ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Μελέτη και ανάπτυξη σύγχρονων μεθόδων σχεδίασης με FPGA

Φοιτητής: Ταουλάντ Θ. Ντόνη

Εισηγητές: Δρ. Αναστασία Βελώνη ,Καθηγήτρια
Ιωάννης Ντουμπάκης,Καθηγητής

ΑΘΗΝΑ, ΙΑΝΟΥΑΡΙΟΣ 2020

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Μελέτη και ανάπτυξη σύγχρονων μεθόδων σχεδίασης με FPGA

Ταουλάντ Θ. Ντόνη
Α.Μ. 41786

Εισηγητής: Δρ. Αναστασία Βελώνη ,Καθηγήτρια
Ιωάννης Ντουμπάκης,Καθηγητής

Ημερομηνία εξέτασης / /2020

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω πρώτα από όλα την καθηγήτρια μου, Αναστασία Βελώνη και τον καθηγητή Ιωάννη Ντουμπάκη για την βοήθεια στην πραγματοποίηση αυτής της πτυχιακής εργασίας. Επίσης, θα ήθελα να ευχαριστήσω όλους τους υπόλοιπους καθηγητές του τμήματός μου για την αφοσίωση τους όπως και τους συμφοιτητές μου για την άψογη συνεργασία κατά την διάρκεια των σπουδών μου. Ένα μεγάλο ευχαριστώ και στην οικογένειά μου για την στήριξη και την υπομονή τους όλα αυτά τα χρόνια.

ΠΕΡΙΛΗΨΗ

Στα πλαίσια της πτυχιακής εργασίας θα μελετηθούν τρόποι σχεδίασης σύγχρονων και σύνθετων ψηφιακών συστημάτων με τις γλώσσες περιγραφής υλικού και τα εργαλεία τα οποία χρησιμοποιούνται για την ανάπτυξη αυτών τον συστημάτων όπως η γλώσσα VHDL το λογισμικό Quartus II και η αναπτυξιακές πλακέτες FPGA για την υλοποίηση κυκλωμάτων.

ΕΠΙΣΤΗΜΟΝΙΚΗ ΠΕΡΙΟΧΉ: Προγραμματισμός,Ψηφιακή Σχεδίαση

ΛΕΞΕΙΣ ΚΛΕΙΔΙΆ: FPGA, VHDL, Verilog, Quartus II, CPLD.

ABSTRACT

In this present thesis is presented a general research and development of modern design methods of complex digital systems with hardware description languages (HDL) and the tools we use to design these systems such as Quartus 2 design suite, hardware description language VHDL ,Verilog and FPGAs development boards.

SCIENTIFIC AREA: Programming, Digital Design

KEY WORDS: FPGA, VHDL, Verilog, Quartus II, CPLD.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

ΣΥΝΤΟΜΟΓΡΑΦΊΕΣ

PLD	**Programmable Logic Devices**
CPLD	**Complex Programmable Logic Devices**
JTAG	**JoinTestActionGroup**
PAL	**Programmable Array Logic**
RTL	**Register transfer level**
FPGA	**Field Programmable Gate Arrays**
VHDL	**VHSIC Hardware Description Language**
TCL	**ToolCommandLanguage**

ΕΙΣΑΓΩΓΉ

Σε αυτό το κεφάλαιο αναλύεται το αντικείμενο της πτυχιακής εργασίας, το οποίο είναι μελέτη και ανάπτυξη σύγχρονων μεθόδων σχεδίασης με FPGA. Επιπλέον, γίνεται μία αναδρομή γύρω από τις μεθόδους που έχουν αναπτυχθεί με σκοπό να υλοποιηθεί η πτυχιακή εργασία.

ΠΕΡΙΓΡΑΦΉ ΤΟΥ ΑΝΤΙΚΕΙΜΈΝΟΥ

Το αντικείμενο της πτυχιακής εργασίας είναι η μελέτη και η ανάπτυξη σύγχρονων μεθόδων σχεδίασης ψηφιακών ηλεκτρονικών κυκλωμάτων με τα πιο σύγχρονα εργαλεία όπως οι γλώσσες περιγραφής υλικού και οι πλατφόρμες σχεδίασης και προσομοίωσης ηλεκτρονικών κυκλωμάτων.

ΟΡΓΆΝΩΣΗ ΚΕΊΜΕΝΟΥ

Στην ενότητα αυτή θα γίνει η ανάλυση και ο τρόπος δομής και σύνταξης της εργασίας.

Αρχικά, περιλαμβάνονται οι ευχαριστίες και η περίληψη, περιέχοντας και τη μετάφραση στα αγγλικά. Επιπλέον, περιλαμβάνονται τα περιεχόμενα, ο κατάλογος εικόνων, ο κατάλογος πινάκων, καθώς και οι συντομογραφίες.

Στο πρώτο κεφάλαιο, παρουσιάζεται γενικά η έννοια του προγραμματισμού ηλεκτρονικών διατάξεων και στο δεύτερο κεφάλαιο των γλωσσών περιγραφής υλικού.

Το κεφάλαιο 3 αφορά μία σύντομη εισαγωγή στην γλώσσα VHDL και την βασική δομή ,τα χαρακτηριστικά και τον τρόπο σύνταξής της. Στο τέταρτο κεφάλαιο γίνεται μία σύγκριση με την άλλη γλώσσα περιγραφής υλικού Verilog.

Στο κεφάλαιο 5 γίνεται μια αναφορά στην πλατφόρμα σχεδίασης Quartus2 ως βασικό εργαλείο σχεδίασης σύνθετων ψηφιακών κυκλωμάτων και κάποιον βασικών λειτουργιών του ενώ στο αμέσως επόμενο κεφάλαιο γίνεται και μια μικρή παρουσίαση στην άλλη γνώστη πλατφόρμα σχεδίασης Vivado της Xilinx.

Στο έβδομο κεφάλαιο αναπτύσσεται μία ολοκληρωμένη εφαρμογή σε γλώσσα VHDL και υλοποιείτε σε αναπτυξιακή πλακέτα της Altera και με FPGA της οικογένειας Cyclone IV.

Τέλος,παρατίθενται τα συμπεράσματα της πτυχιακής εργασίας και η βιβλιογραφία.

ΚΕΦΑΛΑΙΟ 1
ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΟΛΟΚΛΗΡΩΜΕΝΩΝ ΚΥΚΛΩΜΑΤΩΝ

1.1. Τα πρώτα προγραμματιζόμενα ολοκληρωμένα κυκλώματα

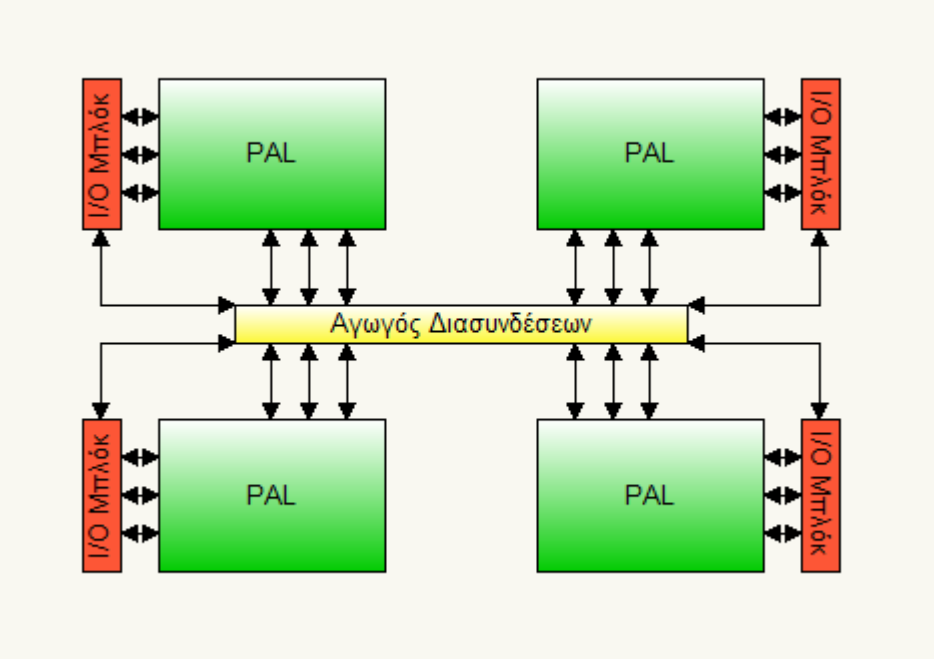
Στις αρχές της δεκαετίας το 70’ πρωτοεμφανίστηκαν τα προγραμματιζόμενα ολοκληρωμένα κυκλώματα που σε σύγκριση με μέχρι τότε ολοκληρωμένα δεν είχαν μια προσδιορισμένη λειτουργία από την εταιρία παραγωγής αλλά ο χρήστης ήταν αυτός που καθόριζε την λειτουργία του ολοκληρωμένου κυκλώματος σύμφωνα με τις ανάγκες του συστήματος που ήθελε να δημιουργήσει.

Οι Προγραμματιζόμενες Λογικές Διατάξεις (PLD) έχουν ως αρχή λειτουργίας την οργάνωση των πυλών σε πίνακες με αποτέλεσμα να εκτελούνται διάφορες λογικές συναρτήσεις .Τα PLD είναι οργανωμένα σε μπλοκ από πύλες και προγραμματιζόμενους διακόπτες που ανάλογα με τον συνδυασμό τους καθορίζεται και η λειτουργία του ολοκληρωμένου κυκλώματος. Οι συνδέσεις μεταξύ των μπλοκ μπορούν να αλλάξουν άρα γίνετε επαναπρογραμματισμός του κυκλώματος. Αυτή είναι και η μεγάλη διαφορά από τα προγραμματισμένα ολοκληρωμένα κυκλώματα και μεγάλο πλεονέκτημα επειδή δίνει την δυνατότητα στον χρήστη να αλλάξει την λειτουργιά του συστήματος όποτε χρειαστεί.

Σημαντικό μειονέκτημα είναι οι ταχύτητα εκτέλεσης λογικών πράξεων λόγο της μεγάλης χωρητικότητας που έχουν οι λογικοί διακόπτες που βρίσκονται στο ολοκληρωμένο.

1.2. Σύνθετα ολοκληρωμένα κυκλώματα

Με το πέρασμα των χρόνων οι απαιτήσεις αυξήθηκαν και η ανάγκη για πιο σύνθετα και περίπλοκα κυκλώματα οδήγησε στην δημιουργία των Σύνθετων Προγραμματιζόμενων Λογικών Διατάξεων (CPLD). Τα CPLD είναι σύνθετα κυκλώματα που σε σχέση με τα PLD διαθέτουν πολλά λογικά μπλοκ ,ουσιαστικά είναι πολλά PLD μαζί σε ένα μοναδικό ολοκληρωμένο κύκλωμα που συνδέονται μεταξύ τους και διαθέτουν πολλούς εισόδους και πολλούς εξόδους.



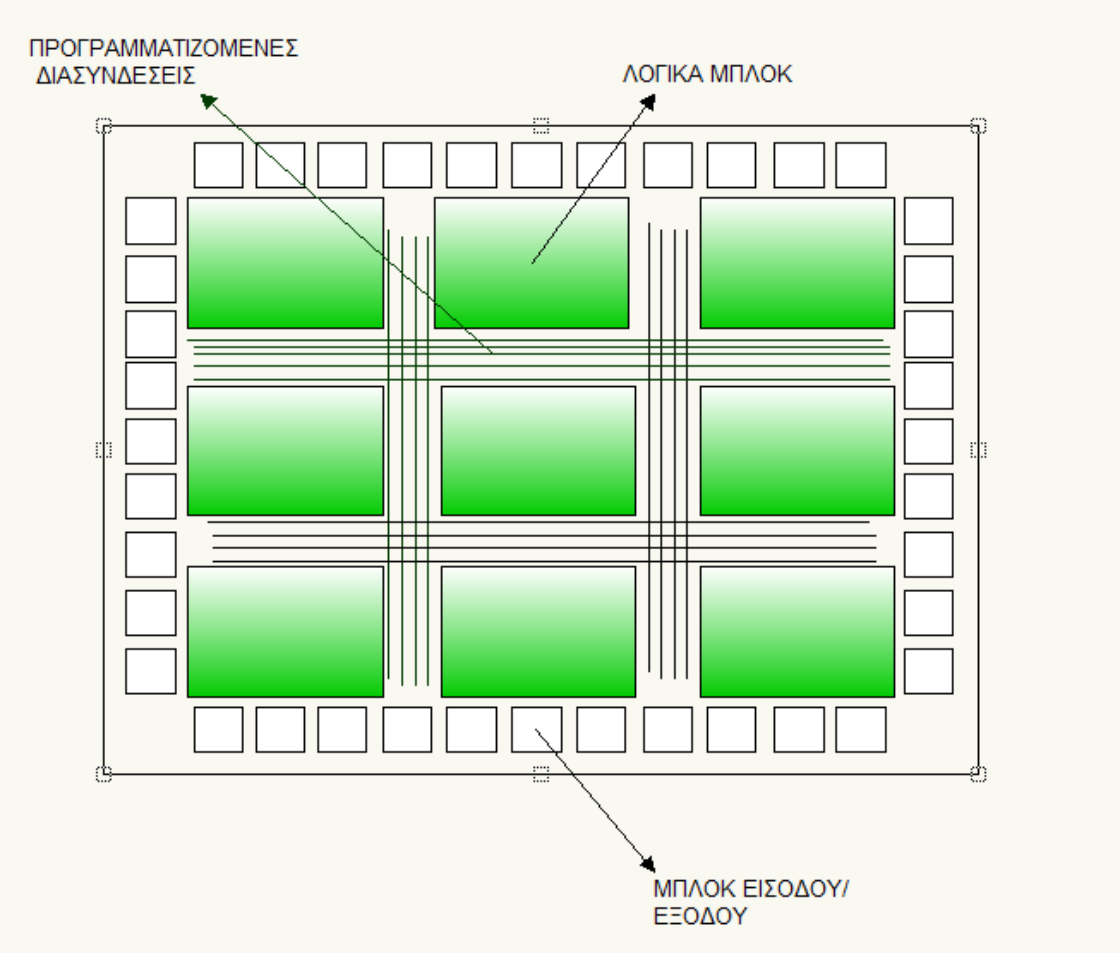
Εικόνα 1.1 : Δομή Διάταξης Πολύπλοκης Προγραμματιζόμενης Λογικής (CPLD)

Ο αριθμός των λογικών μπλοκ δεν είναι σταθερός στα CPLD και από αυτό εξαρτάτε και το φυσικό μέγεθος του ολοκληρωμένου .Στο κέντρο υπάρχει ο αγωγός διασύνδεσης των μπλοκ ενώ δεξιά και αριστερά υπάρχουν οι είσοδοι - έξοδοι. Ο προγραμματισμός τους γίνεται μέσω υπολογιστή και με ενσύρματο τρόπο με το σύστημα ISP (In System Programming) δηλαδή χωρίς να αφαιρεθεί από την πλακέτα το ολοκληρωμένο κύκλωμα εφόσον αυτή διαθέτει θύρα JTAG (JoinTestActionGroup) .Το σχεδιαστικό αρχείο μεταφέρεται από τον υπολογιστή

στο ολοκληρωμένο κύκλωμα το οποίο το διατηρεί ακόμα κι αν διακοπεί η τροφοδοσία του.

1.3. Λογικές Διατάξεις Προγραμματιζόμενες στο Πεδίο (FPGA)

Η Λογικές Διατάξεις Προγραμματιζόμενες στο Πεδίο (Field Programable Gate Array) είναι ολοκληρωμένα κυκλώματα τα οποία διαθέτουν ένα πάρα πολύ μεγάλο αριθμό από πύλες ,λογικά μπλοκ,,σειριακοί δίαυλοι οι οποίοι είναι ασύνδετοι μεταξύ τους όπως και στα CPLD ο προγραμματισμός γίνεται στην πλακέτα για αυτό τον λόγο λέγονται προγραμματιζόμενες στο πεδίο .



Εικόνα 1.2 : Λογικές Διατάξεις Προγραμματιζόμενες στο Πεδίο (FPGA)

Τα FPGA σε σχέση με τα CPLD δεν διατηρούν τον προγραμματισμό τους αν διακοπεί η τροφοδοσία τους. Αυτό συμβαίνει επειδή τα FPGA διαθέτουν μνήμη ROM η οποία μηδενίζεται κάθε φορά που διακόπτετε η τροφοδοσία του κυκλώματος ,έχουν μεγαλύτερη κατανάλωση ρεύματος σε σχέση με τα CPLD. Η χρήση των FPGA είναι ιδανική για σύνθετα ψηφιακά συστήματα σε σχέση με τα CPLD που χρησιμοποιούνται σε πιο απλές εφαρμογές. Τα FPGA μπορούν να προγραμματιστούν εν λειτουργία ,δηλαδή όταν αυτό τρέχει σε αντίθεση με τα CPLD που θα πρέπει να γίνει επανεκκίνηση τους.

ΚΕΦΑΛΑΙΟ 2
ΓΛΩΣΣΕΣ ΠΕΡΙΓΡΑΦΗΣ ΥΛΙΚΟΥ

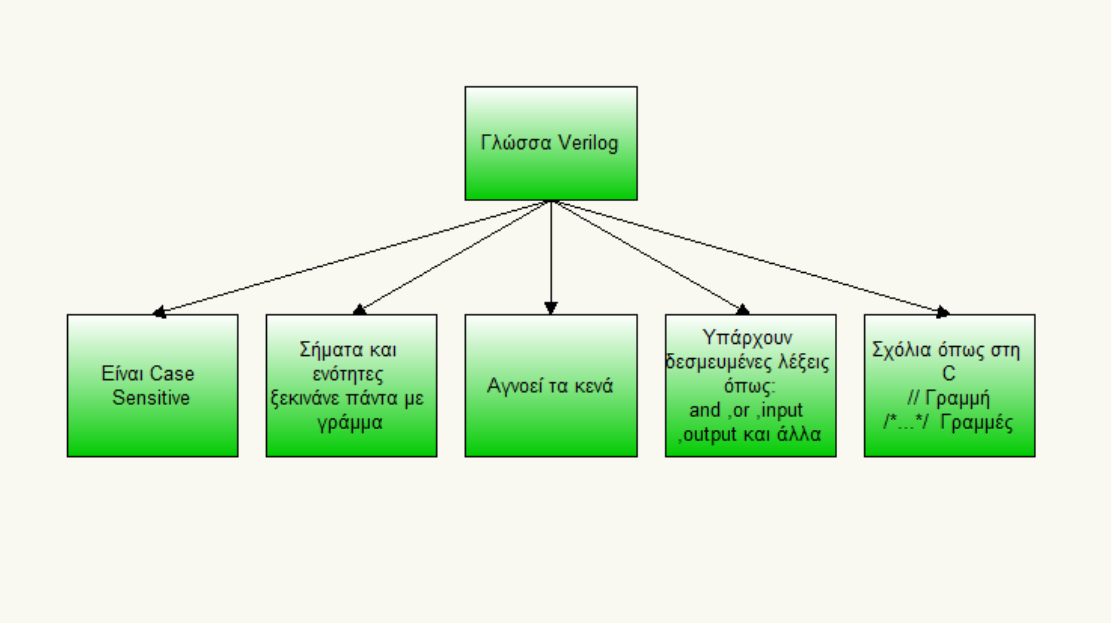
2.1 Εισαγωγή

Στην επιστήμη των υπολογιστών μια γλώσσα περιγραφής υλικού (Hardware Description Language) είναι η γλώσσα προγραμματισμού που περιγράφει την συμπεριφορά ενός ψηφιακού κυκλώματος.,την οργάνωση ,την λειτουργία και την σχεδίαση του κυκλώματος .Διαφέρει από της γλώσσες προγραμματισμού λογισμικού αν και συντακτικά μοιάζουν με κάποιες από αυτές .

Οι γλώσσες περιγραφής υλικού έχουν πολλά πλεονεκτήματα σε σχέση με την παραδοσιακή σχεδίαση ,ένα βασικό είναι η επαλήθευση του κυκλώματος .Οι γλώσσες περιγραφής υλικού δίνουν την δυνατότητα στον χρήστη να ελέγχει και να κάνει διορθώσεις στο κύκλωμά του από τον κώδικα.

Η συμπεριφορά των ψηφιακών συστημάτων στις γλώσσες περιγραφής υλικού εκφράζεται σε μορφή κειμένου όπως και στις γλώσσες προγραμματισμού λογισμικού συμπεριλαμβάνοντας και τον χρόνο ως βασική παράμετρος. Τα κείμενα η εκφράσεις περιγράφουν την συνδεσμολογία του κυκλώματος. Με την βοήθεια λογισμικών σχεδίασης ψηφιακών συστημάτων τα οποία υποστηρίζουν γλώσσες περιγραφής υλικού μπορεί ο χρήστης να περιγράψει ένα κύκλωμα σε μορφή κειμένου με γλώσσες περιγραφής υλικού και να το μοντελοποιήσει αφού έχει κάνει προσομοίωση του κυκλώματος. Οι γλώσσες περιγραφής υλικού κατατάσσονται στην κατηγορία των γλωσσών προδιαγραφών η μοντελοποίησης για τον λόγο που αναφέραμε πιο πάνω. Οι γλώσσες περιγραφής υλικού μπορούν να περιγράψουν ψηφιακά συστήματα η και αναλογικά και να μοντελοποιηθούν με την βοήθεια λογισμικών εργαλείων σχεδίασης.

Η λειτουργία ενός ψηφιακού συστήματος μπορεί να περιγραφεί και με κάποιες γλώσσες προγραμματισμού λογισμικού αλλά σε σχέση με της γλώσσες περιγραφής υλικού έχουν μεγάλα μειονεκτήματα τα οποία την καθιστούν όχι κατάλληλη για τον προγραμματισμό τέτοιον συστημάτων. Το βασικό μειονέκτημα



Εικόνα 2.1 : Μία γλώσσα περιγραφής υλικού είναι μία γλώσσα για Η/Υ που περιγράφει ένα σύστημα σε μορφή κειμένου.

τους είναι ότι οι γλώσσες λογισμικού δεν διαθέτουν εκφράσεις χρόνου το οποίο είναι το πιο σημαντικό στοιχείο ενός ψηφιακού κυκλώματος και επίσης θα πρέπει να συνδυαστούν με διάφορες βιβλιοθήκες. Η γλώσσα C++ σε συνδυασμό με κάποιες κατάλληλες βιβλιοθήκες και έναν λογικό προσομοιωτή ήταν από τους πρώτους τρόπους προσομοίωσης ενός κυκλώματος.

2.2 Ιστορική Αναδρομή

Η τεχνολογία γενικά και η επιστήμη των υπολογιστών σημείωσαν ραγδαία εξέλιξη ειδικά μετά τον Β' Παγκόσμιο Πόλεμο και σε συνδυασμό με την εφεύρεση των τρανζίστορ μερικά χρόνια πιο πριν. Τα τρανζίστορ ίσως να είναι η μεγαλύτερη εφεύρεση του 20ού αιώνα καθώς είναι το βασικό στοιχείο σε ένα ηλεκτρονικό κύκλωμα.

Με το πέρασμα των χρόνων τα κυκλώματα γινόντουσαν περίπλοκα επειδή η ανάγκες μεγάλωναν ως συνεπεία η σχεδίαση των κυκλωμάτων να γίνει μια

δύσκολη και χρονοβόρα διαδικασία. Τα πρώτα ολοκληρωμένα κυκλώματα (Integrated Circuits) δημιουργήθηκαν την δεκαετία του 60’ και περιείχαν πολλά τρανζίστορ συνδεδεμένα μεταξύ τους σε μία μόνο κατασκευή πυριτίου λίγων εκατοστών .

Το μέγεθος των ολοκληρωμένων κυκλωμάτων μίκραινε και ο αριθμός των τρανζίστορ που θα μπορούσαν να τοποθετηθούν σε ένα ολοκληρωμένο μεγάλωνε με το πέρασμα των χρόνων ,αυτό διατυπώνεται και στον νόμο του Μουρ (Moore’s Law) ο οποίος έλεγε ότι τουλάχιστον για μια δεκαετία (‘65-’75) ο αριθμός τον τρανζίστορ μέσα σε ένα ολοκληρωμένο θα διπλασιάζεται κάθε χρόνο. Δέκα χρόνια αργότερα ,παρατηρώντας όλα τα δεδομένα ξανά άλλαξε τον χρόνο που απαιτείται για να διπλασιαστούν τα τρανζίστορ σε ένα ολοκληρωμένο στα δύο χρόνια .Ουσιαστικά ο Νόμος του Μουρ περιγράφει τον ρυθμό της εξέλιξης των ολοκληρωμένων κυκλωμάτων αυτό συνεπάγεται ότι με την αύξηση των τρανζίστορ σε ένα ολοκληρωμένο αυξάνεται και η δυσκολία σχεδίασης ενός κυκλώματος για αυτό τον λόγο έπρεπε να βρεθούν τρόποι για αυτοματοποιημένα συστήματα σχεδίασης τους.

Η πρώτη τεχνολογία που χρησιμοποιήθηκε για την δημιουργία περίπλοκων ολοκληρωμένων κυκλωμάτων ήταν η VLSI (Very Large-Scale Integration) ενώ παράλληλα γινόντουσαν προσπάθειες για άλλους τρόπους σχεδίασης όπως οι γλώσσες περιγραφής υλικού .

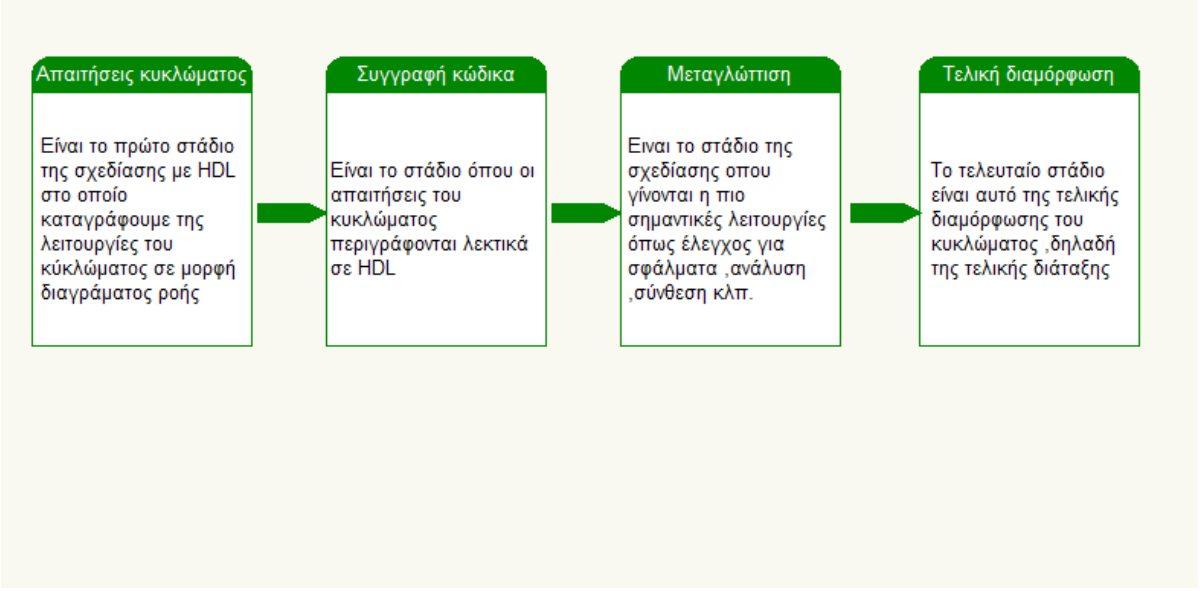
Η δεκαετία του 80’ είναι η δεκαετία στην οποία έγιναν σημαντικά βήματα στη σχεδίαση με γλώσσες περιγραφής υλικού καθώς τότε δημιουργήθηκαν η περισσότερες γλώσσες περιγραφής υλικού. Το 1983 αναπτύσσεται από το υπουργείο άμυνας των ΗΠΑ η γλώσσα περιγραφής VHDL (VHSIC Hardware Description Language) για την περιγραφή συμπεριφοράς κυκλωμάτων ASIC (Application Specific Integrated Circuits) ενώ ένα χρόνο αργότερα πρωτοεμφανίζεται η γλώσσα περιγραφής υλικού Verilog από τους δημιουργούς Prabhu Goel και Phil Moorby .

Η Verilog και η VHDL είναι η κυρίαρχες γλώσσες περιγραφής υλικού μέχρι σήμερα , η Verilog κατέχει την πρωτιά .Και οι δύο γλώσσες έχουν ως κοινό

χαρακτηριστικό την λεκτική περιγραφή της λειτουργίας ενός συστήματος ενώ οι διαφορές τους είναι στους τρόπους περιγραφής, τις δομές που χρησιμοποιούν και διάφορα άλλα στοιχεία.

2.3 Περιγραφή ψηφιακών συστημάτων

Οι HDL γλώσσες προσφέρουν μεγάλη αποδοτικότητα για αυτό και είναι ο βασικός τρόπος σχεδίασης ψηφιακών συστημάτων. Για να σχεδιάσουμε ένα ψηφιακό σύστημα θα πρέπει πρώτα να γνωρίζουμε τις απαιτήσεις του δηλαδή τι ακριβώς θέλουμε να κάνει το σύστημα που θέλουμε να σχεδιάσουμε αυτό μπορούμε να το παρουσιάσουμε ως ένα διάγραμμα ροής .Το επόμενο βήμα η επιλογή της γλώσσας περιγραφής υλικού θα χρησιμοποιήσουμε για την σχεδίαση το κυκλώματος ,δεν υπάρχει κάποια (μεταξύ Verilog και VHDL) γλώσσα περιγραφής υλικού που θεωρείται καλή η κακή γλώσσα ,έχουν διαφορετικά χαρακτηριστικά και για αυτό τον λόγο είναι στο χέρι του σχεδιαστή για να επιλέξει ποια γλώσσα θέλει να χρησιμοποιήσει.



Εικόνα 2.2 :Βασικά βήματα σχεδίασης με HDL

Στην συνέχεια ακολουθεί η συγγραφή του κώδικα το οποίο εξαρτάτε από την πολυπλοκότητα του κυκλώματος και την γλώσσα περιγραφής υλικού χρησιμοποιούμε καθώς έχουν διαφορετικές δομές και διαφορετικό στυλ συγγραφής.

Σε περίπλοκα κυκλώματα οι γλώσσες αυτές χρησιμοποιούν έτοιμες δομές από βιβλιοθήκες που διαθέτουν. Οι συγγραφή του κώδικα γίνεται σε λογισμικά μεταγλώττισης όπου ελέγχεται επίσης ο κώδικας για τυχόν λάθη ,ο έλεγχος γίνεται όταν μεταγλωττίζουμε τον κώδικα ,σε περίπτωση που υπάρχουν λάθη ο μεταγλωττιστής μας ειδοποιεί για σφάλμα στον κώδικα και μια μικρή περιγραφή του σφάλματος.

Το τελευταίο στάδιο της σχεδίασης πριν τελική τοποθέτηση των λογικών στοιχείων σε φυσικό επίπεδο με γλώσσα περιγραφής υλικού είναι αυτό της σύνθεσης όπου γίνεται η αντιστοίχιση της λεκτικής περιγραφής του κώδικα με την λογική διάταξη του κυκλώματος. Ανάλογα την τεχνολογία που προορίζεται το κύκλωμα γίνεται και η αντιστοιχία αυτή καθώς χρησιμοποιούν διαφορετικές διατάξεις από πύλες και διαφορετική αρχιτεκτονικής.

2.4 Πλεονεκτήματα σχεδίασης με γλώσσες περιγραφής υλικού

Οι γλώσσες περιγραφής υλικού και γενικά όλα τα εργαλεία σχεδίασης ψηφιακών συστημάτων όπως εφαρμογές σχεδίασης ,προσομοίωσης και λογικές διατάξεις έχουν εξελιχθεί πάρα πολύ σε σχέση με περασμένες δεκαετίες καθιστώντας τον βασικό τρόπο σχεδίασης ψηφιακών κυκλωμάτων. Η εξέλιξη αυτή οφείλεται κατά κύριο λόγο στην δημιουργία νέων τεχνολογιών και στην εξέλιξη των υπολογιστών .

Η σύγχρονη σχεδίαση γενικά βασίζεται στην σχεδίαση με την βοήθεια υπολογιστών και τον διάφορων εφαρμογών σχεδίασης που κυκλοφορούν CAD (Computer Aided Design). Οι εφαρμογές η οι πλατφόρμες σχεδίασης ψηφιακών συστημάτων που υπάρχουν στην βιομηχανία αυτοματοποίησης ηλεκτρονικών σχεδιάσεων (Electronic design automation, EDA), διαθέτουν πολλά χρήσιμα

εργαλεία όπως η δυνατότητα προσομοίωσης του σχεδίου .Η αυτοματοποιημένη ηλεκτρονική σχεδίαση προσφέρει πολλά πλεονεκτήματα όπως:

- Η σωστή διαχείριση της πολυπλοκότητας της κάθε σχεδίασης.
- Επιτρέπει στον σχεδιαστή να μοντελοποιεί την απόδοση του ψηφιακού συστήματος.
- Δίνει την δυνατότητα στον χρήστη να τοποθετήσει και να δρομολογήσει εκατομμύρια τρανζίστορ σε ένα ολοκληρωμένο κύκλωμα.
- Απαιτείται λιγότερος χρόνος για να σχεδιαστεί ένα ολοκληρωμένο κύκλωμα.
- Σχεδίαση σύμφωνα με τους κανόνες σχεδίασης ολοκληρωμένων κυκλωμάτων.

Η προσομοίωση είναι σημαντική καθώς λειτουργεί ως ένα εργαλείο επαλήθευσης του σχεδίου και σε περιπτώσεις που η προσομοίωση μας δίνει διαφορετικά αποτελέσματα από της αρχικές απαιτήσεις υπάρχει δυνατότητα διόρθωσης . Στην σχεδίαση ενός συστήματος η επαλήθευση ήταν πάντα η πιο δύσκολη και χρονοβόρα διαδικασία επειδή το κύκλωμα που σχεδιάζουμε συνήθως χρειάζεται διόρθωση . Ένα κύκλωμα μπορεί να δοκιμαστεί αν λειτουργεί σύμφωνα με της αρχικές απαιτήσεις σε πραγματική προγραμματιζόμενη λογική διάταξη (PLD-CPLD) η σε FPGA.

ΚΕΦΑΛΑΙΟ 3
ΕΙΣΑΓΩΓΗ ΣΤΗ ΓΛΩΣΣΑ VHDL

3.1. Εισαγωγή

Η VHDL (Very High Speed Integrated Circuits Hardware Description Language) είναι μια γλώσσα προγραμματισμού υλικού η πιο σωστά ,είναι μια γλώσσα με την οποία περιγράφονται ψηφιακά συστήματα με σκοπό την δημιουργία η την αυτοματοποιημένη σχεδίαση σύνθετων ψηφιακών συστημάτων. Η VHDL είναι η πιο γνωστή γλώσσα περιγραφής υλικού και μια από τις πρώτες που κυκλοφορήσαν .

Η διαφορά μιας γλώσσας περιγραφής υλικού και μιας γλώσσας προγραμματισμού είναι στο ότι η γλώσσες περιγραφής υλικού έχουν δημιουργηθεί για να περιγράφουν την συμπεριφορά ενός κυκλώματος. Με την συμπεριφορά του κυκλώματος εννοούμε όλες της λειτουργίες ενός ηλεκτρονικού κυκλώματος δηλαδή όλα τα σήματα είτε εισόδου είτε εξόδου και το τι ακριβός θέλουμε να κάνει το κύκλωμα συμπεριλαμβάνοντας τον χρόνο ως βασικό παράγοντα στην πραγματική περιγραφή και στην συνέχεια την προσομοίωση του ψηφιακού συστήματος .

Η VHDL αναπτύχθηκε από το Υπουργείο Άμυνας των ΗΠΑ αρχικά για εναλλακτική λύση στην περίπλοκη περιγραφή των τότε ψηφιακών κυκλωμάτων και σταδιακά γεννήθηκε η ιδέα της προσομοίωσης συμπεριφοράς των συστημάτων αυτών και στην συνέχεια την σύνθεση όλων των ψηφιακών στοιχείων σε φυσικό επίπεδο υλοποίησης του κυκλώματος .

Η VHDL βασίζεται στην γλώσσα προγραμματισμού λογισμικού Ada επειδή το Υπουργείο Άμυνας των ΗΠΑ χρησιμοποίησε τεχνικές που είχε ήδη χρησιμοποιήσει στην ανάπτυξη της γλώσσας Ada, για αυτό τον λόγο μοιάζουν συντακτικά .Έγινε πρώτη φορά πρότυπη το 1987 ως IEEE 1076-1987 και στην συνέχεια με τις ανανεωμένες εκδόσεις IEEE 1076-1993, IEEE 1076-2002 και IEEE 1076-2008. Η VHDL και γενικά η γλώσσες περιγραφής υλικού είναι στην

ουσία το βασικό εργαλείο των λογισμικών σχεδίασης ψηφιακών συστημάτων καθώς όσο μεγαλώνουν οι απαιτήσεις στην σχεδίαση τέτοιων συστημάτων τόσο αναγκαίο γίνεται και η χρήση τους στα ψηφιακά συστήματα όπως είναι τα Ολοκληρωμένα Κυκλώματα Ειδικών Εφαρμογών ASIC (Application Specific Integrated Circuit) η Επαναπρογραμματιζόμενες Διατάξεις όπως είναι τα CPLD και FPGA.

3.2 Σχεδίαση με VHDL

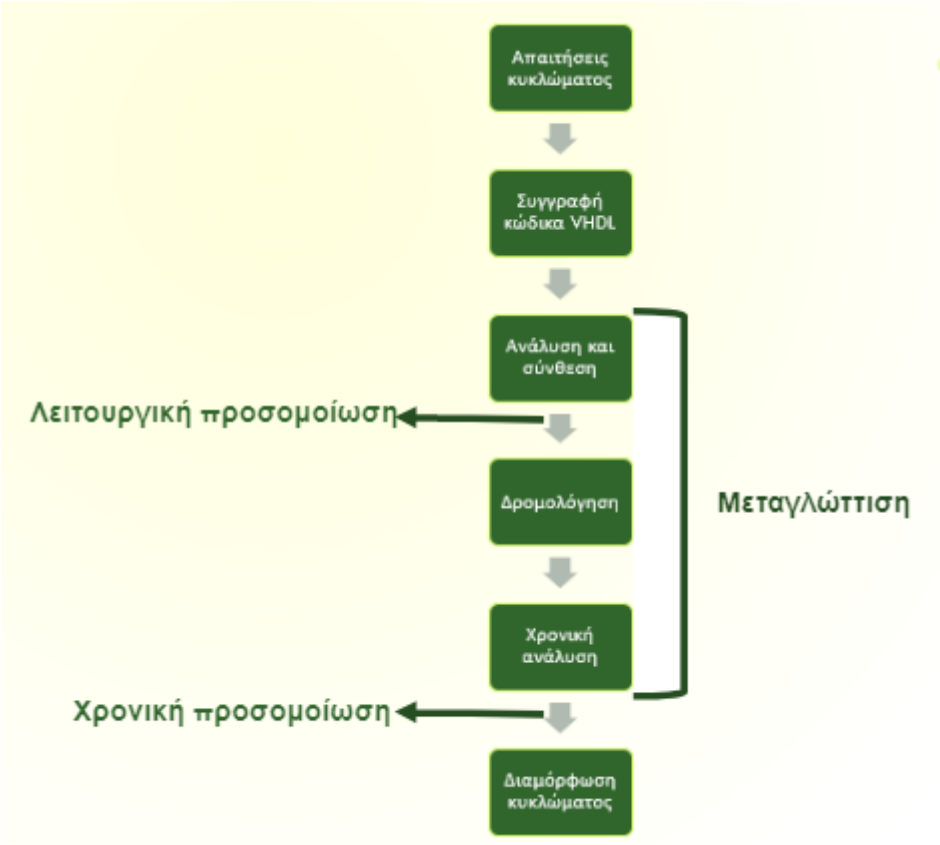
Η σχεδίαση με γλώσσες περιγραφής υλικού γίνεται σε συνδυασμό με κάποιες πλατφόρμες σχεδίασης CAD (Computer Aided Design) τα οποία μπορεί να αποτελούνται από διάφορα εργαλεία. Αυτά τα προγράμματα σχεδίασης δίνουν την δυνατότητα στον χρήστη να επιλέξει τον τρόπο σχεδίασης καθώς υποστηρίζουν και την σχηματική σχεδίαση αλλά για περίπλοκα ψηφιακά συστήματα η γλώσσες περιγραφής υλικού είναι οι κατάλληλες.

Για την σχεδίαση ενός συστήματος σε VHDL ο χρήστης θα πρέπει να καθορίσει τις απαιτήσεις του συστήματος δηλαδή την συμπεριφορά που θέλει να έχει το σύστημα .Το επόμενο βήμα είναι η συγγραφή κώδικα σε VHDL, οι πλατφόρμες σχεδίασης διαθέτουν εργαλεία για την συγγραφή και επεξεργασία πηγαίου κώδικα καθώς και για την αποσφαλμάτωση του. Κατά την διάρκεια της συγγραφής του κώδικα μπορούμε να ελέγξουμε τον κώδικα για σφάλματα .Τα αρχείο της VHDL που δημιουργείται έχει την κατάληξη .vhd αλλά ένα πρότζεκτ αποτελείται από πολλά και διαφορετικά αρχεία. Το επόμενο βήμα αφού έχουμε τελειώσει με την συγγραφή του κώδικα είναι μεταγλώττιση (Compilation).

Η μεταγλώττιση χωρίζεται σε τρεις φάσεις : Ανάλυση και σύνθεση ,δρομολόγηση και η χρονική ανάλυση .Στην φάση της ανάλυσης και σύνθεσης γίνεται η ανάλυση του κώδικα δηλαδή ο μεταγλωττιστής ελέγχει τον κώδικα για λάθη και τα εμφανίζει σε μορφή μηνυμάτων με στόχο να διορθωθούν από τον χρήστη για να προχωρήσει η διαδικασία. Η διαδικασία της ανάλυσης και σύνθεσης είναι η διαδικασία στην οποία γίνεται ο σχεδιασμός του κυκλώματος

σύμφωνα με την περιγραφή που κάναμε στον κώδικα και αναλόγως την διάταξη που θα χρησιμοποιηθεί ,αν δηλαδή θα είναι FPGA η CPLD.

Η επόμενη φάση της μεταγλώττισης είναι η δρομολόγηση και τοποθέτηση που είναι η διαδικασία της φυσικής σχεδίασης της διάταξης .Σε αυτή την διαδικασία ο μεταγλωττιστής τοποθετεί όλα τα λογικά στοιχεία που χρειάζονται για την υλοποίηση του σχεδίου, ουσιαστικά γίνεται μια προσομοίωση του σχεδίου σε φυσικό επίπεδο έτσι ώστε να υπολογιστεί και ο χρόνος μετάβασης των σημάτων .



Εικόνα 3.1 : Βασική ροή εργασιών κατά τη σχεδίαση με τη γλώσσα VHDL

Η τρίτη φάση της μεταγλώττισης περιλαμβάνει την χρονική ανάλυση του τελικού κυκλώματος η οποία είναι η διαδικασία ελέγχου καθυστερήσεων των

σημάτων .Αν χρονική ανάλυση ολοκληρωθεί με επιτυχία και η διαδικασία της μεταγλώττισης τότε περνάμε στην φάση της διαμόρφωσης (Configuration) της διάταξης φορτώνοντας το αρχείο σε αυτό μέσο των κατάλληλων συνδέσεων ανάλογα την διάταξη η στην δημιουργία μασκών ολοκληρωμένων κυκλωμάτων ειδικών εφαρμογών (ASIC) με την βοήθεια κάποιον πρόσθετων εργαλείων

3.3. Ιδιότητες και χαρακτηριστικά της γλώσσας VHDL

Όπως αναφέρθηκε και σε προηγούμενη ενότητα η VHDL είναι μια γλώσσα η οποία είναι παρόμοια έως ένα βαθμό με την γλώσσα προγραμματισμού Ada η οποία υποστηρίζει τον προγραμματισμό λογισμικού παράλληλων διεργασιών έτσι και η VHDL βασίζεται στις αρχές παράλληλου και δομημένου προγραμματισμού.

Η εκτέλεση του κώδικα στην VHDL γίνεται παράλληλα ,κάθε τμήμα του κώδικα εκτελείται ανεξάρτητα τη θέση που βρίσκετε και στο τέλος γίνεται η σύνθεση όλων των τμημάτων. Στην VHDL μπορούμε να αποθηκεύσουμε προγράμματα τα οποία έχουμε γράψει και να τα εισάγουμε ως τμήμα ενός άλλου σχεδίου, μπορούμε δηλαδή να δημιουργήσουμε βιβλιοθήκες με δικές μας δομές.

Η χρήση των τμημάτων αυτών σε ένα πιο σύνθετο σχέδιο γίνεται αναφέροντας την οντότητα του τμήματος που θέλουμε να εισάγουμε χωρίς να επαναλάβουμε όλο τον κώδικα. Ένα ακόμα χαρακτηριστικό της γλώσσας VHDL είναι η μη αναγνώριση των πεζών και κεφαλαίων γραμμάτων δηλαδή είναι Not Case Sensitive γλώσσα επίσης δεν αναγνωρίζει και τα κενά χαρακτήρες. Για να προσθέσουμε σχόλια στον κώδικα στην VHDL ξεκινάμε πάντα με δυο διαδοχικές παύλες “--” και ότι ακολουθεί μετά ο μεταγλωττιστής το αντιλαμβάνεται ως σχόλιο.

Η VHDL δεν υποστηρίζει σχόλιο πολλών γραμμών για αυτόν τον λόγο σε περίπτωση που το σχόλιο που θέλουμε να γράψουμε επεκτείνεται περισσότερο από μια γραμμή θα πρέπει σε κάθε γραμμή να ξεκινήσουμε με τις δύο διαδοχικές παύλες “--”. Η χρήση σχολίων σε πηγαίο κώδικα είναι πολύ σημαντικό σε όλες τις γλώσσες προγραμματισμού βοηθάει στην ανάγνωση και την κατανόηση του κώδικα από αναγνώστες και τον δημιουργό. Η κάθε εντολή τελειώνει με των

χαρακτήρα του ερωτηματικού ";" (Semicolon) όπως σε πολλές γλώσσες προγραμματισμού.

Ένα σημαντικό στοιχείο στην VHDL είναι και τα αναγνωριστικά (Identifiers) τα οποία είναι τα ονόματα που δίνονται σε διάφορα στοιχεία του κώδικα όπως τα σήματα και μεταβλητές. Υπάρχουν κάποιοι κανόνες στην ονομασία αυτών των στοιχείων:

Το αναγνωριστικό μπορεί να είναι πολλών χαρακτήρων η λίγων χαρακτήρων ανάλογα τις προτιμήσεις του χρήστη. Όσο πιο πολλούς χαρακτήρες τόσο πιο κατανοητό θα είναι .

Το αναγνωριστικό θα πρέπει να περιέχει πληροφορία για την λειτουργία του η αυτό που αντιπροσωπεύει.

Το αναγνωριστικό πρέπει να ξεκινάει με αλφαβητικό γράμμα και μπορεί να περιέχει συνδυασμό γραμμάτων (A-Z και a-z) ,αριθμών (0-9) και τον χαρακτήρα κάτω παύλα ''_" η οποία δεν μπορεί να είναι ποτέ ο τελευταίος χαρακτήρας και να είναι δυο συνεχόμενα.

Ένα κατάλληλο αναγνωριστικό για μια εφαρμογή ξυπνητηριού θα ήταν AlarmClock η alarm_clock.

3.3.1 Δεσμευμένες λέξεις στην γλώσσα VHDL

Στη VHDL όπως σε όλες τις γλώσσες περιγραφής υλικού υπάρχουν δεσμευμένες λέξεις τις οποίες δεν μπορούμε να χρησιμοποιήσουμε ως αναγνωριστικά στον κώδικα επειδή περιγράφουν μια συγκεκριμένη λειτουργία. Στον παρακάτω πίνακα παρουσιάζονται κάποιες δεσμευμένες λέξεις:

and	process	next	entity	array	case	use
or	function	return	signal	bus	package	variable
end	for	architecture	select	buffer	port	downto

begin	library	In	open	file	loop	elsif

Πίνακας 3.1 : Δεσμευμένες λέξεις της VHDL

3.3.2 Τρόπος συγγραφής κώδικα σε γλώσσα VHDL

Όπως αναφέραμε σε προηγούμενη ενότητα η VHDL δεν είναι Case Sensitive και δεν αναγνωρίζει τα κενά, αυτό την κάνει πιο ‘’ελεύθερη’’ στην συγγραφή από τον χρήστη αλλά χωρίς να σημαίνει ότι είναι απαραίτητα καλό.

Για να θεωρηθεί ένας κώδικας καλό γραμμένος θα πρέπει πρωτίστως να είναι αναγνώσιμος από τον ίδιο τον συγγραφέα και τους χρήστες που θα έχουν πρόσβαση σε αυτό για διάφορους λόγους όπως για παράδειγμα στην περαιτέρω ανάπτυξη του κώδικα η την αποσφαλμάτωση του.

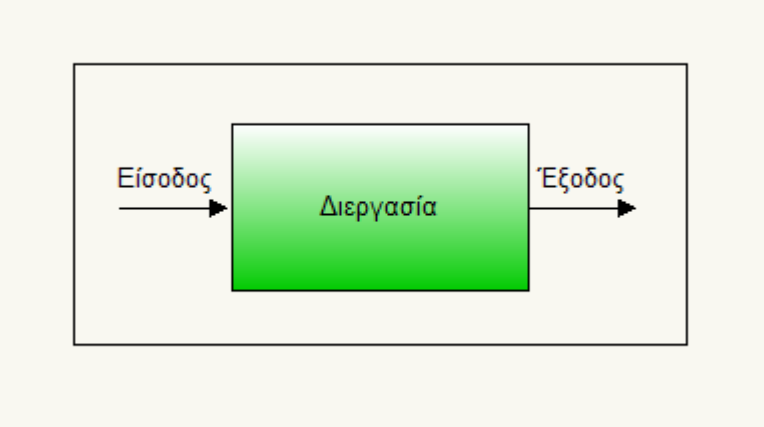
Ένας καλό γραμμένος κώδικας βοηθάει στην ελαχιστοποίηση των σφαλμάτων στον κώδικα και στην πιο γρήγορη εύρεση σφαλμάτων σε περίπτωση που υπάρχουν.

Συνοψίζοντας, η VHDL δεν έχει αυστηρούς περιορισμούς στον τρόπο και το στυλ της συγγραφής του κώδικα φθάνει αυτό λειτουργικά να μην περιέχει σφάλματα και οπτικά να είναι αναγνώσιμο και κατανοητό.

3.4 Βασικές δομές γλώσσας VHDL

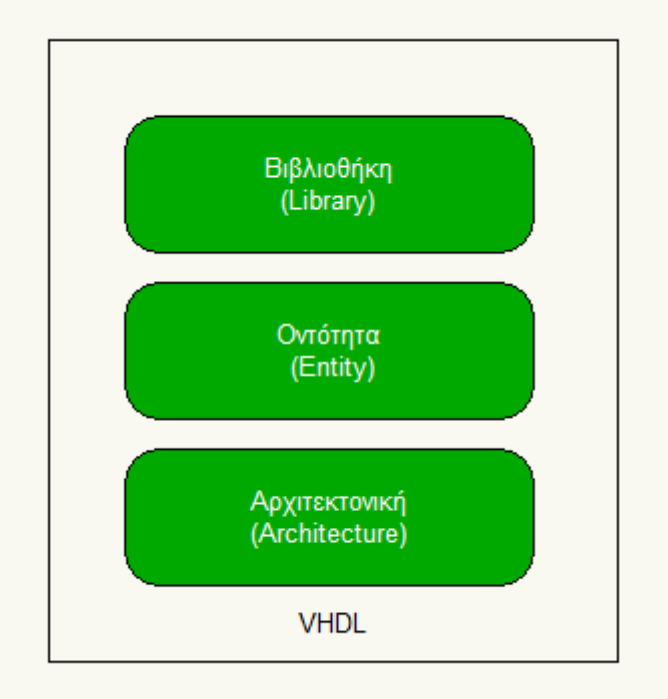
Η VHDL ως γλώσσα περιγραφής υλικού ,δομικά μοιάζει με ένα σύστημα το οποίο εκτελεί συγκεκριμένη λειτουργία και διαθέτει είσοδο και έξοδο καθώς και κάποια συγκεκριμένα στοιχεία για να λειτουργήσει το σύστημα αυτό.

Ως είσοδο και έξοδο στην VHDL υπάρχει η δομή Entity (Οντότητα) όπου δηλώνονται και λεπτομέρειες για τον τύπο τους. Η δομή Architecture (Αρχιτεκτονική) περιγράφει την λειτουργία του συστήματος έτσι ώστε να έχουμε το επιθυμητό αποτέλεσμα στην έξοδο.



Εικόνα 3.2: Παρομοίωση της VHDL με ένα σύστημα

Για να λειτουργήσει ένα σύστημα θα πρέπει να διαθέτουμε και τα κατάλληλα εξαρτήματα για να το κατασκευάσουμε αυτό, στης γλώσσες προγραμματισμού υπάρχουν έτοιμες βιβλιοθήκες οι οποίες μπορεί να διαθέτουν από εντολές μέχρι έτοιμες δομές .



Εικόνα 3.3 :Βασικά τμήματα ενός αρχείου VHDL

3.4.1 Βιβλιοθήκες στην γλώσσα VHDL

Η βασική βιβλιοθήκη της VHDL με πρότυπο IEEE Standard 1164 διαθέτει ένα πλήρες σύνολο από διάφορους τύπους δεδομένων, συναρτήσεις, εντολές και πολλά άλλα .Στην VHDL πάντα ξεκινάμε να γράφουμε τον κώδικα δηλώνοντας τις βιβλιοθήκες που θα χρησιμοποιήσουμε .Για απλές εφαρμογές της VHDL είναι αρκετό να χρησιμοποιήσουμε την IEEE.STD_LOGIC_1164.ALL και IEEE.NUMERIC_STD.ALL και η δήλωση τους γίνεται όπως παρακάτω:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
```

Μπορούμε να δημιουργήσουμε δικιά μας βιβλιοθήκη από προγράμματα που έχουμε γράψει και να τα επαναχρησιμοποιήσουμε σε άλλα πρότζεκτ απλά δηλώνοντάς τα στην αρχή του κώδικα όπως κάνουμε και με τις βιβλιοθήκες της VHDL.

3.4.2 Entity (Οντότητα)

Η οντότητα (Entity) είναι η δομή του κώδικα στην οποία γίνεται η δήλωση των σημάτων εισόδου και εξόδου του κυκλώματος. Στο παρακάτω παράδειγμα παρουσιάζεται η δήλωση μιας οντότητας:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity JK is
 Port ( j: in  STD_LOGIC;
              k : in  STD_LOGIC;
```

```
            clk  : in STD_LOGIC;
            clr: in STD_LOGIC;
            q: buffer STD_LOGIC;
            nq: buffer STD_LOGIC
            );
end JK;
```

Entity είναι η δεσμευμένη λέξη για την οντότητα ακολουθεί το αναγνωριστικό της δηλαδή το όνομα που θα δώσουμε εμείς στην οντότητα ,Is είναι δεσμευμένη λέξη όπως και η λέξη Port για να μας δείξει ότι τα ακόλουθα δεδομένα είναι τα σήματα εισόδου και εξόδου του κυκλώματος. Για να δηλώσουμε ένα σήμα γράφουμε το όνομα του ,το σύμβολο ":" ,το είδος σήματος δηλαδή αν είναι εισόδου εξόδου κτλ. και τον τύπο δεδομένων του σήματος και για κάθε σήμα βάζουμε στο τέλος το σύμβολο ";". Για παράδειγμα:

```
Port ( port_1 : in  STD_LOGIC );
```

Κάθε πόρτα πρέπει να έχει μοναδικό όνομα ,το είδος σήματος και τον τύπο δεδομένων . Καλό είναι η κάθε πόρτα να έχει την δικιά της γραμμή στον κώδικα για ευανάγνωστους λόγους και τα σχόλια μπορούν να βοηθήσουν σε αυτό.

3.4.3 Architecture (Αρχιτεκτονική)

Η δομή Αρχιτεκτονική (Architecture) στην γλώσσα VHDL είναι η δομή στην οποία περιγράφεται η λειτουργίας ενός κυκλώματος. Είναι εύκολο να καταλάβουμε ότι η εξωτερική περιγραφή ενός κυκλώματος δηλαδή το τι είσοδο θα έχει ένα κύκλωμα και τι έξοδο ,είναι αρκετά απλό σε σχέση με την περιγραφή της εσωτερικής του λειτουργίας που είναι η συμπεριφορά που θα πρέπει να έχει για να λάβουμε την επιθυμητή έξοδο.

Στο παρακάτω παράδειγμα βλέπουμε την αρχιτεκτονική ενός κυκλώματος J-K Flip-Flop:

```
architecture Behavioral of JK is

begin
    process(clk,clr)
       begin
            if  (clr = '1') then
                q <= '0';
            elsif (clk'event and clk = '1') then
                q <= (j and not q) or (not k and q);
            end if;
    end process;
  nq <= not q;
 end Behavioral;
```

Η δήλωση ξεκινάει με την δεσμευμένη λέξη Architecture και ακολουθεί το αναγνωριστικό της ,όνομα το οποίο το επιλέγουμε εμείς και η δεσμευμένη λέξη Of για να επισημάνει ότι η αρχιτεκτονική ανήκει στην οντότητα που έχουμε δηλώσει .

Η περιγραφή των λειτουργιών ξεκινάει με την δεσμευμένη λέξη Begin όπως και η κάθε διαδικασία (Process) και τελειώνουν με την λέξη End και το αναγνωριστικό της αρχιτεκτονικής .

3.5 Τύποι αντικειμένων στην VHDL

Η VHDL διάφορους τύπους αντικειμένων από τα οποία τα πιο γνωστά είναι το αντικείμενο Σήμα (Signal) ,το αντικείμενο Σταθερά (Constant) και το αντικείμενο Μεταβλητή (Variable), αυτά τα αντικείμενα δεδομένων για αποθήκευση η μεταφορά πληροφορίας σε αριθμητική μορφή. Το αντικείμενο Variable όπως και σε άλλες γλώσσες χρησιμοποιείτε για τοπική αποθήκευση πληροφορίας και ορίζεται μέσα στην διαδικασία. Ο τύπος αντικειμένου Signal χρησιμοποιείτε για την αναπαράσταση των αγωγών διασύνδεσης του κυκλώματος. Ο τύπος αντικειμένου Constant είναι όπως το αντικείμενο Variable αλλά η τιμή του δεν

αλλάζει .Το αντικείμενο δεδομένων Signal μπορεί να είναι διαφορετικών τύπων δεδομένων ,για παράδειγμα :

```
SIGNAL clk1  : STD_LOGIC := '0';
SIGNAL clk2 : STD_LOGIC := '0';
SIGNAL count : INTEGER := 1;
SIGNAL countReset : INTEGER := 0;
```

Ο τύπος προσδιορίζει τι τιμές μπορεί να λάβει το αντικείμενο δεδομένων για αυτό τον λόγο είναι απαραίτητο να γνωρίζουμε οπωσδήποτε να γνωρίζουμε πως χρησιμοποιούμε τους τύπους δεδομένων. Στις μεταβλητές είναι σημαντικό η σειρά ανάθεσης τιμών επειδή η εντολές εκτελούνται ακολουθιακά .Στα σήματα από την άλλη δεν έχει σημασία επειδή εκτελούνται ταυτόχρονα.

Για να δηλώσουμε ένα αντικείμενο Variable,Signal η Constant γράφουμε:

```
variable Άνομα_μεταβλητής : Τύπος_μεταβλητής : Ανάθεση_τιμής ='';

signal Ονομα_μεταβλητής : Τύπος_μεταβλητής : Ανάθεση_τιμής ='';

constant Ονομα_μεταβλητής : Τύπος_μεταβλητής : Ανάθεση_τιμής ='';
```

Οι μεταβλητές και οι σταθερές που χρησιμοποιούνται σε συναρτήσεις και διαδικασίες είναι βασικά αντικείμενα στην διαμόρφωση της συμπεριφοράς ενός κυκλώματος.

3.6 Τύποι δεδομένων στην VHDL

Για να δηλώσουμε ένα αντικείμενο θα πρέπει να καθορίσουμε και τον τύπο δεδομένων που θα έχει .Ο τύπος δεδομένων καθορίζει την περιοχή τιμών και τον περιορισμό λειτουργιών αντικειμένου. Η VHDL διαθέτει διάφορους τύπους δεδομένων ενώ δίνει την δυνατότητα στον χρήστη να δημιουργήσει δικό του

τύπο. Στον παρακάτω πίνακα παρουσιάζονται οι βασικοί τύποι δεδομένων που χρησιμοποιούνται στην VHDL:

std_logic	Δυαδικές τιμές (0 και 1)
integer	Περιοχή τιμών (-2,147,483,648 έως 2,147,483,647)
boolean	Παίρνει τιμές True η False
natural	Υποκατηγορία του τύπου Integer
positive	Υποκατηγορία του τύπου Integer
character	Όλοι οι χαρακτήρες που υποστηρίζει η VHDL
string	Πίνακας χαρακτήρων
integer_vector	Πίνακας ακέραιων αριθμών
std_logic_vector	Πίνακας δυαδικών
boolean_vector	Πίνακας boolean
bit	Δυαδικές τιμές (0 και 1)
bit_vector	Πίνακας δυαδικών

Πίνακας: Τύποι δεδομένων

Για να δημιουργήσουμε δικό μας τύπο δεδομένων μπορούμε να ακολουθήσουμε το παρακάτω παράδειγμα:

```
TYPE display_ROM IS ARRAY (0 TO 10) OF std_logic_vector (6 DOWNTO 0);
CONSTANT convert_to_segments : display_ROM :=
      ("1000000", "1111001", "0100100", "0110000", "0011001", "0010010",
"0000010", "1111000", "0000000", "0010000", "1111111");
```

3.6.1 Βασική τύποι δεδομένων

Ένας από τους βασικούς τύπους δεδομένων είναι ο std_logic το οποίο δέχεται διάφορες τιμές (0,1,u,x,z,l,h,w,-) όπου η βασικές είναι δυαδικές τιμές 0 και 1. Ο τύπος ακέραιων (integer) είναι επίσης βασικός τύπος καθώς είναι απαραίτητο στην περιγραφή συμπεριφοράς περίπλοκων ψηφιακών συστημάτων. Η περιοχή τιμών που μπορεί να πάρει ένας τύπος integer είναι από -2,147,483,648 μέχρι 2,147,483,647. Οι τύποι Natural και Positive είναι ουσιαστικά integer με διαφορετική περιοχή τιμών και μέγεθος 32 bits.

3.6.2 Προσημασμένοι και μη προσημασμένοι τύποι

Για να χρησιμοποιήσουμε προσημασμένο η μη προσημασμένο τύπο δεδομένων στον κώδικα θα πρέπει να δηλώσουμε και την προτυποποιημένη βιβλιοθήκη ieee.numeric std .Ο προσημασμένος τύπος έχει περιοχή τιμών −2^(N−1) έως 2^(N−1) − 1 ενώ ο μη προσημασμένος τύπος έχει περιοχή τιμών από ο έως 2^N-1 όπου N ο αριθμός των bits. Είναι τύποι δεδομένων οι οποίοι χρησιμοποιούντα όσο στης μεταβλητές τόσο και στα σήματα .

3.7 Τελεστές στην VHDL

Οι τελεστές στην VHDL χωρίζονται σε επτά ομάδες με σειρά προτεραιότητας ενώ δεν υπάρχει σειρά προτεραιότητας στους τελεστές που ανήκουν στην ίδια ομάδα. Σε κάθε έκφραση οι τελεστές θα πρέπει να ανήκουν στην ίδια ομάδα, παρακάτω παρουσιάζονται όλες οι ομάδες τελεστών που διαθέτει η VHDL ξεκινώντας από την ομάδα με την μικρότερη προτεραιότητα:

Οι λογικοί τελεστές (Logical Operators) ουσιαστικά αναπαριστούν λογικές πύλες η για την ακρίβεια εκτελούν λογικές πράξεις. Στους λογικούς τελεστές δεν συμπεριλαμβάνεται η λογική πράξη not η οποία ανήκει σε άλλη ομάδα με μεγαλύτερη προτεραιότητα .
Λογικοί τελεστές*:* ***and, or, nand, nor, xor, nxor.***

Οι σχεσιακή τελεστές (Relation Operator) είναι ίδιοι όπως και στις περισσότερες γλώσσες προγραμματισμού.
Σχεσιακοί τελεστές: **=, /=, <, <=, >, >=** .

Τελεστές ολίσθησης (Shift Operator) υπάρχουν τριών ειδών ,λογικοί (Logical) , αριθμητικοί (Arithmetic) και οι περιστροφικοί (Rotate) .
Τελεστές ολίσθησης: Logical: **sll**(shift left logical)**, srl**(shift right logical)
Arithmetic:**sla**(shift left arithmetic)**, sra**(shift rightarithmetic)
Rotate : **rol** (rotate left), **ror** (rotate right)**.**

Η διαφορά της λογικής ολίσθησης με την αριθμητική είναι στο ότι στην λογική τα bit που θέλουμε να μετακινήσουμε προς μια κατεύθυνση χάνονται και συμπληρώνονται στην άλλη άκρη με μηδενικά. Στην αριθμητική ολίσθηση το bit του πρόσημου δεν αλλάζει ενώ τα bit που χάνονται συμπληρώνονται με την τιμή1.

Τελεστές πρόσθεσης (Adding Operator): **+, -, &**
Τελεστές πρόσημου (Sign Operator): **+, -**
Τελεστές πολλαπλασιασμού (Multiplying Operator): ***, /, mod, rem**
Ανάμικτοι τελεστές (Adding Operator): ****, abs, not**

3.8 Τελεστές ανάθεσης τιμών στην VHDL

Για να αναθέσουμε τιμές σε αντικείμενα δεδομένων στην VHDL υπάρχουν ειδικοί τελεστές .Σε γλώσσες προγραμματισμού συνήθως χρησιμοποιείτε το σύμβολο της ισότητας "=" ενώ στην VHDL για αναθέσουμε τιμή σε ένα σήμα χρησιμοποιούμε "<=" το οποίο συμβολίζει την μεταφορά της τιμής που βρίσκετε στην δεξιά μεριά του τελεστή προς την αριστερή μεριά του .

Παρακάτω παρουσιάζονται τρόποι ανάθεσης τιμών σε σήμα:

```
Σήμα_3 <= Σήμα_1 and Σήμα_2;
```

```
Σήμα_α <= 'Τιμή' ;
```

Στην πρώτη περίπτωση η τιμή του σήματος Σήμα_3 προκύπτει από την εκτέλεση της λογικής πράξης and των σημάτων Σήμα_1 και Σήμα_2 ενώ στην δεύτερη περίπτωση το σήμα Σήμα_α παίρνει την τιμή που βρίσκεται μέσα στα εισαγωγικά στην δεξιά μεριά τελεστή ανάθεσης.

Για να αναθέσουμε τιμή σε μια μεταβλητή τα σύμβολα που χρησιμοποιούμε είναι ":=" όπως το παρακάτω παράδειγμα:

```
VARIABLE Μεταβλητή_1 : Τύπος_δεδομένων := Τιμή;
```

```
Μεταβλητή_3 := Μεταβλητή_1 and Μεταβλητή_2;
```

```
Μεταβλητή_α := Τιμή;
```

Όπως φαίνεται στην πρώτη περίπτωση μπορούμε να αναθέσουμε την τιμή σε μια μεταβλητή όταν δηλώνουμε την μεταβλητή αυτή. Η τιμή που βάζουμε δεν τοποθετείτε σε εισαγωγικά όπως στον τελεστή ανάθεσης σήματος. Στην δεύτερη περίπτωση η μεταβλητή Μεταβλητή_3 θα πάρει την τιμή της πράξεις που γίνεται στην δεξιά μεριά του τελεστή ανάθεσης που στο συγκεκριμένο παράδειγμα είναι η λογική πράξη and. Στην τελευταία περίπτωση γίνεται η απευθείας ανάθεση τιμής στην μεταβλητή Μεταβλητή_α.

3.9 Τρόποι περιγραφής ενός κυκλώματος

Σε προηγούμενη ενότητα αναφέραμε ότι η δομή που περιγράφει της λειτουργίες ενός κυκλώματος σε γλώσσα VHDL είναι η αρχιτεκτονική (Architecture) αλλά υπάρχουν τεχνικές η τρόποι ανάπτυξης κώδικα για την περιγραφή ενός ψηφιακού συστήματος; Η απάντηση είναι ναι ,υπάρχουν τρεις βασικοί τρόποι για να περιγράψουμε ένα κύκλωμα οι οποίοι είναι :

- Συμπεριφοράς (Behavioral)
- Ροής δεδομένων (Data-flow)
- Δομημένος (Structural)

Ο τρόπος της ροής δεδομένων (Data-flow) για να περιγράψει ψηφιακό σύστημα χρησιμοποιεί ένα σύνολο λογικές συναρτήσεις, λογικές πύλες και διάφορα άλλα λογικά στοιχεία που διαθέτει η VHDL. Ο τρόπος αυτός περιγράφει στο ψηφιακό σύστημα την σχέση εισόδων—εξόδων μεταξύ των λογικών εκφράσεων της VHDL. Ουσιαστικά περιγράφετε το κύκλωμα πως θα έπρεπε να φαίνεται σε επίπεδο λογικών πυλών. Αυτός ο τρόπος περιγραφής της συμπεριφοράς ενός κυκλώματος είναι κατάλληλο για εφαρμογές σε απλά κυκλώματα.

Ο τρόπος συμπεριφοράς (Behavioral) περιγράφει αναλυτικά ένα κύκλωμα χρησιμοποιώντας διεργασίες (Processes) η οποία είναι η βασική δομή .Σε σχέση με τον τρόπο της ροής δεδομένων δεν αναπαριστά το κύκλωμα αλλά την αντίδραση των εξόδων σε κάθε είσοδο.

Ο δομημένος (Structural) τρόπος περιγραφής συμπεριφοράς περιγράφει ένα σύστημα χρησιμοποιώντας έτοιμες δομές της VHDL. Για περίπλοκα κυκλώματα είναι καλύτερα να γίνει συνδυασμός αυτόν τον τρόπων περιγραφής.

3.10 Δομή διεργασίας (Process)

Η διεργασία (Process) αποτελείτε από διαδοχικές δηλώσεις ενώ η διεργασίες εκτελούνται ταυτόχρονα. Σε μια δομή Process γίνεται δήλωση αντικειμένων τοπικά. Το παρακάτω είναι ένα παράδειγμα δήλωσης διεργασίας :

```
Ετικέτα : Process (Λίστα_ευαισθησίας) is
      Δηλώσεις_διεργασίας
```

```
...
...
...
begin
ακολουθιακές_δηλώσεις
...
...
...
end Process Ετικέτα ;
```

Στην δήλωση της διεργασίας η ετικέτα δεν είναι υποχρεωτική ενώ η λίστα ευαισθησίας είναι σήματα τα οποία αν αλλάξει η τιμή τους εκτελείτε η διαδικασία .Η δηλώσεις της διεργασίας είναι τοπικές και μπορεί να είναι μεταβλητές η σταθερές. Στις ακολουθιακές δηλώσεις μπορούμε να χρησιμοποιήσουμε διάφορες εκφράσεις όπως ανάθεση τιμών σε μεταβλητές η σταθερές ,εκφράσεις ελέγχου με τις εντολές if, elsif κτλ, εκφράσεις επανάληψης και διάφορες άλλες εκφράσεις .

Σημαντικό σε μια διεργασία process είναι και η εντολή wait on που αν δεν συμπεριλάβουμε λίστα ευαισθησίας στην διεργασία και την εντολή wait on η διεργασία θα καταλήξει να εκτελείτε συνέχεια. Στην παρακάτω εικόνα έχουμε ένα παράδειγμα με και χωρίς την χρήση της εντολής wait on.

```
--Διεργασία Process με την χρήση της
--λίστας ευαισθησίας .
  process(clk,clr)
        begin
            if  (clr = '1') then
                q <= '0';
            elsif (clk'event and clk = '1') then
                q <= (j and not q) or (not k and q);
            end if;
     end process;

--Διεργασία Process χωρίς την χρήση της
--λίστας ευαισθησίας αλλά με χρήση της
--εντολής wait on.
     process
       begin
            if  (clr = '1') then
                q <= '0';
            elsif (clk'event and clk = '1') then
                q <= (j and not q) or (not k and q);
            end if;
            wait on clk ,clr
     end process;
```

Εικόνα 3.4 : Χρήση της λίστας ευαισθησίας και της εντολής wait on

Η κάθε διεργασία τελειώνει με την εντολή End Process Ετικέτα; η αν δεν έχουμε δηλώσει ετικέτα γράφουμε μόνο End Process;.

3.11 Δομές ελέγχου και επανάληψης

Όπως όλες οι γλώσσες προγραμματισμού και η VHDL διαθέτει εντολές για να περιγράψει συστήματα με δομές επανάληψης και ελέγχου. Οι βασικές εντολές για να εκφράσουμε τέτοιες δομές είναι η If ,case και η εντολή επανάληψης loop. Αυτές η εντολές τις έχουμε δει σε άλλες γλώσσες αλλά και λειτουργικά οι εντολές αυτές είναι παρόμοιες και στην VHDL. Χρησιμοποιούνται στην περιγραφή κυκλωμάτων μέσα σε διεργασίες Process.

3.11.1 Εντολές ελέγχου If και case

Η εντολή if χρησιμοποιείτε σε εκφράσεις για να ελέγχει μια συγκεκριμένη συνθήκη αν είναι αληθές .Σε κάθε κάθε δήλωση της if υπάρχει τουλάχιστον μια then και η κάθε δήλωση τελειώνει με end if ενώ υπάρχει και η εντολή elsif η αντίστοιχη της else if που χρησιμοποιούν άλλες γλώσσες. Το παρακάτω μας δείχνει πως συντάσσεται μια εντολή if στην VHDL:

```
if      (Συνθήκη)    then
        ακολουθιακή_δήλωση;
elsif   (Συνθήκη)   then
        ακολουθιακή_δήλωση;
elsif   (Συνθήκη)   then
        ακολουθιακή_δήλωση;
...        ...        ...
...        ...        ...
else    ακολουθιακή_δήλωση;

end if ;
```

Η συνθήκη δεν είναι υποχρεωτική να είναι σε παρένθεση είναι στο χέρι το προγραμματιστή αν θα βάλει η όχι .Για ευανάγνωστους λόγους καλό είναι να βάζουμε την συνθήκη σε παρένθεση. Όπως φαίνεται και στο παράδειγμα μετά την else ακολουθεί η τελευταία έκφραση και το τέλος της δομής ελέγχου. Μια άλλη εντολή ελέγχου είναι και η Case η σύνταξη της γίνετε με τον παρακάτω τρόπο:

```
case (έκφραση) is

when επιλογή => ακολουθιακή_δήλωση;
```

```
when επιλογή => ακολουθιακή_δήλωση;

...   ...        ...

when others => ακολουθιακή_δήλωση;
end case
```

3.11.2 Δομές επανάληψη

Δύο εντολές που χρησιμοποιούνται στην VHDL για υλοποίηση δομών επανάληψης είναι η εντολή for και η εντολή while. Όπως και οι εντολές ελέγχου έτσι και οι εντολές επανάληψης είναι γνωστές από άλλες γλώσσες προγραμματισμού. Οι εντολές for και η while χρησιμοποιούνται μέσα στην διεργασία Process και εκτελούνται ακολουθιακά.

Οι δομές αυτές είναι κατάλληλες όταν θέλουμε να περιγράψουμε ένα σύστημα που για να ικανοποιείτε κάποια συγκεκριμένη συνθήκη θα πρέπει να εκτελείτε κάποια η κάποιες εντολές πολλές φορές.

Η εντολή επανάληψης for χρησιμοποιείται σε δομές τις οποίες γνωρίζουμε τον αριθμό επαναλήψεων σε αντίθεση με την εντολή while η οποία χρησιμοποιείται σε δομές που ο αριθμός επαναλήψεων είναι άγνωστος. Παρακάτω ακολουθεί τρόποι σύνταξης μιας δομής for:

```
for δείκτης in 0 to  αρ_επ loop
ακολουθιακές δηλώσεις
end loop;
type εύρος_επ is range 0 to αρ_επ;
for δείκτης in εύρος_επ loop
ακολουθιακές δηλώσεις
end loop;
```

Στην πρώτη περίπτωση το εύρος επαναλήψεων δηλώνεται απευθείας στην δομή επανάληψης ενώ στην δεύτερη περίπτωση δηλώνετε πριν. Το εύρος επαναλήψεων μπορεί να δηλωθεί και με την εντολή downto. Ο δείκτης είναι μια μεταβλητή η οποία δεν χρειάζεται να δηλωθεί.

Η εντολή while είναι πιο απλή στην σύνταξη διότι δεν διαθέτει δείκτη για τις επαναλήψεις.

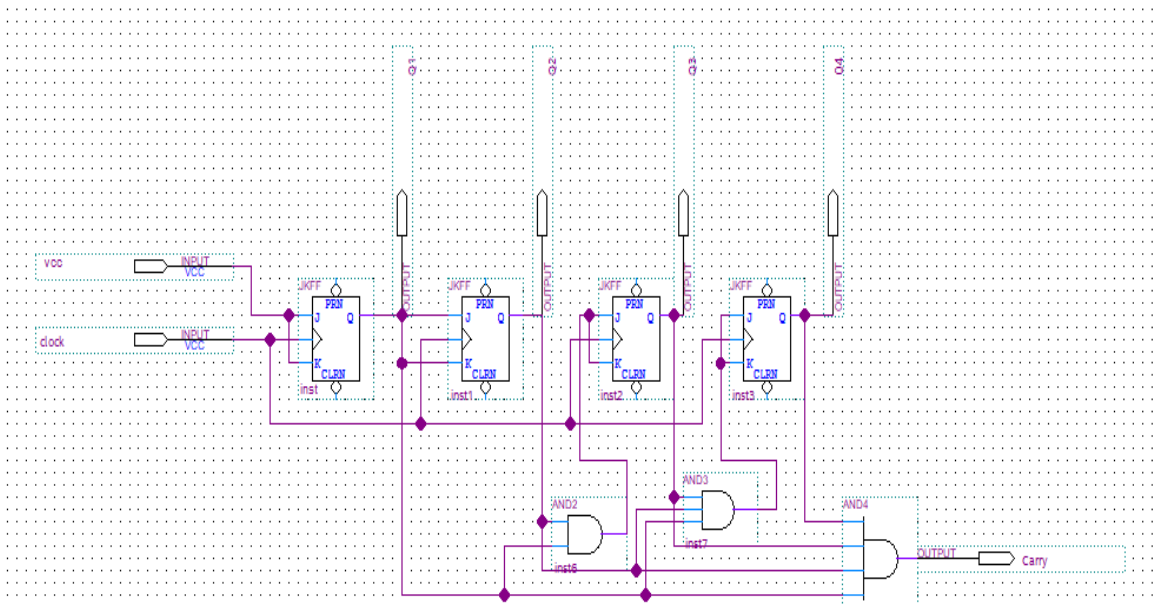
```
while (Συνθήκη) loop
ακολουθιακές δηλώσεις
....
....
end loop
```

3.12 Παραδείγματα κυκλωμάτων σε γλώσσα VHDL

Σε αυτή την ενότητα θα δούμε παραδείγματα κυκλωμάτων σε VHDL γλώσσα χρησιμοποιώντας δομές και εντολές που έχουν αναφερθεί σε αυτό το κεφάλαιο.

3.12.1 Σύγχρονος δυαδικός απαριθμητής προς τα πάνω (Up counter)

Το κύκλωμα ενός δυαδικού απαριθμητή που μετράει προς τα πάνω αποτελείται από τέσσερεις JK Flip-Flop συνδεδεμένοι μεταξύ τους έτσι ώστε οι έξοδοι του κυκλώματος οι οποίοι είναι οι έξοδοι των τεσσάρων JK Flip-Flop όπως φαίνονται στην παρακάτω εικόνα:



Εικόνα 3.5 : Δυαδικός απαριθμητής προς τα πάνω

Η είσοδος είναι οι παλμοί του ρολογιού του συστήματος ,με κάθε αλλαγή του παλμού αλλάζει και η έξοδος κατά ένα προς τα πάνω ενώ όταν φθάσει 15 με τον επόμενο παλμό μηδενίζεται η έξοδος. Στον παρακάτω κώδικα υλοποιείται το κύκλωμα του δυαδικού απαριθμητή προς τα πάνω (Up counter):

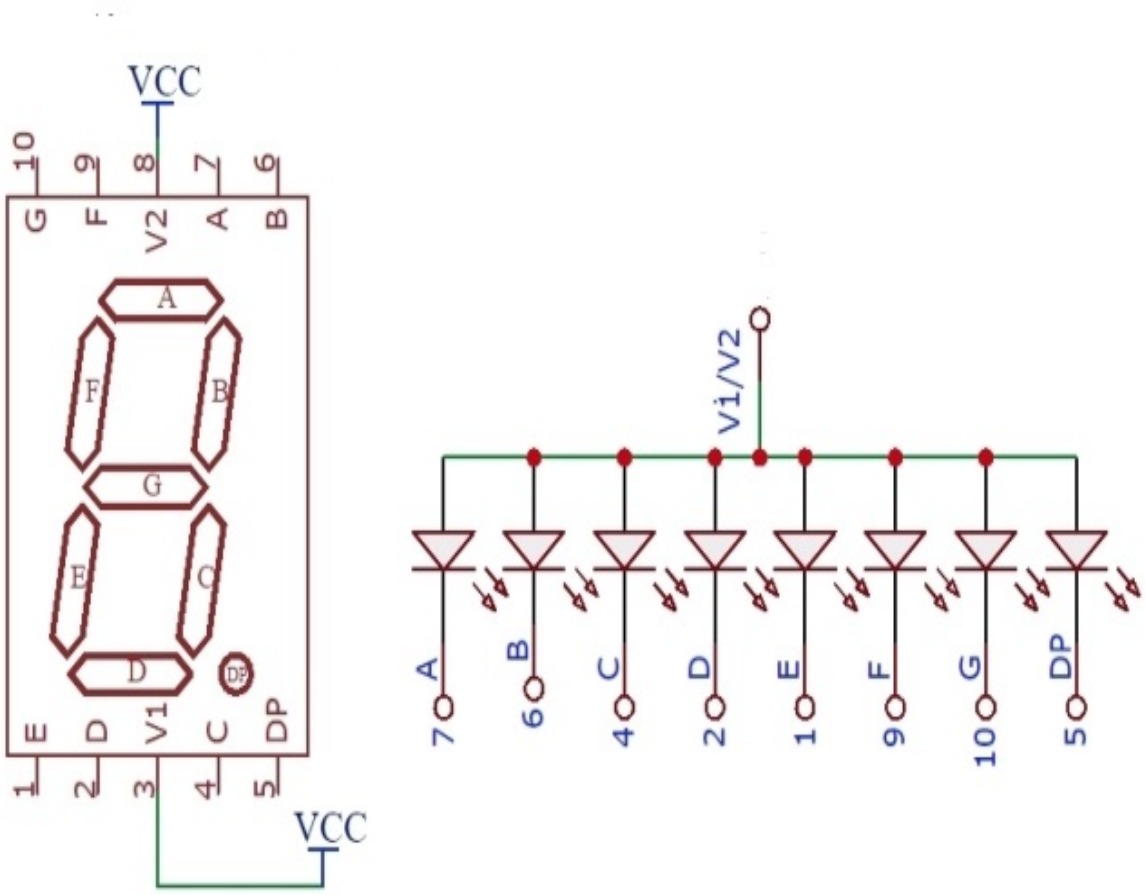
```
--Δήλωση βιβλιοθήκης
library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

--Οντότητα απαρηθμητής
entity aparathmitis is
    port(Clk,Clr : in  std_logic; --Δήλωσεις οντότητας
         eksodos :       out std_logic_vector(3 downto 0));
end aparathmitis;
```

```
architecture behaviour of aparathmitis is
   signal deiktis: std_logic_vector(3 downto 0);--Δήλωση δείκτη για κάθε
bit
       begin
             process (Clk, Clr) --Αρχή διαδικασίας Process
             begin
                   if (Clr='1') then
                         deiktis <= "0000";  -- Αν η είσοδος Clr είναι
1 τότε όλοι έξοδοι μηδενήζωνται
                   elsif (Clk'event and Clk='1') then -- Σε κάθε παλμό
του ρολογιού
                         deiktis <= deiktis + 1;--Ο δείκτης αυξάνεται
κατά ένα
                   end if;
             end process;
        eksodos <= deiktis;--Η έξοδος παίρνει την τιμή του δείκτη
end behaviour;             --κάθε φορά που εκτελείτε η διεργασία
```

Όπως φαίνεται και στα σχόλια στην διεργασία Process όταν το σήμα Clr είναι 1 τότε μηδενίζεται η έξοδος του κυκλώματος ενώ κάθε φορά που αλλάζει ο παλμός του ρολογιού αλλάζει και ο δείκτης κατά 1 .Κάθε φορά που ολοκληρώνεται μια διεργασία η έξοδος παίρνει την τιμή του δείκτη.

3.12.2 Κύκλωμα μετατροπής δυαδικού σε 7 segment displays

Το 7 segment είναι ηλεκτρονικό στοιχείο η εξάρτημα το οποίο αποτελείται από επτά τμήματα που το κάθε τμήμα αποτελείται από ένα Led κοινής ανόδου η κοινής καθόδου .Στην ουσία το 7 segment είναι ένα ηλεκτρονικό κύκλωμα το οποίο αποτελείται από Led και αντιστάσεις και χρησιμοποιείται για την αναπαράσταση δεκαδικών και δεκαεξαδικών αριθμών. Υπάρχουν κυκλώματα για την μετατροπή του δυαδικών αριθμών σε δεκαδικό η δεκαεξαδικό. Στην παρακάτω εικόνα φαίνεται ένας 7 segment display:



Εικόνα 3.6: Ηλεκτρονικό σχέδιο ενός 7 segment display.

Όπως φαίνεται στην εικόνα το κύκλωμα διαθέτει δέκα ακροδέκτες ,τα επτά για κάθε τμήμα ,το ένα είναι η τελεία (DP) και τα άλλα δύό είναι τροφοδοσία η οποία είναι κοινή άρα το 7 segment είναι κοινής ανόδου .Στην εικόνα βλέπουμε και την αντιστοιχία των τμημάτων με το κάθε ακροδέκτη.

Για να δημιουργήσουμε ένα κύκλωμα που να μετατρέπει έναν δυαδικό αριθμό σε δεκαεξαδικό θα πρέπει πρώτα να δημιουργήσουμε έναν πίνακα αντιστοιχίας των δυαδικών ,δεκαεξαδικών και των τμημάτων του 7 segment :

Δυαδική μορφή	Δεκαεξαδική	ABCDEFG
0000	0	0000001
0001	1	1001111
0010	2	0010010
0011	3	0000110
0100	4	1001100
0101	5	0100100
0110	6	0100000
0111	7	0001111
1000	8	0000000
1001	9	0000100
1010	A	0001000
1011	B	1100000
1100	C	0110001
1101	D	1000010
1110	E	0110000
1111	F	0111000

Πίνακας 3.2 : Αντιστοιχίας δυαδικών και δεκαεξαδικών σε 7 segments

Στον πίνακα βλέπουμε την δυαδική μορφή του αριθμού ,την δεκαεξαδική και το κάθε τμήμα Led που πρέπει να ανάψει για να απεικονίσει το δεκαεξαδικό

αριθμό .Ο πίνακας έχει συμπληρωθεί με βάση την συνδεσμολογία των τμημάτων Led που στην περίπτωση μας είναι κοινού ανόδου.

Ο κώδικας για την υλοποίηση του κυκλώματος μετατροπής παρουσιάζεται παρακάτω:

```
--Δήλωση βιβλιοθήκης
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

--Δήλωση οντότητας
entity bcd_to_hex is
  Port ( binary        : in STD_LOGIC_VECTOR (3 downto 0);--Δήλωση
είσόδου
         display     : out STD_LOGIC_VECTOR (6 downto 0));--Δήλωση
εξόδου
end bcd_to_hex;

--Δήλωση αρχητεκτονικής
architecture decoder of bcd_to_hex is
  begin
    process(binary) --Δήλωση διεργασίας binary
     begin
        case binary is --Απεικόνιση στο 7 segment για κάθε
                       --δυαδικό αριθμό τεσσάρων bit
           when "0000" =>display <= "0000001"; --Εμφανίζει το 0
           when "0001" =>display <= "1001111"; --Εμφανίζει το 1
           when "0010" =>display <= "0010010"; --Εμφανίζει το 2
           when "0011" =>display <= "0000110"; --Εμφανίζει το 3
           when "0100" =>display <= "1001100"; --Εμφανίζει το 4
           when "0101" =>display <= "0100100"; --Εμφανίζει το 5
           when "0110" =>display <= "0100000"; --Εμφανίζει το 6
```

```
            when "0111" =>display <= "0001111"; --Εμφανίζει το 7
            when "1000" =>display <= "0000000"; --Εμφανίζει το 8
            when "1001" =>display <= "0000100"; --Εμφανίζει το 9
            when "1010" =>display <= "0001000"; --Εμφανίζει το A
            when "1011" =>display <= "1100000"; --Εμφανίζει το B
            when "1100" =>display <= "0110001"; --Εμφανίζει το C
            when "1101" =>display <= "1000010"; --Εμφανίζει το D
            when "111O" =>display <= "0110000"; --Εμφανίζει το E
            when "1111" =>display <= "0111000"; --Εμφανίζει το F
            when others =>display <= "1111111"; --Δεν εμφανίζει τίποτα
        end case;
    end process;
end decoder;
```

Στην αρχή δηλώνουμε την βιβλιοθήκη IEEE.STD_LOGIC_1164.ALL της VHDL και στην συνέχεια την οντότητα με όνομα bcd_to_hex που είναι μια περιληπτική περιγραφή της λειτουργίας του κυκλώματος, Στην συνέχεια δηλώνουμε τις πόρτες ,έχουμε δύο πόρτες ,μια είσοδο και μια έξοδο.

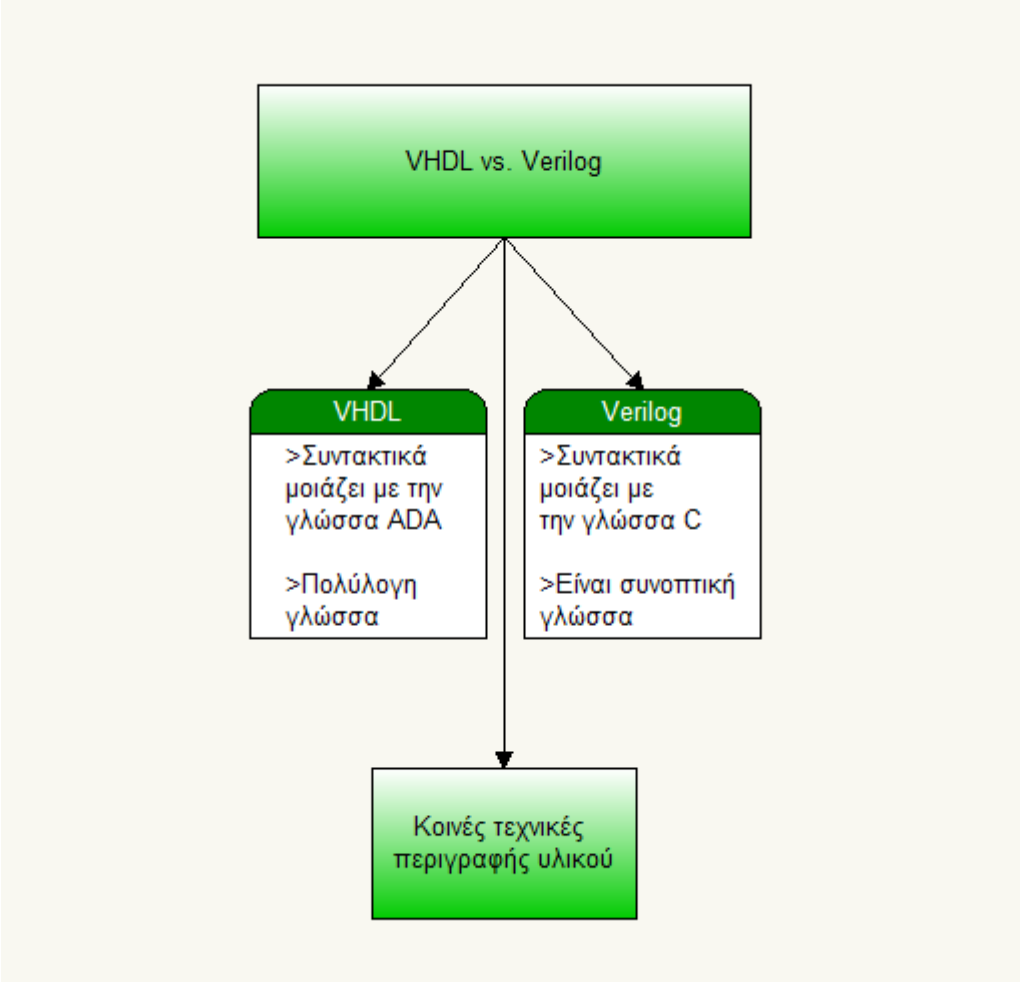
Ως είσοδο θα έχουμε θα έχουμε έναν μονοδιάστατο πίνακα τεσσάρων στοιχείων εφόσον ο δυαδικός αριθμός θα είναι τεσσάρων bit.

Ως σήμα εξόδου με όνομα display δηλώνουμε επίσης έναν μονοδιάστατο πίνακα επτά στοιχείων και τύπο αντικειμένου std_logic_vector. Δηλώνουμε την αρχιτεκτονική και μετά την διεργασία Process στην οποία χρησιμοποιούμε την δομή επανάληψης Case για την περιγραφή της κάθε περίπτωσης.

ΚΕΦΆΛΑΙΟ 4
VERILOG HDL

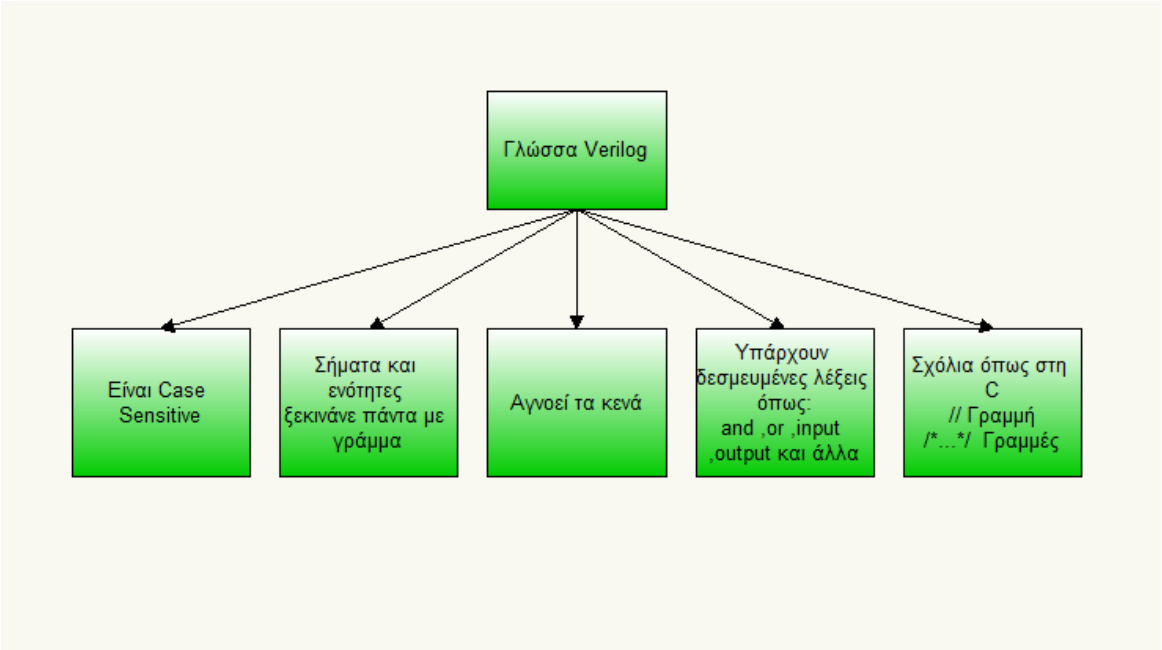
4.1. Γλώσσα Verilog

Η γλώσσα περιγραφής υλικού Verilog είναι μια από τις γνωστές και πιο σημαντικές στο χώρο της σχεδίασης των ψηφιακών συστημάτων .Όπως όλες η γλώσσες περιγραφής υλικού έτσι και η Verilog διαφέρει από της γλώσσες προγραμματισμού λογισμικού διότι περιγράφει την συμπεριφορά ενός κυκλώματος και του χρόνου που διαδίδονται τα σήματα σε αυτό .



Εικόνα 4.1 : VHDL vs. Verilog

Συντακτικά μοιάζει με τη γλώσσα C και είναι μια συνοπτική γλώσσα σε σχέση με την VHDL η οποία είναι πολύλογη .



Εικόνα 4.2 : Χαρακτηριστικά γλώσσας Verilog

Όπως και στη C η Verilog χρησιμοποιεί κάποιους συντακτικούς κανόνες όπως είναι ευαίσθητη στον διαχωρισμό των πεζών και κεφαλαίων (Case sensitive) ,όλες οι λέξεις κλειδιά είναι με πεζά,όλα τα σήματα και η ενότητες ξεκινάνε πάντα με γράμμα και ποτέ με αριθμό η άλλο σύμβολο. Η Verilog διαθέτει λέξεις οι οποίες είναι δεσμευμένες και δεν μπορούν να χρησιμοποιηθούν στον κώδικα εκτός από την ιδιότητα που έχουν ενώ αγνοεί τα κενά .

Τα σχόλια γίνονται με τον ίδιο ακριβώς τρόπο όπως και στην γλώσσα C,δηλαδή το σχόλιο μιας γραμμής γίνεται εισάγοντας στην αρχή του σχόλιου “//” ενώ το σχόλιο πολλών γραμμών εισάγοντας ‘’/*’’ στην αρχή του σχόλιου και ‘’*/’’ στο τέλος του σχολίου.

Στη γλώσσα Verilog μπορούμε να χρησιμοποιήσουμε ακεραίους αριθμούς (integer) προκαθορισμένου μεγέθους (Sized) η (Unsized) μη προκαθορισμένου μεγέθους, στην πρώτη περίπτωση δηλώνουμε το μέγεθος του αριθμού σε bits

μετά την μορφή του αριθμού ,για παράδειγμα αν θα είναι δυαδικό βάζουμε ένα b αν είναι δεκαεξαδικό βάζουμε h ,και στην συνέχεια γράφουμε τον αριθμό .

Στην δεύτερη περίπτωση όπου δεν προκαθορίζεται το μέγεθος των αριθμών δεσμεύεται μνήμη 32 Bit για κάθε μη προκαθορισμένο μεγέθους ακέραιο αριθμό. Μπορούμε να δηλώσουμε τους ακέραιους αριθμούς σε δυαδικό, οκταδικό ,δεκαδικό και δεκαεξαδικό. Οι πραγματικοί αριθμοί που υποστηρίζονται στην Verilog μπορεί να είναι σταθερές η μεταβλητές τιμές.

Η γλώσσα Verilog επίσης υποστηρίζει και τους δυο τύπους αριθμών ,προσημασμένους (Signed) και τους μη προσημασμένους (Unsigned) αριθμούς αλλά δεν τους δηλώνουμε με μορφή κειμένου αν είναι η δεν είναι προσημασμένο απλά γράφουμε το μέγεθος του σταθερού αριθμού και αν θέλουμε να έχει και αρνητικές τιμές τότε τοποθετούμε το αρνητικό πρόσημο πριν από το μέγεθος του αριθμού, αν δεν βάλουμε πρόσημο τότε ο αριθμός είναι μη προσημασμένος.

4.1.1. Ιστορία

Η γλώσσα περιγραφής υλικού Verilog είναι από τις πρώτες γλώσσες που κυκλοφόρησαν την δεκαετία του 80’ και δημιουργοί της ήταν ο Prabhu Goel και Phil Moorby .Τα αρχικά ονόματα ήταν Automated Integrated Design Systems και αργότερα Automated Integrated Design Systems ενώ στα τέλη της δεκαετίας του 80’ η Cadence Design System αγοράζει όλα τα δικαιώματα και γίνεται η πιο γνωστή και πιο χρήσιμη γλώσσα στον τομέα της ανάπτυξης ψηφιακών κυκλωμάτων και συστημάτων.

Έχουν κυκλοφορήσει διάφορες εκδόσεις της Verilog αφού η γλώσσα γινόταν όλο και πιο γνωστή στο ευρύ κοινό. Η Verilog 95 με πρότυπο IEEE 1364-1995 είναι μια έκδοση της Verilog αλλά αναγνωρίζεται ως αυτόνομη γλώσσα με υποσύνολα την Verilog AMS και Verilog A.

4.1.2 Verilog 2001

Η Verilog 2001 είναι μια αναβαθμισμένη έκδοση της Verilog -95 με σημαντικές αλλαγές ως προς την ευκολία στον χειρισμό προσημασμένων μεταβλητών . Σε αυτή την έκδοση υπάρχουν και οι τελεστές ελέγχου (case/if/else) όπως στην γλώσσα VHDL καθώς και εύχρηστες αλλαγές στον τρόπο σύνταξης .Η Verilog 2001 είναι η πιο διαδεδομένη έκδοση της γλώσσας Verilog και υποστηρίζεται από τις περισσότερες πλατφόρμες σχεδίασης ψηφιακών συστημάτων.

4.2 Εισαγωγή στην Verilog

Ένα πρόγραμμα στην Verilog ξεκινάει με την λέξη module και τελειώνει με την λέξη endmodule. Η module είναι η δομή παρόμοια με την οντότητα στην γλώσσα VHDL δηλαδή είναι σαν ένα μαύρο κουτί με εισόδους και εξόδους χωρίς να ξέρουμε την ακριβή λειτουργία του.

Κάθε module θα πρέπει να έχει και όνομα πού θα είναι και το όνομα του προγράμματος, στην συνέχεια δηλώνουμε τις πόρτες με τις δεσμευμένες λέξεις input και output μέσα σε παρενθέσεις ,κάθε input και output ακολουθείται από το αναγνωριστικό του. Τα αναγνωριστικά αποτελούνται πάντα από γράμμα η αριθμούς (A-Z και a-z) ,(0-9) η και το σύμβολο της κάτω παύλας (Underscore) θα πρέπει όμως να ξεκινάνε πάντα με γράμμα.

4.2.1 Τύποι δεδομένων στην Verilog

Οι δύο βασικοί τύποι δεδομένων στην Verilog είναι ο τύπος wire και ο τύπος reg .Ο τύπος δεδομένων wire χρησιμοποιείται για να συνδέσει δύο σημεία ενώ ο τύπος δεδομένων reg (register) χρησιμοποιείται για να αποθηκεύει τιμές.

ΚΕΦΑΛΑΙΟ 5
ΑΝΑΠΤΥΞΙΑΚΗ ΠΛΑΤΦΟΡΜΑ Quartus II

5.1 Το λογισμικό Quartus II

Η πλατφόρμα ανάπτυξης σχεδίου και προσομοίωσης Quartus II είναι ένα πολύ σημαντικό εργαλείο στον τομέα της σχεδίασης των ψηφιακών συστημάτων. Παρέχει στον χρήστη πάρα πολλές δυνατότητες για την σχεδίαση πολύπλοκων και σύνθετων συστημάτων που δεν θα μπορούσαν να σχεδιαστούν με άλλο τρόπο. Η πλατφόρμα Quartus II είναι το βασικό εργαλείο προγραμματισμού των ψηφιακών διατάξεων που κυκλοφορούν από την Altera είτε είναι CPLD η FPGA είτε για την δημιουργία σχεδίου που προορίζονται για ολοκληρωμένα κυκλώματα ASIC (application-specific integrated circuit).Εργαλεία σχεδιασμού όπως το Quartus II δίνουν την δυνατότητα να προσομοιώσει το κύκλωμα που σχεδιάζει σε λειτουργικό επίπεδο και χρονικό ο χρήστης .Παρέχει επίσης και την δυνατότητα αυτόματου εντοπισμού λαθών στην σχεδίαση είτε είναι με γλώσσες περιγραφής υλικού είτε είναι σχηματική σχεδίαση όπως και χρονική ανάλυση του κυκλώματος.

Το Quartus II περιορίζεται στον προγραμματισμό FPGA ,CPLD και αναπτυξιακών πλακετών FPGA όλων των μοντέλων και οικογενειών της εταιρίας Altera. Οι τρόποι σχεδίασης που υποστηρίζει το λογισμικό είναι η σχεδίαση με λογικά σχήματα,σύμβολα και εκφράσεις τα οποία η εφαρμογή τα διαθέτει σε μια πλούσια βιβλιοθήκη. Ο δεύτερος τρόπος είναι η σχεδίαση με τις γλώσσες περιγραφής υλικού ,οι γλώσσες που υποστηρίζονται είναι η Altera HDL (AHDL), η VHDL και η Verilog. Το Quartus II διαθέτει επίσης έναν διαδραστικό οδηγό εκμάθησης με οπτικοακουστικό υλικό μοιρασμένο σε ενότητες το οποίο βοηθάει τον χρήστη στην γρήγορη κατανόηση της λειτουργίας τον εργαλείων της εφαρμογής.

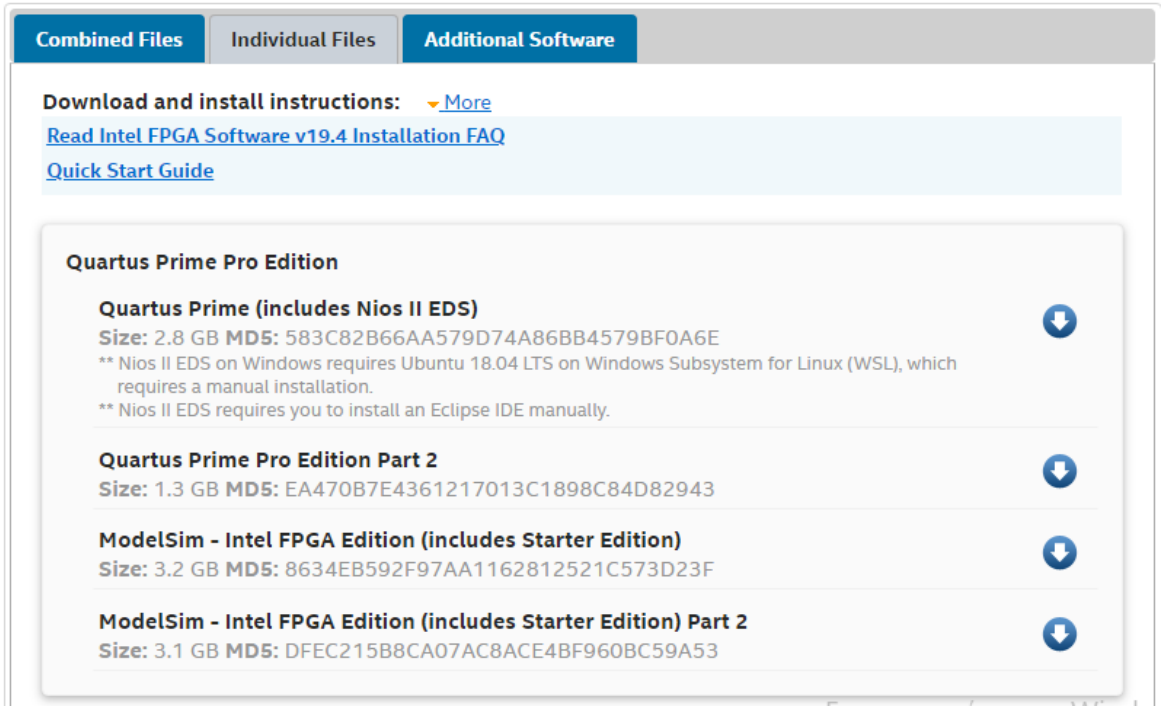
5.2. Βασικός οδηγός εγκατάσταση πλατφόρμας QuartusII

Το Quartus II είναι εφαρμογή η οποία δεν διατίθεται δωρεάν για λειτουργικό σύστημα Windows αν και κάποιες εκδόσεις της πλατφόρμας μπορείς να τις κατεβάσεις για δοκιμή 30 ημερών και δεν διαθέτουν όλες τις δυνατότητες της αλλά και πάλι είναι απαραίτητη η εγγραφή στην ιστοσελίδα της Intel η οποία είναι https://fpgasoftware.intel.com/ . Υπάρχουν πάρα πολλές εκδόσεις της εφαρμογής που διαθέτει η Intel μπορούμε να επιλέξουμε την έκδοση που είναι συμβατή με το λειτουργικό μας σύστημα. Η τελευταία έκδοση που κυκλοφορεί είναι η 19.4.

Β΄ΗΜΑ 1: Επιλέγουμε την έκδοση που θέλουμε,υπάρχουν και τρεις επιλογές ,όσο αναφορά τις δυνατότητες, Pro,Standard και Lite επιλέγουμε ανάλογα τις ανάγκες μας .

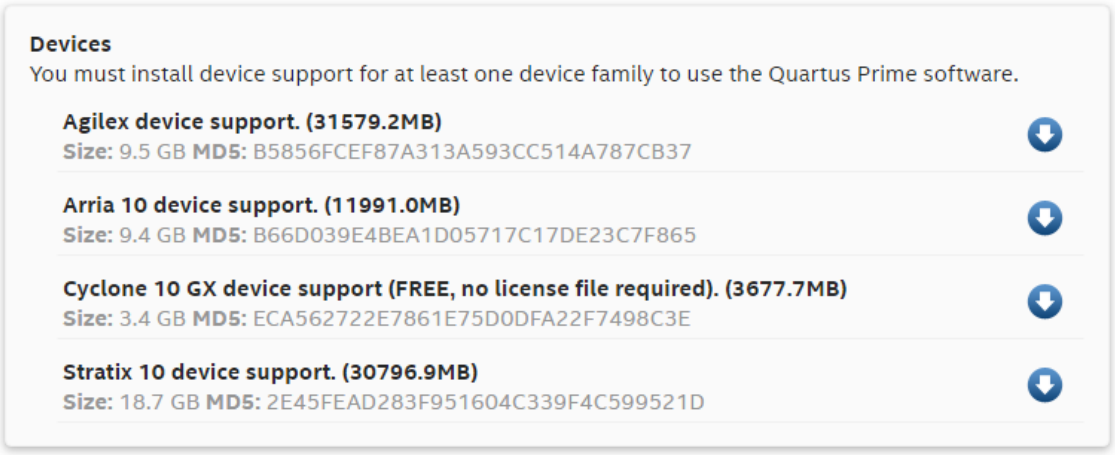
Εικόνα 5.1 : Εγκατάσταση Πλατφόρμας Quartus βήμα 1

ΒΗΜΑ 2: Στη συνέχεια επιλέγουμε το λογισμικό που θα κατεβάσουμε,στην περίπτωσή μας θα επιλέξουμε το Quartus Prime και το κατεβάζουμε.



Εικόνα 5.2: Εγκατάσταση Πλατφόρμας Quartus βήμα 2

ΒΗΜΑ 3:Για να χρησιμοποιήσουμε το Quartus θα πρέπει να έχουμε έχουμε εγκαταστήσει τουλάχιστον ένα λογισμικό υποστήριξης συσκευής οποιαδήποτε οικογένειας της Altera. Αν διαθέτουμε κάποια αναπτυξιακή πλακέτα της Altera κατεβάσουμε την συγκεκριμένη οικογένεια.



Εικόνα 5.3: Εγκατάσταση Πλατφόρμας Quartus βήμα 3

5.3 Σχεδίαση σε Quartus II

Στην αναπτυξιακή πλατφόρμα σχεδίασης Quartus 2 μας δίνεται η δυνατότητα να σχεδιάσουμε ένα ψηφιακό κύκλωμα με δύο τρόπους , ο ένας είναι ο σχηματικός όπου ο χρήστης τοποθετεί όλα τα στοιχεία και τις συνδέσεις του κυκλώματος ,ενώ ο δεύτερος τρόπος είναι η ανάπτυξη πηγαίου κώδικα σε γλώσσες περιγραφής υλικού στον οποίον περιγράφεται η συμπεριφορά του κυκλώματος .

5.3.1 Σχηματική σχεδίαση

Η σχηματική σχεδίαση ψηφιακών κυκλωμάτων είναι ένας εύκολος τρόπος σχεδίασης αλλά δεν είναι κατάλληλος για περίπλοκες εφαρμογές .Το Quartus 2 διαθέτει βιβλιοθήκες που υποστηρίζουν αυτόν τον τρόπο σχεδίασης. Οι βιβλιοθήκες περιέχουν σύμβολα από ηλεκτρονικά στοιχεία όπως πύλες ,εισόδους ,εξόδους και διάφορα άλλα. Για να σχεδιάσουμε σε σχηματική σχεδίαση στο Quartus 2 ανοίγουμε νέο αρχείο και επιλέγουμε από το παράθυρο που ανοίγει το “Block Diagram / Schematic File” ,στη συνέχεια ανοίγει το παράθυρο σχεδίασης ,στη πάνω μπάρα του παράθυρου βρίσκονται όλα τα σύμβολα - στοιχεία που χρειάζονται για να σχεδιάσουμε. Τα στοιχεία μεταξύ τους

συνδέονται από αγωγούς .Στο Quartus 2 υπάρχει δυνατότητα αποθήκευσης αρχείων που έχει δημιουργήσει ο χρήστης στη βιβλιοθήκη και η χρήση τους σε άλλα αρχεία.

5.3.2 Σχεδίαση με γλώσσες περιγραφής υλικού

Για να σχεδιάσουμε με γλώσσα περιγραφής υλικού σε Quartus 2 επιλέγουμε από το παράθυρο “New” τον τύπο αρχείου θα δημιουργήσουμε. Επιλέγουμε την γλώσσα που θέλουμε να γράψουμε ,VHDL ,Verilog κάποια άλλη HDL γλώσσα. Στη συνέχεια ανοίγει το παράθυρο που θα συντάξουμε τον κώδικα που θα περιγράφει το κύκλωμά μας . Η γλώσσες περιγραφής υλικού είναι κατάλληλες για την ανάπτυξη περίπλοκων κυκλωμάτων και σύνθετων συστημάτων,περιγράφει ένα λογικό κύκλωμα σε μορφή απλού κειμένου .

5.4 Ανάλυση και σύνθεση

Στη σχεδίαση με γλώσσες περιγραφής υλικού το τελικό σχέδιο προκύπτει από κάποιες αυτοματοποιημένες διαδικασίες του λογισμικού. Το ένα από αυτές είναι και η ανάλυση και σύνθεση. Στην ανάλυση το λογισμικό επεξεργάζει τον κώδικα για τυχών λάθη στην σύνταξή του τα οποία πρέπει να διορθώσει ο χρήστης. Στην σύνθεση σχεδιάζεται το λογικό κύκλωμα που περιγράφεται από τον κώδικα και ανάλογα την τεχνολογία της τελικής διάταξης διαμορφώνεται το τελικό σχέδιο. Με την τεχνολογία διάταξης η αρχιτεκτονική διάταξης εννοούμε αν ο κώδικας προορίζεται για CPLD διάταξη η FPGA διάταξη .Στην CPLD διάταξη οι βαθμίδες για την δημιουργία λογικών συναρτήσεων βασίζονται στους πίνακες που αποτελούνται από πύλες AND και OR. Στην FPGA διάταξη οι λογικές συναρτήσεις βασίζονται στους πίνακες αναφοράς (Look Up Tables LUT).Στην σύνθεση ουσιαστικά γίνεται μία αρχική προσομοίωση λειτουργίας, του βελτιστοποιημένου σε σχέση με το αρχικό, κυκλώματος και της αρχιτεκτονικής που προορίζεται αλλά χωρίς να υπολογίζεται ο χρόνος ως παράγοντας .

5.5 Τοποθέτηση και δρομολόγηση

Μετά την ανάλυση και την σύνθεση ,η επόμενη διαδικασία που ακολουθεί είναι η τοποθέτηση και δρομολόγηση (Place and Route) η διαδικασία προσαρμογής (Fitting).Αυτή η διαδικασία χρησιμοποιεί τα δεδομένα από την φάση της ανάλυσης και σύνθεσης για αντιστοίχιση των λογικών στοιχείων στην τελική διάταξη . Σε αυτή την διαδικασία συμπεριλαμβάνονται και οι χρονικές απαιτήσεις δηλωμένες στον κώδικα. Μετά την τοποθέτηση όλων των λογικών στοιχείων μέσα στην διάταξη ,το επόμενο βήμα είναι η σύνδεση μεταξύ τους (Routing) .

5.6. Χρονική ανάλυση

Στη διαδικασία της χρονικής ανάλυσης (Timing Analyzer) είναι η φάση στην οποία γίνεται ο έλεγχος στους χρόνους του κυκλώματος .Υπολογίζονται οι χρόνοι και οι καθυστερήσεις των σημάτων κατά την διάρκεια της εκτέλεσης της εφαρμογής. Οι καθυστερήσεις προκύπτουν από τις συνδέσεις των λογικών στοιχείων με αγωγούς και το μήκος τους. Οι καθυστερήσεις που δημιουργούνται δεν πρέπει να ξεπερνάνε τα όρια που έχουμε θέσει στο ρολόι του συστήματος,σε αντίθετη περίπτωση θα πρέπει να γίνει διόρθωση στη φάση της δρομολόγησης. Η χρονική ανάλυση όπως και η ανάλυση-σύνθεση ,τοποθέτηση-δρομολόγηση ανήκουν σε μία πιο γενική διαδικασία που ονομάζεται μεταγλώττιση (Compilation). Η μεταγλώττιση δημιουργεί το κατάλληλο αρχείο για το επόμενο στάδιο το οποίο είναι η προσομοίωση (Simulation).

5.7 Προσομοίωση

Στην φάση της προσομοίωσης γίνεται ο τελικός έλεγχος της σωστής λειτουργίας του κυκλώματος χωρίς το κύκλωμα να δέχεται κάποια βελτιστοποίηση. Στη προσομοίωση η λειτουργία του κυκλώματος θα πρέπει να έχει την ίδια συμπεριφορά με την συμπεριφορά που έχει σχεδιάσει ο χρήστης. Ο προσομοιωτής δεν λαμβάνει υπόψη τις καθυστερήσεις που προκύπτουν κατά την

εκτέλεση του ψηφιακού συστήματός,ο χρόνος που ένα σήμα χρειάζεται να πάει από την είσοδο μέχρι την έξοδο του συστήματος στην προσομοίωση είναι μηδενικός ενώ στην πραγματικότητα δεν ισχύει.

5.8 Προγραμματισμός της συσκευής

Η τελευταία φάση στην δημιουργία ενός ψηφιακού συστήματος είναι ο προγραμματισμός της συσκευής για την οποία προορίζεται σύστημα αυτό, είτε είναι CPLD είτε είναι FPGA. Για τον προγραμματισμό της συσκευής δημιουργείται ένα αρχείο .sof στην φάση της χρονικής ανάλυσης το οποίο φορτώνεται στην συσκευή CPLD η FPGA μέσο ενσύρματης σύνδεσης και της θύρας JTAG που διαθέτουν αυτές οι συσκευές. Στον υπολογιστή το καλώδιο προγραμματισμού, το οποίο διαθέτει οδηγό USB Blaster, συνδέεται σε θύρα USB .Όπως έχουμε πει και στο πρώτο κεφάλαιο τα CPLD διαθέτουν μνήμες EEPROM που σημαίνει ότι διατηρούν την διαμόρφωσή τους και μετά την διακοπή της τροφοδοσίας ενώ υπάρχει η δυνατότητα της επανεγγραφής της συσκευής. Στα FPGA αν διακοπεί η τροφοδοσία η διαμόρφωση της συσκευής χάνεται.

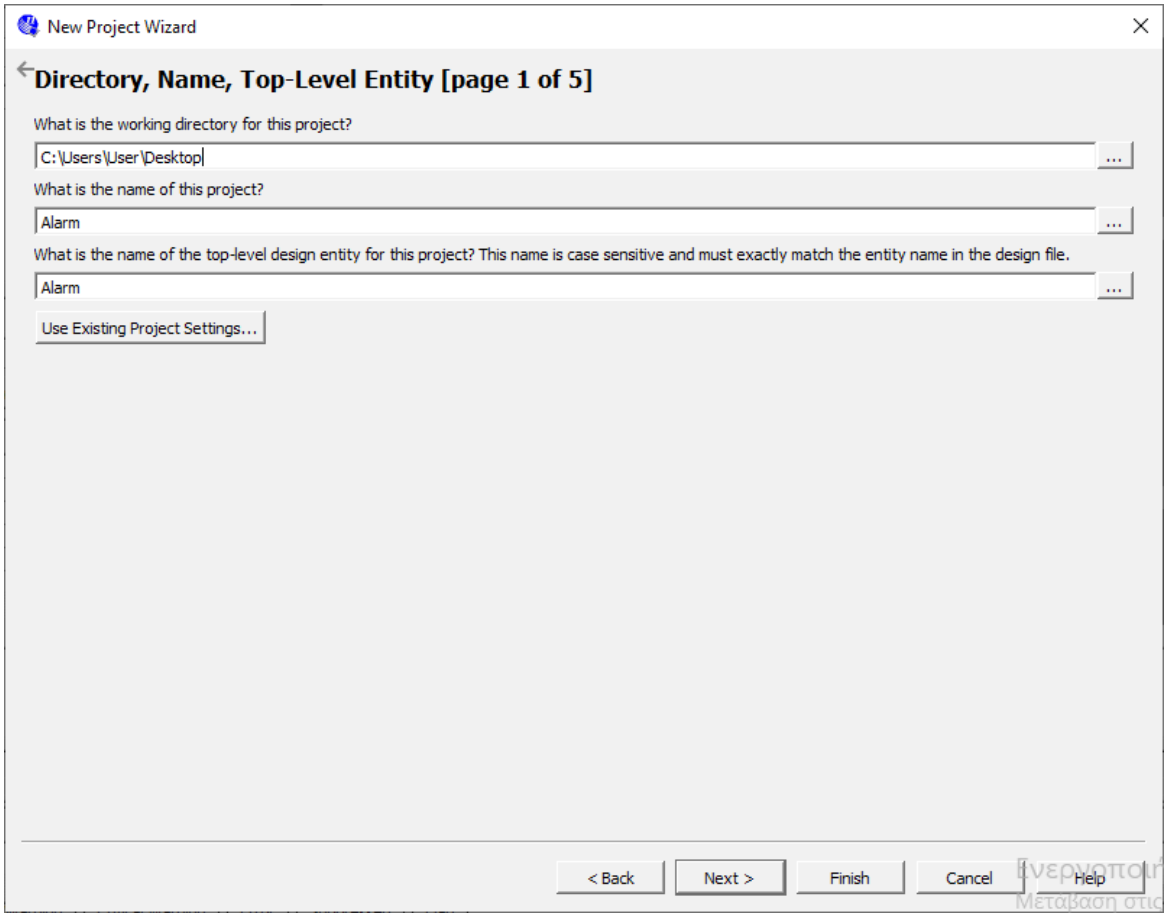
5.9 Ξεκινώντας με το Quartus II

Το αρχικό παράθυρο που εμφανίζεται στο Quartus φαίνεται στην παρακάτω εικόνα. Υπάρχει η επιλογή δημιουργίας νέου πρότζεκτ ,άνοιγμα υπάρχοντος η άνοιγμα εκπαιδευτικού προγράμματος.

Εικόνα 5.4: Βασικό περιβάλλον εργασίας του εργαλείου Quartus

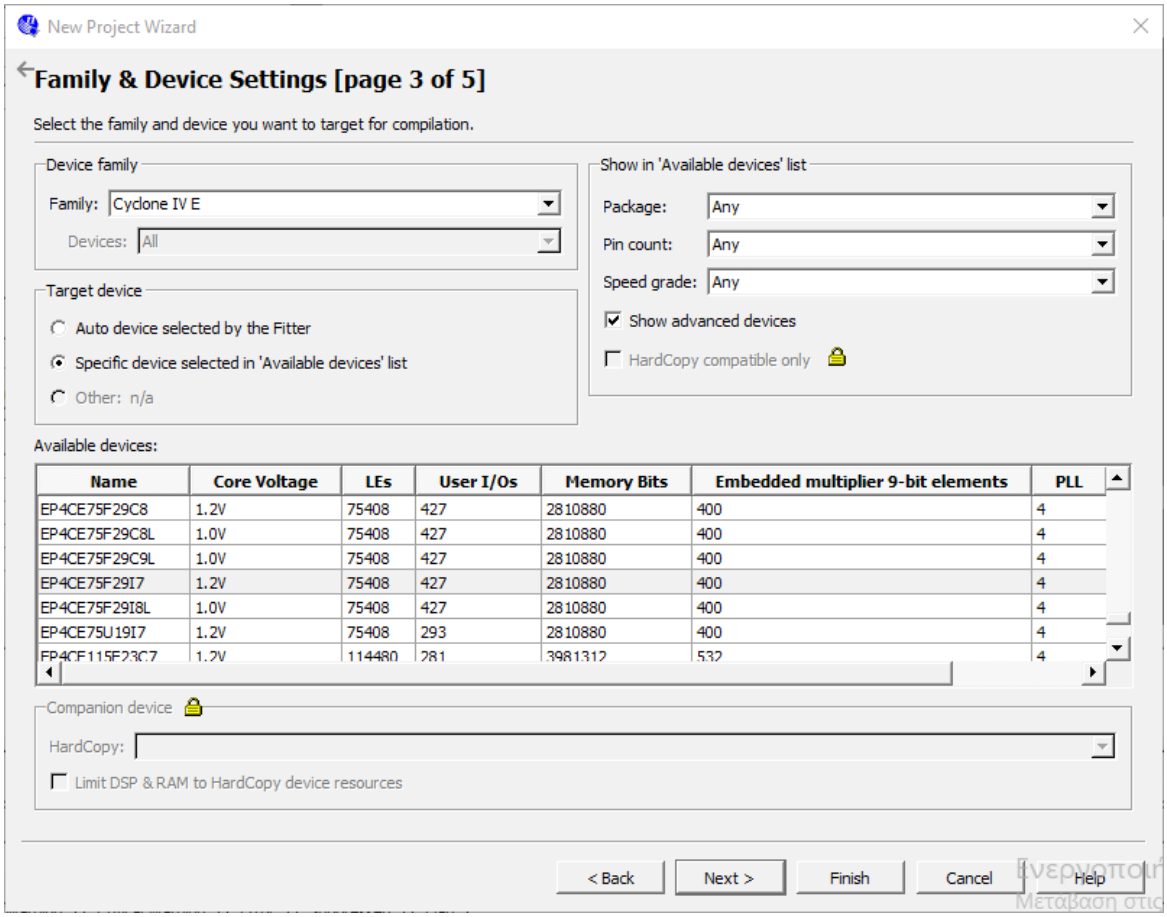
5.9.1. Δημιουργία νέου πρότζεκτ

Για την δημιουργία νέου πρότζεκτ στο Quartus II επιλέγουμε New Project από την επιλογή File η από το αρχικό παράθυρο του προγράμματός. Στη συνέχεια συμπληρώνουμε στα παράθυρα που ανοίγουν όλα τα στοιχειά του νέου πρότζεκτ που δημιουργούμε, όπως το όνομα ,η θέση που θα αποθηκευτεί ο φάκελος του πρότζεκτ και για ποια συσκευή προορίζεται το σχέδιο δηλαδή το μοντέλο του FPGA.



Εικόνα 5.5 : Νέο Project

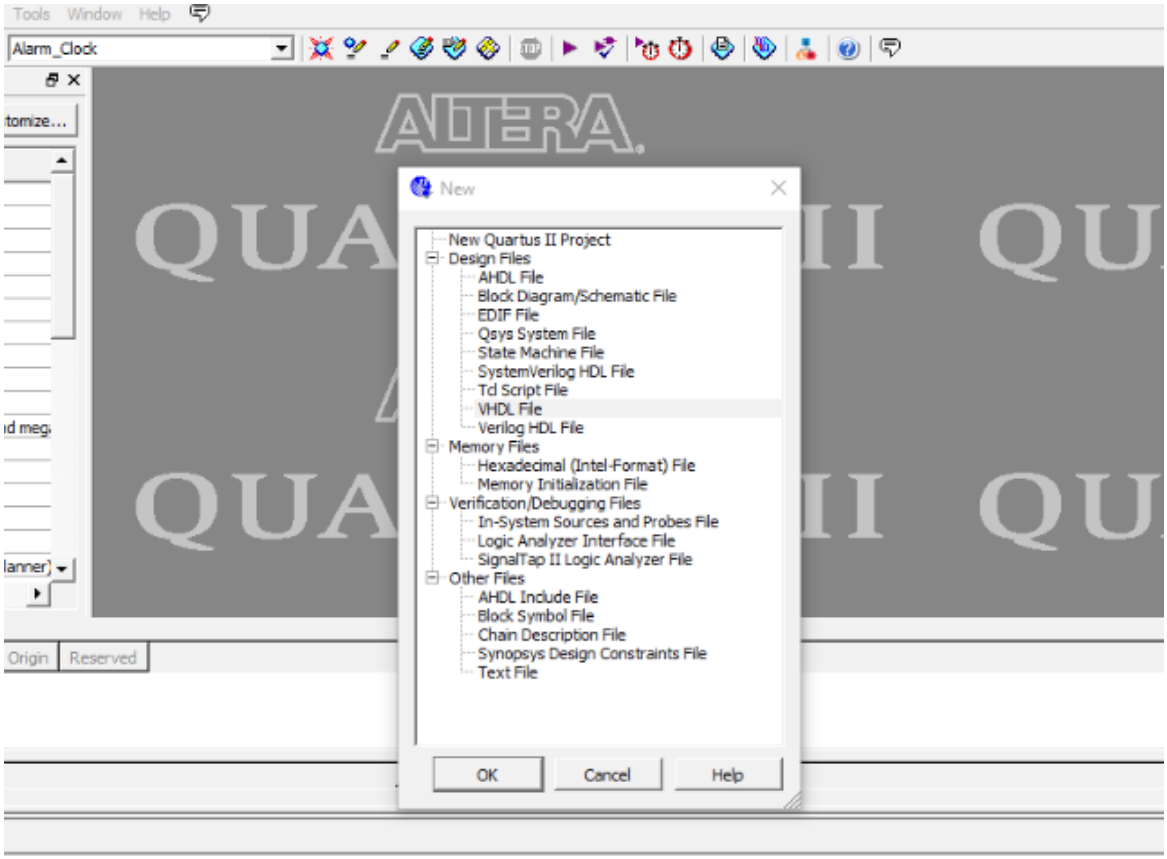
Μόλις τελειώσουμε με την διαδικασία δημιουργίας νέου πρότζεκτ θα έχει δημιουργηθεί ένας φάκελος με το όνομα του πρότζεκτ και θα περιέχει όλα τα αρχεία που δημιουργούνται κατά την σχεδίαση του ψηφιακού συστήματος.



Εικόνα 5.6 : Επιλογή Συσκευής

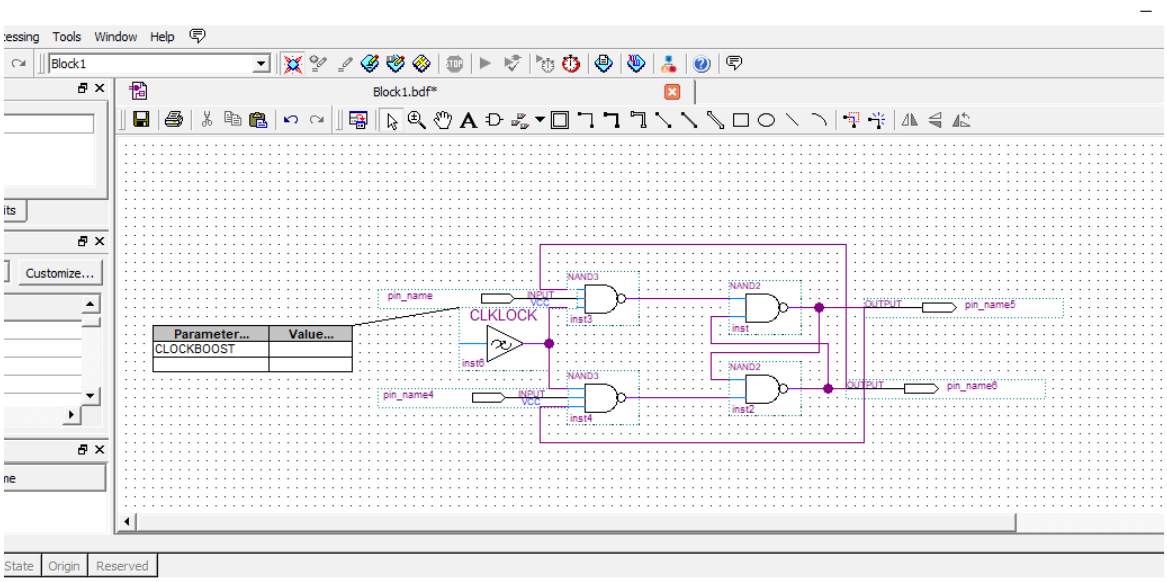
5.9.2 Επιλογή τρόπου σχεδίασης

Αν η δημιουργία του πρότζεκτ πραγματοποιήθηκε με επιτυχία στη συνέχεια θα επιλέξουμε με ποιόν τρόπο θα σχεδιάσουμε το κύκλωμά μας .Ο πρώτος τρόπος είναι η σχηματική σχεδίαση Block Diagram / Schematic File ενώ ο δεύτερος τρόπος είναι με την συγγραφή κώδικα σε γλώσσες περιγραφής υλικού όπως η VHDL και η Verilog. Για να επιλέξουμε τρόπο σχεδίασης πηγαίνουμε File → New.



Εικόνα 5.7 : Νέο VHDL Αρχείο

Εάν επιλέξουμε τον πρώτο τρόπο δηλαδή την περιγραφή του κυκλώματος με σχήματα η εισαγωγή συμβόλων γίνεται από την μπάρα εργαλείων επιλέγοντας το σύμβολο της λογικής πύλης . Στη συνέχεια ανοίγει ένα παράθυρο που περιέχει την βιβλιοθήκη συμβόλων του Quartus II. Υπάρχουν τρεις φάκελοι Megafunctions περιέχει κυκλώματα, αποκωδικοποιητές , Primitives περιέχει τις λογικές πύλες και others περιέχει διάφορα ολοκληρωμένα κυκλώματα όπως flip-flop και άλλα .



Εικόνα 5.8 : Σχηματική σχεδίαση κυκλώματος Flip-Flop

Σε περίπτωση που θέλουμε να περιγράψουμε το κύκλωμα μας με γλώσσες περιγραφής υλικού επιλέγουμε μία από αυτές για να ανοίξει το παράθυρου συγγραφής κώδικα.

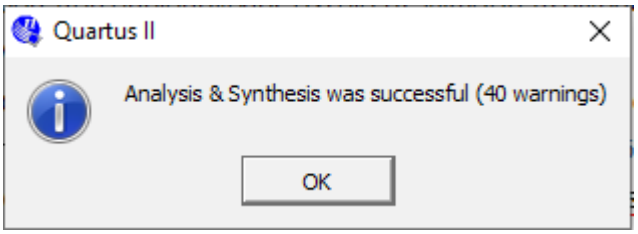
5.9.3 Μεταγλώττιση

Το επόμενο βήμα μετά την ολοκλήρωση της περιγραφής του κυκλώματος είναι η μεταγλώττιση του πηγαίου κώδικα η της σχηματικής σχεδίασης .Ο μεταγλωττιστής ελέγχει τον κώδικα η την σχηματική σχεδίαση για τυχόν λάθη και στην συνέχεια δημιουργούνται αρχεία για την χρονική ανάλυση ,την προσομοίωση και των προγραμματισμό του FPGA. Στην μεταγλώττιση εκτελούνται οι εξής διαδικασίες :

- Ανάλυση και σύνθεση (Analysis & Synthesis)
- Προσαρμογή (Fitter)

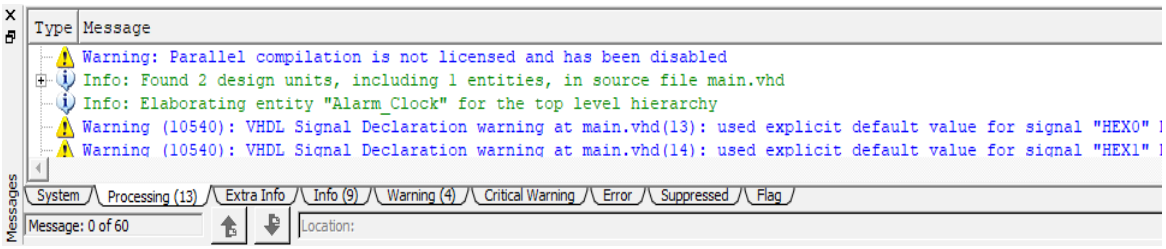
- Συναρμολόγηση (Assembler)
- Χρονική ανάλυση (Timing Analyzer)

Στην περίπτωση που δημιουργούμε σχέδιο με γλώσσα περιγραφής υλικού και θέλουμε να ελέγξουμε τον κώδικα για ορθογραφικά και συντακτικά λάθη χωρίς να κάνουμε μεταγλώττιση του κώδικα δηλαδή να εκτελεστούν όλες οι διαδικασίες της μεταγλώττισης που θα είχε ως συνέπεια την μεγαλύτερη χρονική διάρκεια μέχρι την εκτέλεση όλων των διαδικασιών για τον λόγο αυτό στο Quartus II υπάρχει η δυνατότητα ανάλυσης και ελέγχου του κώδικα εκτελώντας μόνο την φάση της ανάλυσης και σύνθεσης (Analysis and Synthesis).Για να επιλέξουμε αυτή την επιλογή πηγαίνουμε στο Processing → Start → Start Analysis and Synthesis η για πιο σύντομα πατάμε Ctrl + K. Όταν ολοκληρωθεί η διαδικασία της Ανάλυσης και Σύνθεσης εμφανίζεται μήνυμα από το Quartus II που μας ειδοποιεί αν η διαδικασία ήταν επιτυχής η όχι.



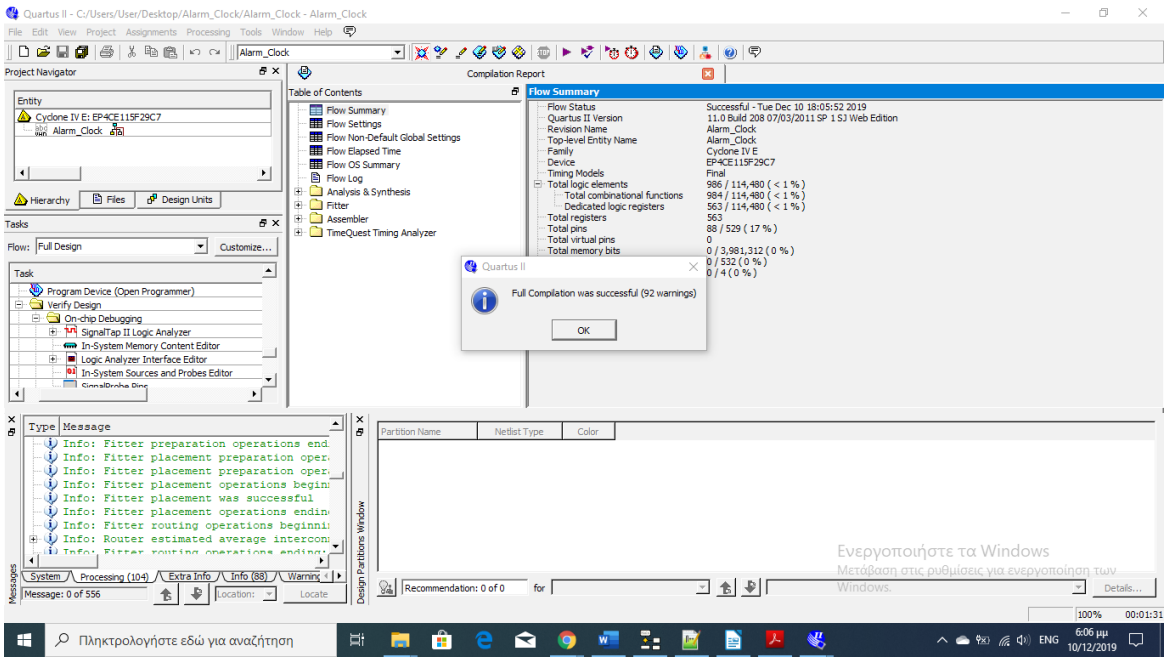
Εικόνα 5.9 : Μήνυμα Ανάλυσης και σύνθεσης

Στην συνέχεια πατάμε Ok για να κλείσει το μήνυμα και βλέπουμε στο κάτω μέρος της οθόνης του προγράμματος το παράθυρο μηνυμάτων της εφαρμογής που εμφανίζονται όλα τα μηνύματα κατά τη μεταγλώττιση του κώδικα. Υπάρχουν διάφορα είδη μηνυμάτων όπως φαίνονται στην παρακάτω εικόνα. Σε περίπτωση που ο κώδικας έχει σφάλματα (errors) η διαδικασία σταματάει και το πρόγραμμα μας εμφανίζει τον αριθμό σφαλμάτων του κώδικα.



Εικόνα 5.10 : Παράθυρο μηνυμάτων

Για διόρθωση σφαλμάτων πηγαίνουμε στην οθόνη μηνυμάτων και βρίσκουμε τα μηνύματα σφαλμάτων ,κάνουμε διπλό αριστερό κλικ στο μήνυμα λάθους και το ποντίκι πηγαίνει αυτόματα στην γραμμή που βρίσκεται το σφάλμα του κώδικα .Επαναλαμβάνουμε την διαδικασία μέχρι να διορθώσουμε όλα τα σφάλματα που έχει ο κώδικας. Κάποιες φορές όπως και σε άλλες γλώσσες προγραμματισμού όταν διορθώνουμε ένα σφάλμα να μειωθεί και ο συνολικός αριθμός των σφαλμάτων που έχει ο κώδικας. Αυτό συμβαίνει επειδή ένα συγκεκριμένο σφάλμα μπορεί να προκαλέσει κι άλλα σφάλματα σε διάφορα κομμάτια του κώδικα. Όταν ολοκληρωθεί η μεταγλώττιση το σύστημα μας εμφανίσει ένα παράθυρο με την αναφορά της μεταγλώττισης (Compilation Report)

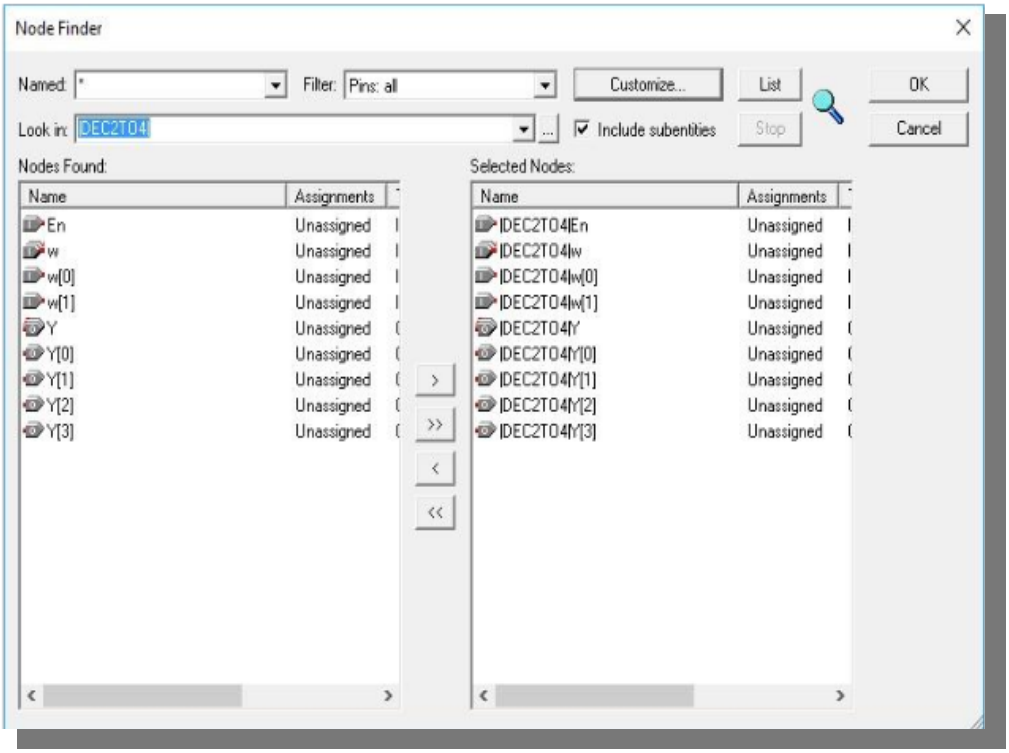


Εικόνα 5.11: Πληροφοριακά και στατιστικά στοιχεία για τη διαδικασία της μετάφρασης του κυκλώματος

5.9.4 Προσομοίωση

Σημαντικό εργαλείο του Quartus II είναι και η προσομοίωση που μας δίνει την δυνατότητα να δοκιμάσουμε το κύκλωμα που έχουμε σχεδιάσει σε επίπεδο προσομοίωσης και να ελέγξουμε την συμπεριφορά του .Για να κάνουμε προσομοίωση του κυκλώματος που έχουμε σχεδιάσει θα πρέπει πρώτα να δημιουργήσουμε τις εισόδους του κυκλώματος σε κυματομορφή .Για την δημιουργία κυματομορφών πηγαίνουμε File -> New -> Other Files -> Vector Wave form File.

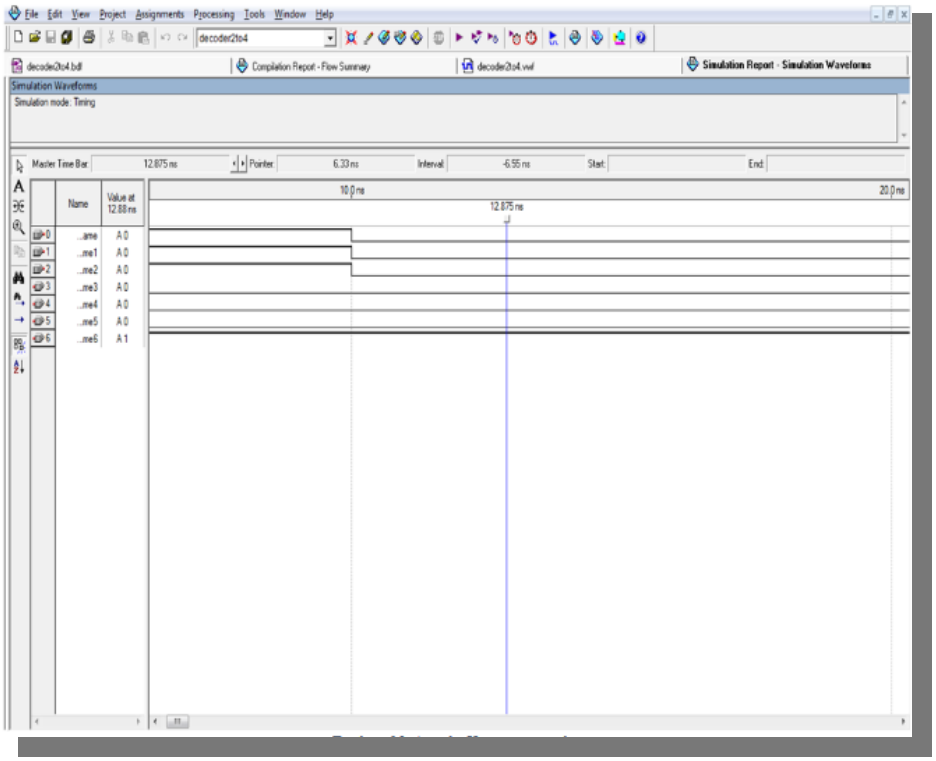
Για να κάνουμε προσομοίωση του κυκλώματος πρέπει να εισάγουμε τους κόμβους εισόδου και εξόδου του ψηφιακού κυκλώματος ,αυτό το πετυχαίνουμε από το παράθυρο Node Finder όπως φαίνεται και στην παρακάτω εικόνα.



Εικόνα 5.12 : Node finder

Αποθηκεύουμε το αρχείο με τις κυματομορφές εισόδου (Wave form vector file) με όνομα αυστηρά ίδιο με το όνομα της οντότητας .Στη συνέχεια πηγαίνουμε να εκτελέσουμε την προσομοίωση από το βασικό μενού επιλέγουμε Processing και στην συνέχεια Start Simulation. Όταν ολοκληρωθεί η διαδικασία της προσομοίωσης ανοίγει ένα παράθυρο αναφοράς προσομοίωσης (Simulation

Report) το οποίο μας παρουσιάζει στατιστικά στοιχειά από την προσομοίωση του κυκλώματος καθώς και αποτελέσματα λειτουργίας του ψηφιακού κυκλώματος σε επίπεδο προσομοίωσης. Μπορούμε να διασταυρώσουμε τα αποτελέσματα της προσομοίωσης με τα θεωρητικά αποτελέσματα από τους πίνακες αληθείας.

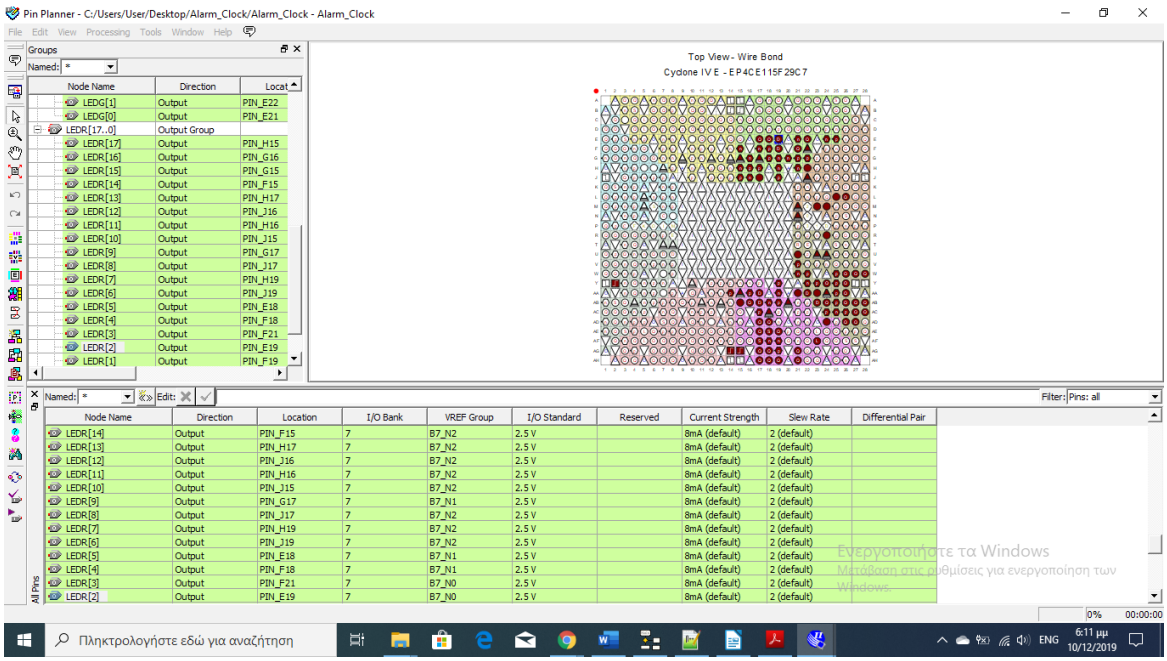


Εικόνα 5.13: Αποτελέσματα της προσομοίωσης της λειτουργίας του κυκλώματος

5.9.5 Ορισμός ακροδεκτών

Μετά την διαδικασία της προσομοίωσης η επόμενη φάση είναι αντιστοίχιση των εισόδων και εξόδων του ψηφιακού κυκλώματος που σχεδιάσαμε με τους φυσικούς ακροδέκτες της διάταξης που προορίζεται το σχέδιο. Ανάλογα το μοντέλο της διάταξης γίνεται και η ανάθεση ακροδεκτών, πηγαίνουμε στο βασικό μενού και επιλέγουμε Assignments → Pins. Στο παράθυρο που ανοίγει υπάρχουν

τρία πεδία,το ένα πεδίο που βρίσκετε στο κάτω μέρος του παραθύρου και εμφανίζονται όλοι οι ακροδέκτες και όλα τα στοιχεία τους όπως το όνομα,τον τύπο του σήματος (εισόδου η εξόδου),την διεύθυνση του κάθε ακροδέκτη κ.α. Το δεύτερο πεδίο είναι εμφανίζονται τα σήματα εισόδου -εξόδου σε ομάδες ενώ το τρίτο πεδίο εμφανίζει την διάταξη σε μορφή κάτοψης και εμφανίζει τις περιοχές που χρησιμοποιούνται. Μπορούμε να πατήσουμε πάνω σε δεσμευμένες περιοχές της διατάξεις και να μας εμφανίσει όλα τα στοιχεία του ακροδέκτη .

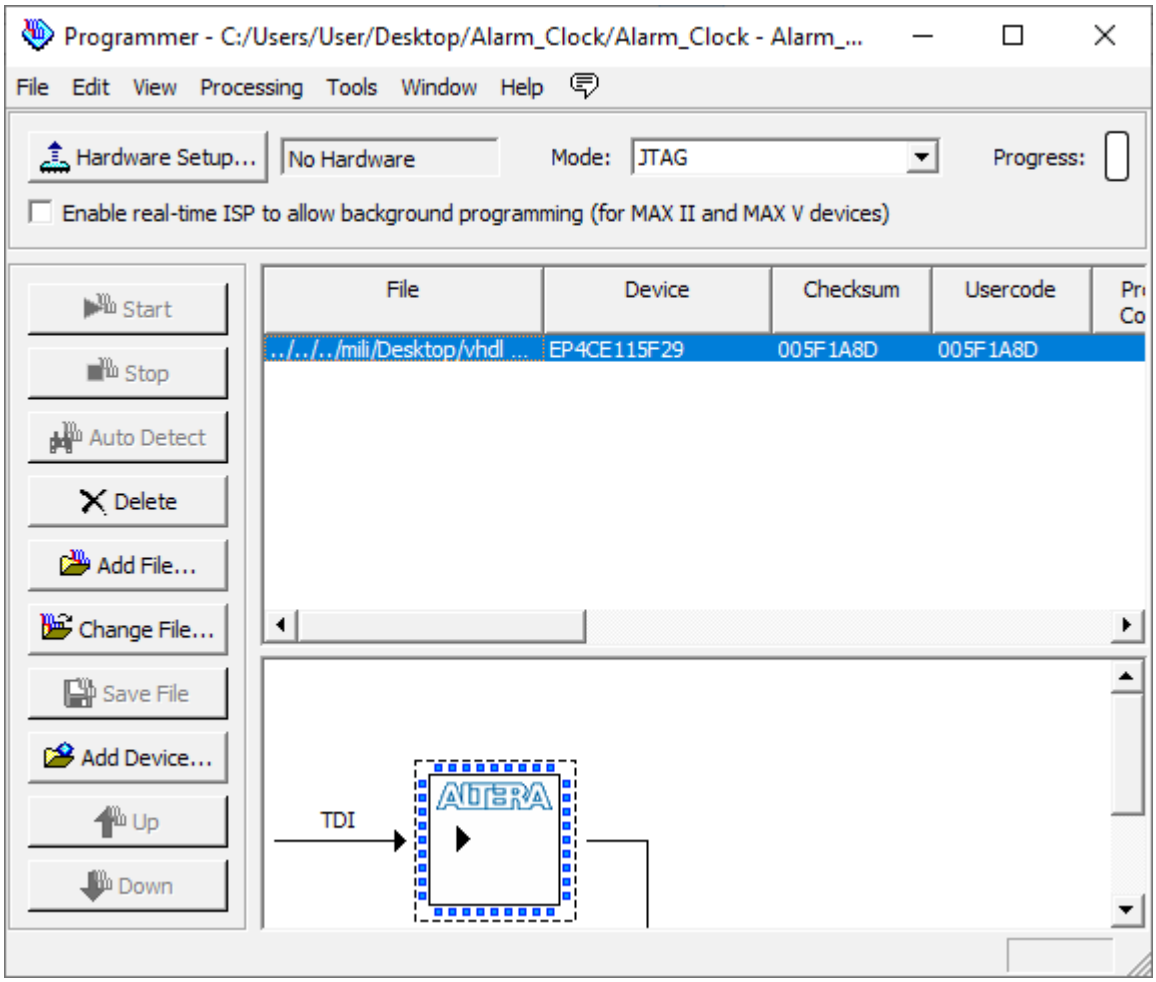


Εικόνα 5.14: Ανάθεση ακροδεκτών

5.9.6 Προγραμματισμός κυκλώματος

Το τελευταίο βήμα για την διαμόρφωση της διάταξης είναι ο προγραμματισμός της ,δηλαδή η φόρτωση του αρχείου .sof που δημιουργείται και περιέχει τις απαραίτητες πληροφορίες για την διαμόρφωση του κυκλώματος. Για να ξεκινήσουμε την διαδικασία του προγραμματισμού του FPGA θα πρέπει πρώτα να έχουμε συνδέσει το FPGA με τον υπολογιστή μέσο θύρας USB και να διαθέτει

οδηγό (driver) συνήθως χρησιμοποιείται το USB Blaster ενώ η θύρα στο FPGA είναι JTAG.



Εικόνα 5.15 : Programmer

Στην συνέχεια κι αφού έχουμε κάνει τις απαραίτητες συνδέσεις μεταξύ του υπολογιστή και του FPGA προχωράμε στη διαδικασία προγραμματισμού της διάταξης. Μπορούμε να ανοίξουμε το Programmer με δύο τρόπους, ο πρώτος πηγαίνοντας στο βασικό μενού επιλέγουμε Tools → Programmer. 'Ο δεύτερος τρόπος είναι από το εικονίδιο συντόμευσης στο μενού εικονιδίων που βρίσκετε

κάτω ακριβώς από το βασικό μενού. Στη συνέχεια ανοίγει το παράθυρο Programmer όπου γίνονται οι απαραίτητες ρυθμίσεις όπως η επιλογή της θύρας του FPGA ,η ρύθμισή του και διάφορες άλλες επιλογές.

5.10 Απλό παράδειγμα σε Quartus

Σε αυτή την ενότητα θα παρουσιάσουμε ένα απλό παράδειγμα υλοποίησης ενός κυκλώματος JK flip-flop παρακάτω πίνακα βλέπουμε τον πίνακα αληθείας του J-K flip flop και στον επόμενο τον χαρακτηριστικό του πίνακα.

J	K	Qt	Qt+1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Πίνακας 5.1 : J-K flip flop πίνακας αληθείας

J	K	Qt+1
0	0	Qt
0	1	0
1	0	1

1	1	Qt’

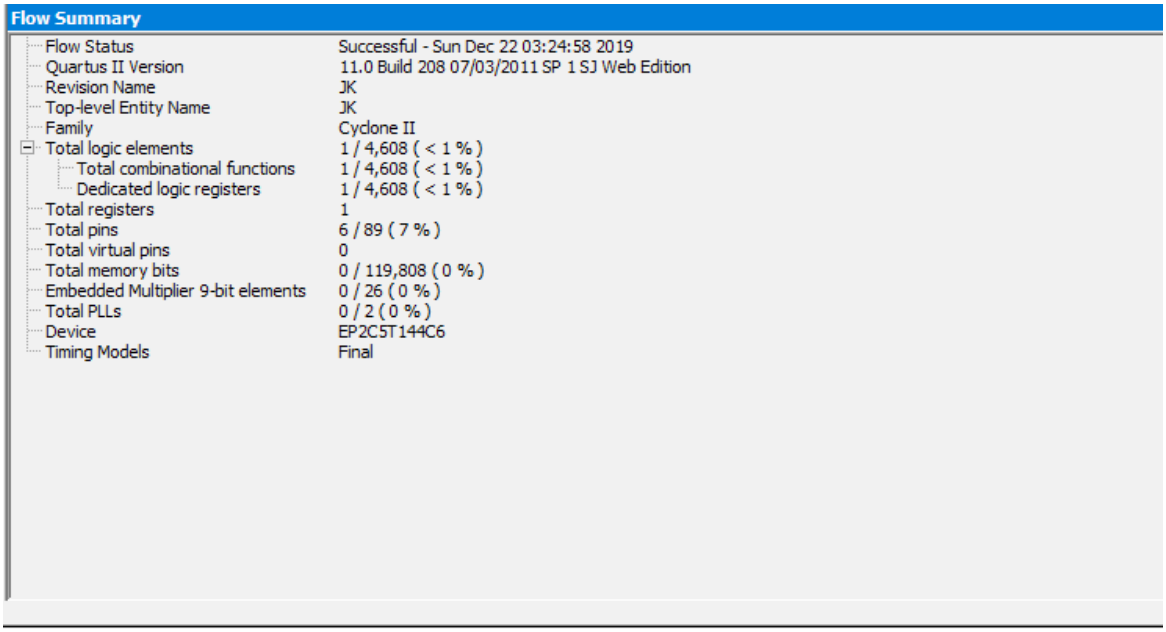
Πίνακας 5.2 : Χαρακτηριστικός πίνακας J-K flip flop

Για την δημιουργία νέου πρότζεκτ ακολουθούμε τα βήματα που ακολουθήσαμε στις προηγούμενες ενότητες. Όπως ξέρουμε ένα κύκλωμα J-K Flip-Flop αποτελείται από τέσσερεις εισόδους και δύο εξόδους . Στο παρακάτω κείμενο περιγράφεται το κύκλωμα J-K Flip-Flop σε γλώσσα VHDL.

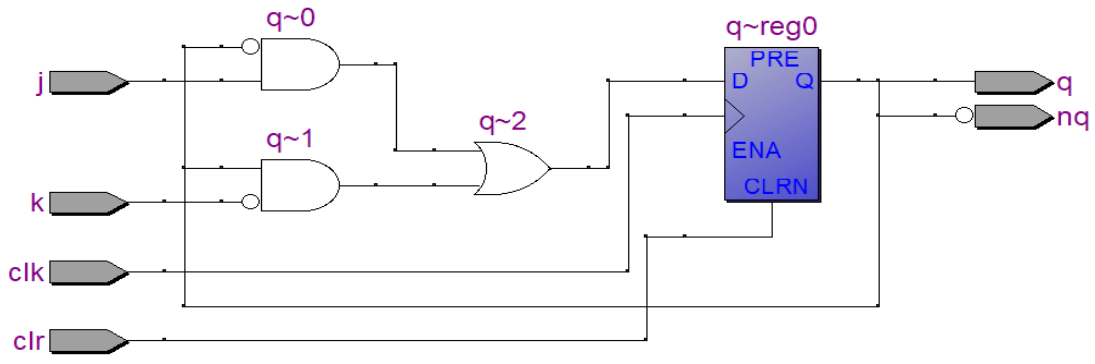
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity JK is
 Port ( j: in  STD_LOGIC;
              k : in  STD_LOGIC;
              clk  : in STD_LOGIC;
              clr: in STD_LOGIC;
              q: buffer STD_LOGIC;
              nq: buffer STD_LOGIC
              );
end JK;
architecture Behavioral of JK is
begin
    process(clk,clr)
       begin
           if  (clr = '1') then
               q <= '0';
           elsif (clk'event and clk = '1') then
               q <= (j and not q) or (not k and q);
           end if;
    end process;
  nq <= not q;
```

```
end Behavioral;
```

Στην συνέχεια κάνουμε compile τον κώδικα:



Εικόνα 5.16 : Αποτελέσματα μετά την μεταγλώττιση

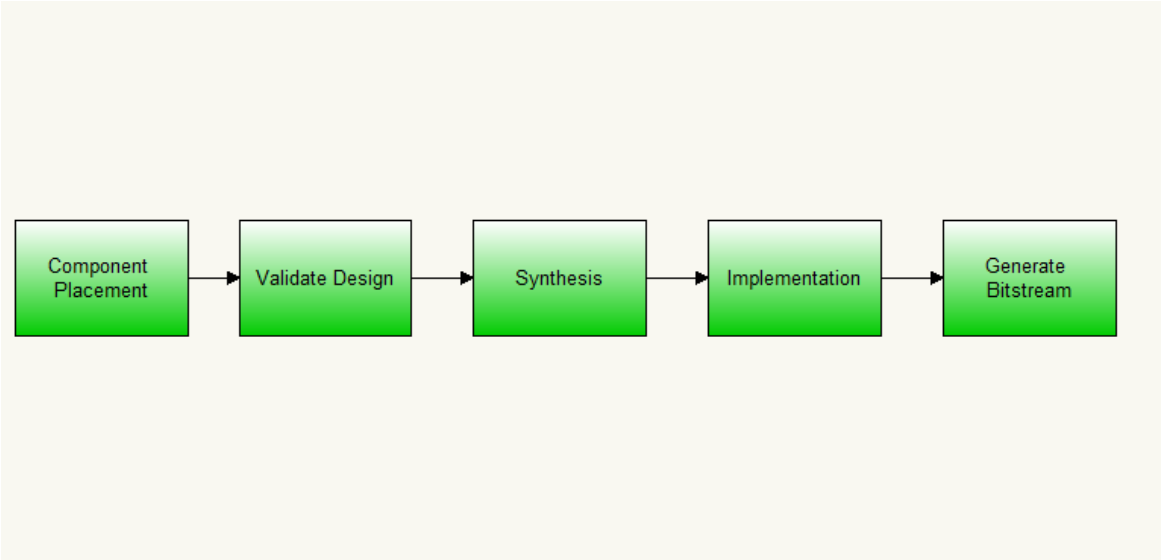


Εικόνα 5.17 : Τελικό κύκλωμα σε επίπεδο RTL

ΚΕΦΑΛΑΙΟ 6
ΑΝΑΠΤΥΞΙΑΚΗ ΠΛΑΤΦΟΡΜΑ ΣΧΕΔΙΑΣΗΣ VIVADO

6.1 Vivado Design Suite

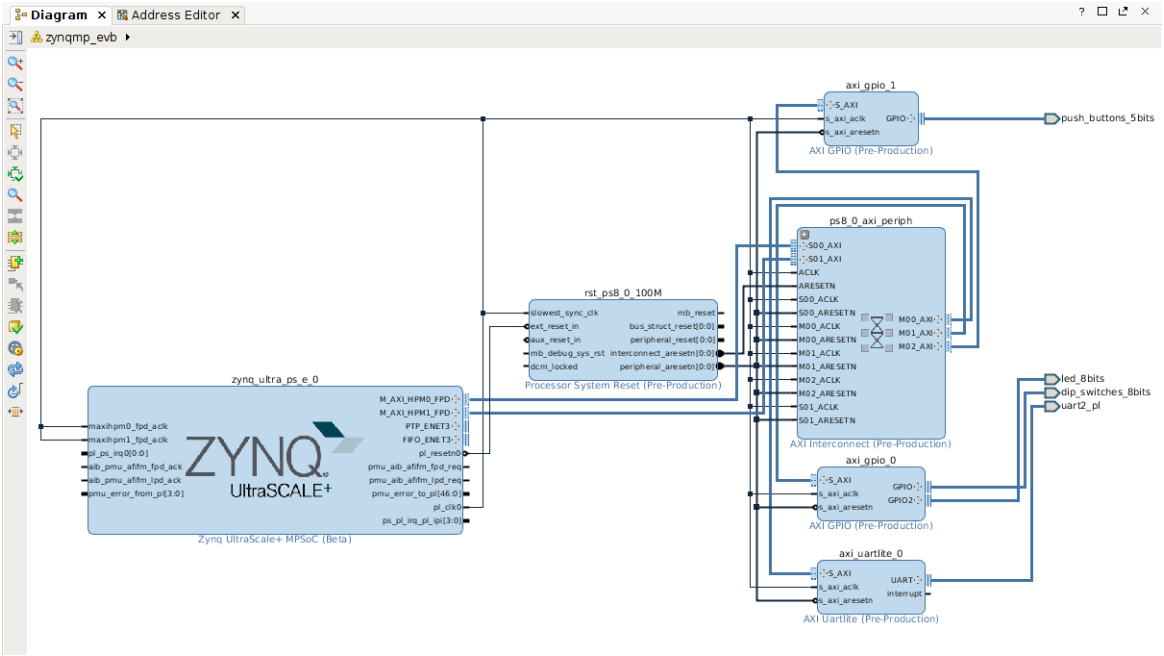
Το Vivado Design Suite είναι μία αναπτυξιακή πλατφόρμα σχεδίασης ψηφιακών συστημάτων η οποία, όπως και τα άλλα εργαλεία που διαθέτει , έχει δημιουργηθεί και συντηρείται από την εταιρία Xilinx η οποία είναι από τις πρωτοπόρες στην παραγωγή FPGAs . Είναι ένα εργαλείο, το οποίο συνθέτει και αναλύει σχεδιασμούς γλώσσας υλικό εφαρμογή δίνει την δυνατότητα στον χρήστη να κάνει σύνθεση , προσομοίωση και υλοποίηση του σχεδίου ενώ η γραφή διεπαφή (GUI) την κάνει εύκολη στην χρήση ενώ υπάρχει δυνατότητα ο χρήστης να χρησιμοποιήσει την γλώσσα TCL (ToolCommandLanguage) για εισαγωγή εντολών από την γραμμή εισαγωγής εντολών η με εκτέλεση αρχείων TCL . Το Vivado Design Suite προορίζεται για τον προγραμματισμό CPLD,FPGA και αναπτυξιακές πλακέτες όλων των μοντέλων και οικογενειών της εταιρίας Xilinx.



Εικόνα 6.1: Διαδικασία υλοποίησης ενός συστήματος στο Vivado

Το Vivado Design Suite υποστηρίζει σχεδίαση με γλώσσες περιγραφής υλικού όπως η VHDL και η Verilog οι οποίες είναι και η βασικές γλώσσες περιγραφής υλικού.

Ο δεύτερος τρόπος σχεδίασης ενός ψηφιακού συστήματος στο Vivado Design Suite είναι η σχηματική σχεδίαση δηλαδή το κύκλωμα περιγράφεται σε μορφή μπλοκ διαγράμματος . Vivado Design Suite διαθέτει έτοιμες βιβλιοθήκες από IP Blocks ενώ μας δίνει την δυνατότητα να δημιουργήσουμε δικές μας IP Blocks από εργαλεία που διαθέτει η εφαρμογή. Ένα παράδειγμα μπλοκ διαγράμματος:



Εικόνα 6.2 : Μπλοκ διαγράμματος στο Vivado

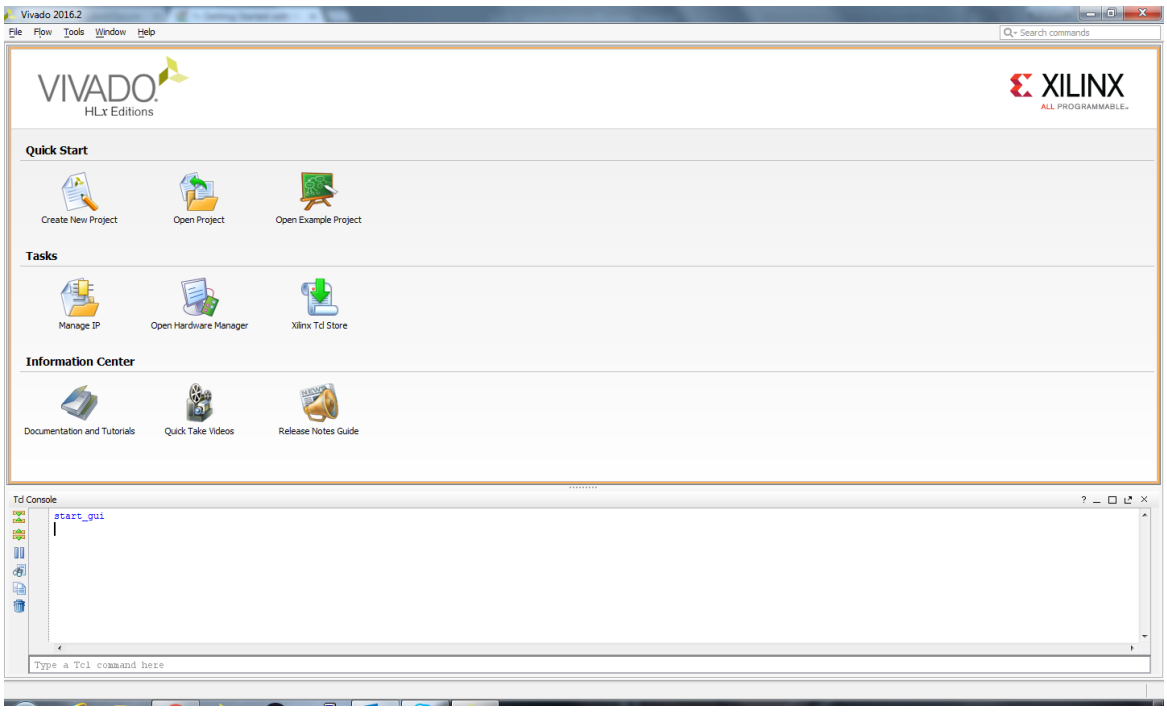
Πριν σχεδιάσουμε το σύστημα πρέπει να εισάγουμε κάποιους περιορισμούς (constraints) που έχουν σχέση με τους χρονισμούς και με τους ακροδέκτες του ψηφιακού συστήματος.

Η επόμενη φάση είναι ο έλεγχος που γίνεται στο σχεδιασμένο ψηφιακό σύστημα (design validation) .Σε αυτή την φάση της σχεδίασης γίνεται ο έλεγχος για λάθη στην σχεδίαση στην σχεδίαση του κυκλώματος και αφού διορθωθούν

περνάμε στην επόμενη φάση η οποία είναι η σύνθεση (synthesis) η οποία μετατρέπει τα αρχεία από τις γλώσσες περιγραφής υλικού σε λίστες συνδέσεων πυλών (gate level netlist). Αν σε αυτή την φάση της σχεδίασης δεν υπάρχουν σφάλματα τότε το σχέδιο είναι σε επίπεδο λογικών πυλών και είμαστε έτοιμοι για την υλοποιήσει του.

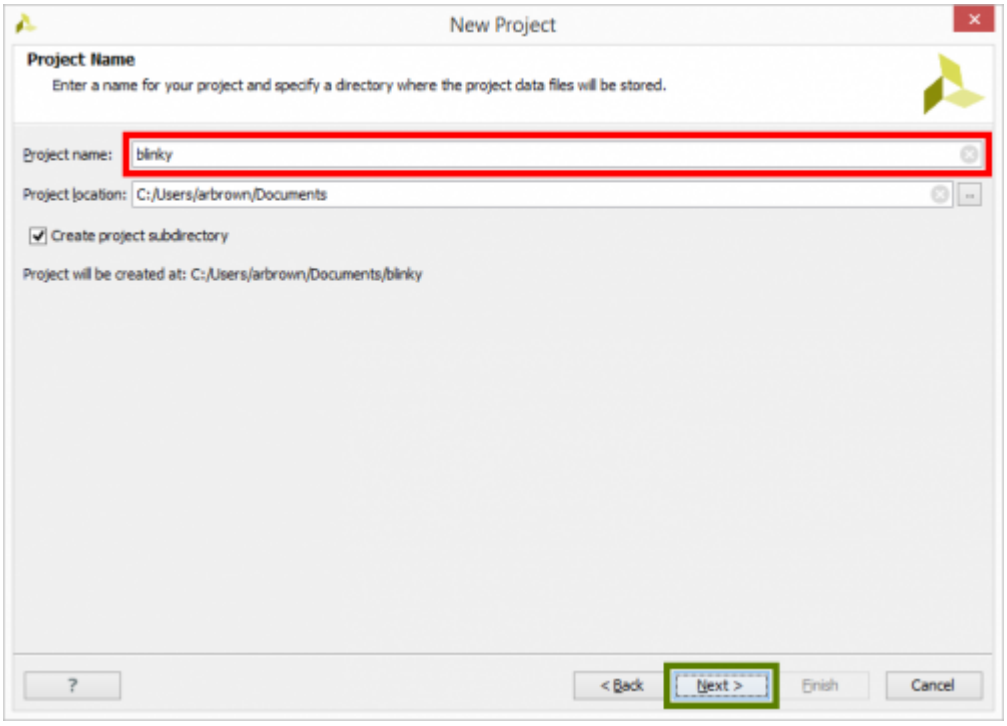
6.2. Ξεκινώντας με την πλατφόρμα Vivado

Τα πιο βασικά κουμπιά στην αρχική οθόνη της εφαρμογής είναι “Create new project” ,”Open project”,”Open Example Project” και “Open Hardware Manager” δηλαδή δημιουργία νέου σχέδιο,άνοιγμα υπάρχοντος η άνοιγμα παραδείγματος. Το “Open Hardware Manager” συνδέει το λογισμικό με το FPGA για φόρτωση του σχέδιο στην πλακέτα.



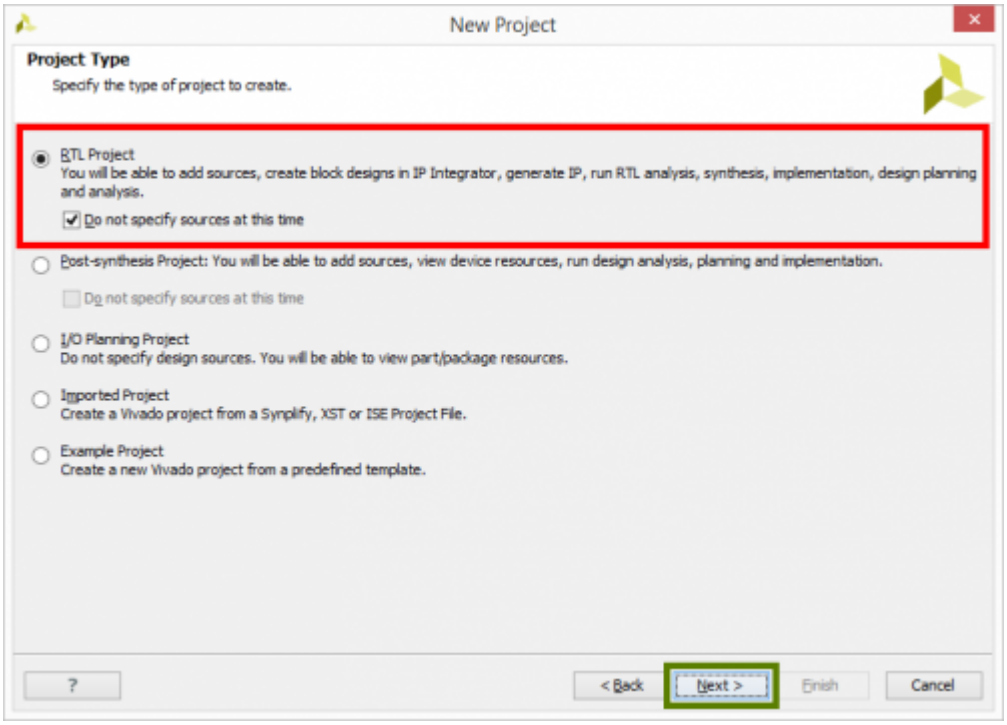
Εικόνα 6.3 : Αρχική οθόνη λογισμικού Vivado

Δημιουργούμε νέο πρότζεκτ επιλέγοντας “Create new project” και στην συνέχεια πατάμε “Next” στο παράθυρο που εμφανίζεται. Στην συνέχεια ονομάζουμε το σχέδιο που θα δημιουργήσουμε και επιλέγουμε που ακριβώς θέλουμε να το αποθηκεύσουμε. Δεν βάζουμε κενό χαρακτήρα στο όνομα η στο χώρο αποθήκευσης.



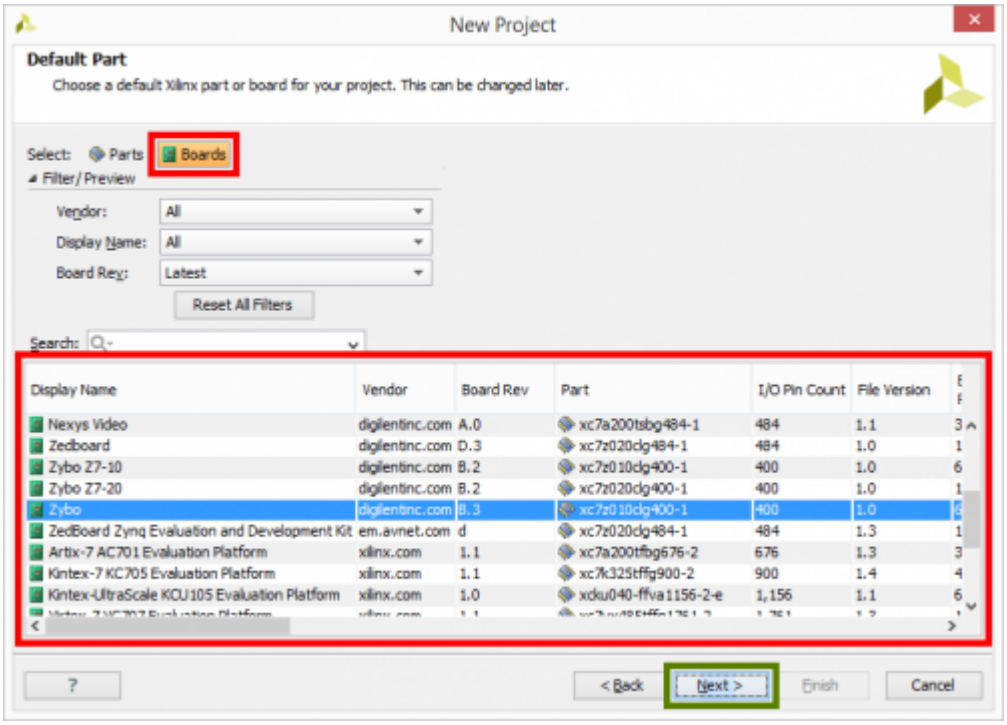
Εικόνα 6.4 : Δημιουργία νέου σχέδιο

Στο επόμενο παράθυρο πρέπει να επιλέξουμε τον τύπου του σχέδιο θέλουμε. Επιλέγουμε το RTL project και το Do not specify sources at this time ώστε ο πηγαίος κώδικας να προστεθεί μετά την δημιουργία του σχέδιο στη συνέχεια πατάμε Next.



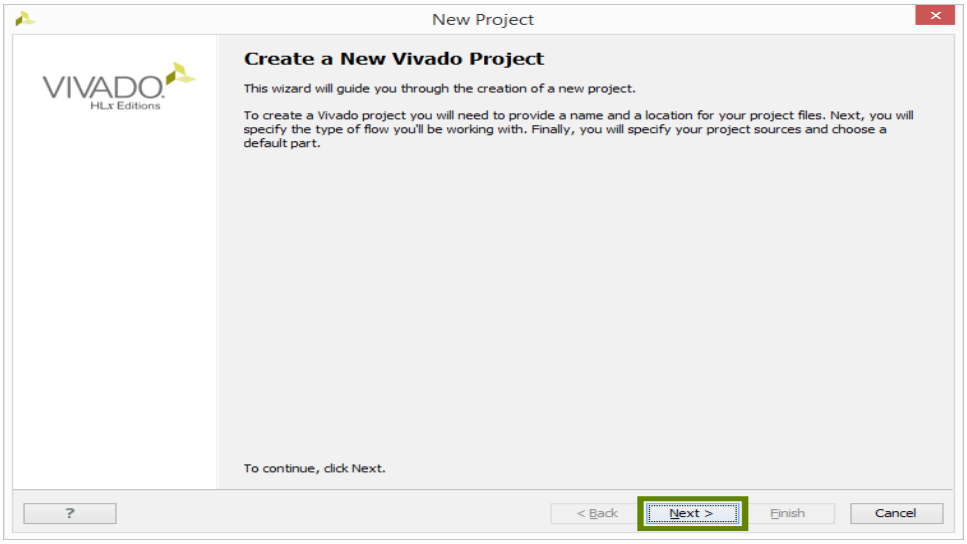
Εικόνα 6.5 : Επιλογή τύπου σχέδιο

Στο επόμενο παράθυρο που εμφανίζεται θα πρέπει να επιλέξουμε τον τύπο της πλακέτας που θα φορτωθεί το πρόγραμμα. Σε περίπτωση που δεν υπάρχει στην λίστα θα πρέπει να το κατεβάσουμε.



Εικόνα 6.6 : Επιλογή τύπου πλακέτας

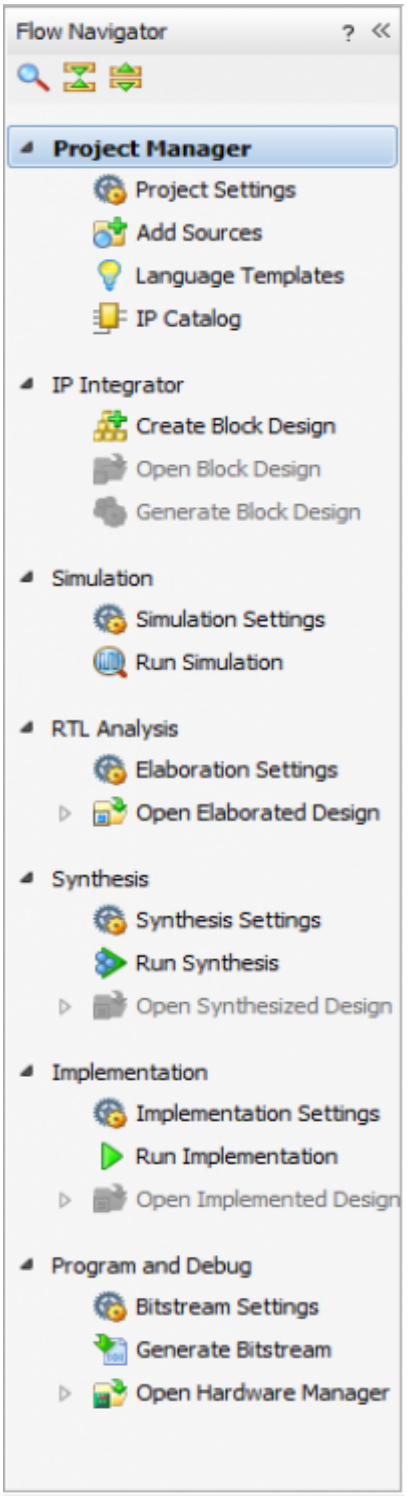
Στο τελευταίο παράθυρο που εμφανίζεται γίνεται μία σύνοψη των επιλογών που κάναμε .Ελέγχουμε αν όλα είναι εντάξει και πατάμε Finish.



Εικόνα 6.7 :Έλεγχος επιλογών του σχέδιο

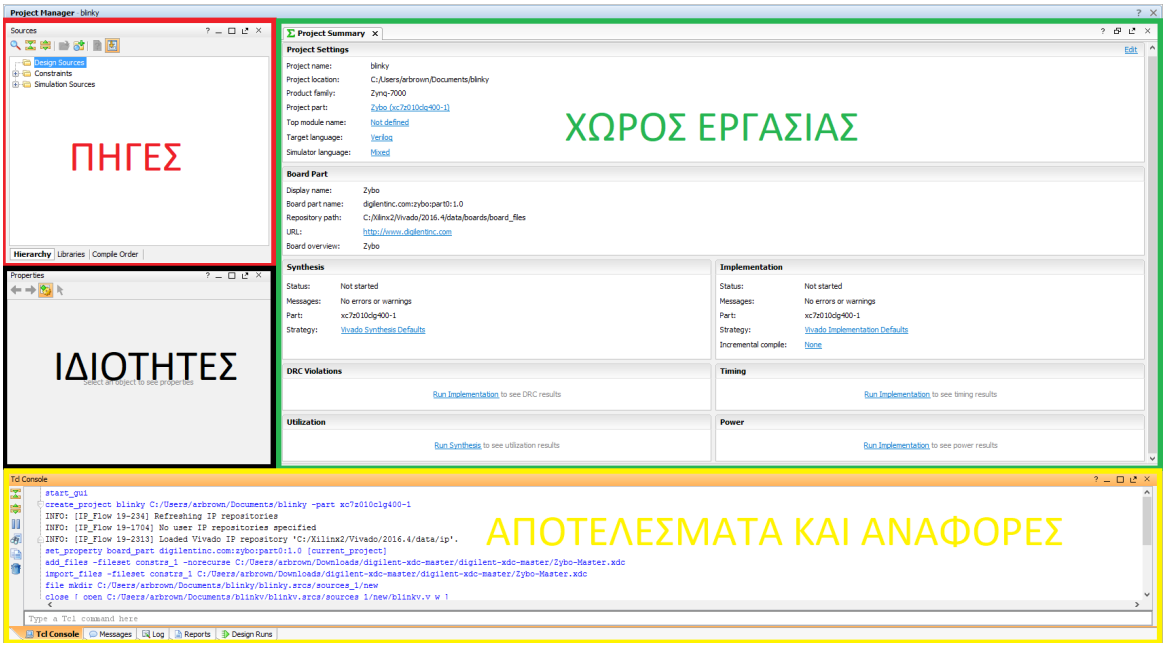
Το Flow Navigator είναι το πιο σημαντικό παράθυρο του Vivado. Είναι ο τρόπος με τον οποίο ένας χρήστης περιηγείται μεταξύ διαφορετικών εργαλείων της εφαρμογής. Το Flow Navigator είναι χωρισμένο σε επτά τμήματα:

- *Project Manager*
- *IP Integrator*
- *Simulation*
- *RTL Analysis*
- *Synthesis*
- *Implementation*
- *Program and Debug*



Εικόνα 6.8 : Flow navigator

Το βασικό παράθυρο που ανοίγει αφού έχουμε δημιουργήσει το σχέδιο είναι το Project Manager το οποίο αποτελείται από τέσσερα παραθυράκια, χώρος εργασίας ,πηγές,ιδιότητες και αποτελέσματα.



Εικόνα 6.9 : Το Project Manager

Το παράθυρο πηγές περιέχει την ιεραρχία του σχέδιο και από κει ανοίγουμε τα αρχεία.

Το παράθυρο ιδιότητες μας επιτρέπει να δούμε και να επεξεργαστούμε ιδιότητες του αρχείου που έχουμε επιλέξει.

Το πιο σημαντικά παράθυρο στο Project Manager είναι ο χώρος εργασίας ,εκεί ανοίγουν οι αναφορές για προβολή και τα αρχεία HDL για επεξεργασία.

Το τελευταίο παράθυρο που εμφανίζεται στο κάτω μέρος της εφαρμογής περιέχει διάφορα εργαλεία αποσφαλμάτωσης και προβολής μηνυμάτων και σφαλμάτων.

6.2.1 Προσθήκη αρχείου περιορισμού (Constraint File)

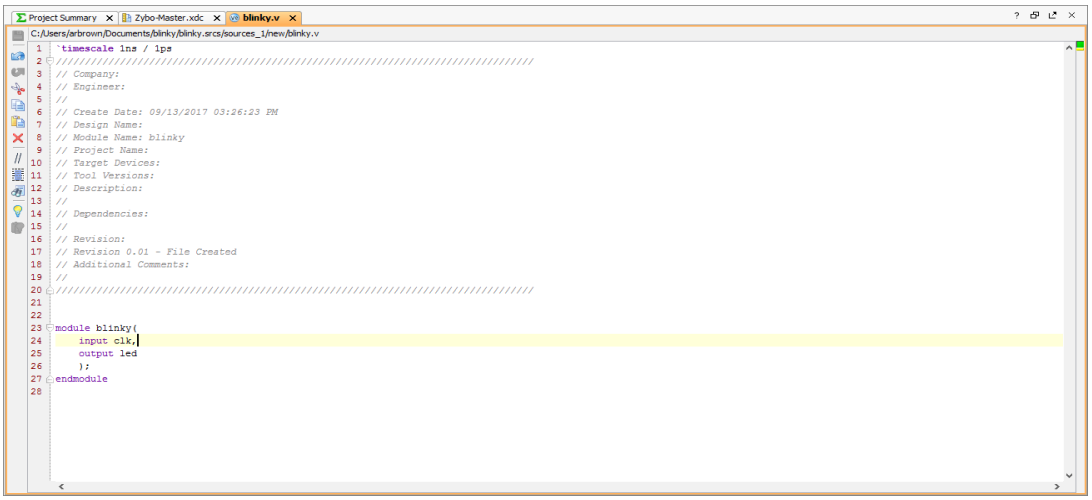
Για την δημιουργία σύνδεσης μεταξύ του κώδικα HDL και του FPGA θα πρέπει να προστεθεί η να δημιουργηθεί ένα αρχείο περιορισμού Xilinx Design Constraint (XDC) .Για κάθε πλακέτα αντιστοιχεί το δικό του XDC αρχείο, για να προσθέσουμε ένα τέτοιο αρχείο πηγαίνουμε στο παράθυρο Flow navigator και επιλέγουμε Add source. Στο παράθυρο που εμφανίζεται επιλέγουμε Add or create constraints και πατάμε το κουμπί Next.

Στο επόμενο παράθυρο επιλέγουμε Add files για να φορτώσουμε το αρχείο XDC στο σχέδιο .

6.2.2 Δημιουργία αρχείου Verilog

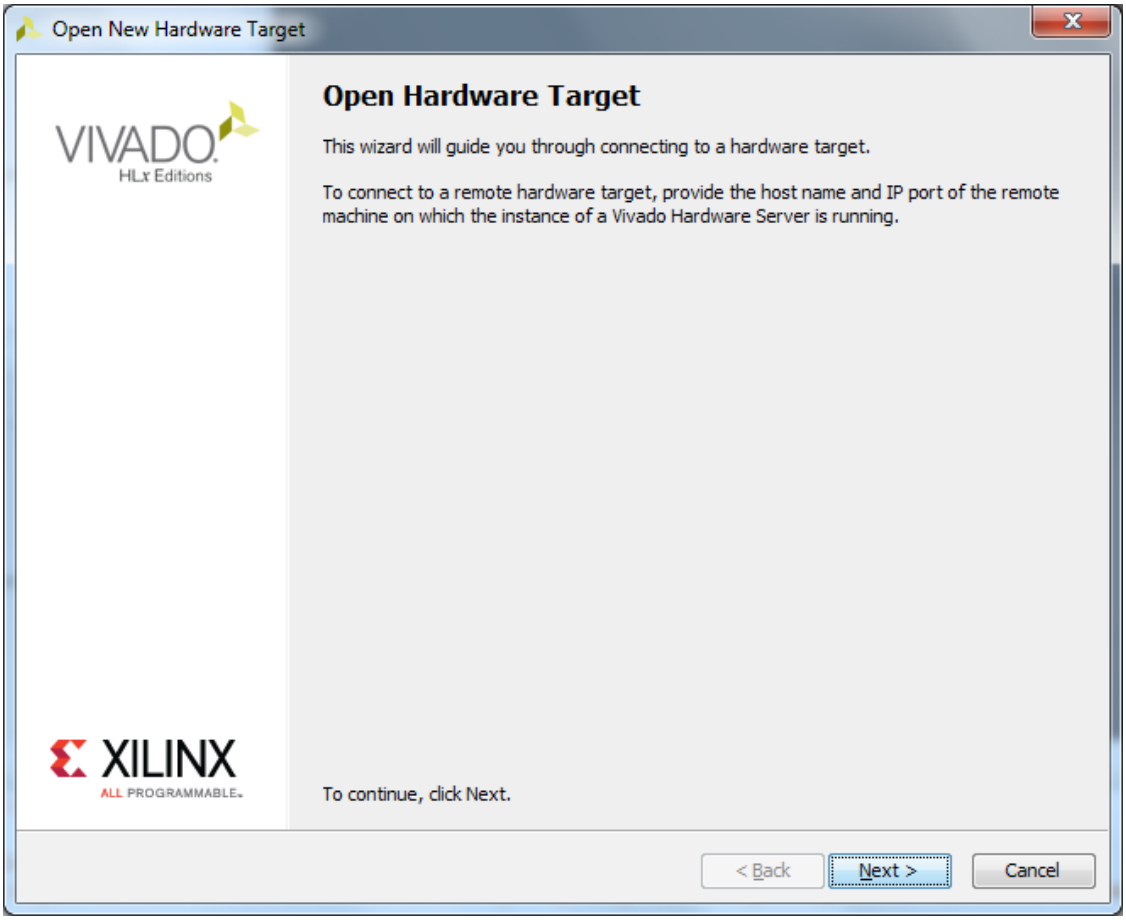
Πηγαίνουμε ξανά στο Flow navigator και επιλέγουμε Add sources στο παράθυρο που εμφανίζεται επιλέγουμε Add or create design sources και πατάμε Next.

Στο επόμενο παράθυρο επιλέγουμε Create File και πατάμε Next. Στην συνέχεια επιλέγουμε τον τύπο αρχείου ,το ονομά του και τον χώρο αποθήκευσης του, πατάμε Ok και μετά Finish.



Εικόνα 6.10 : Απλό παράδειγμα με γλώσσα Verilog

Γράφουμε ένα απλό παράδειγμα να αναβοσβήνει ένα LED και στη συνέχεια το τρέχουμε,αν δεν έχουμε σφάλματα στον κώδικα ανοίγουμε το Hardware Manager και επιλέγουμε το Open target και μετά πατάμε Next.

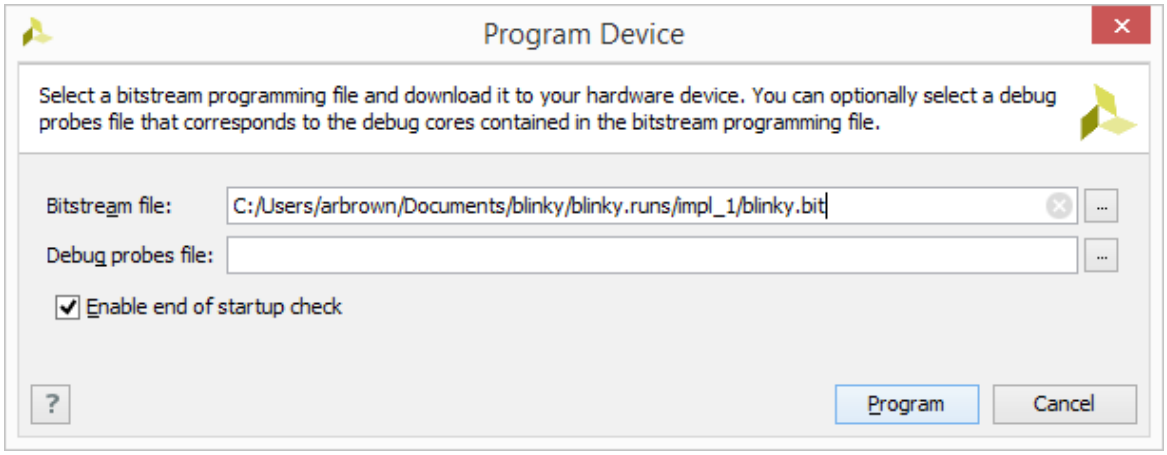


Εικόνα 6.11: Open Hardware Target

Το επόμενο παράθυρο μας ρωτάει τον τρόπο σύνδεσης με την συσκευή δηλαδή τοπική η απομακρυσμένη σύνδεση,πατάμε επόμενο και θα μας εμφανίζει την συσκευή η τις συσκευές οι οποίες είναι συνδεδεμένες με τον υπολογιστή ,πατάμε επόμενο και εμφανίζεται το τελευταίο παράθυρο με όλες τις επιλογές που κάναμε προηγουμένως αν είναι όλα εντάξει πατάμε Finish.

Για την σύνδεση με την συσκευή υπάρχει κι άλλος τρόπος αυτόματης σύνδεσης πατώντας την επιλογή Auto Connect.

Στην συνέχεια αφού έχουμε κάνει όλα τα βήματα σωστά φορτώνουμε το πρόγραμμα στην πλακέτα επιλέγοντας Program Device ,μετά την φόρτωση του προγράμματος στην πλακέτα θα πρέπει να αναβοσβήνει το LED που είχαμε ορίσει στον κώδικα.



Εικόνα 6.12 :Φόρτωση προγράμματος στην πλακέτα

ΚΕΦΆΛΑΙΟ 7

ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΓΗΣ ΣΕ ΓΛΩΣΣΑ VHDL ΚΑΙ ΥΛΟΠΟΙΗΣΗ ΣΕ FPGA

7.1 Αναπτυξιακές πλακέτες FPGA

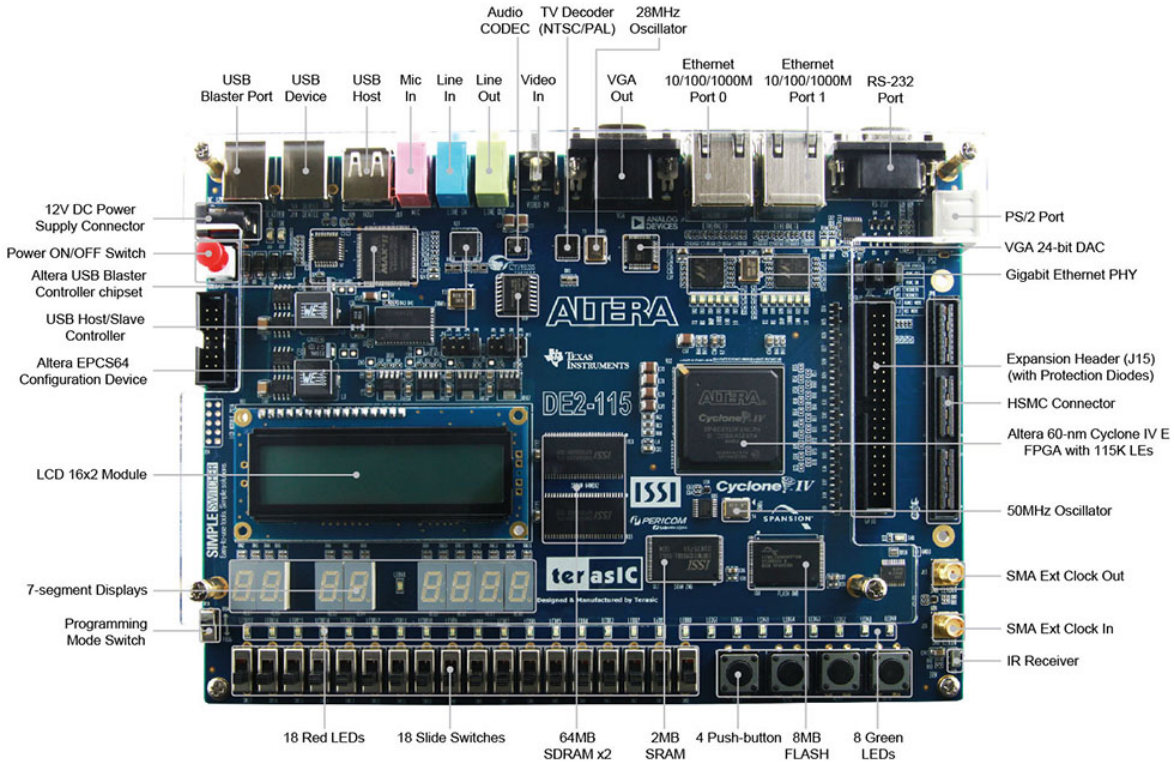
Αναπτυξιακές πλακέτες είναι ολοκληρωμένα συστήματα τα οποία διαθέτουν ενσωματωμένα FPGA και πολλά άλλα εξαρτήματα ενσωματωμένα όπως LEDs ,7 segments,buttons, switches και διάφορα άλλα εξαρτήματα. Υπάρχουν πάρα πολλά μοντέλα πλακετών τα οποία διαφέρουν μεταξύ τους από τις δυνατότητες τους ,τα χαρακτηριστικά και το κόστος τους.

Η Αναπτυξιακές πλακέτες χρησιμοποιούνται κυρίως για εκπαιδευτικό σκοπό .Η επιλογή μίας αναπτυξιακής πλακέτας γίνεται βάσει της χρήσης που προορίζεται, υπάρχουν οικονομικές πλακέτες υπάρχουν και οι ακριβές λύσεις για ποιο σύνθετες εφαρμογές κυρίαρχες εταιρίας στον χώρο αυτό είναι η Altera η οποία ανήκει πλέον στην Intel και η Xilinx που κατέχει περίπου 55% της αγοράς.

Στο κεφάλαιο αυτό θα σχεδιάσουμε με τη γλώσσα VHDL μια ολοκληρωμένη εφαρμογή ψηφιακού ρολογιού -ξυπνητηριού.
Η εφαρμογή θα υλοποιείται στην αναπτυξιακή πλακέτα D2-115 της Altera η οποία ανήκει στην οικογένεια Cyclone IV της ίδιας εταιρίας.

7.2 Η αναπτυξιακή πλακέτα D2-115

Η αναπτυξιακή πλακέτα D2-115 διαθέτει πολλά χαρακτηριστικά που επιτρέπουν στους χρήστες να εφαρμόζουν ένα ευρύ φάσμα
σχεδιασμένων κυκλωμάτων από τα πιο άπλα κυκλώματα μέχρι τις πιο περίπλοκες εφαρμογές πολυμέσων. Διαθέτει ενσωματωμένο το FPGA Cyclone IV EP4CE115F29 .

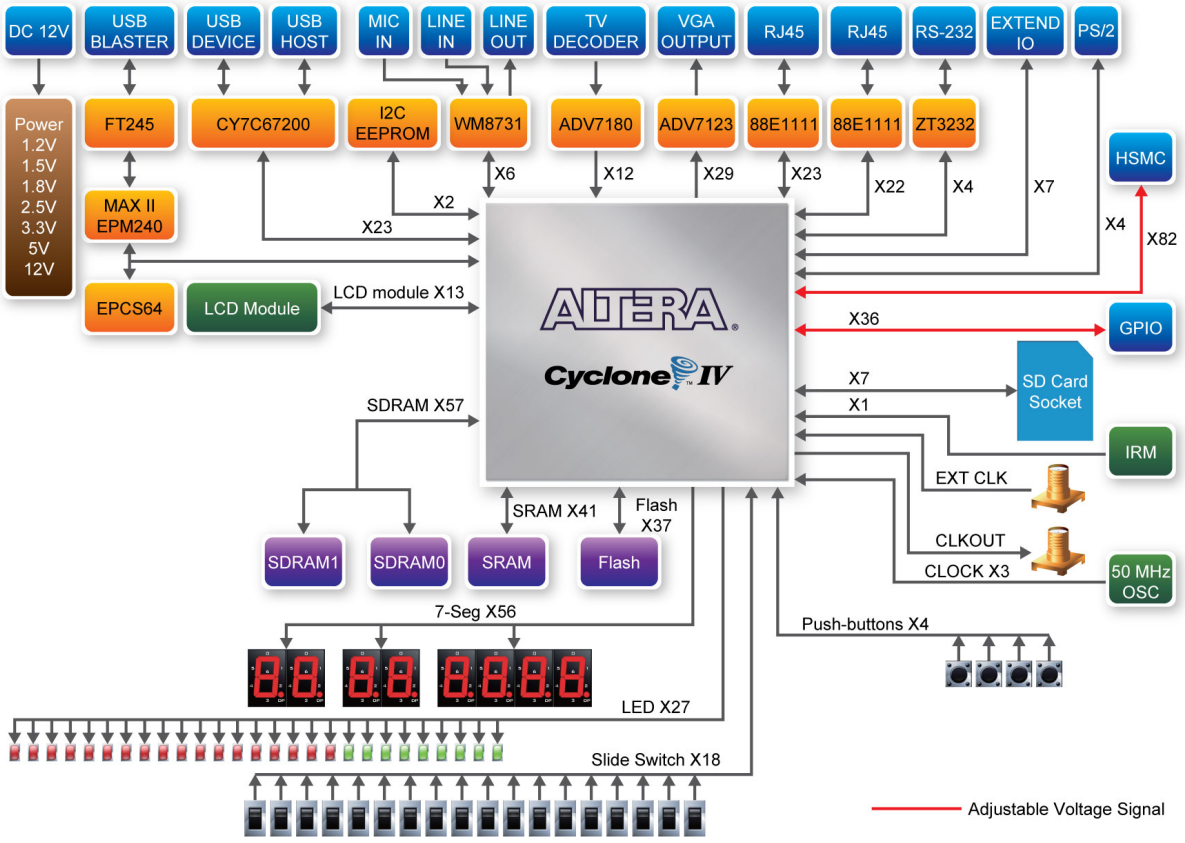


7.1 : Η αναπτυξιακή πλακέτα DE2-115 της Altera

Η πλακέτα DE-115 έχει ενσωματωμένα τα παρακάτω εξαρτήματα :

- Altera Cyclone IV 4CE115 FPGA
- Altera Serial Configuration device – EPCS64
- USB Blaster (on board) για προγραμματισμό πλακέτας
- 2MB SRAM
- 2 x 64MB SDRAM
- 8MB Flash μνήμη
- Υποδοχή κάρτας SD
- 4 Push-buttons
- 18 Διακόπτες
- 18 Κόκκινα LED

- 9 Πράσινα user LED
- 50MHz Κρύσταλλο ρολογιού και διάφορα άλλα εξαρτήματα .



Εικόνα 7.2 : Μπλοκ διαγράμματος της πλακέτας DE2-115

7.3 Υλοποίηση

Για την υλοποίηση της εφαρμογής θα χρησιμοποιήσουμε τα παρακάτω εξαρτήματα της πλακέτας D2-115:

- ✓ Το ρολόι της πλακέτας
- ✓ Τα μπουτόν KEY0 έως KEY3

- ✓ τα έξι από τα οχτώ seven segments displays που διαθέτει η πλακέτα HEX2 έως HEX7
- ✓ τα οχτώ πράσινα LED
- ✓ Τα οχτώ κόκκινα LED

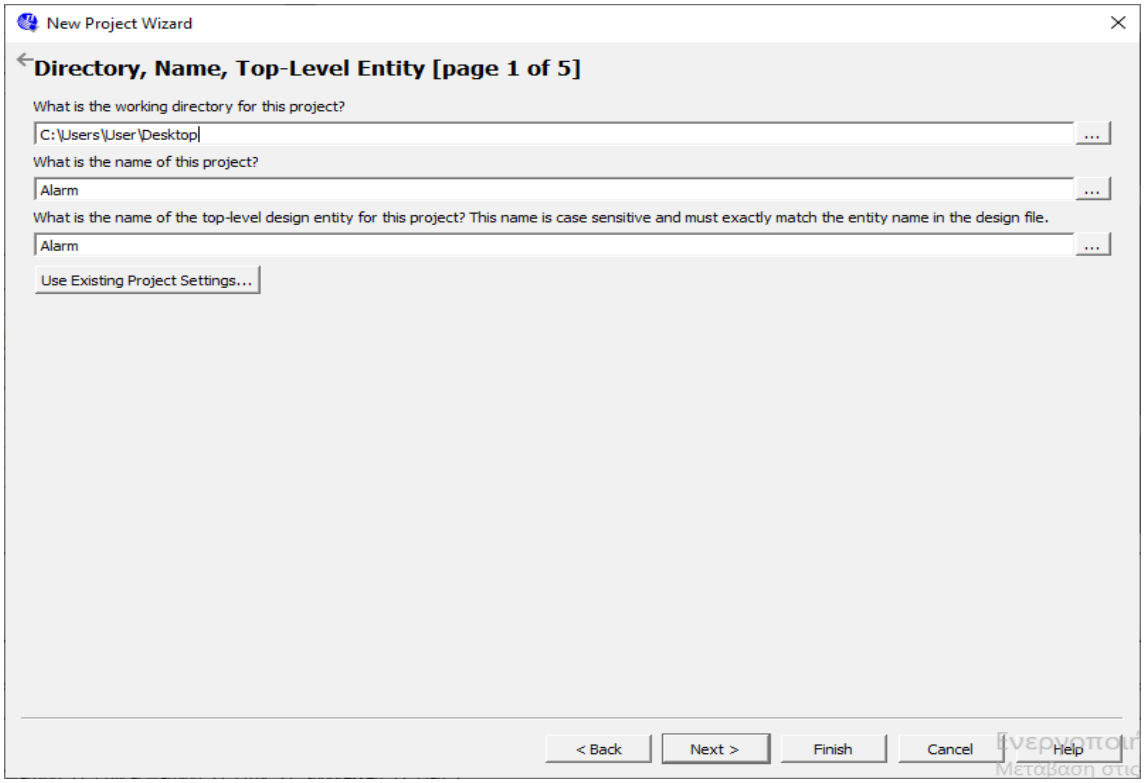
7.4 Λειτουργία

Η εφαρμογή θα έχει την εξής λειτουργία :

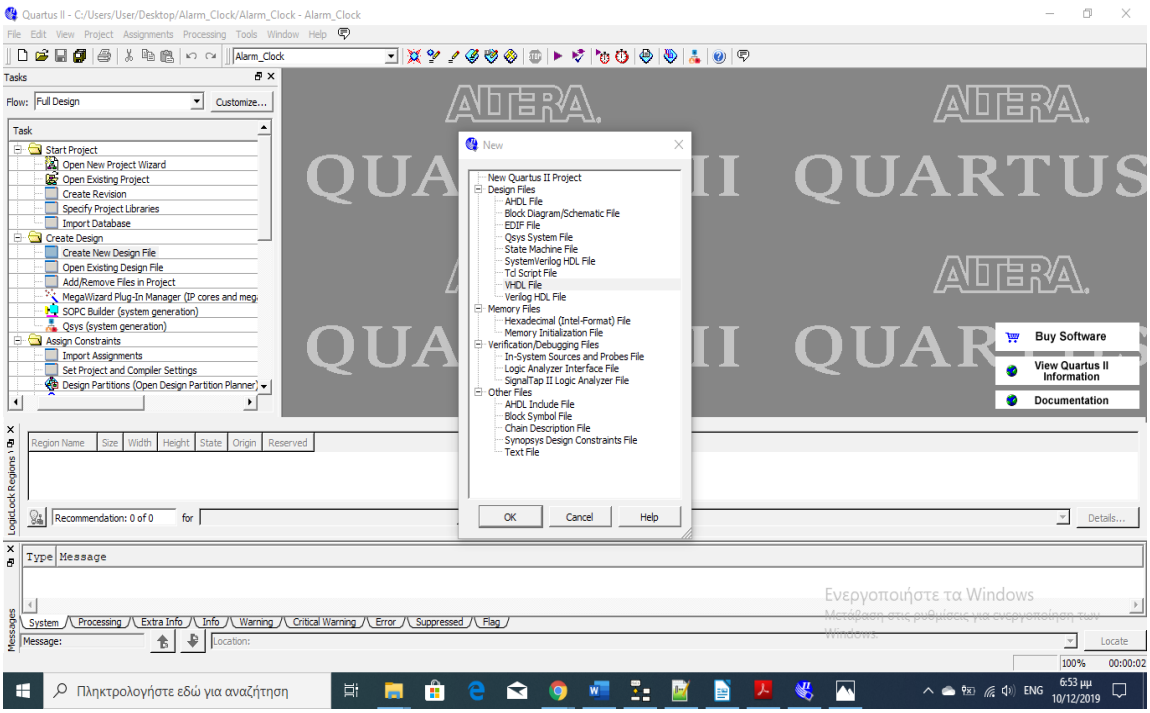
Θα είναι ένα ψηφιακό ρολόι, τα ψηφία του θα εμφανίζονται στα έξι από τα οχτώ 7 segments display της πλακέτας Τα δυο θα δείχνουν την ώρα τα δυο θα δείχνουν τα λεπτά και τα άλλα δυο τα δευτερόλεπτα. Η εφαρμογή θα διαθέτει και τέσσερα κουμπιά με τα οποία θα ρυθμίζεται το ρολόι και το ξυπνητήρι. Το ένα κουμπί θα είναι για να καθορίζει την λειτουργιά δηλαδή κρατώντας το πατημένο για 0.5 δευτερόλεπτα θα μπαίνει σε λειτουργιά ρύθμισης της ώρας ενώ κρατώντας το πατημένο για 0.25 δευτερόλεπτα θα μπαίνει σε λειτουργία ρύθμισης ξυπνητηριού. Το δεύτερο κουμπί αλλάζει τον "δείκτη" κατά την ρύθμιση της ώρας δηλαδή τις ώρες ,τα λεπτά η τα δευτερόλεπτα. Το τρίτο κουμπί είναι είναι για να αλλάζει τα ψηφία και το τέταρτο κουμπί για να κάνει reset το ρολόι η το ξυπνητήρι. Όταν το σύστημα θα είναι σε λειτουργία ρολογιού θα είναι αναμμένα 4 κόκκινα LED ενώ όταν το ένα πράσινο LED είναι αναμένω το σύστημα βρίσκετε στην λειτουργία ρύθμισης ξυπνητηριού.

7.5 Δημιουργία νέου σχεδίου στο Quartus II

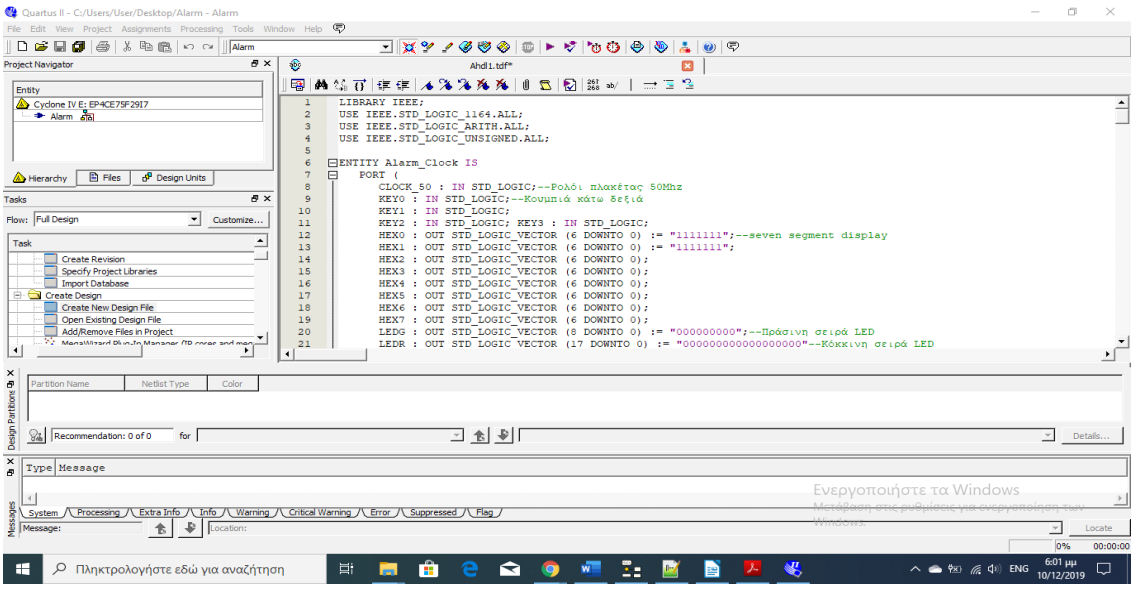
Για την υλοποίηση της εφαρμογής ακολουθούμε τα βήματα του κεφαλαίου 5 για την δημιουργία νέου σχεδίου στο Quartus II. Στην συνέχεια πηγαίνουμε στο Create Design και επιλέγομαι Create New Design File ,στο παραθυράκι που εμφανίζεται επιλέγουμε τον τρόπο σχεδίασης η την γλώσσα περιγραφής υλικού που θα χρησιμοποιήσουμε, στην περίπτωσή μας επιλέγουμε την γλώσσα VHDL. Πατώντας το κουμπί Ok ανοίγει το παράθυρο συγγραφής κώδικα.



Εικόνα 7.3 : Δημιουργία νέου σχεδίου σε Quartus II

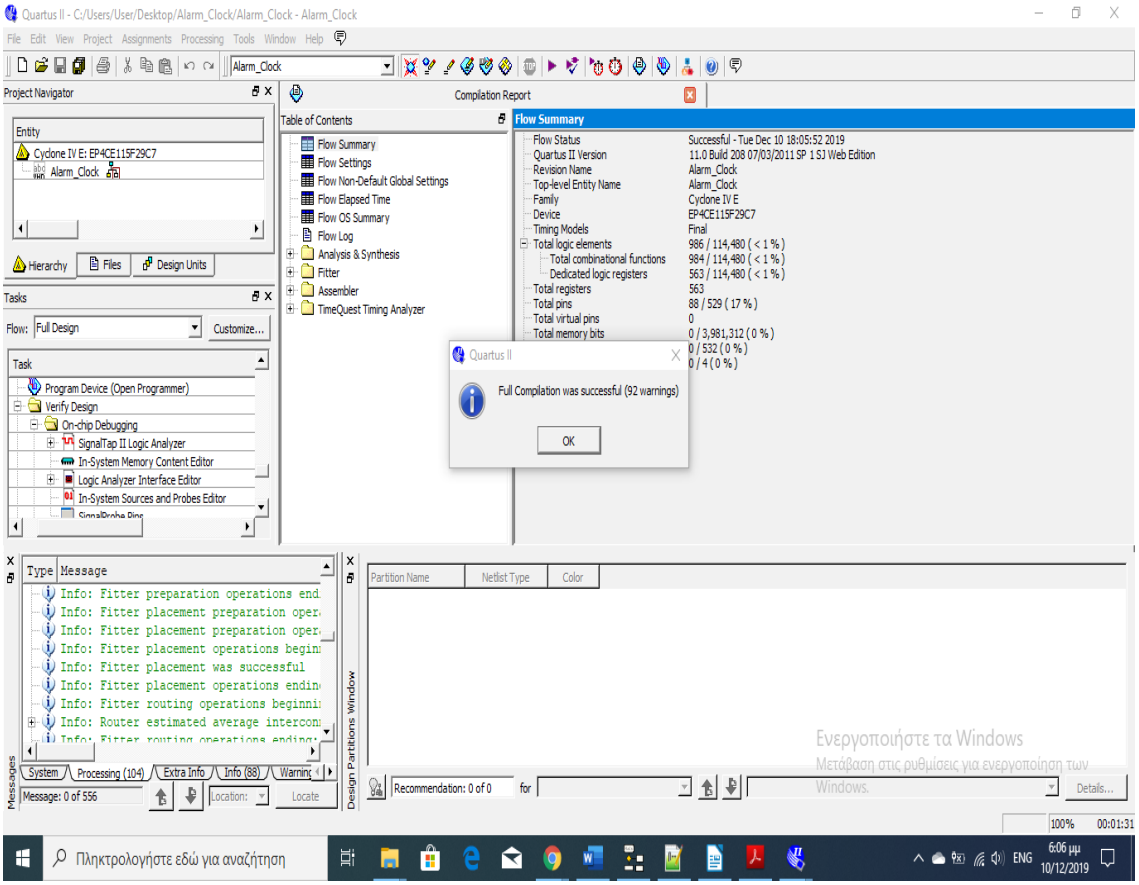


Εικόνα 7.4 : Επιλογή γλώσσας περιγραφής στο Quartus II

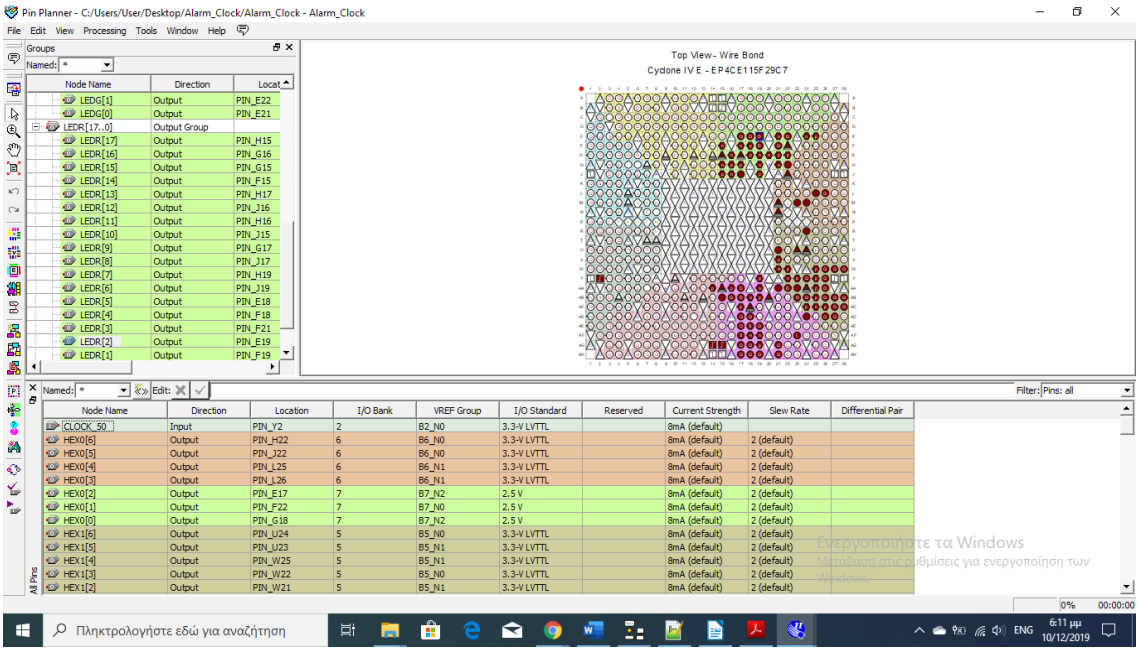


Εικόνα 7.5 : Συγγραφή κώδικα στο Quartus II

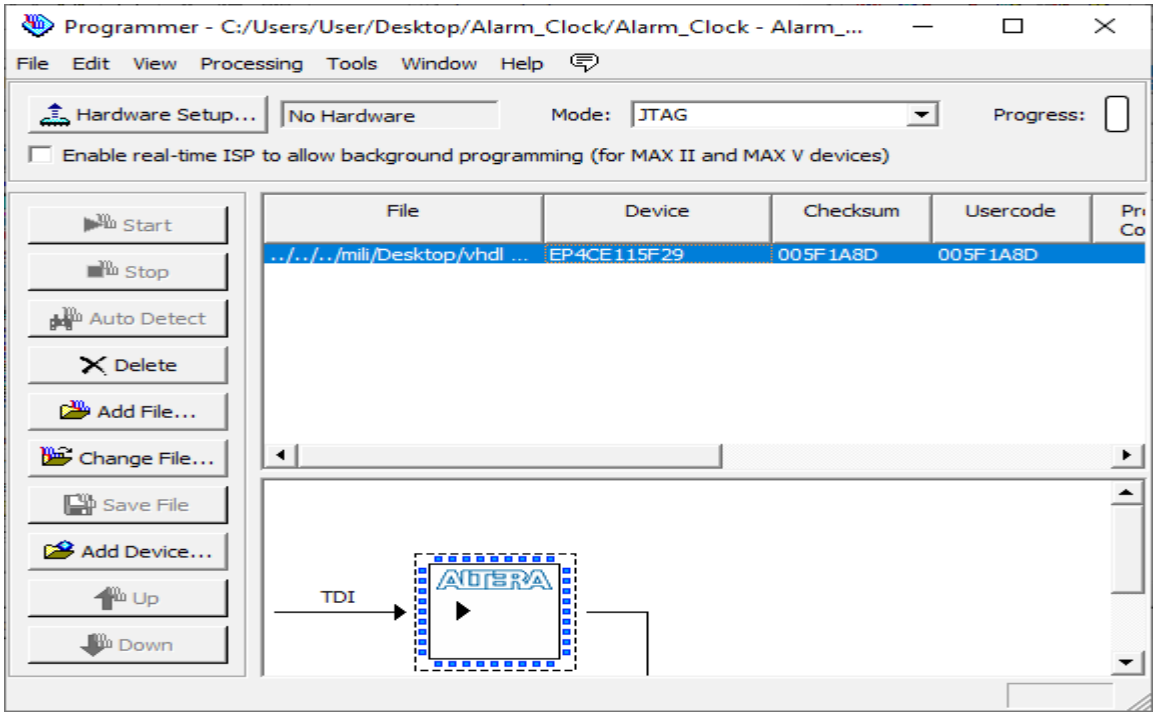
Δηλώνουμε τις βιβλιοθήκες που θα χρησιμοποιήσουμε στο πρόγραμμά μας ,τις εισόδους -εξόδους,την συμπεριφορά του κυκλώματος και τις διαδικασίες που θα τρέχουν στο πρόγραμμα. Για να κάνουμε Compile το πρόγραμμα πατάμε το Start Compilation.



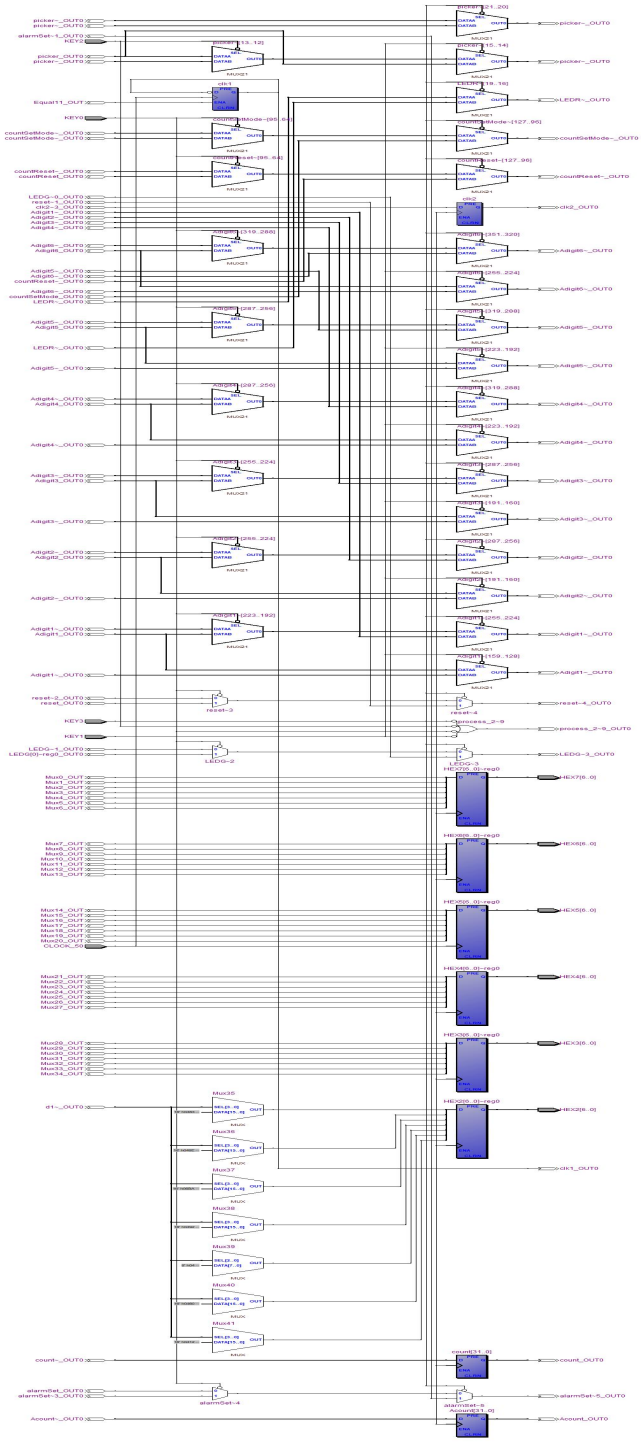
Εικόνα 7.6 : Αποτελέσματα μετά το Compile



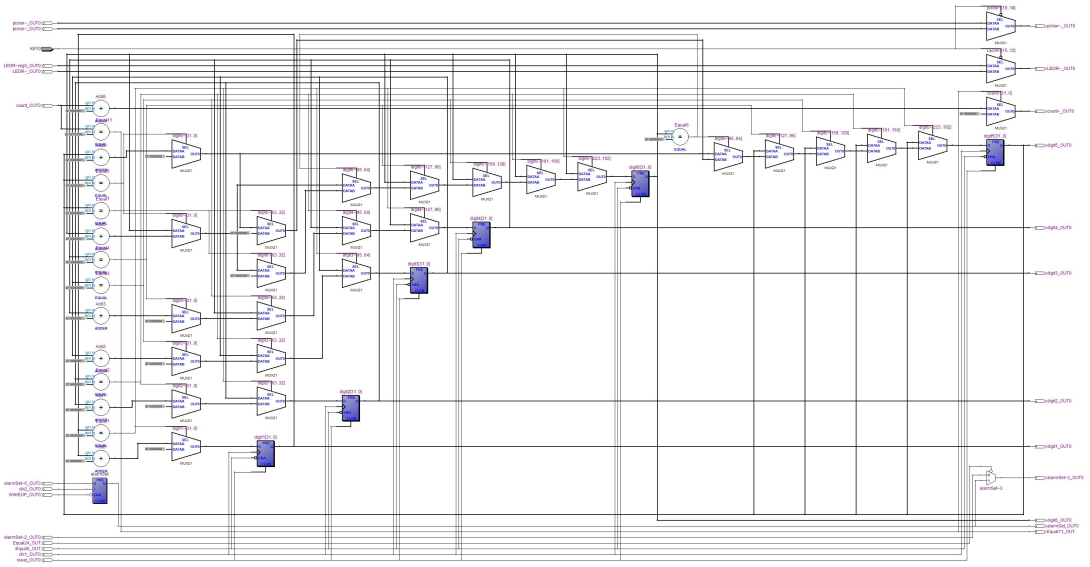
Εικόνα 7.7 : Pin planner -Περιοχές του FPGA που χρησιμοποιούνται



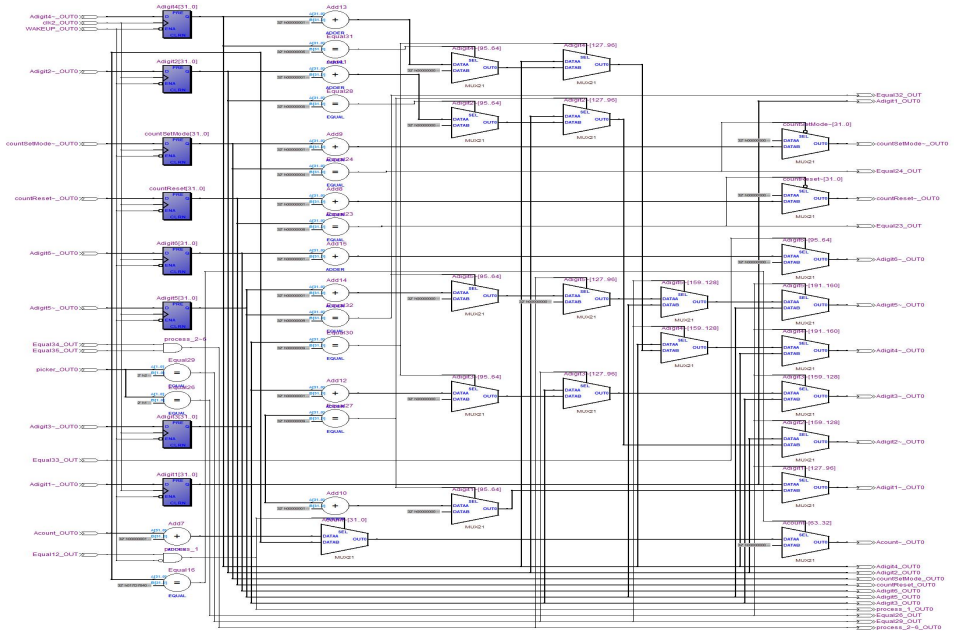
Εικόνα 7.8 Φόρτωση του προγράμματος στο FPGA



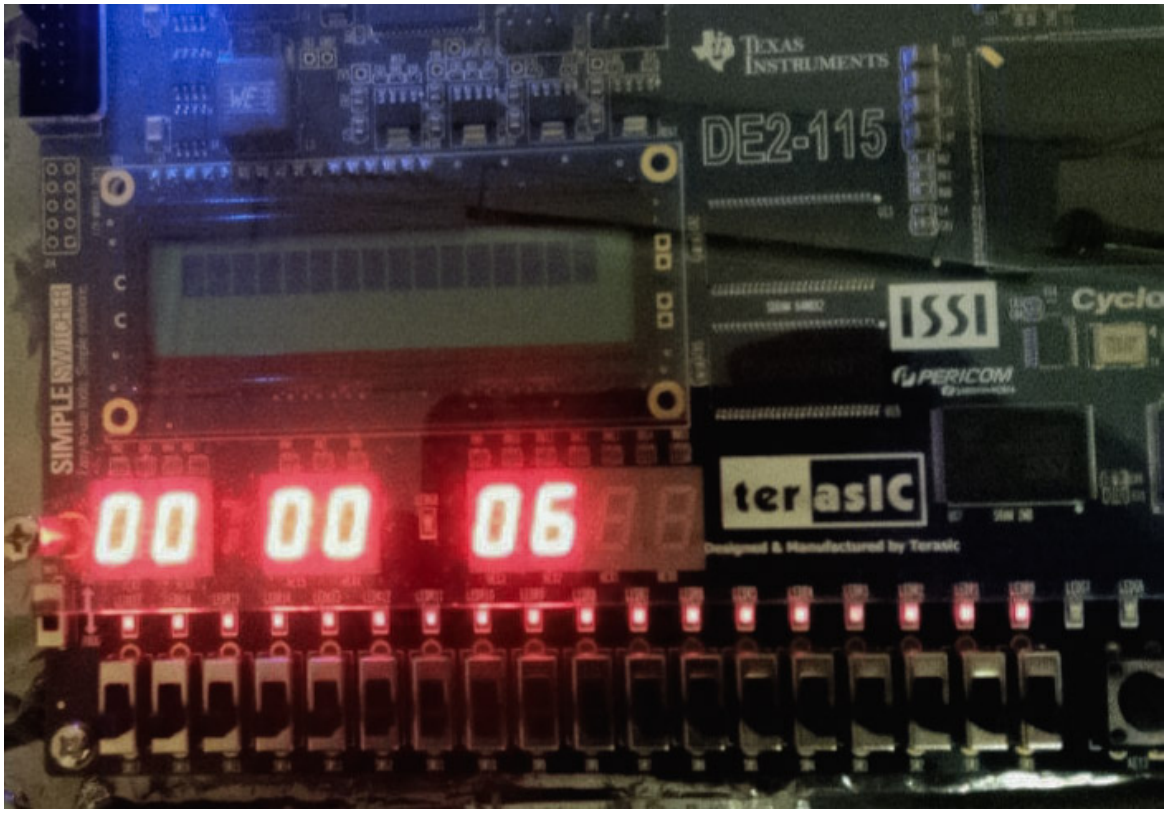
Εικόνα 7.9 : Η εφαρμογή σε επίπεδο RTL (1)



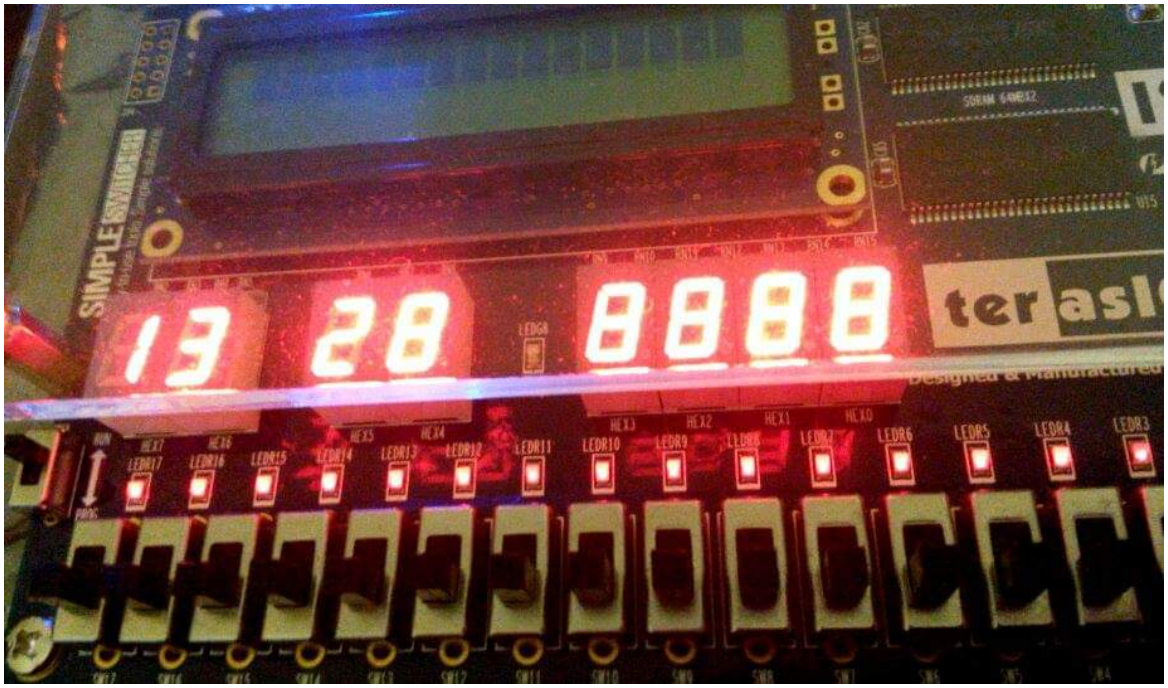
Εικόνα 7.10 : Η εφαρμογή σε επίπεδο RTL (2)



Εικόνα 7.9 : Η εφαρμογή σε επίπεδο RTL (3)



Εικόνα 7.12: Η εφαρμογή σε λειτουργία (1)



Εικόνα 7.13: Η εφαρμογή σε λειτουργία (2)

7.6 Κώδικας της εφαρμογής

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY Alarm_Clock IS
	PORT (
		CLOCK_50 : IN STD_LOGIC;--Ρολόι πλακέτας 50Mhz
		KEY0 : IN STD_LOGIC;--Κουμπιά κάτω δεξιά
		KEY1 : IN STD_LOGIC;
		KEY2 : IN STD_LOGIC;	KEY3 : IN STD_LOGIC;
		HEX0 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0) := "1111111";--
seven segment display
		HEX1 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0) := "1111111";
		HEX2 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
		HEX3 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
		HEX4 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
		HEX5 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
		HEX6 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
		HEX7 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
		LEDG : OUT STD_LOGIC_VECTOR (8 DOWNTO 0) := "000000000";--
Πράσινη σειρά LED
		LEDR : OUT STD_LOGIC_VECTOR (17 DOWNTO 0) :=
"00000000000000000"--Κόκκινη σειρά LED
	);
END Alarm_Clock;

ARCHITECTURE counter OF Alarm_Clock IS
	SIGNAL clk1  : STD_LOGIC := '0';--Πέρνουμε event κάθε 0.5 sec
	SIGNAL clk2 : STD_LOGIC := '0';--Πέρνουμε event κάθε 0.25 sec
	SIGNAL count : INTEGER := 1;
	SIGNAL countReset : INTEGER := 0;
	SIGNAL countSetMode : INTEGER := 0;
	SIGNAL digit1 : INTEGER := 0;--Ψηφια ρολογιου
	SIGNAL digit2 : INTEGER := 0;
	SIGNAL digit3 : INTEGER := 0;
	SIGNAL digit4 : INTEGER := 0;
	SIGNAL digit5 : INTEGER := 0;
	SIGNAL digit6 : INTEGER := 0;
	SIGNAL Adigit1 : INTEGER := 0;--Ψηφία ξυπνητηριού
	SIGNAL Adigit2 : INTEGER := 0;
	SIGNAL Adigit3 : INTEGER := 0;
	SIGNAL Adigit4 : INTEGER := 0;
	SIGNAL Adigit5 : INTEGER := 0;
	SIGNAL Adigit6 : INTEGER := 0;
	SIGNAL picker : STD_LOGIC_VECTOR(1 DOWNTO 0) := "00";--Καταστάσεις
ρολογιου/ξυπνητηριου
	SIGNAL Acount : INTEGER := 0;
```

```
	SIGNAL reset : STD_LOGIC := '0';
	SIGNAL WAKEUP : STD_LOGIC := '0'; --Οταν γινει 1 χτυπαει το
ξυπνητηρι
	SIGNAL alarmSet : STD_LOGIC := '0';--Ρυθμιση ξυπνητηριου
	TYPE display_ROM IS ARRAY (0 TO 10) OF std_logic_vector (6 DOWNTO
0);
	CONSTANT convert_to_segments : display_ROM :=
	("1000000", "1111001", "0100100", "0110000", "0011001", "0010010",
"0000010", "1111000",
		"0000000", "0010000", "1111111"); --0,1,2,3,4,5,6,7,8,9,OFF
BEGIN
	PROCESS (clk1, reset)
	BEGIN
		IF (reset = '1') THEN
			digit1 <= 0;--Μηδενισμος ολων των ψηφιων του ρολογιου
			digit2 <= 0;
			digit3 <= 0;
			digit4 <= 0;
			digit5 <= 0;
			digit6 <= 0;
		ELSIF (clk1'EVENT AND clk1 = '1') THEN
			IF (countReset /= 4) THEN
				digit1 <= digit1 + 1;
				IF (digit1 = 9) THEN
					digit1 <= 0;
					digit2 <= digit2 + 1;
					IF (digit2 = 5) THEN
						digit2 <= 0;
						digit3 <= digit3 + 1;
						IF (digit3 = 9) THEN
							digit3 <= 0;
							digit4 <= digit4 + 1;
							IF (digit4 = 5) THEN
								digit4 <= 0;
								digit5 <= digit5 + 1;
								IF (digit5 = 9) THEN
									digit5 <= 0;
									digit6 <= digit6 +
1;

								END IF;
								IF (digit6 = 2) THEN
									IF (digit5 = 3)
THEN

										digit1 <= 0;
										digit2 <= 0;
										digit3 <= 0;
										digit4 <= 0;
										digit5 <= 0;
										digit6 <= 0;
									END IF;
								END IF;
```

```
                                        END IF;
                                    END IF;
                                END IF;
                            END IF;
                        END IF;
                END IF;
        END PROCESS;
PROCESS (CLOCK_50)
VARIABLE d1 : INTEGER := 0;
VARIABLE d2 : INTEGER := 0;
VARIABLE d3 : INTEGER := 0;
VARIABLE d4 : INTEGER := 0;
VARIABLE d5 : INTEGER := 0;
VARIABLE d6 : INTEGER := 0;
BEGIN
        IF (CLOCK_50'EVENT AND CLOCK_50 = '1') THEN
                count <= count + 1;
                IF (count = 6250000) THEN --events για clk1,clk2
                        clk2 <= NOT clk2;       --
                END IF;                         --
                IF (count = 12500000) THEN--
                        clk2 <= NOT clk2;       --
                END IF;                         --
                IF (count = 18750000) THEN--
                        clk2 <= NOT clk2;       --
                END IF;                         --
                IF (count = 25000000) THEN--
                        clk1 <= NOT clk1;       --
                        clk2 <= NOT clk2;       --
                        count <= 1;             --
                END IF;                         --
                IF (picker = "00" AND WAKEUP = '0') THEN
                        d6 := digit6;
                        d5 := digit5;
                        d4 := digit4;
                        d3 := digit3;
                        d2 := digit2;
                        d1 := digit1;
                ELSE
                        Acount <= Acount + 1;
                        IF (Acount < 12500000) THEN
                                d6 := Adigit6;
                                d5 := Adigit5;
                                d4 := Adigit4;
                                d3 := Adigit3;
                                d2 := Adigit2;
                                d1 := Adigit1;
                        ELSIF (picker = "01") THEN
                                d6 := Adigit6;
                                d5 := Adigit5;
                                d4 := Adigit4;
```

```
                        d3 := Adigit3;
                        d2 := 10; --OFF
                        d1 := 10; --OFF
                  ELSIF (picker = "10") THEN
                        d6 := Adigit6;
                        d5 := Adigit5;
                        d4 := 10; --OFF
                        d3 := 10; --OFF
                        d2 := Adigit2;
                        d1 := Adigit1;
                  ELSIF (picker = "11") THEN
                        d6 := 10; --OFF
                        d5 := 10; --OFF
                        d4 := Adigit4;
                        d3 := Adigit3;
                        d2 := Adigit2;
                        d1 := Adigit1;
                  ELSE -- IS TIME TO WAKEUP
                        d6 := 10; --OFF
                        d5 := 10; --OFF
                        d4 := 10; --OFF
                        d3 := 10; --OFF
                        d2 := 10; --OFF
                        d1 := 10; --OFF
                  END IF;
            END IF;
            IF (Acount = 25000000) THEN
                  Acount <= 0;
            END IF;
            HEX7 <= convert_to_segments(d6);
            HEX6 <= convert_to_segments(d5);
            HEX5 <= convert_to_segments(d4);
            HEX4 <= convert_to_segments(d3);
            HEX3 <= convert_to_segments(d2);
            HEX2 <= convert_to_segments(d1);

      END IF;

END PROCESS;

PROCESS (clk2)--Διαδικασία ξυπνητηριού
BEGIN
      IF (clk2'EVENT AND clk2 = '1') THEN
            IF (WAKEUP = '0') THEN
                  IF (Adigit1 = digit1 AND Adigit2 = digit2 AND
Adigit3 = digit3 AND Adigit4 = digit4 AND Adigit5 = digit5 AND Adigit6 =
digit6 AND alarmSet = '1') THEN
                        WAKEUP <= '1';
                  ELSE
                        IF (KEY3 = '0') THEN
```

```
                        IF (countReset /= 8) THEN
                              countReset <= countReset + 1;
                        ELSE
                              LEDG(0) <= '1';
                              countReset <= 0;
                              IF (picker /= "00" ) THEN
                                    Adigit1 <= 0;
                                    Adigit2 <= 0;
                                    Adigit3 <= 0;
                                    Adigit4 <= 0;
                                    Adigit5 <= 0;
                                    Adigit6 <= 0;
                                    LEDR(1) <= '0';
                                    LEDR(2) <= '0';
                                    LEDR(3) <= '0';
                                    LEDR(4) <= '0';
                                    alarmSet <= '0';
                                    picker <= "00";
                              ELSE
                                    reset <= '1';
                              END IF;
                        END IF;
                  ELSIF (KEY0 = '0') THEN
                        IF (countSetMode /= 4) THEN
                              countSetMode <= countSetMode +
1;
                        ELSE
                              IF (picker = "00") THEN
                                    LEDR(1) <= '1';
                                    LEDR(2) <= '1';
                                    LEDR(3) <= '1';
                                    LEDR(4) <= '1';
                                    picker <= "01";
                                    countSetMode <= 0;
                              ELSE
                                    LEDR(1) <= '0';
                                    LEDR(2) <= '0';
                                    LEDR(3) <= '0';
                                    LEDR(4) <= '0';
                                    picker <= "00";
                                    countSetMode <= 0;
                                    alarmSet <= '1';
                              END IF;
                        END IF;
                  ELSIF (picker /= "00") THEN
                        IF (KEY1 = '0') THEN
                              IF (picker = "01") THEN
                                    Adigit1 <= Adigit1 + 1;
                                    IF (Adigit1 = 9) THEN
                                          Adigit1 <= 0;
                                          Adigit2 <= Adigit2
```

```
+ 1;
                                                    IF (Adigit2 = 5)
THEN
                                                        Adigit2 <=
0;
                                                    END IF;
                                                END IF;
                                            ELSIF (picker = "10") THEN
                                                Adigit3 <= Adigit3 + 1;
                                                IF (Adigit3 = 9) THEN
                                                    Adigit3 <= 0;
                                                    Adigit4 <= Adigit4
+ 1;
                                                    IF (Adigit4 = 5)
THEN
                                                        Adigit4 <=
0;
                                                    END IF;
                                                END IF;
                                            ELSE
                                                Adigit5 <= Adigit5 + 1;
                                                IF (Adigit5 = 9) THEN
                                                    Adigit5 <= 0;
                                                    Adigit6 <= Adigit6
+ 1;
                                                    IF (Adigit6 = 3)
THEN
                                                        Adigit6 <=
0;
                                                    END IF;
                                                END IF;
                                                IF (Adigit6 = 2 AND
Adigit5 = 3) THEN
                                                    Adigit5 <= 0;
                                                    Adigit6 <= 0;
                                                END IF;
                                            END IF;
                                        ELSIF (KEY2 = '0') THEN--Αλλαγη σε
ωρα ,λεπτα η δευτερολεπτα
                                            IF (picker <= "01") THEN
                                                picker <= "10";
                                            ELSIF (picker <= "10") THEN
                                                picker <= "11";
                                            ELSIF (picker <= "11") THEN
                                                picker <= "01";
                                            END IF;
                                        END IF;
                                    ELSE
                                        LEDG(0) <= '0';
                                        countReset <= 0;
                                        countSetMode <= 0;
```

```
                                    reset <= '0';
                                END IF;
                            END IF;
                        ELSE -- IS TIME TO WAKEUP
                            IF (KEY0 = '0' OR KEY1 = '0' OR KEY2 = '0' OR
KEY3 = '0') THEN
                                WAKEUP <= '0';
                            END IF;
                        END IF;
                END IF;
        END PROCESS;

END counter;
```

ΚΕΦΆΛΑΙΟ 8
ΣΥΜΠΕΡΆΣΜΑΤΑ

8.1 Συμπεράσματα πτυχιακής εργασίας

Σε μία εποχή που η σχεδίαση ψηφιακών κυκλωμάτων γίνεται ολοένα και πιο απαιτητικός σε σχέση με την πολυπλοκότητα και την αποδοτικότητα ,τα σύγχρονα συστήματα σχεδίασης τέτοιων κυκλωμάτων ίσως να αποτελούν τον μοναδικό τρόπο υλοποιήσεις τους. Τα σύγχρονα συστήματα σχεδίασης ψηφιακών συστημάτων είναι ένα σύνολο από εργαλεία τα οποία ο συνδυασμός τους μας προσφέρει πολλές δυνατότητες και ευκολίες στη σχεδίαση ,στην επαλήθευση και στην προσομοίωση του σχεδίου. Βασικά στοιχεία σύγχρονών συστημάτων σχεδίασης είναι η γλώσσες περιγραφής υλικού ,το λογισμικό σχεδίασης και τα FPGAs-CPLDs. Ο κλάδος της τεχνολογίας και ειδικά η επιστήμη των υπολογιστών εξελίσσονται με πάρα πολύ γρήγορους ρυθμούς αυτό συνεπάγεται ότι και η σχεδίαση ολοκληρωμένων κυκλωμάτων που αποτελεί πολύ σημαντικό κομμάτι της επιστήμης των υπολογιστών αναγκαστικά κινείται στους ίδιους ρυθμούς εξέλιξης.

Η σχεδίαση ολοκληρωμένων κυκλωμάτων και ψηφιακών κυκλωμάτων γενικά είναι μια διαδικασία που απαιτεί αρχική μελέτη όσον αφορά τις απαιτήσεις και αυτές τις απαιτήσεις να εκφραστούν λεκτικά με κάποια γλώσσα περιγραφής υλικού. Άρα για να μπορέσουμε να σχεδιάσουμε ένα περίπλοκο ψηφιακό κύκλωμα θα πρέπει να ξέρουμε κάποια γλώσσα περιγραφής υλικού. Για πιο απλά κυκλώματα μπορεί να γίνει και χωρίς χρήση γλώσσας περιγραφής υλικού αλλά με χρήση της σχηματικής σχεδίασης δηλαδή σε επίπεδο λογικών πυλών και διαφόρων άλλων στοιχείων. Η συγγραφή του κώδικα που θα περιγράφει την συμπεριφορά ενός κυκλώματος γίνεται σε κάποιο λογισμικό μεταγλώττισης γλωσσών περιγραφής υλικού κάποια από τα οποία διατίθενται δωρεάν. Για την πλήρη κάλυψη των αναγκών στην ψηφιακή σχεδίαση θα πρέπει να αποκτήσουμε κάποιο από τα πακέτα λογισμικού σχεδίασης τα οποία διαθέτουν όλα τα απαραίτητα εργαλεία για την σχεδίαση ενός περίπλοκου ψηφιακού συστήματος.

Συνεπώς ένας σχεδιαστής ψηφιακών συστημάτων θα πρέπει να ξέρει να διαχειρίζεται κάποια από αυτά τα πακέτα λογισμικού σχεδίασης. Ένα ακόμα στοιχείο της ψηφιακής σχεδίασης που πρέπει να γνωρίσουμε είναι και η τεχνολογία που θα χρησιμοποιεί το κύκλωμα που σχεδιάζουμε. Υπάρχουν δύο είδη ολοκληρωμένων κυκλωμάτων που μπορούμε να σχεδιάσουμε, οι επαναπρογρμματιζόμενες διατάξεις (FPGA-CPLD) και οι διατάξεις ειδικού σκοπού (ASIC) .Οι πρώτες μπορούν να προγραμματιστούν πολλές φορές για διαφορετικές εφαρμογές ενώ οι διατάξεις ειδικού σκοπού είναι ολοκληρωμένα κυκλώματα τα οποία προγραμματίζονται μόνο μια φορά και για μια συγκεκριμένη εφαρμογή. Οι κυρίαρχες εταιρίες στην παραγωγή ψηφιακών συστημάτων έχουν δημιουργήσει κάποια κυκλώματα τα οποία διατίθενται στην αγορά και αποτελούνται από επαναπρογρμματιζόμενες διατάξεις (FPGA-CPLD) και διάφορα ηλεκτρονικά στοιχεία ,είναι κατάλληλες για εκπαιδευτικούς σκοπούς αλλά και για διάφορες εφαρμογές.

Συνοψίζοντας ,για να σχεδιάσουμε ένα ψηφιακό σύστημα βασιζόμενο στους κανόνες της σύγχρονης σχεδίασης θα πρέπει να:

- Να έχουμε σαφή εικόνα των απαιτήσεων του συστήματος που θα σχεδιάσουμε.
- Να έχουμε καλές γνώσεις ηλεκτρονικών και λογικών εκφράσεων.
- Για περίπλοκα κυκλώματα να γνωρίσουμε καλά κάποια γλώσσα περιγραφής υλικού.
- Να είμαστε σε θέση να χειριζόμαστε κάποιο από τα πακέτα σχεδίασης ψηφιακών συστημάτων όπως το Quartus η το Vivado.
- Να είμαστε σε θέση να προγραμματίσουμε την διάταξη για την οποία προορίζεται το σχέδιο.
- Να έχουμε τουλάχιστον βασικές γνώσεις σχετικά με τεχνολογίες που χρησιμοποιούν τα ολοκληρωμένα κυκλώματα είτε είναι επαναπρογραμματιζόμενα είτε για ειδικές εφαρμογές.

8.2 Το μέλλον στην σύγχρονη σχεδίαση κυκλωμάτων

Τις τελευταίες δεκαετίες η σχεδίαση των ψηφιακών συστημάτων έχει απτελέσει τον πιο σημαντικό κλάδο στην εξέλιξη της τεχνολογίας και αυτό διαπιστώνεται εύκολα παρατηρώντας καθημερινές μας συνήθειες σε ότι έχει σχέση με την τεχνολογία. Αυτή την εξέλιξη ,που στην οποία η ψηφιακή σχεδίαση έχει παίξει βασικό ρόλο, την παρατηρούμε στις ηλεκτρονικές συσκευές που χρησιμοποιούμε καθημερινά όπως τα κινητά τηλέφωνα ,υπολογιστές ,τηλεοράσεις κλπ. Το μέλλον στην ψηφιακή σχεδίαση συστημάτων δεν μπορεί να μην είναι εξίσου σημαντικό καθώς είναι μια βιομηχανία η οποία εξελίσσεται με πολύ γρήγορους ρυθμούς.

Η παρούσα πτυχιακή εργασία αποτελεί έναν βασικό οδηγό για όσους θέλουν να ασχοληθούν με την σχεδίαση των ολοκληρωμένων κυκλωμάτων και γενικά ψηφιακών συστημάτων καθώς αναφέρονται όλα τα εργαλεία τα οποία χρειάζονται για την σχεδίαση αυτών των συστημάτων.

ΒΙΒΛΙΟΓΡΑΦΙΑ

1) https://en.wikipedia.org/wiki/Hardware_description_language

2) https://en.wikipedia.org/wiki/VHDL

3) https://www.csd.uoc.gr/~hy225/10a/verilog_basics.pdf

4) http://users.sch.gr/jabatzo/files/yliko/live%20ebooks/psifiaka_G_2018_final/__5.html

5) https://www.nandland.com/

6) https://allaboutfpga.com/

7) http://arch.icte.uowm.gr/courses/digital_design/lecture13.pdf

8) http://www.cs.uoi.gr/~kabousia/pdf/LogicDesignII/VHDL_Types2.pdf

9) http://freerangefactory.org/pdf/df344hdh4h8kjfh3500ft2/free_range_vhdl.pdf?fbclid=IwAR11dNVRForzfkd0CH3GeqfL9pB1j8fv6bDRB86oi4Oh3oX6aVSrPiLq-ms

10) https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/qts/qts_qii5v1.pdf

11) https://reference.digilentinc.com/reference/software/vivado/start

12) https://www.xilinx.com/

13) https://www.electronicdesign.com/resources

14) https://en.wikipedia.org/wiki/VHDL#Design

15) https://www.ics.uci.edu/~jmoorkan/vhdlref/vhdl_golden_reference_guide.pdf

16) http://teachers.cm.ihu.gr/kalomiros/Proigmena_Psifiaka_Theoria/didaktiko_yliko/Intro_VHDL_Kalomiros_teuxos2.pdf

17) https://www.fpga4student.com/