MATLAB. Προσέχουμε στο σημείο αυτό, ώστε ο φάκελος εργασίας του έργου να είναι σωστά ορισμένος και πατάμε το πλήκτρο RUN.



Βήμα 6: Μπορούμε γνωρίζοντας τον τύπο δεδομένων να ορίσουμε κάθε είσοδο της οντότητας μας ή να πατήσουμε το πλήκτρο RUN όπου το MATLAB καθορίζει αυτόματα τον τύπο κάθε εισόδου. Το MATLAB χρησιμοποιώντας τις πληροφορίες του αρχείου testbech προσδιορίζει τον τύπο δεδομένων για κάθε στοιχείο της συνάρτησης.



Βήμα 7: Στο στάδιο αυτό γίνεται η μετατροπή του κώδικα MATLAB. Στο βήμα αυτό μπορούμε να τροποποιήσουμε το μέγεθος και τον τύπο των δεδομένων.



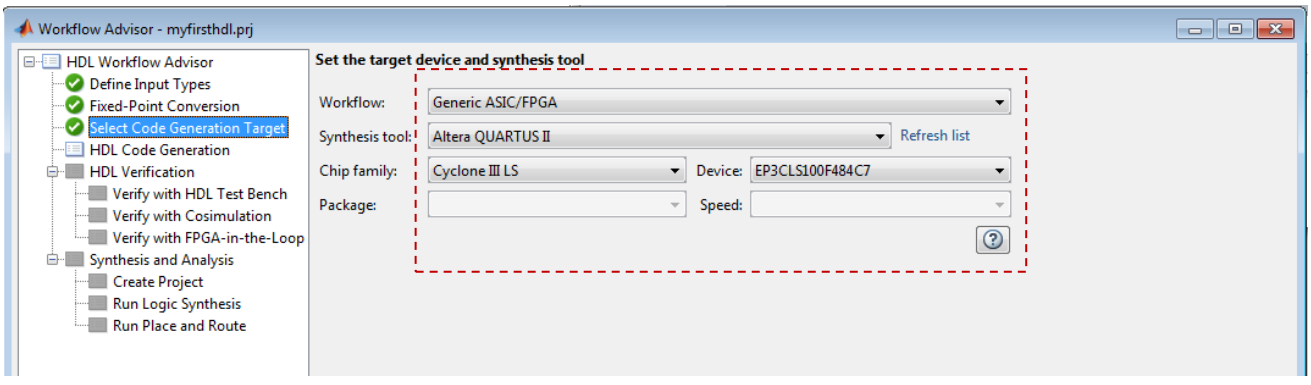
Βήμα 8: Εκκινούμε τη προσομοίωση κατά την οποία το εργαλείο αποδίδει τον τύπο δεδομένου για κάθε είσοδο και έξοδο της οντότητας.

Μπορούμε όμως να τροποποιήσουμε το μήκος της λέξης που θα χρησιμοποιεί ο τύπος του δεδομένου, καθώς και άλλες τροποποιήσεις που κρίνονται αναγκαίες.

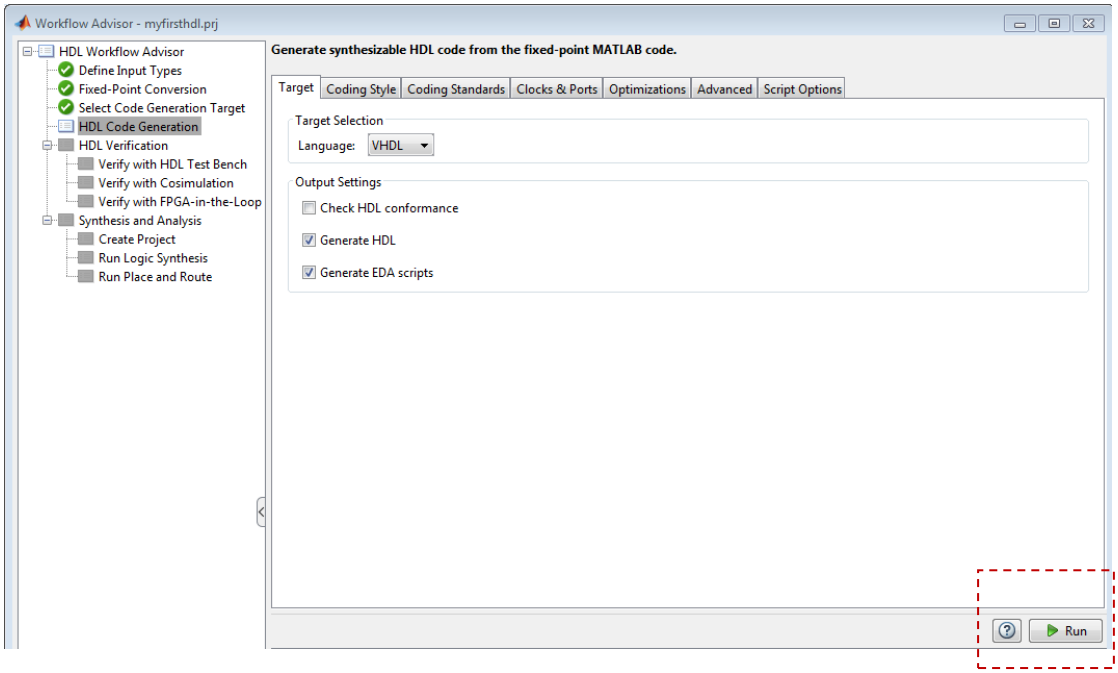
Βήμα 9: Καθορίζουμε το μήκος κάθε λέξης σε 4bit και επιλέγουμε Validate.

Εάν δεν υπάρξει σφάλμα εμφανίζεται το μήνυμα Validation succeeded.

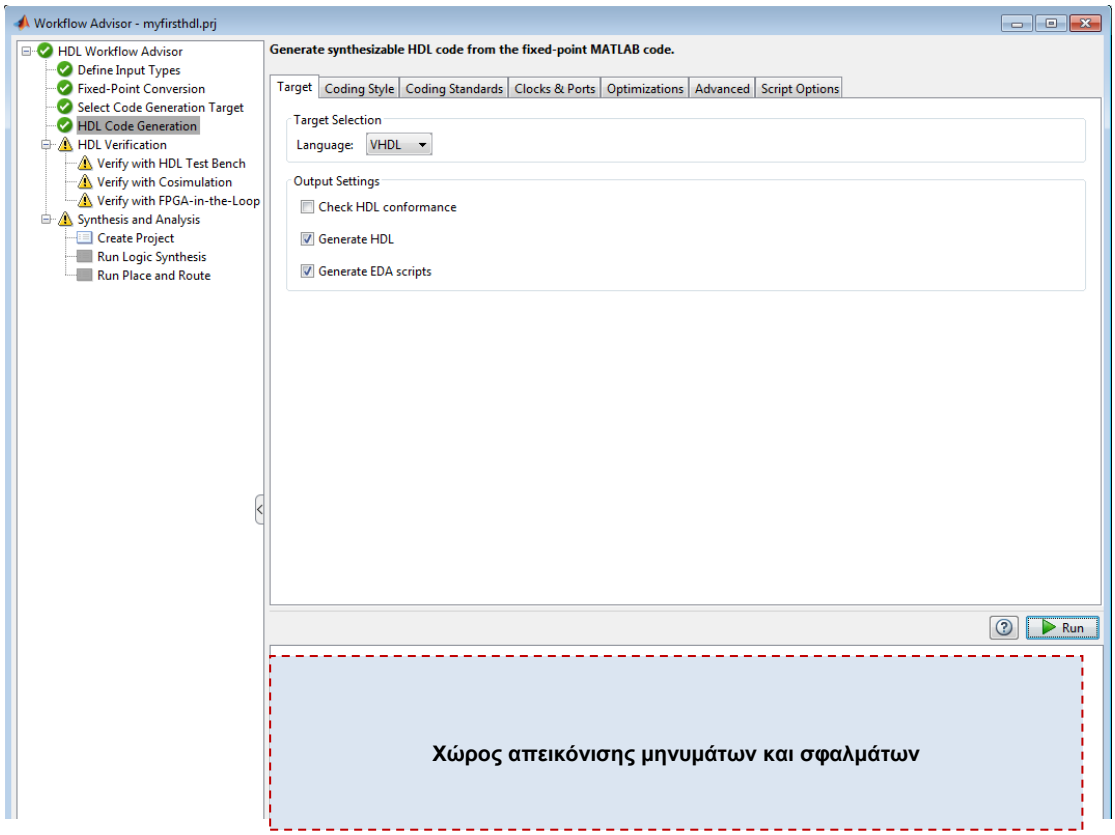
Βήμα 10: Στο πεδίο Select Code Generation Target ορίζουμε σύστημα και το ολοκληρωμένο στο οποίο θα υλοποιήσουμε την εφαρμογή μας. Επιλέγουμε με βάση το υλικό μας.



Βήμα 11: Στο πεδίο HDl Code Genetration επιλέγουμε παραμέτρους σύμφωνα με το υλικό μας, όπως τον τύπο του ρολογιού του συστήματος, μεταγλώττισης κώδικα se VHDL ή Verilog κ.α. Στη συνέχεια πατάμε το πλήκτρο RUN.



Εάν δεν υπάρξει σφάλμα εμφανίζονται πληροφορίες σχετικές με τα αρχεία που δημιουργήθηκαν.

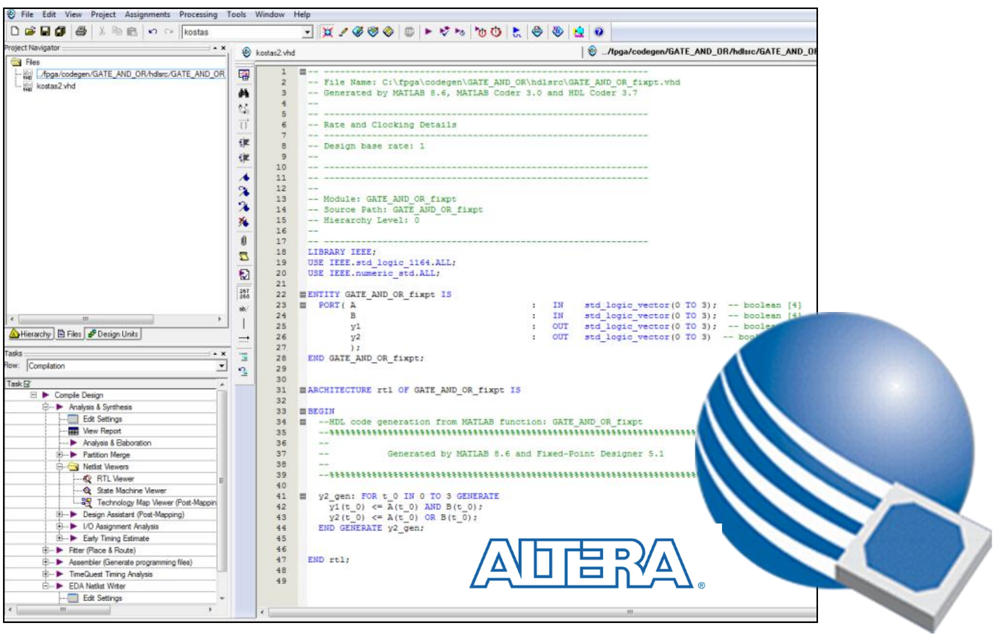


Επιλέγοντας το αρχείο από το παράθυρο των πληροφοριών μας ανοίγει στον Editor το αρχείο vhdl που δημιουργήθηκε.

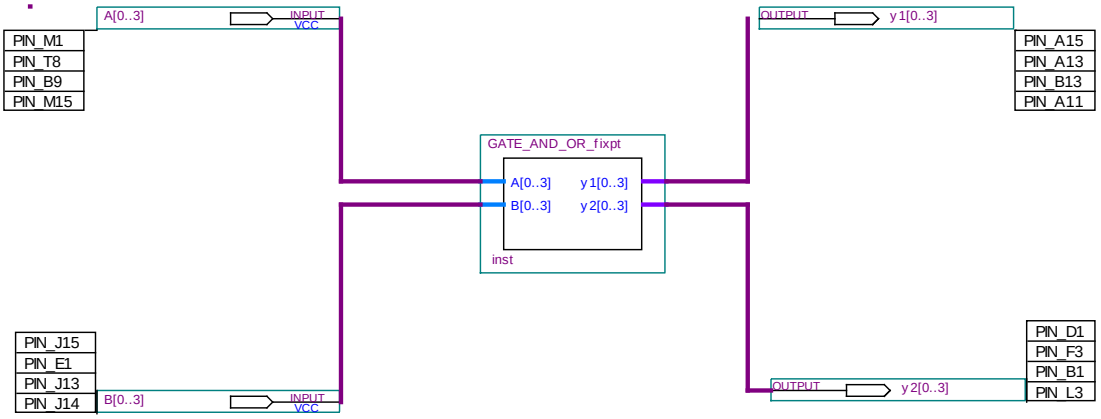
```
-- Generated by MATLAB 8.6, MATLAB Coder 3.0 and HDL Coder 3.7
-- -------------------------------------------------------------
-- Module: GATE_AND_OR_fixpt
-- Source Path: GATE_AND_OR_fixpt
-- -------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY GATE_AND_OR_fixpt IS
  PORT( A :   IN    std_logic_vector(0 TO 3);  -- boolean [4]
        B :   IN    std_logic_vector(0 TO 3);  -- boolean [4]
        y1:   OUT   std_logic_vector(0 TO 3);  -- boolean [4]
        y2:   OUT   std_logic_vector(0 TO 3)  -- boolean [4]
        );
END GATE_AND_OR_fixpt;


ARCHITECTURE rtl OF GATE_AND_OR_fixpt IS

BEGIN
  --HDL code generation from MATLAB function: GATE_AND_OR_fixpt
  --%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  --                                                                          %
  --           Generated by MATLAB 8.6 and Fixed-Point Designer 5.1           %
  --                                                                          %
  --%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  y2_gen: FOR t_0 IN 0 TO 3 GENERATE
    y1(t_0) <= A(t_0) AND B(t_0);
    y2(t_0) <= A(t_0) OR B(t_0);
  END GENERATE y2_gen;

 END rtl;
```

Για τη σύνθεση και την υλοποίηση του παραπάνω κώδικα μπορούμε να εργαστούμε στο λογισμικό Quarts II της Altera, όπως απεικονίζεται στην παρακάτω εικόνα.

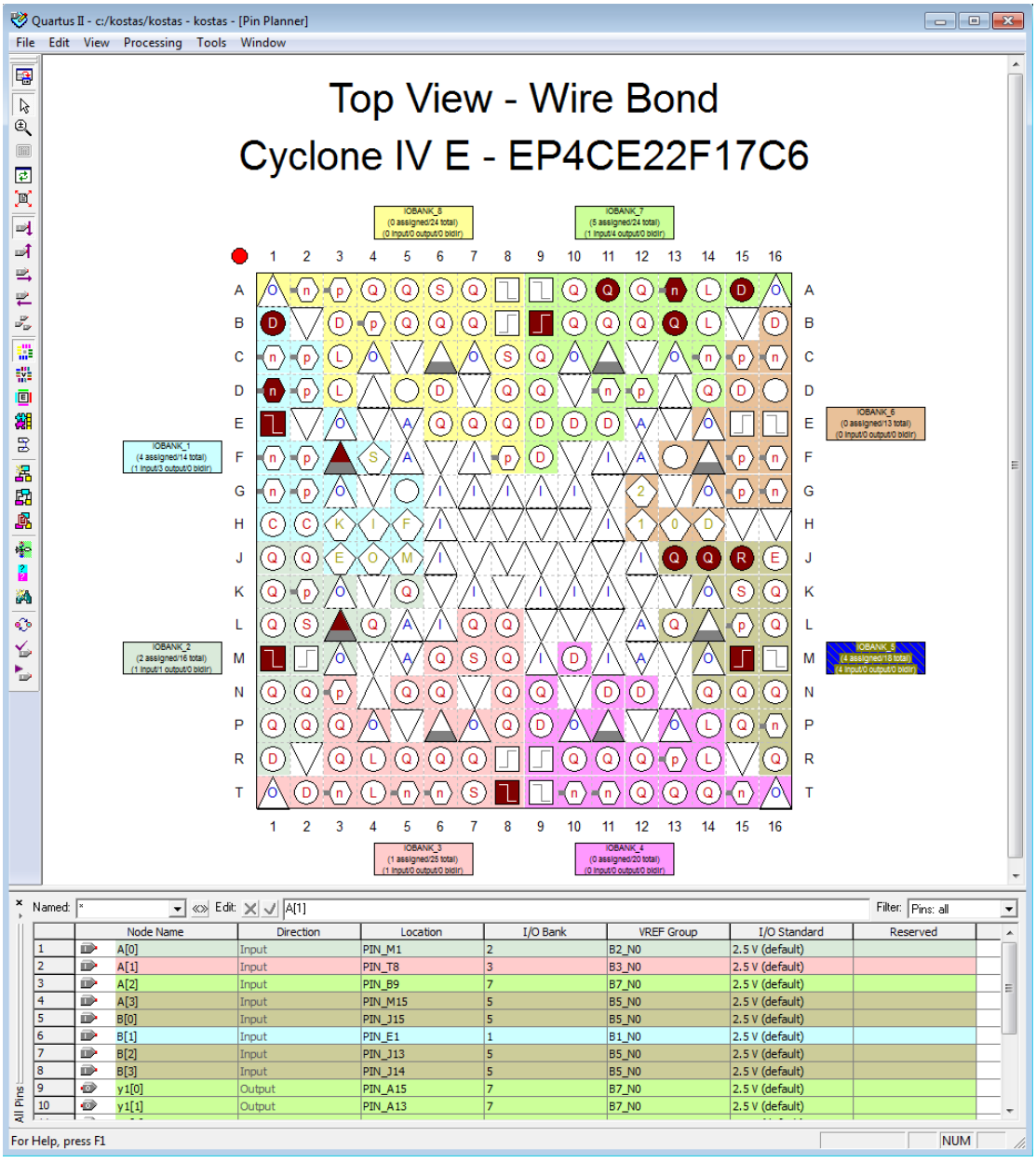
Ο κώδικας που δημιουργήσαμε από το MATLAB στο περιβάλλον του Quart II



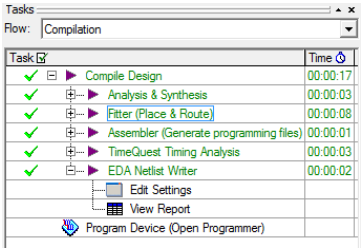
Μετατροπή σε μορφή λειτουργικού διαγράμματος



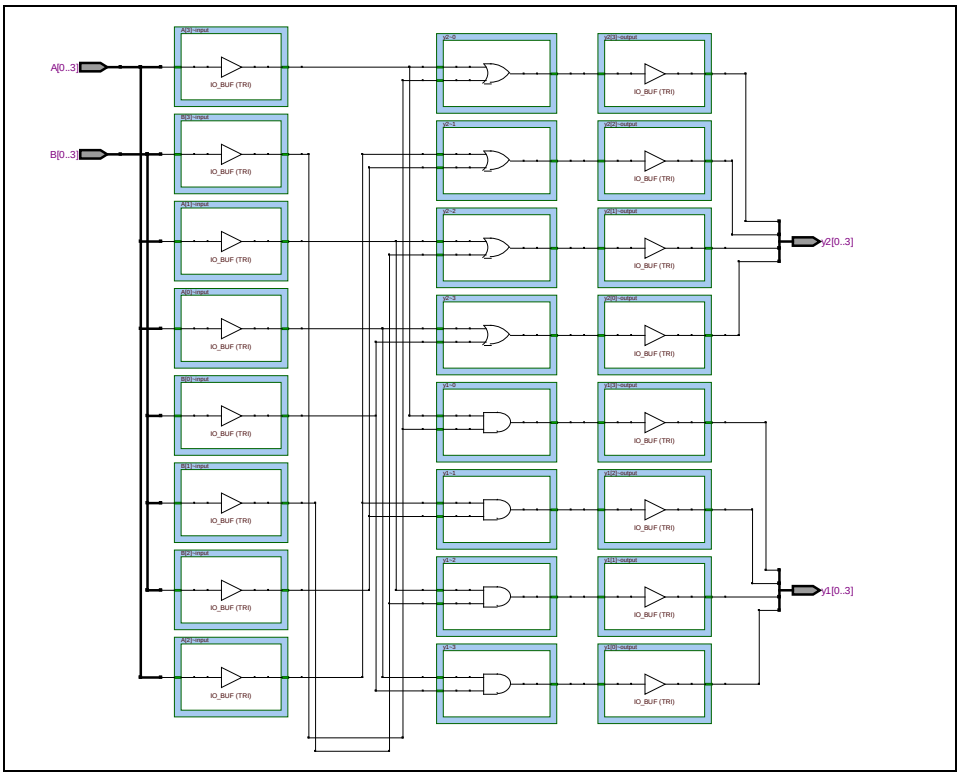
Επιλογή ακροδεκτών της σύνθεσης για το ολοκληρωμένο EP4CE22F17C6



Στάδιο μεταγλώττισης της εφαρμογής



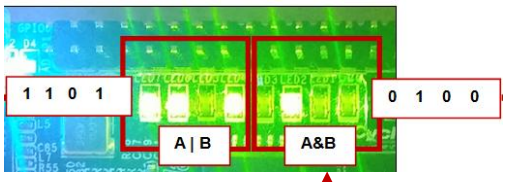
Προβολή του κυκλώματος της υλοποίησης στο ολοκληρωμένο EP4CE22F17C6



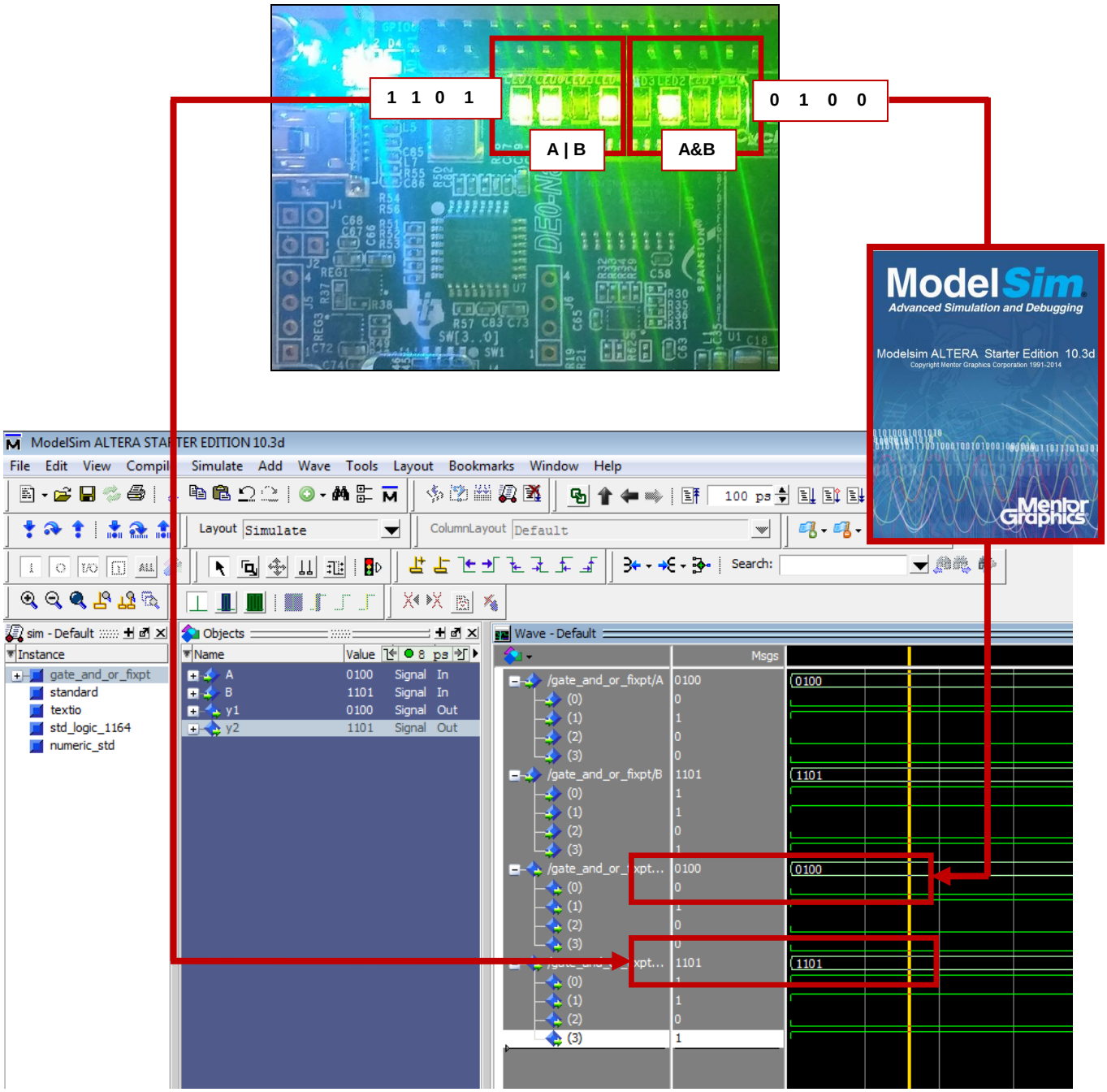
Αποτελέσματα εξόδου στο MATLAB

Σήματα εισόδου A= 0100 B=1101

```
Command Window
>> A=[0 1 0 0]

A =
     0     1     0     0

>> B=[1 1 0 1]

B =
     1     1     0     1

>> A&B

ans =
     0     1     0     0

>> A|B
ans =
     1     1     0     1
```



Στιγμιότυπο από τη λειτουργία του παραδείγματος στο FPGA DE0-nano και με επαλήθευση στο λογισμικό ModelSIM



30.2 Υλοποίηση συνδυαστικού κυκλώματος με MATLAB

Να υλοποιήσετε ψηφιακή σχεδίαση σε μονάδα FPGA κατά την οποία επιλέγεται για έξοδο μεταξύ δύο εισόδων 4 bit, τη λογική πράξη AND, OR ή NAND. Η επιλογή της πράξης καθορίζεται από σήμα επιλογής select. Η ψηφιακή διάταξη απεικονίζεται παρακάτω. Για την υλοποίηση της διάταξης ακολουθούμε τα βήματα του προηγούμενου παραδείγματος με τη διαφορά ότι δεν δημιουργουμε αρχείο testbench.