ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΠΡΟΓΡΑΜΜΑ ΣΠΟΥΔΩΝ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ Τ.Ε.

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Σχεδιασμός Αλγορίθμου Ελέγχου Μετατροπέα Ηλεκτρονικών Ισχύος σε FPGA με τη χρήση HDL

Αθανασιάδης Βασίλειος Α.Μ.: ΗΝ06086

Επιβλέπων: Γεώργιος Χ. Χριστοφορίδης, Καθηγητής

Κοζάνη, Οκτώβριος 2023

Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία του φοιτητή (Βασίλειου Αθανασιάδη) που την εκπόνησε. Στο πλαίσιο της πολιτικής ανοικτής πρόσβασης ο συγγραφέας/δημιουργός εκχωρεί στο Πανεπιστήμιο Δυτικής Μακεδονίας, μη αποκλειστική άδεια χρήσης του δικαιώματος αναπαραγωγής, προσαρμογής, δημόσιου δανεισμού, παρουσίασης στο κοινό και ψηφιακής διάχυσής τους διεθνώς, σε ηλεκτρονική μορφή και σε οποιοδήποτε μέσο, για διδακτικούς και ερευνητικούς σκοπούς, άνευ ανταλλάγματος και για όλο το χρόνο διάρκειας των δικαιωμάτων πνευματικής ιδιοκτησίας. Η ανοικτή πρόσβαση στο πλήρες κείμενο για μελέτη και ανάγνωση δεν σημαίνει καθ' οιονδήποτε τρόπο παραχώρηση δικαιωμάτων διανοητικής ιδιοκτησίας του συγγραφέα/δημιουργού ούτε επιτρέπει την αναπαραγωγή, αναδημοσίευση, αντιγραφή, αποθήκευση, πώληση, εμπορική χρήση, μετάδοση, διανομή, έκδοση, εκτέλεση, «μεταφόρτωση» (downloading), «ανάρτηση» (uploading), μετάφραση, τροποποίηση με οποιονδήποτε τρόπο, τμηματικά ή περιληπτικά της εργασίας, χωρίς τη ρητή προηγούμενη έγγραφη συναίνεση του συγγραφέα/δημιουργού. Ο συγγραφέας/δημιουργός διατηρεί το σύνολο των ηθικών και περιουσιακών του δικαιωμάτων.

Περιληψη

Σε αυτήν την εργασία περιγράφεται λεπτομερώς η υλοποίηση κώδικα HDL (Hardware Description Language) ενός κυκλώματος Pusle-Width Modulation (PWM) σε FPGA. Το πρόγραμμα που χρησιμοποιήθηκε είναι το Quartus Prime 22.1 Lite Edition και τέλος προσομοιώθηκε όλο το σύστημα στο ModelSim.

Λέξεις Κλειδιά: FPGA, HDL, Verilog, Quartus, PWM

ABSTRACT

This paper provides a detailed description of the implementation of HDL (Hardware Description Language) code for a Pulse-Width Modulation (PWM) circuit on an FPGA. The program used for this purpose was Quartus Prime 22.1 Lite Edition, and the entire system was simulated using ModelSim.

Keywords: FPGA, HDL, Verilog, Quartus, PWM

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω τον καθηγητή μου Δρ. Γεώργιο Χριστοφορίδη για την ανάθεση αυτής της πτυχιακής εργασίας, τον υποψήφιο διδάκτορα Παύλο Παπαγεωργίου για την καθοδήγηση που μου προσέφερε και τον Δρ. Κίμων Καρρά για τη βοήθεια στην πραγματοποίηση της.

Πινακας Περιεχομενων

Περιεχόμενα

Περίληψη 3
Abstract 4
Ευχαριστίες 6
Πίνακας Περιεχομένων 7
Κεφάλαιο 1: Εισαγωγικά Στοιχεία 8
1.1 FPGA 8
1.2 Γλώσσα Περιγραφής Υλικού (Hardware Description Language – HDL) 8
1.3 Η πλακέτα ανάπτυξης DE0-Nano 9
Κεφάλαιο 2: Υλοποίηση 11
Υλοποίηση Συστήματος σε FPGA 11
Το Κύκλωμα 11
Δημιουργία Κώδικα των Επιμέρους Στοιχείων στο Quartus με Verilog 12
Ο ADC128S022 18
Phase-Locked Loop (PLL) 19
Δημιουργία Συμβόλων και Σύνδεση 20
Κεφαλαίο 3: Testbench και Αποτελέσματα 22
Μελλοντική Εργασία 25
Βιβλιογραφία 26
Παράρτημα 27
Κώδικας Verilog Αφαιρέτη 27
Κώδικας Verilog Σταθεράς 27
Κώδικας Verilog PI ελεγκτή 28
Κώδικας Verilog Limiter 29
Κώδικας Verilog Συγκριτή 29
Κώδικας Verilog Τριγωνικού Σήματος 30
Ολόκληρο το Κύκλωμά σε μορφή Κώδικα Verilog 31
Testbench 34

ΚΕΦΑΛΑΙΟ 1: ΕΙΣΑΓΩΓΙΚΑ ΣΤΟΙΧΕΙΑ

1.1 FPGA

Οι προγραμματιζόμενες συστοιχίες πύλης πεδίου (FPGA) είναι συσκευές ημιαγωγών που βασίζονται γύρω από μια μήτρα διαμορφώσιμων λογικών μπλοκ, configurable logic block (CLB) που συνδέονται μέσω προγραμματιζόμενων διασυνδέσεων. Τα FPGA μπορούν να επαναπρογραμματιστούν στις επιθυμητές απαιτήσεις εφαρμογής ή λειτουργικότητας μετά την κατασκευή. Αυτή η δυνατότητα διακρίνει τα FPGA από τα Ολοκληρωμένα Κυκλώματα Ειδικών Εφαρμογών - Application-Specific Integrated Circuit (ASIC), τα οποία κατασκευάζονται κατά παραγγελία για συγκεκριμένες εργασίες σχεδίασης. Παρόλο που είναι διαθέσιμα τα FPGA με δυνατότητα προγραμματισμού μιας χρήσης - One-Time Programmable (OTP), οι κυρίαρχοι τύποι βασίζονται σε static random access memory (SRAM) και μπορούν να αναπρογραμματιστούν καθώς εξελίσσεται η σχεδίαση.

1.2 Γλώσσα Περιγραφής Υλικού (Hardware Description Language – HDL)

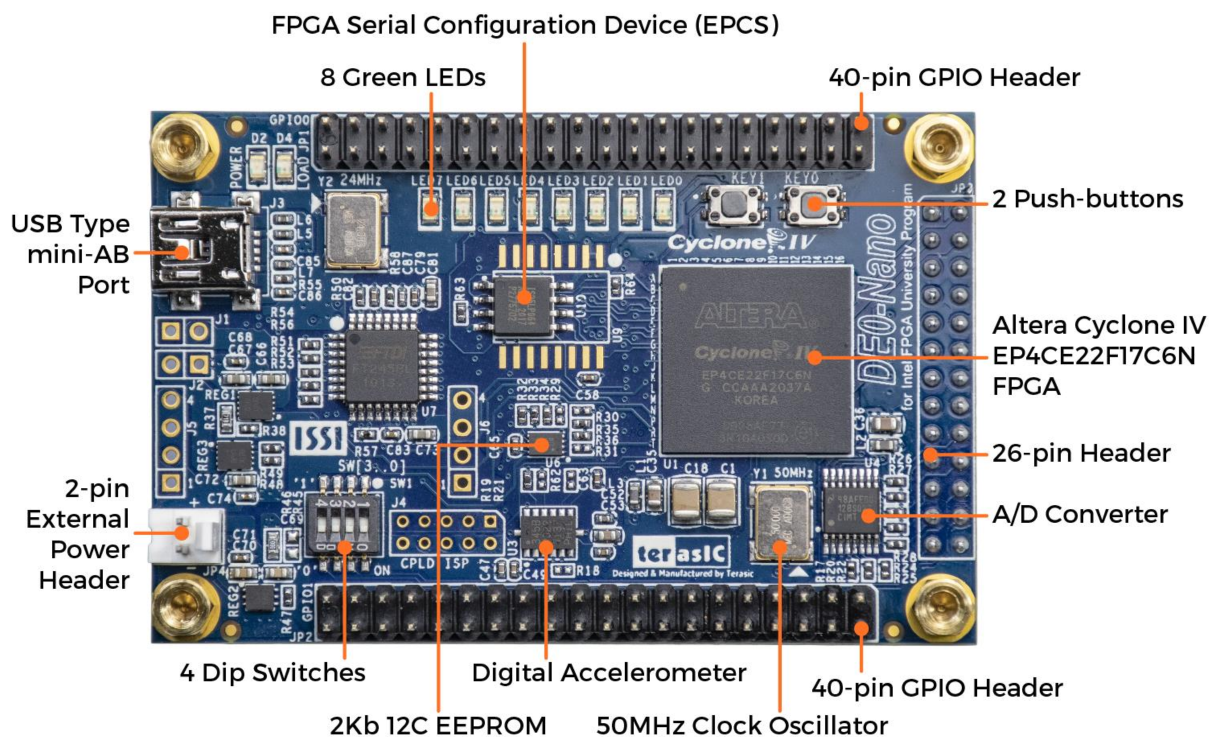
Η γλώσσα περιγραφής υλικού (Hardware Description Language - HDL) είναι μια γλώσσα προγραμματισμού που χρησιμοποιείται για τον σχεδιασμό, την περιγραφή και τον προσομοιωτή ψηφιακών κυκλωμάτων και συστημάτων. Οι HDL επιτρέπουν στους μηχανικούς σχεδίασης να περιγράψουν τη λειτουργία ενός κυκλώματος ή ενός συστήματος χρησιμοποιώντας γλώσσες προγραμματισμού αντί να ζωγραφίζουν τα κυκλώματα χειροκίνητα.

Οι δύο πιο γνωστές HDL είναι η VHDL (VHSIC Hardware Description Language) και η Verilog. Αυτές οι γλώσσες επιτρέπουν στους σχεδιαστές να περιγράψουν τη συμπεριφορά και τη δομή ενός ψηφιακού κυκλώματος με τρόπο που μπορεί να μεταφραστεί σε ψηφιακό λογικό σχεδιασμό. Αυτό διευκολύνει τον αυτοματοποιημένο σχεδιασμό, την προσομοίωση, τον έλεγχο και τη σύνθεση ψηφιακών κυκλωμάτων.

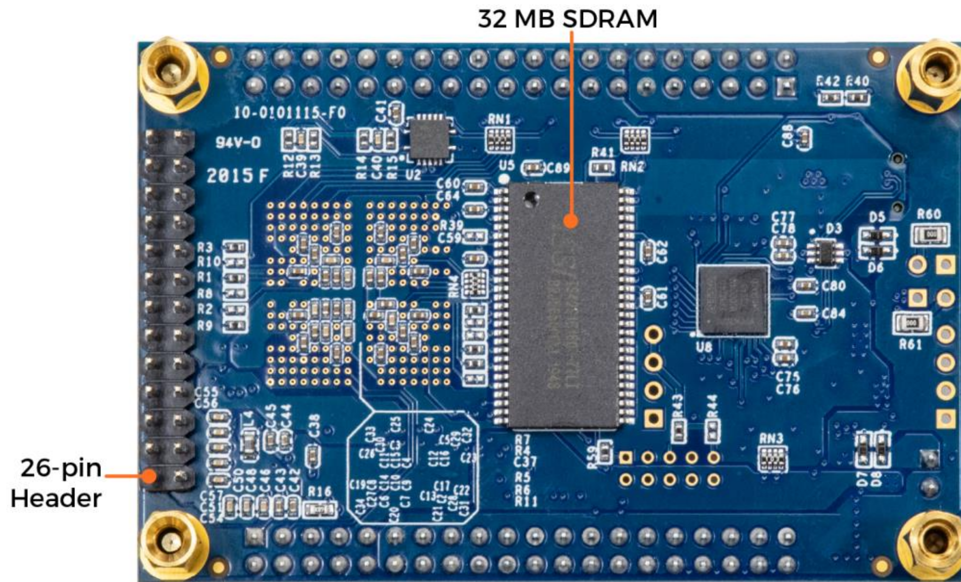
Οι HDL χρησιμοποιούνται ευρέως στον σχεδιασμό και την ανάπτυξη ενσωματωμένων κυκλωμάτων, FPGA (Field-Programmable Gate Array) και ASIC (Application-Specific Integrated Circuit), καθώς και σε εφαρμογές που απαιτούν ψηφιακή ψηφιακή επεξεργασία και σχεδιασμό hardware.

1.3 Η πλακέτα ανάπτυξης DE0-Nano

Η πλακέτα που χρησιμοποιήθηκε για την ψηφιακή σχεδίαση του συστήματος είναι η de0-nano της εταιρίας Terasic. Εκτός του επεξεργαστή η πλακέτα έχει και άλλα επιμέρους στοιχεία όπως ο ADC128S022 (Analog to Digital Converter) της National Instruments τον οποίο χρησιμοποιήσαμε και θα δούμε πιο αναλυτικά παρακάτω.



Εικόνα 1 - de0-nano top view



Εικόνα 2- de0-nano bottom view

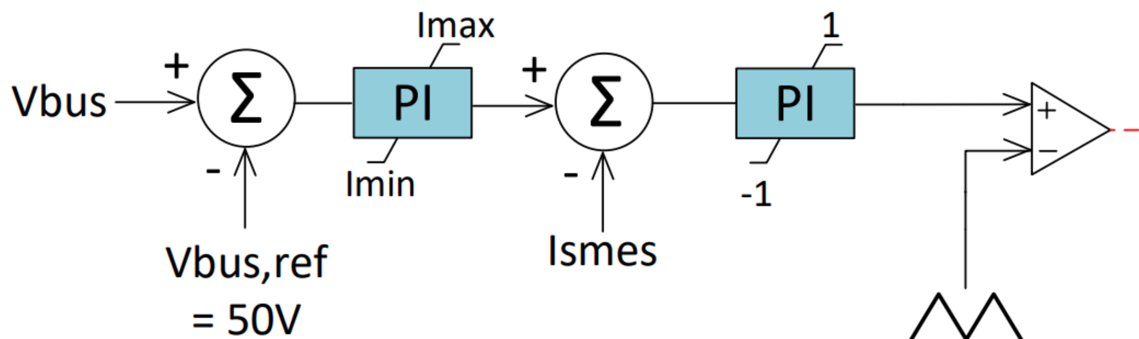
ΚΕΦΑΛΑΙΟ 2: ΥΛΟΠΟΙΗΣΗ

Σε αυτό το κεφάλαιο θα γίνει η περιγραφή της κάθε υλοποίησης ξεχωριστά, καθώς και τυχόν παραδοχές που έχουν τεθεί σε κάθε μία από αυτές.

Υλοποίηση Συστήματος σε FPGA

Για την υλοποίηση του κυκλώματος χρησιμοποιήθηκε το Quartus Prime Lite Edition (22.1). Το εργαλείο προσφέρει την δυνατότητα σύνθεσης και ανάλυσης Hardware Design Language (HDL) σχεδίων και βοηθάει στην σύνθεση στοιχείων σε επίπεδο πυλών ή και με κώδικα, την εκτέλεση ανάλυσης χρονισμού, την δημιουργία Register-Transfer Level (RTL) διαγραμμάτων και την προσομοίωση αντιδράσεων ενός σχεδίου με διαφορετικές παραμέτρους.

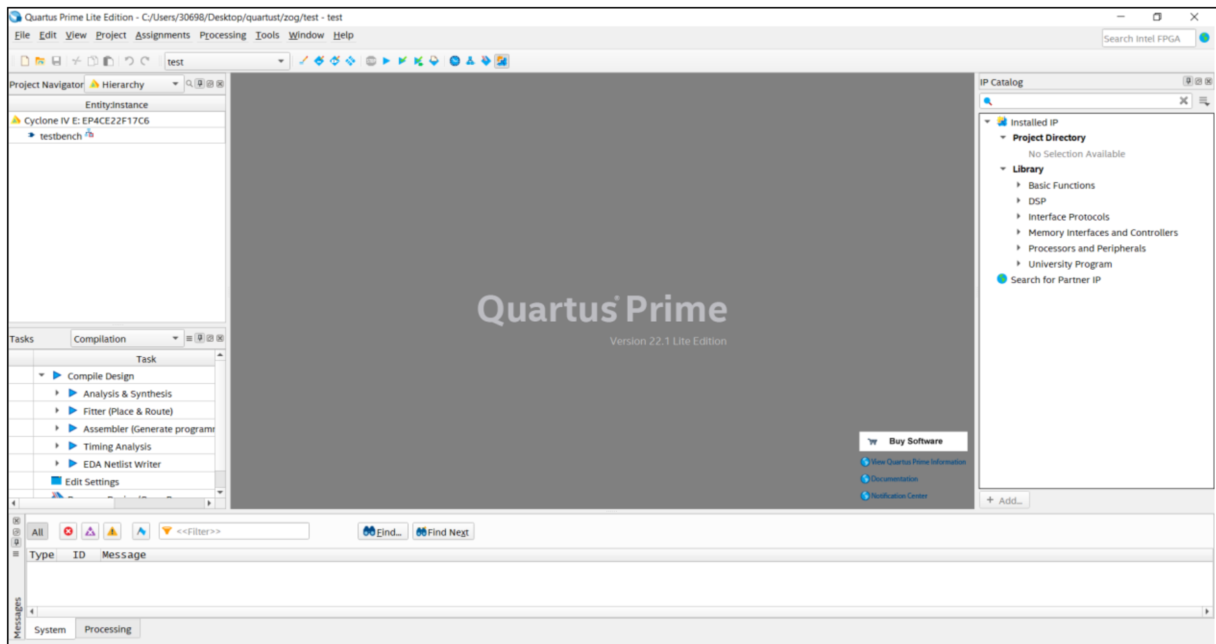
Το Κύκλωμα



Εικόνα 3 – Κύκλωμα

Το κύκλωμα αποτελείται από δυο αφαιρέτες, δυο PI ελεγκτές με μεταβλητές Kp=1 & Ki=1000 και Kp=0.1 & Ki=100 αντίστοιχα, δυο limiter στις εξόδους των ελεγκτών και έναν συγκριτή όπου έρχεται και ένα τριγωνικό σήμα και μας δίνει την έξοδο του συστήματος (PWMout) που παίρνει τις λογικές τιμές 0 και 1.

Δημιουργία Κώδικα των Επιμέρους Στοιχείων στο Quartus με Verilog



Εικόνα 4 - Quartus UI

Ανοίγοντας το Quartus βλέπουμε το γραφικό του περιβάλλον (**εικόνα 4**). Από εκεί πάμε στο **File > New > Verilog HDL File** για να δημιουργήσουμε ένα Verilog αρχείο.

Για το πρώτο στοιχείο, που είναι ο αφαιρέτης, ο κώδικας είναι αυτός στην **εικόνα 5** όπου στις γραμμές 2,3 και 4 ορίζουμε τις εισόδους και τις εξόδους και μέσα έχει μια απλή εντολή **always** όπου αφαιρεί τις τιμές των δυο εισόδων.

```
1  module subtractor(
2       input signal1,
3       input [7:0] input_pin,
4       output reg [31:0] result
5  );
6
7  always @(*) begin
8       result = signal1 - input_pin;
9  end
10
11 endmodule
```

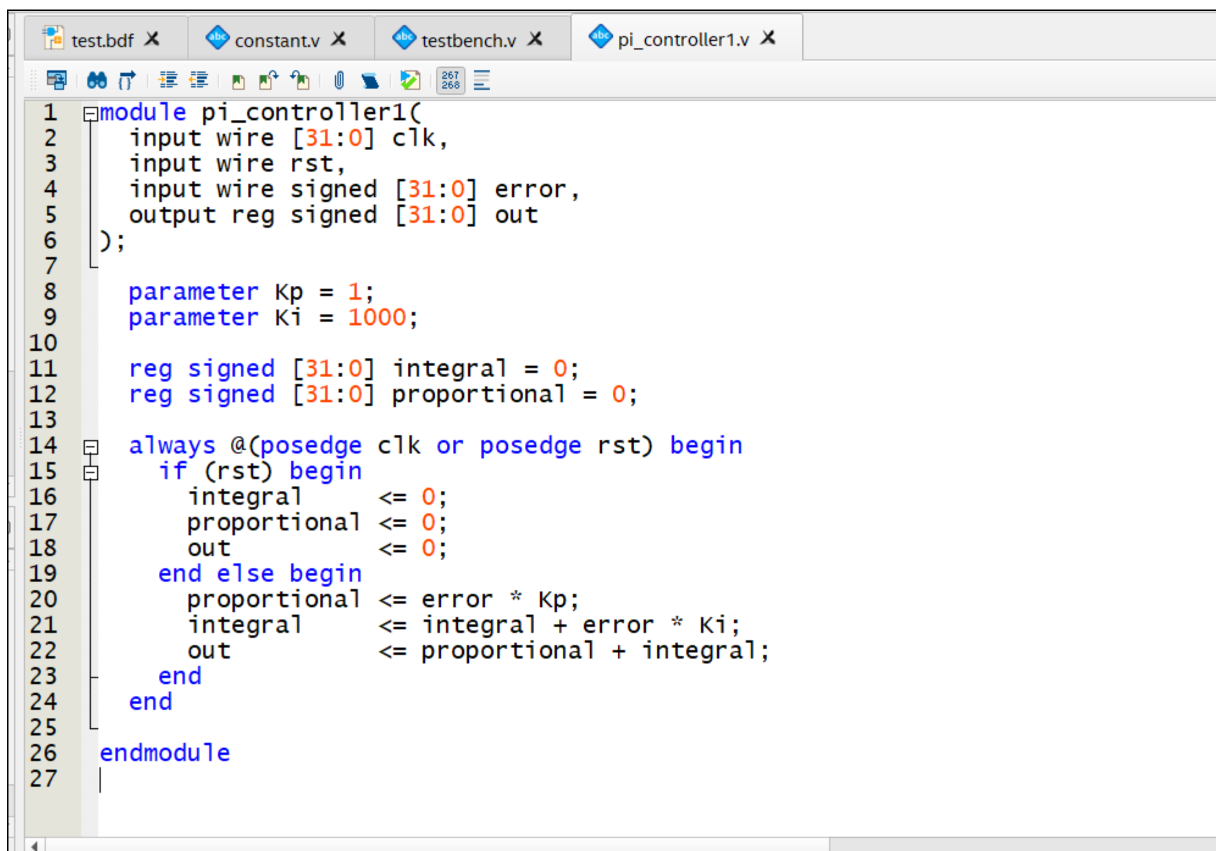
Εικόνα 5 - Subtractor Verilog Code

Δημιουργούμε ένα κώδικα για το Vbus(reference) που θέλουμε να είναι μια σταθερά στα 50 volt στο δεκαδικό σύστημα. (**εικόνα 6**)

```
1  module constant(
2       input [31:0] clk,
3       output reg [7:0] result
4  );
5
6  localparam MY_CONST = 50; // Define a constant value of 50 in decimal format
7
8  always @(posedge clk[31]) begin
9       result <= MY_CONST; // Assign the constant value to the output port
10 end
11
12 endmodule
```

Εικόνα 6 - Vref

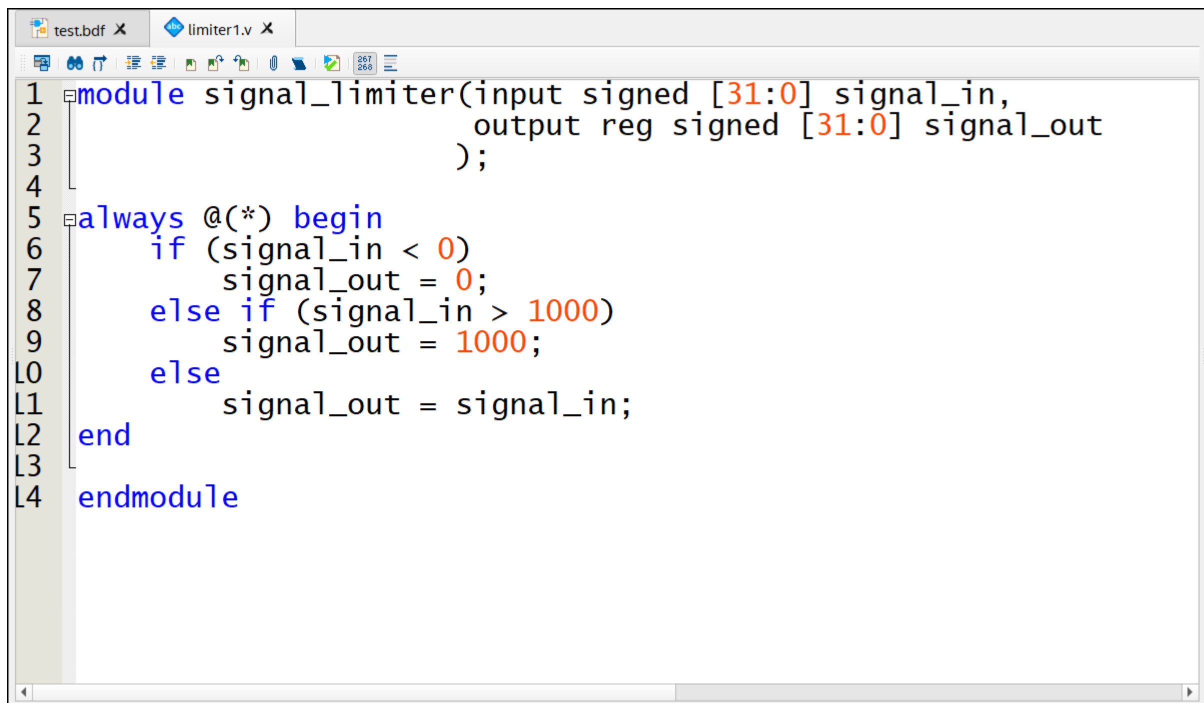
Το επόμενο στοιχείο είναι ο PI ελεγκτής με μεταβλητές Kp=1 & Ki=1000 που δίνεται στις γραμμές 8 και 9. (**εικόνα 7**)



```
module pi_controller1(
  input wire [31:0] clk,
  input wire rst,
  input wire signed [31:0] error,
  output reg signed [31:0] out
);

  parameter Kp = 1;
  parameter Ki = 1000;

  reg signed [31:0] integral = 0;
  reg signed [31:0] proportional = 0;

  always @(posedge clk or posedge rst) begin
    if (rst) begin
      integral     <= 0;
      proportional <= 0;
      out          <= 0;
    end else begin
      proportional <= error * Kp;
      integral     <= integral + error * Ki;
      out          <= proportional + integral;
    end
  end

endmodule
```

Εικόνα 7 - PI controller

Αμέσως μετά έχουμε έναν limiter που περιορίζει τις τιμές από την έξοδο του PI ελεγκτή μεταξύ 0 και 1000. (**εικόνα 8**)



Εικόνα 8 – Limiter

Αμέσως μετά έχουμε τον δεύτερο αφαιρέτη που ο κώδικας είναι ο ίδιος με αυτόν στην **εικόνα 5.**

Στον δεύτερο PI ελεγκτή έχουμε πάλι τον ίδιο κώδικα με τον προηγούμενο απλά αλλάζουμε τις μεταβλητές Kp και Ki όπως φαίνεται στην **εικόνα 9**.

```verilog
module pi_controller2(
   input wire [31:0] clk,
   input wire rst,
   input wire signed [31:0] error,
   output reg signed [31:0] out
);

   parameter Kp = 0.1;
   parameter Ki = 100;

   reg signed [31:0] integral = 0;
   reg signed [31:0] proportional = 0;

   always @(posedge clk or posedge rst) begin
     if (rst) begin
       integral <= 0;
       proportional <= 0;
       out <= 0;
     end else begin
       proportional <= error * Kp;
       integral <= integral + error * Ki;
       out <= proportional + integral;
     end
   end

endmodule
```

Εικόνα 9 - PI controller 2

Στην έξοδο του δεύτερου PI ελεγκτή έχουμε τον δεύτερο limiter 9 (**εικόνα 10**) που περιορίζει τις τιμές από -1 μέχρι 1 και έρχονται στην μια είσοδο ενός συγκριτή (**εικόνα 11**).

```verilog
module signal_limiter2(
     input [31:0] clk,
     input signed [31:0] signal_in,
     output reg signed [31:0] signal_out
);

always @(posedge clk[31]) begin
     if (signal_in < -1)
          signal_out = -1;
     else if (signal_in > 1)
          signal_out = 1;
     else
          signal_out = signal_in;
end

endmodule
```

Εικόνα 10 - limiter 2

```
module comparator (
    input [31:0] A,
    input [31:0] B,
    output reg EQ
);

always @* begin
    EQ = (A > B) ? 1 : 0;
end

endmodule
```

Εικόνα 11 – comparator

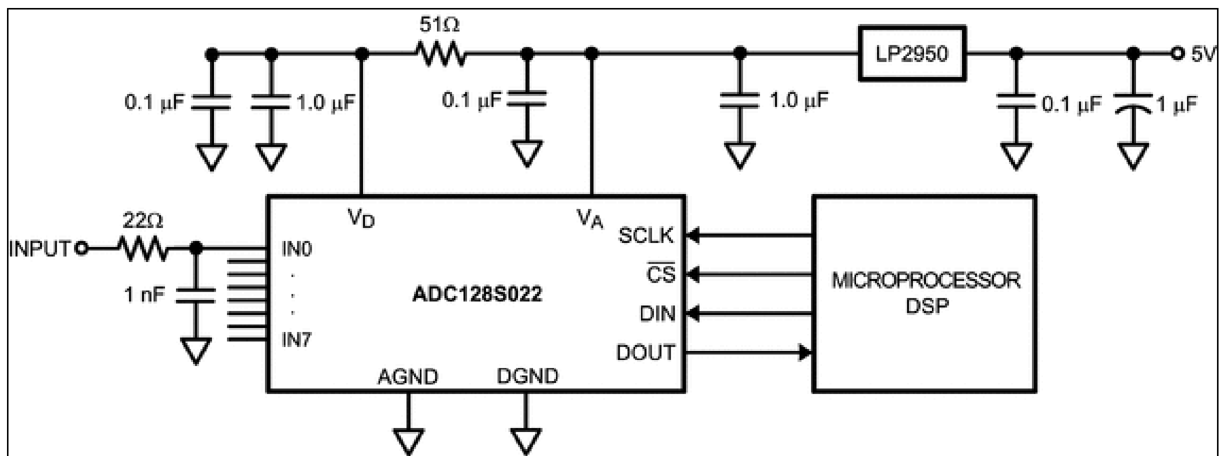
Στην άλλη είσοδο του συγκριτή έρχεται ένα τριγωνικό σήμα (**εικόνα 12**) που παίρνει τιμές από -1 έως 1. Στο τριγωνικό αυτό σήμα έχουμε δώσει συχνότητα 10khz.

```
module triangular_signal(
  input [31:0] clk,
  output reg signed [31:0] triangular_output
);

  parameter CNT_MAX = 2500; // sets frequency of the signal to 10 kHz
  reg signed [31:0] cnt_reg = 0;
  reg direction = 1'b1;

  always @(posedge clk[31]) begin
    if (cnt_reg == CNT_MAX - 1) begin
      direction <= ~direction;
      cnt_reg <= 0;
    end else begin
      cnt_reg <= cnt_reg + 1;
    end
    if (direction) begin
      triangular_output <= cnt_reg * 2 - CNT_MAX;
    end else begin
      triangular_output <= CNT_MAX - (cnt_reg * 2);
    end
  end
endmodule
```

Εικόνα 12 - triangular wave

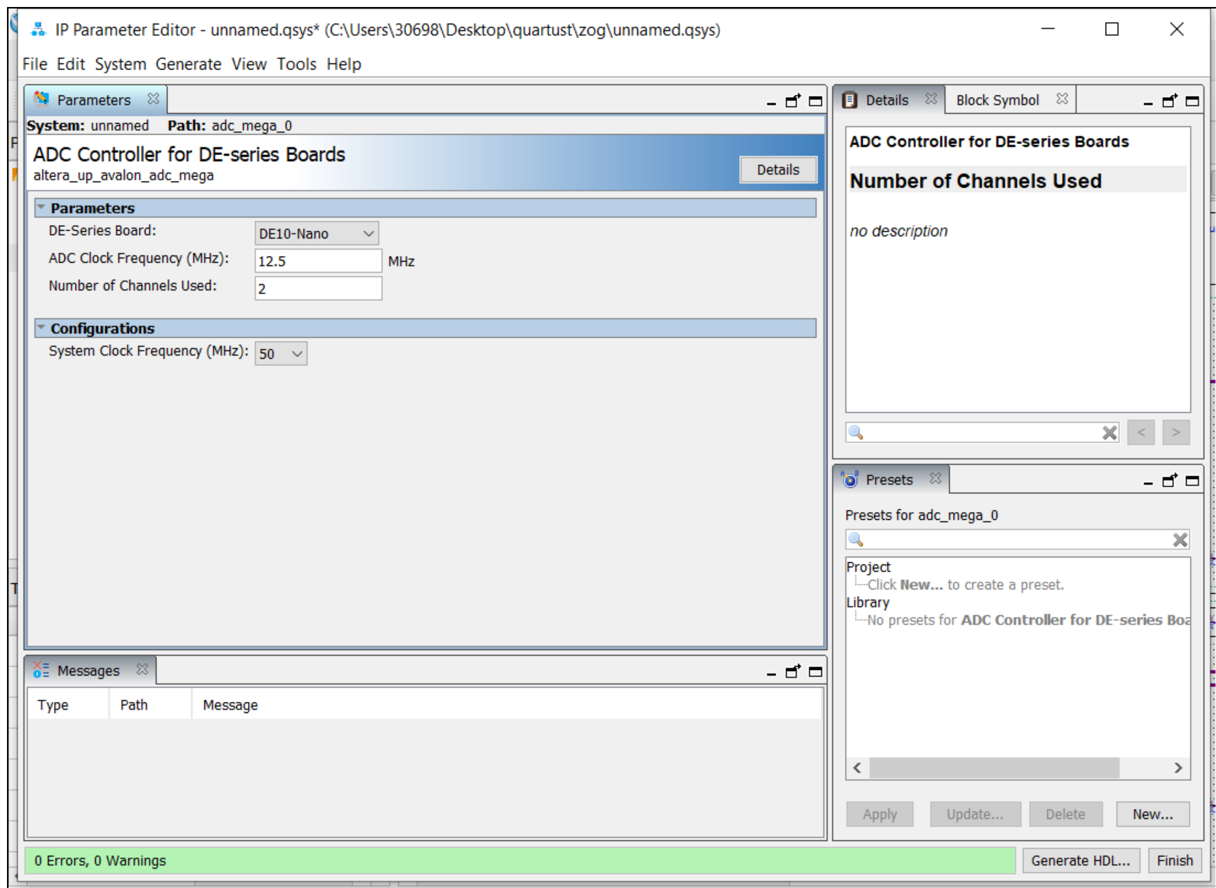
Ο ADC128S022

Ο ADC128S022 είναι ένας 12-bit A/D converter με 8 κανάλια και 50-200 ksps και επικοινωνεί με τον επεξεργαστή με το πρωτόκολλο SPI (**εικόνα 13**).
Το SPI(Serial Peripheral Interface) είναι ένα πρότυπο για σύγχρονη σειριακή επικοινωνία, που χρησιμοποιείται κυρίως σε ενσωματωμένα συστήματα για ενσύρματη επικοινωνία μικρής απόστασης μεταξύ ολοκληρωμένων κυκλωμάτων.



Εικόνα 13 - ADC - spi

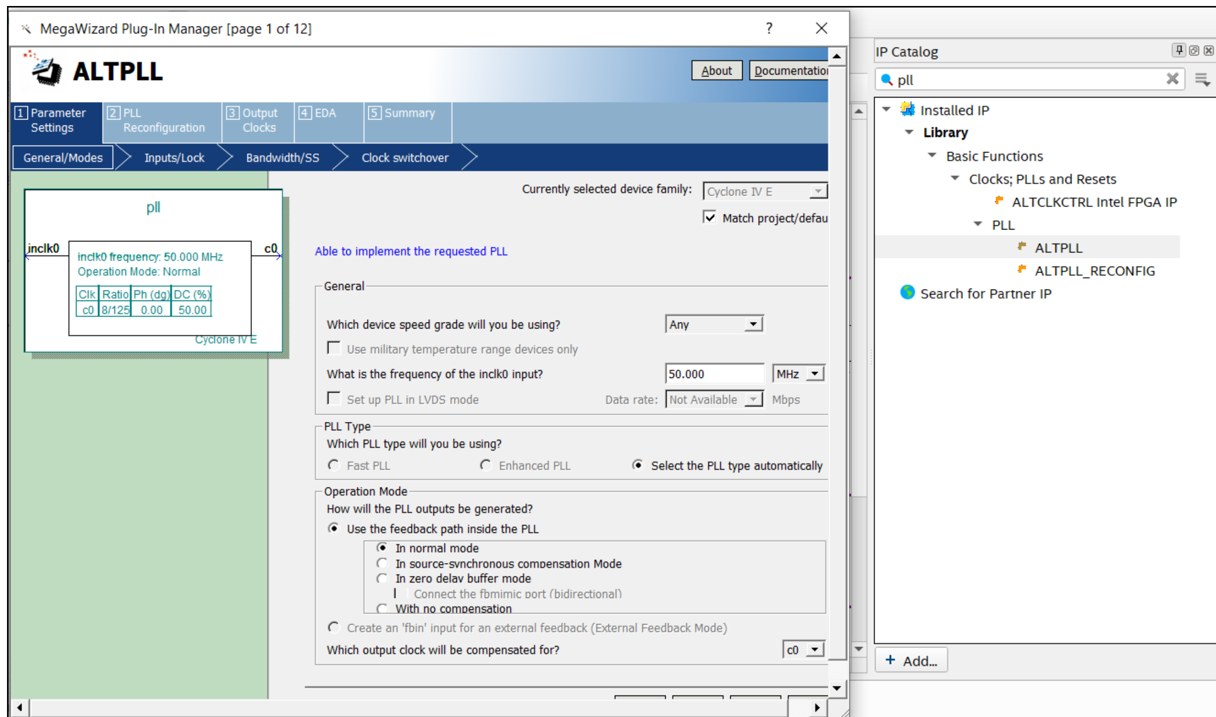
Για να τον ενσωματώσουμε στο σύστημα μας το Quartus προσφέρει μια τύπου εφαρμογή από το ip catalog που βρίσκεται στο δεξιά μέρος του προγράμματος και ονομάζεται ADC Controller for DE-series Boards. Εκεί επιλέγουμε το μοντέλο της πλακέτας και πόσα κανάλια θέλουμε να χρησιμοποιήσουμε (**εικόνα 14**). Πατώντας Generate HDL παράγεται ο κώδικας αυτόματα. Δυο από τα 8 κανάλια του ADC που καταλήγουν σε εξωτερικά pins της πλακέτας θα τα χρησιμοποιήσουμε για να δώσουμε τη τάση Vbus και Ismes στις άλλες εισόδους των αφαιρετών όπως φαίνεται στο κύκλωμα της **εικόνας 3**.



Εικόνα 14 - ADC controller

Phase-Locked Loop (PLL)

Για να συγχρονιστούν όλα τα στοιχεία μεταξύ τους και με το ρολόι της κάρτας θα συνδέσουμε ένα **PLL** στην είσοδο **clk** όλων των στοιχείων. Πάλι στο ip catalog υπάρχει έτοιμο και επιλέγοντας το, ανοίγει ένα παράθυρο (**εικόνα 15**) που απλά επιλέγουμε το μοντέλο της πλακέτας και τη συχνότητα του ρολογιού της που στην δικιά μας περίπτωση είναι 50mhz.

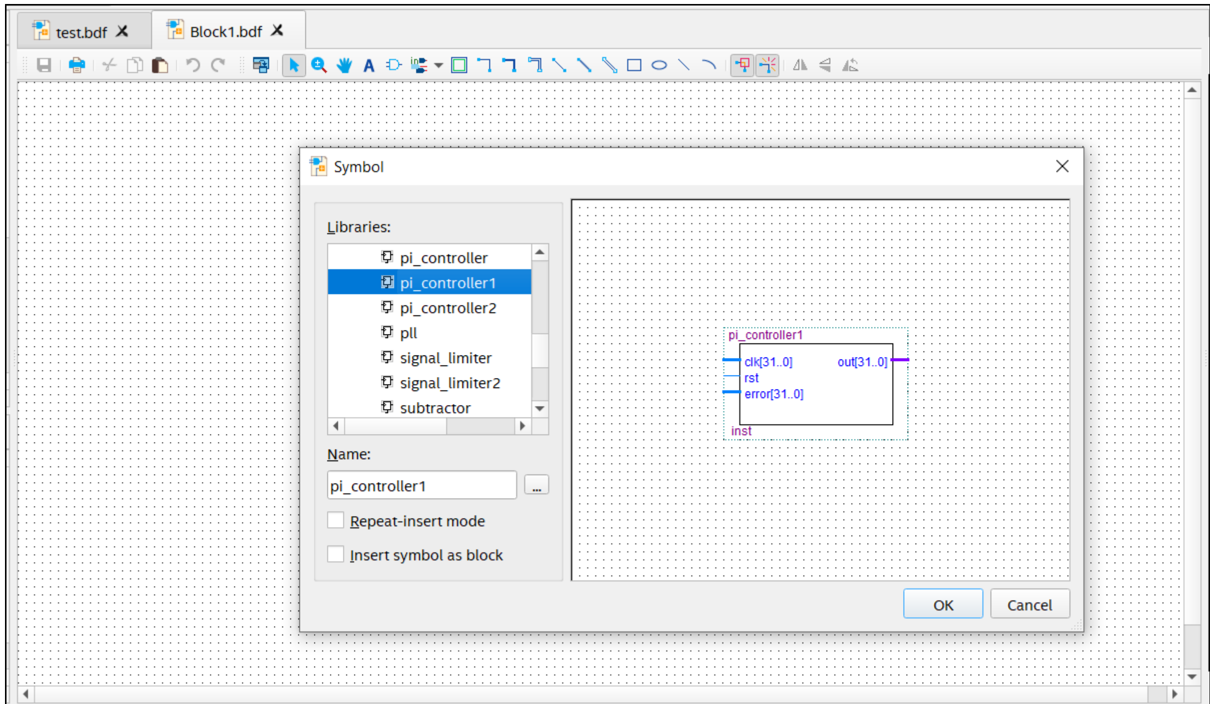


Εικόνα 15 - PLL

Δημιουργία Συμβόλων και Σύνδεση

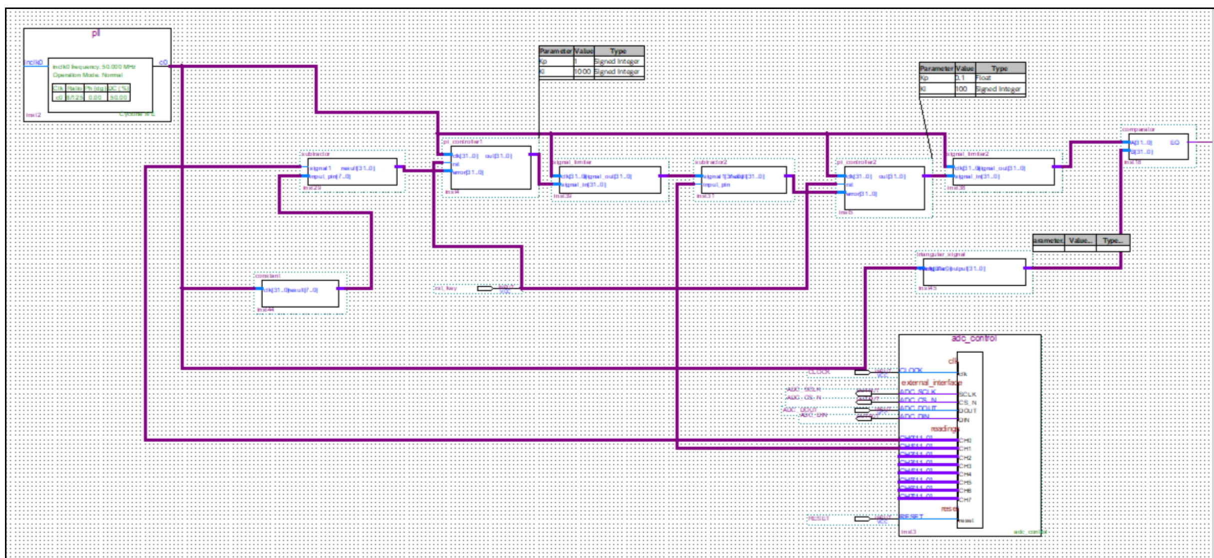
Για την καλύτερη κατανόηση και μια πιο οπτική επαφή με το σύστημα, το Quartus μας δίνει τη δυνατότητα δημιουργίας συμβόλων.

Στον φάκελο που έχουμε γραμμένο τον κώδικα για κάθε ένα ξεχωριστό στοιχείο, πατώντας στο **file** > **Create/Update** > **Create Symbol File for Current File** δημιουργείται ένα σύμβολο. Για να εισάγουμε το σύμβολο πρέπει πρώτα να δημιουργήσουμε τον κατάλληλο φάκελο, πηγαίνοντας στο **file** > **New** > **Block Diagram/Schematic file,** στο παράθυρο που εμφανίζεται πατώντας δεξί κλικ **Insert** > **Symbol** εμφανίζεται μια λίστα με τα σύμβολα που έχουμε δημιουργήσει (**εικόνα 16**).



Εικόνα 16 - Symbol File

Ακολουθώντας την ίδια διαδικασία για όλα τα στοιχεία, τα εισάγουμε στο **block diagram.** Για τη **σύνδεση** απλά σέρνουμε τον κέρσορα από τις εξόδους στις εισόδους των συμβόλων και έχουμε το κύκλωμα ολοκληρωμένο όπως φαίνεται στην **εικόνα 17**.



Εικόνα 17 - Κύκλωμα Quartus

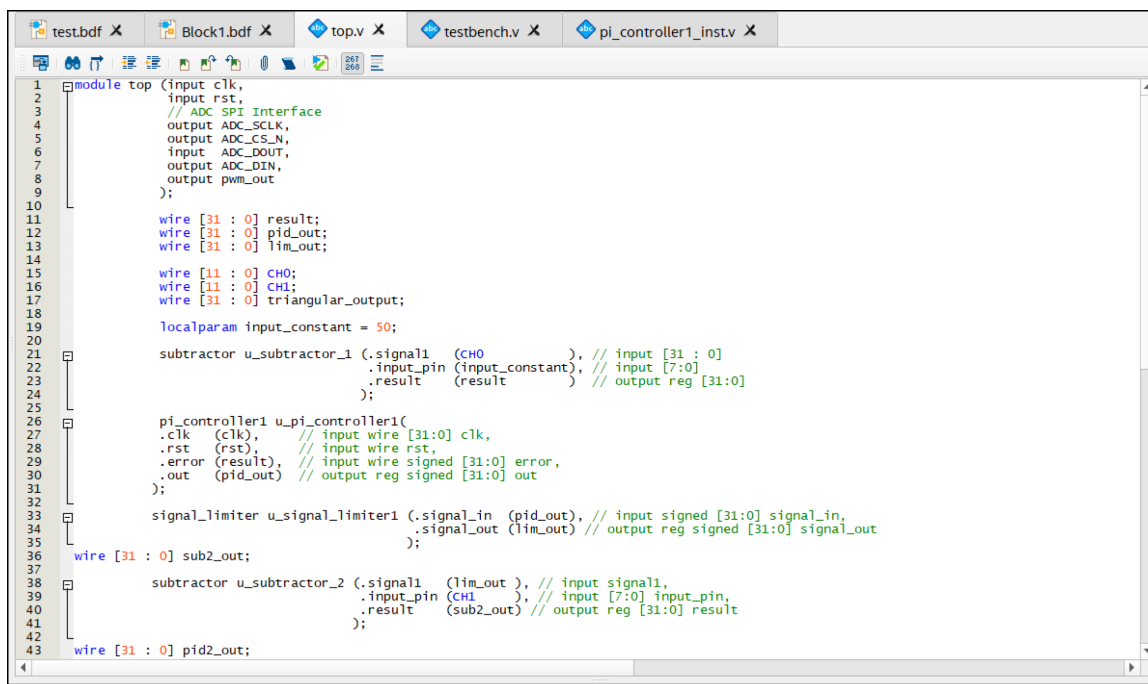
ΚΕΦΑΛΑΙΟ 3: TESTBENCH ΚΑΙ ΑΠΟΤΕΛΕΣΜΑΤΑ

Ένα πολύ σημαντικό βήμα πριν τον προγραμματισμό ενός FPGA είναι το **testbench.**

Το **testbench** είναι ένα εργαλείο που χρησιμοποιείται κατά την ανάπτυξη και τη δοκιμή του FPGA για να εξασφαλίσει ότι ο σχεδιασμός λειτουργεί σύμφωνα με τις προδιαγραφές και τις απαιτήσεις του. Συμβάλλει στην εξοικονόμηση χρόνου και πόρων, καθώς επιτρέπει την εντόπιση και την αντιμετώπιση προβλημάτων πριν από τον φυσικό προγραμματισμό του FPGA.

Το πρόγραμμα που χρησιμοποιήθηκε είναι το **ModelSim** της εταιρίας Siemens.

Για να φτιάξουμε ένα testbench πρέπει να αποτυπώσουμε το κύκλωμα της **εικόνας 17** σε μορφή κώδικα. Αυτό γίνεται πηγαίνοντας σε κάθε στοιχείο ξεχωριστά στο **project navigator** στο αριστερό μέρος του Quartus, πατώντας δεξί κλικ > Create Verilog Instantiation File. Παίρνουμε τον instantiation κώδικα από όλα τα στοιχεία και δηλώνουμε τη συνδεσμολογία μέσα στις παρενθέσεις και έχουμε αυτή τη μορφή (**εικόνα 18**)




```
module top (input clk,
            input rst,
            // ADC SPI Interface
            output ADC_SCLK,
            output ADC_CS_N,
            input  ADC_DOUT,
            output ADC_DIN,
            output pwm_out
           );

           wire [31 : 0] result;
           wire [31 : 0] pid_out;
           wire [31 : 0] lim_out;

           wire [11 : 0] CH0;
           wire [11 : 0] CH1;
           wire [31 : 0] triangular_output;

           localparam input_constant = 50;

           subtractor u_subtractor_1 (.signal1   (CH0           ), // input [31 : 0]
                                      .input_pin (input_constant), // input [7:0]
                                      .result    (result        )  // output reg [31:0]
                                     );

           pi_controller1 u_pi_controller1(
           .clk   (clk),     // input wire [31:0] clk,
           .rst   (rst),     // input wire rst,
           .error (result),  // input wire signed [31:0] error,
           .out   (pid_out)  // output reg signed [31:0] out
          );

          signal_limiter u_signal_limiter1 (.signal_in  (pid_out), // input signed [31:0] signal_in,
                                            .signal_out (lim_out) // output reg signed [31:0] signal_out
                                           );
wire [31 : 0] sub2_out;

          subtractor u_subtractor_2 (.signal1   (lim_out ), // input signal1,
                                     .input_pin (CH1     ), // input [7:0] input_pin,
                                     .result    (sub2_out) // output reg [31:0] result
                                    );

wire [31 : 0] pid2_out;
```

Εικόνα 18 - instatiation file

Αφού σώσουμε το προηγούμενο αρχείο δίνοντας το το όνομα top, δημιουργούμε ένα άλλο Verilog file γράφοντας πάνω την εντολή **`timescale** που προσομοιώνει το ρολόι του συστήματος. Έπειτα διαβάζοντας το manual [1] του ADC προσομοιώνουμε τον τρόπο λειτουργείας του για να μας δίνει κάποιες τυχαίες τιμές στις εισόδους των δυο αφαιρετών (**εικόνα 19**) (**εικόνα 20**).

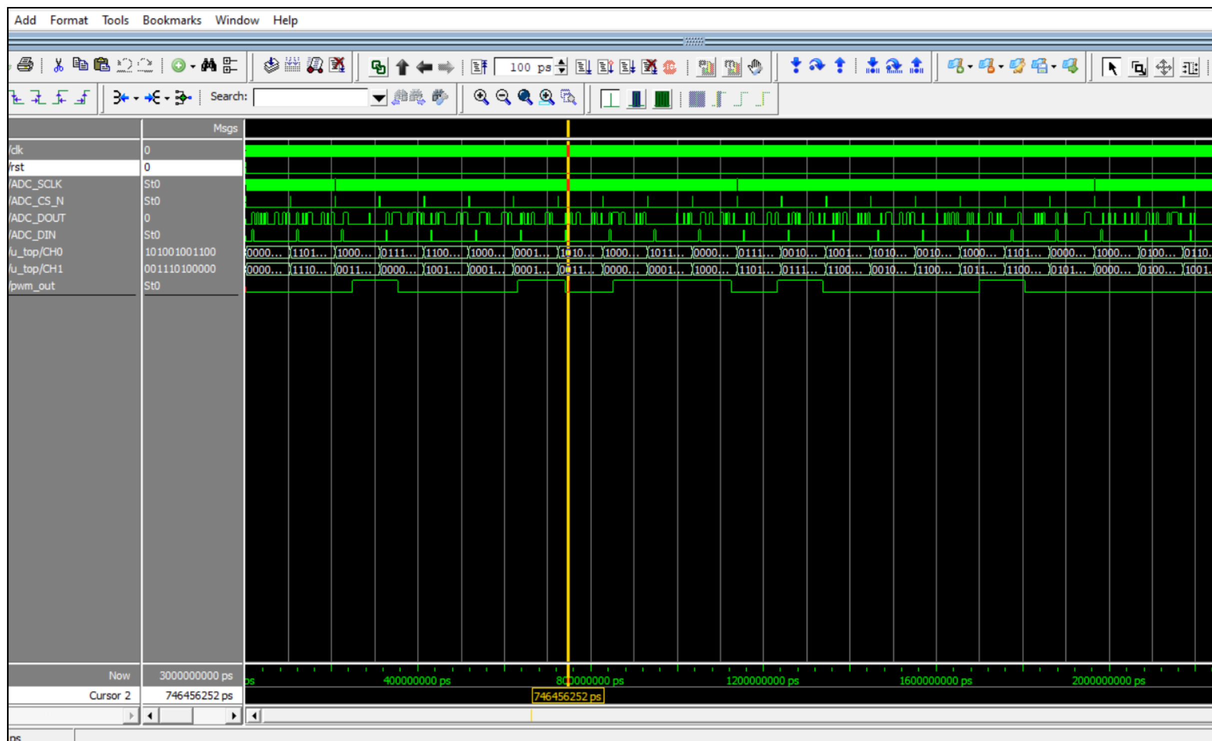
```
1   `timescale 10ns/100ps
2
3   module testbench();
4
5   wire ADC_SCLK;
6   wire ADC_CS_N;
7   reg ADC_DOUT;
8   wire ADC_DIN;
9   wire pwm_out;
10
11  reg clk;
12
13  reg rst;
14
15  integer cycle_cnt;
16
17  initial
18  begin
19     cycle_cnt = 0;
20      clk = 0;
21      rst = 0;
22      #80
23      rst = 1;
24      #80
25      rst = 0;
26      wait (ADC_CS_N == 1'b0);
27      ADC_DOUT = 0;
28     while(cycle_cnt < 100) begin
29       while (cycle_cnt < 4) begin
30         @(negedge ADC_SCLK)
31         cycle_cnt++;
32       end
33       cycle_cnt = 0;
34        while (cycle_cnt < 12) begin
```

Εικόνα 19 – testbench

```
test.bdf    Block1.bdf    top.v    testbench.v    pi_controller1_inst.v

28     while(cycle_cnt < 100) begin
29       while (cycle_cnt < 4) begin
30         @(negedge ADC_SCLK)
31         cycle_cnt++;
32       end
33       cycle_cnt = 0;
34        while (cycle_cnt < 12) begin
35         @(negedge ADC_SCLK)
36        ADC_DOUT = $urandom_range(0, 1);
37         cycle_cnt++;
38       end
39       cycle_cnt = 0;
40        ADC_DOUT = 0;
41     end
42
43  end
44
45  always
46  begin
47  #10 clk = !clk;
48  end
49
50             top u_top(
51             .clk       (clk       ),        // In
52             .rst       (rst       ),        // In
53             .ADC_SCLK  (ADC_SCLK  ),        // Out
54             .ADC_CS_N  (ADC_CS_N  ),        // Out
55             .ADC_DOUT  (ADC_DOUT  ),        // In
56             .ADC_DIN   (ADC_DIN   ),        // Out
57             .pwm_out   (pwm_out   )         // Out
58            );
59
60  endmodule
```

Εικόνα 20 – testbench

Τέλος, αφού ανοίξουμε το ModelSim, περάσουμε όλα τα αρχεία και πατήσουμε Simulate θα εμφανιστούν αυτές οι κυματομορφές (**εικόνα 21**).



Εικόνα 21 - ModelSim Simulation

Παρατηρούμε ότι ανάλογα με τις τυχαίες τιμές που παίρνουν τα CH0 και CH1 η έξοδος PWM_OUT άλλοτε παίρνει τη τιμή 1 και άλλοτε τη τιμή 0 άρα το σύστημα μας φαινομενικά δουλεύει.

ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ

Μελλοντικά μπορεί να εξεταστεί η σύνδεση της κάρτας FPGA με ένα προσομοιωτή πραγματικού χρόνου για την υλοποίηση ενός πειράματος Controller Hardware in the Loop, CHIL (συσκευή ελέγχου σε βρόχο). Ο προσομοιωτής πραγματικού χρόνου θα προσομοιώνει τον μετατροπέα ηλεκτρονικών ισχύος και θα παράγει αναλογικά σήματα τα οποία θα εισάγονται στις αναλογικές εισόδους της κάρτας. Η κάρτα θα τρέχει τον αλγόριθμο ελέγχου που θα έχει υλοποιηθεί σε αυτήν, και θα παράγει ψηφιακά σήματα τα οποία θα εισάγονται στις ψηφιακές εισόδους του προσομοιωτή πραγματικού χρόνου. Με τον τρόπο αυτό, θα υλοποιηθεί ένα πείραμα σε κλειστό βρόχο όπου ο προσομοιωτής πραγματικού χρόνου θα αλληλεπιδρά με την κάρτα.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] *https://www.ti.com/document-viewer/ADC128S022/datasheet*

https://docplayer.net/52158896-Using-the-de0-nano-adc-controller-1-introduction-for-quartus-ii-12-1.html

Implementation of an FPGA-Based Current Control and SVPWM ASIC with Asymmetric Five-Segment Switching Scheme for AC Motor Drives
Department of Electrical Engineering, Minghsin University of Science and Technology, No. 1, Xinxing Rd., Xinfeng, Hsinchu 30401, Taiwan

Multiple Closed Loop System Control with Digital PID Controller Using FPGA
Şirin AKKAYA, Control and Automation Engineering Department, Istanbul Technical University
Onur Akbatı and Haluk Görgün, Control and Automation Engineering Department, Yildiz Technical University
Istanbul, Turkey

ΠΑΡΑΡΤΗΜΑ

Κώδικας Verilog Αφαιρέτη

```
module subtractor(
    input signal1,
    input [7:0] input_pin,
    output reg [31:0] result
);

always @(*) begin
    result = signal1 - input_pin;
end

endmodule
```

Κώδικας Verilog Σταθεράς

```
module constant(
    input [31:0] clk,
    output reg [7:0] result
);

localparam MY_CONST = 50; // Define a constant value of 50 in decimal format

always @(posedge clk[31]) begin
    result <= MY_CONST; // Assign the constant value to the output port
end

endmodule
```

Κώδικας Verilog PI ελεγκτή

```
module pi_controller1(
  input wire [31:0] clk,
  input wire rst,
  input wire signed [31:0] error,
  output reg signed [31:0] out
);

  parameter Kp = 1;
  parameter Ki = 1000;

  reg signed [31:0] integral = 0;
  reg signed [31:0] proportional = 0;

  always @(posedge clk or posedge rst) begin
    if (rst) begin
      integral     <= 0;
      proportional <= 0;
      out          <= 0;
    end else begin
      proportional <= error * Kp;
      integral     <= integral + error * Ki;
      out          <= proportional + integral;
    end
  end

endmodule
```

Κώδικας Verilog Limiter

```
module signal_limiter(input signed [31:0] signal_in,
                     output reg signed [31:0] signal_out
                    );

always @(*) begin
    if (signal_in < 0)
        signal_out = 0;
    else if (signal_in > 1000)
        signal_out = 1000;
    else
        signal_out = signal_in;
end

endmodule
```

Κώδικας Verilog Συγκριτή

```
module comparator (
    input [31:0] A,
    input [31:0] B,
    output reg EQ
);

always @* begin
    EQ = (A > B) ? 1 : 0;
end

endmodule
```

Κώδικας Verilog Τριγωνικού Σήματος

```
module triangular_signal(
  input [31:0] clk,
  output reg signed [31:0] triangular_output
);

  parameter CNT_MAX = 2500; // sets frequency of the signal to 10 kHz
  reg signed [31:0] cnt_reg = 0;
  reg direction = 1'b1;

  always @(posedge clk[31]) begin
    if (cnt_reg == CNT_MAX - 1) begin
      direction <= ~direction;
      cnt_reg <= 0;
    end else begin
      cnt_reg <= cnt_reg + 1;
    end
    if (direction) begin
      triangular_output <= cnt_reg * 2 - CNT_MAX;
    end else begin
      triangular_output <= CNT_MAX - (cnt_reg * 2);
    end
  end
endmodule
```

Ολόκληρο το Κύκλωμά σε μορφή Κώδικα Verilog

```
module top (input clk,
        input rst,
                                    // ADC SPI Interface
                                    output ADC_SCLK,
                                    output ADC_CS_N,
                                    input  ADC_DOUT,
                                    output ADC_DIN,
                                    output pwm_out
        );

                            wire [31 : 0] result;
                            wire [31 : 0] pid_out;
                            wire [31 : 0] lim_out;

                            wire [11 : 0] CH0;
        wire [11 : 0] CH1;
        wire [31 : 0] triangular_output;

                            localparam input_constant = 50;

                            subtractor u_subtractor_1 (.signal1   (CH0          ), // input [31 : 0]
                            .input_pin (input_constant), // input [7:0]
                            .result    (result       )  // output reg [31:0]
                                                                                    );

        pi_controller1 u_pi_controller1(
        .clk   (clk),     // input wire [31:0] clk,
        .rst   (rst),     // input wire rst,
        .error (result),  // input wire signed [31:0] error,
        .out   (pid_out)  // output reg signed [31:0] out
        );
```

```
signal_limiter u_signal_limiter1 (.signal_in   (pid_out), // input signed
[31:0] signal_in,
                .signal_out (lim_out) // output reg signed [31:0] signal_out
               );
wire [31 : 0] sub2_out;

subtractor u_subtractor_2 (.signal1   (lim_out ), // input signal1,
               .input_pin (CH1     ), // input [7:0] input_pin,
               .result    (sub2_out) // output reg [31:0] result
              );

wire [31 : 0] pid2_out;

        pi_controller1 u_pi_controller2(
 .clk   (clk    ),   // input wire [31:0] clk,
 .rst   (rst    ),   // input wire rst,
 .error (sub2_out ), // input wire signed [31:0] error,
 .out   (pid2_out)   // output reg signed [31:0] out
);

wire [31 : 0] lim2_out;

signal_limiter u_signal_limiter2 (.signal_in  (pid2_out), // input signed
[31:0] signal_in,
                .signal_out (lim2_out) // output reg signed [31:0] signal_out
               );

comparator u_comparator (.A  (lim2_out        ),  // input [31:0] A,
               .B  (triangular_output),  // input [31:0] B,
               .EQ (pwm_out          )   // output reg EQ
              );
```

```
triangular_signal u_triangular_signal (.clk (clk), //
input [31:0] clk,
.triangular_output (triangular_output) // output reg signed
[31:0] triangular_output
);
adc_control u_adc_control(.CLOCK (clk), // input wire CLOCK, // clk.clk
.ADC_SCLK (ADC_SCLK), // output wire
ADC_SCLK, // external_interface.SCLK
.ADC_CS_N (ADC_CS_N), // output wire
ADC_CS_N, // .CS_N
.ADC_DOUT (ADC_DOUT), // input wire
ADC_DOUT, // .DOUT
.ADC_DIN (ADC_DIN ), // output wire ADC_DIN, //
.DIN
.CH0 (CH0 ), // output wire [11:0] CH0, //
readings.CH0
.CH1 (CH1 ), // output wire [11:0] CH1, //
.CH1
.CH2 (), // output wire [11:0] CH2, //
.CH2
.CH3 (), // output wire [11:0] CH3, //
.CH3
.CH4 (), // output wire [11:0] CH4, //
.CH4
.CH5 (), // output wire [11:0] CH5, //
.CH5
.CH6 (), // output wire [11:0] CH6, //
.CH6
.CH7 (), // output wire [11:0] CH7, //
.CH7
.RESET (rst ) //input wire RESET //
reset.reset
);
Endmodule
```

Testbench

```
`timescale 10ns/100ps

module testbench();

wire ADC_SCLK;
wire ADC_CS_N;
reg ADC_DOUT;
wire ADC_DIN;
wire pwm_out;

reg clk;

reg rst;

integer cycle_cnt;

initial
begin
        cycle_cnt = 0;
  clk = 0;
  rst = 0;
  #80
  rst = 1;
  #80
  rst = 0;
  wait (ADC_CS_N == 1'b0);
  ADC_DOUT = 0;
        while(cycle_cnt < 100) begin
         while (cycle_cnt < 4) begin
          @(negedge ADC_SCLK)
          cycle_cnt++;
         end
```

```
            cycle_cnt = 0;
      while (cycle_cnt < 12) begin
              @(negedge ADC_SCLK)
                     ADC_DOUT = $urandom_range(0, 1);
              cycle_cnt++;
            end
            cycle_cnt = 0;
      ADC_DOUT = 0;
          end

end

always
begin
#10 clk = !clk;
end

          top u_top(
                        .clk        (clk      ),      // In
                        .rst        (rst      ),      // In
                        .ADC_SCLK  (ADC_SCLK ),       // Out
                        .ADC_CS_N  (ADC_CS_N ),       // Out
                        .ADC_DOUT  (ADC_DOUT ),       // In
                        .ADC_DIN   (ADC_DIN  ),       // Out
                        .pwm_out   (pwm_out  )        // Out
                       );

endmodule
```