ΑΛΕΞΑΝΔΡΕΙΟ ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ
ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΝΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΕ HANDEL-C ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΑ FPGA (09330ΕΜ)

ΓΚΑΧΟΠΟΥΛΟΥ ΚΥΡΙΑΚΗ
ΣΑΒΒΟΥΔΗΣ ΧΡΗΣΤΟΣ

Επιβλέπων καθηγητής: Χατζηγκάιδας Αθανάσιος

Ανάληψη Π.Ε. : Φεβρουάριος 2010
Περάτωση Π.Ε. : Φεβρουάριος 2011

Θεσσαλονίκη, Φεβρουάριος 2011

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΕΡΙΛΗΨΗ

Η παρούσα εργασία έχει σαν αντικείμενο τον προγραμματισμό των FPGA με την χρήση Handel-C και του πακέτου ISE της Xilinx.Το σχεδιαστικό περιβάλλον της εταιρείας Xilinx και τα εργαλεία της χρησιμοποιούνται ευρύτατα για την υποστήριξη τόσο του εκπαιδευτικού μέρους σχεδίασης ψηφιακών κυκλωμάτων, όσο και για την υποστήριξη του ερευνητικού έργου.

Η σχεδίαση της αρχιτεκτονικής της διάταξης και ο ορισμός της περιγραφής όλων των υποκυκλωμάτων και στοιχείων της έγινε με κώδικα στη γλώσσα περιγραφής υλικού Handel-C.

Στην εργασία αυτή αναλύθηκε η σχεδίαση και η υλοποίηση κυκλωμάτων FPGA.

SYMMARY

The present work has as object the programming of FPGA by usage Handel-C and the Xilinx ISE package. The designing environment of company Xilinx and her tools are used very widely for the support of so much educational part of designing of digital circuits, as much as for the support of inquiring work.

The designing of her architectural provision and the definition of description all her part circuits and elements became with code in the language of description of material Handel-C.

In this word were analyzed the designing and the concretization in circuits FPGA.

1. ΕΙΣΑΓΩΓΗ ΣΤΗ ΓΛΩΣΣΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

1.1 Γενική εισαγωγή

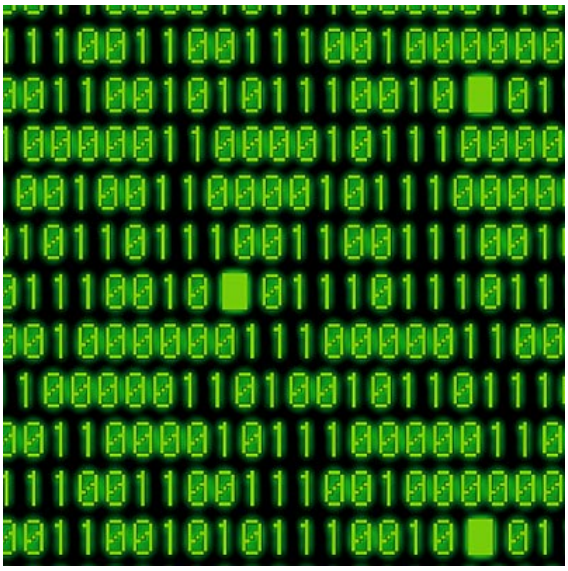
Ως γλώσσα προγραμματισμού ορίζεται γενικά η συστηματική σημειογραφία με την οποία περιγράφουμε υπολογιστικές διεργασίες ή, με απλά λόγια, μια «τεχνητή» γλώσσα που έχει σχεδιαστεί κατάλληλα ώστε να περιγράφει το σύνολο των βημάτων που μπορεί να εκτελέσει ένα μηχάνημα για να λύσει ένα πρόβλημα. Η περιγραφή της λύσης ενός προβλήματος σε ένα μηχάνημα, όπως ένας ηλεκτρονικός υπολογιστής, προϋποθέτει ένα σύνολο εντολών που «καταλαβαίνει» και «εκτελεί» ο ηλεκτρονικός υπολογιστής. Αυτό το σύνολο των εντολών καλείται αλγόριθμος και είναι υπεύθυνο να περιγράφει μεθόδους και διαδικασίες για τον τρόπο χειρισμού των δεδομένων που εισέρχονται σε έναν ηλεκτρονικό υπολογιστή, με σκοπό την ικανοποίηση και επίτευξη των επιθυμητών στόχων.

Η επίλυση ενός προβλήματος με τον ηλεκτρονικό υπολογιστή περιλαμβάνει τρία στάδια:

- τον ακριβή προσδιορισμό του προβλήματος
- την ανάπτυξη του αντίστοιχου αλγορίθμου
- τη διατύπωση του αλγορίθμου σε κατανοητή μορφή από τον υπολογιστή.

Ο προγραμματισμός ουσιαστικά ασχολείται με το τρίτο στάδιο και είναι αυτός που δίνει την εντύπωση ότι οι ηλεκτρονικοί υπολογιστές είναι «έξυπνες» μηχανές που επιλύουν τα πολύπλοκα προβλήματα.

Στην πραγματικότητα ένας ηλεκτρονικός υπολογιστής δεν είναι τίποτα παραπάνω από ένα μηχάνημα που κατανοεί μόνο δύο καταστάσεις, το μηδέν και το ένα, δηλαδή τα ψηφία του δυαδικού συστήματος. Ο υπολογιστής μπορεί απλά να αποθηκεύει στη μνήμη του ακολουθίες δυαδικών ψηφίων, να τις ανακτά, να εκτελεί στοιχειώδεις αριθμητικές πράξεις με αυτές και να τις συγκρίνει. Οι γλώσσες προγραμματισμού είναι αυτές που τελικά επιτρέπουν την «μετάφραση» του εκάστοτε προβλήματος από την ανθρώπινη γλώσσα στη γλώσσα που κατανοούν οι ηλεκτρονικοί υπολογιστές.



Εικόνα 1: *Πληροφορία σε δυαδικό κώδικα.*

1.2 Ταξινόμηση γλωσσών προγραμματισμού

Ένα από τα στοιχεία που κάνουν τις γλώσσες προγραμματισμού τόσο γοητευτικό αντικείμενο μελέτης είναι η ποικιλομορφία τους. Η αλήθεια είναι ότι η ταξινόμηση των γλωσσών προγραμματισμού τελικά δεν είναι εύκολη υπόθεση, και αυτό αποδεικνύεται από το πλήθος των διαφορετικών μοντέλων ταξινόμησης που υπάρχουν στη σχετική βιβλιογραφία. Μια γλώσσα προγραμματισμού συνήθως έχει περισσότερες από μία προγονικές γλώσσες, ενώ οι σύγχρονες γλώσσες προγραμματισμού προκύπτουν από τον συνδυασμό καινούργιων ιδεών και εννοιών με στοιχεία και χαρακτηριστικά διάφορων προγενέστερων γλωσσών, οι οποίες πολλές φορές μπορεί να μην είναι καν «συγγενικές» μεταξύ τους.

Το εγχείρημα της ταξινόμησης των γλωσσών προγραμματισμού περιπλέκεται ακόμα περισσότερο αν αναλογισθεί κανείς ότι αυτές μπορούν να ταξινομηθούν με βάση διάφορες παραμέτρους και οπτικές γωνίες, όπως είναι για παράδειγμα ο τομέας χρησιμοποίησής τους ή το στυλ προγραμματισμού, δηλαδή στον τρόπο σκέψης του δημιουργού της εκάστοτε γλώσσας. Στη συνέχεια της συγκεκριμένης παραγράφου θα γίνει μια προσπάθεια παρουσίασης κάποιων από αυτές.

Όλες οι γλώσσες προγραμματισμού που έχουν αναπτυχθεί μέχρι σήμερα αντιπροσωπεύουν διάφορες ιδέες πάνω στον προγραμματισμό, με την κάθε μια από αυτές να είναι συνήθως καλύτερα προσαρμοσμένη σε ορισμένες κατηγορίες προβλημάτων. Με άξονα αναφοράς το στυλ προγραμματισμού, οι γλώσσες προγραμματισμού διακρίνονται σε δύο βασικές κατηγορίες:

- προστακτικές, κατηγορία στην οποία ανήκει η μεγάλη πλειοψηφία των γλωσσών προγραμματισμού και σύμφωνα με την οποία ο προγραμματισμός προσεγγίζεται με όρους προστακτικών προτάσεων, για παράδειγμα υποβολής εντολών ή διαταγών, οι οποίες μεταβάλουν την κατάσταση του εκάστοτε προγράμματος. Οι συγκεκριμένες γλώσσες προγραμματισμού είναι επίσης γνωστές και ως αλγοριθμικές, επειδή είναι σχεδιασμένες για να επιτρέπουν την υλοποίηση αλγορίθμων.
- επεξηγηματικές, οι οποίες στην ουσία διαφέρουν από τις προστακτικές ως προς το γεγονός ότι περιγράφουν στον ηλεκτρονικό υπολογιστή τι πρέπει να κάνει και όχι πως πρέπει να το κάνει.

Οι συγκεκριμένες κατηγορίες μπορούν να διακριθούν σε διάφορες υποκατηγορίες, με την ταξινόμηση των γλωσσών σε αυτές βέβαια πολλές φορές να διαφέρει από την μία βιβλιογραφική πηγή στην άλλη, εξαιτίας των λόγων που προαναφέρθηκαν. Κάποιες από τις πιο συνηθισμένες που αναφέρονται είναι οι:

- αντικειμενοστραφείς, οι οποίες ανήκουν στην ευρύτερη κατηγορία των προστακτικών και βασίζονται σε εκείνο το στυλ προγραμματισμού που χρησιμοποιεί αντικείμενα (δηλαδή δομές δεδομένων που αποτελούνται από πεδία δεδομένων και μεθόδους σε συνδυασμό με τις αλληλεπιδράσεις τους) για τον σχεδιασμό εφαρμογών και προγραμμάτων.
- συναρτησιακές, οι οποίες ανήκουν στις επεξηγηματικές και δίνουν έμφαση στην εφαρμογή συναρτήσεων και όχι στην αλλαγή καταστάσεων όπως οι προστακτικές.
- λογικές ή λογικοκεντρικές, οι οποίες ανήκουν επίσης στις επεξηγηματικές και, όπως αναφέρει άλλωστε και το όνομά τους βασίζονται στη χρήση Λογικής για τη δημιουργία προγραμμάτων με σκοπό την επίλυση προβλημάτων.

Αυτές οι κατηγορίες δεν είναι όμως προσδιορισμένες με αυστηρότητα και γι' αυτό δεν είναι πάντα σαφές ποιος είναι ο σωστότερος τρόπος ταξινόμησης μιας γλώσσας. Υπάρχουν πάρα πολλές γλώσσες που κινούνται στα όρια μεταξύ αυτών των κατηγοριών, ενώ στην πράξη συναντάμε πολύ περισσότερες κατηγορίες από αυτές που προαναφέρθηκαν.

Μια άλλη ταξινόμηση μπορεί να προκύψει με βάση τον τομέα χρήσης της κάθε γλώσσας. Με βάση αυτό το κριτήριο θα μπορούσαν να διακριθούν σε:

- γλώσσες γενικής χρήσης, οι οποίες θεωρητικά μπορούν να χρησιμοποιηθούν για την επίλυση οποιουδήποτε προβλήματος, ωστόσο στην πράξη κάθε γλώσσα γενικής χρήσης έχει σχεδιαστεί για να ανταποκρίνεται καλύτερα σε συγκεκριμένη κατηγορία προβλημάτων, όπως για παράδειγμα:
  - γλώσσες επιστημονικής κατεύθυνσης
  - γλώσσες εμπορικής κατεύθυνσης
  - γλώσσες προγραμματισμού συστημάτων
  - γλώσσες τεχνητής νοημοσύνης
- γλώσσες ειδικής χρήσης, οι οποίες χρησιμοποιούνται σε ειδικές περιοχές εφαρμογών, όπως για παράδειγμα στα γραφικά με υπολογιστή, στη ρομποτική ,στα συστήματα διαχείρισης βάσεων δεδομένων και άλλα.

1.3 Σύντομη ιστορική αναδρομή γλωσσών προγραμματισμού

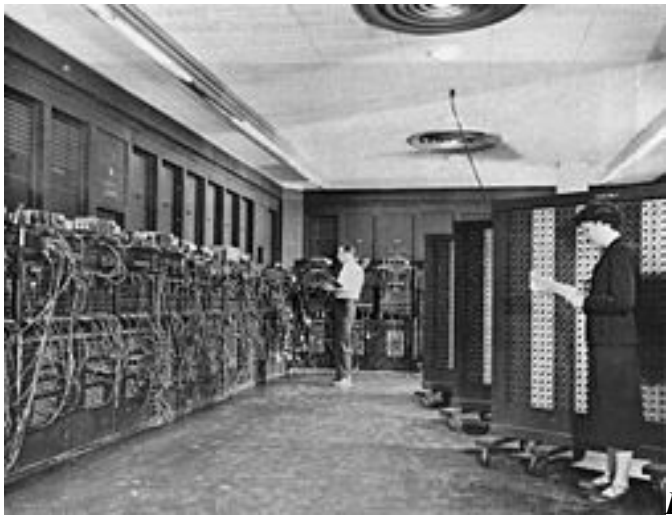
Ένα πρόβλημα που προκύπτει κατά τη μελέτη της ιστορίας των γλωσσών προγραμματισμού είναι σχετικά με την αρχή της. Αυτό οφείλεται στο γεγονός ότι οι πρώτες γλώσσες προγραμματισμού προηγήθηκαν των υπολογιστικών μηχανών με την

σύγχρονη έννοια. Για παράδειγμα, κατά τον 19ο αιώνα χρησιμοποιούνταν προγραμματιζόμενοι αργαλειοί και πιανόλες που θα μπορούσε να υποστηρίξει κανείς ότι λειτουργούσαν βάσει αυτού που σήμερα ονομάζουμε γλώσσες ειδικής χρήσης. Ωστόσο, ο πρώτος ηλεκτρονικός υπολογιστής
τέθηκε σε λειτουργία τη δεκαετία του 1940.

Όπως αναφέρθηκε και προηγουμένως, ένας ηλεκτρονικός υπολογιστής δεν είναι τίποτα παραπάνω από μια μηχανή που αναγνωρίζει απλά τα ψηφία του δυαδικού συστήματος. Έτσι, στους πρώτους υπολογιστές έπρεπε να δίνονται κατευθείαν κατάλληλες ακολουθίες από 0 και 1, δηλαδή εντολές σε μορφή κατανοητή από τον υπολογιστή. Εντούτοις, αυτό ήταν κάτι που ελάχιστοι μπορούσαν να κάνουν, αφού απαιτούσε βαθιά γνώση του υλικού και της αρχιτεκτονικής του υπολογιστή. Είναι χαρακτηριστικό ότι ο προγραμματισμός του πρώτου ηλεκτρονικού υπολογιστή, του γνωστού ENIAC, για την εκτέλεση ορισμένων υπολογισμών απαιτούσε την αλλαγή της θέσης εκατοντάδων διακοπτών και την αντίστοιχη ρύθμιση όλων των καλωδιώσεών του.

Εικόνα 2: *Μια αναπαλαιωμένη πιανόλα.*

Γι' αυτό και αυτές οι γλώσσες προγραμματισμού 1ης γενιάς, όπως ονομάζονται αντικαταστάθηκαν σχετικά γρήγορα. Αξίζει βέβαια να αναφερθεί ότι ακόμα και σήμερα οι εντολές ενός προγράμματος μετατρέπονται σε ακολουθίες που αποτελούνται από 0 και 1, οι οποίες ονομάζονται εντολές σε γλώσσα μηχανής και εκτελούνται από τον ηλεκτρονικό υπολογιστή.



Εικόνα 3: *Προγραμματιστές της εποχής ασχολούνται με τις ρυθμίσεις του ENIAC.*

Οι γλώσσες προγραμματισμού 2ης γενιάς ήταν το πρώιμο αποτέλεσμα της προσπάθειας για τη δημιουργία μιας συμβολικής γλώσσας, η οποία θα είναι κατανοητή από τον άνθρωπο και θα μετατρέπεται

εσωτερικά από τους ηλεκτρονικούς υπολογιστές στις αντίστοιχες ακολουθίες 0 και 1. Χαρακτηριστικό αυτών των γλωσσών ήταν η ευκολότερη απομνημόνευσή τους, αφού για τη σύνταξη των εντολών τους χρησιμοποιούσαν απλές λέξεις σε συνδυασμό με αριθμητικά ψηφία, που μεταφράζονταν από τον υπολογιστή σε ακολουθίες δυαδικών ψηφιών με τη βοήθεια ενός ειδικού προγράμματος, του συμβολομεταφραστή (assembler).

Οι συγκεκριμένες γλώσσες όμως είχαν και ορισμένα βασικά αρνητικά χαρακτηριστικά, όπως το γεγονός ότι παρέμεναν στενά συνδεδεμένες με την αρχιτεκτονική του εκάστοτε υπολογιστή καθώς και ότι δεν διέθεταν εντολές για σύνθετες λειτουργίες, κάτι που απαιτούσε τη σύνταξη μακροσκελών προγραμμάτων που ήταν δύσκολο να γραφτούν και ακόμα περισσότερο να συντηρηθούν. Οι γλώσσες αυτές ονομάζονται συμβολικές ή γλώσσες χαμηλού επιπέδου, επειδή ουσιαστικά η χρήση τους ήταν περιορισμένη στο εκάστοτε μηχάνημα.

Τα μειονεκτήματα των γλωσσών χαμηλού επιπέδου οδήγησαν στα τέλη της δεκαετίας του 50 στην εμφάνιση των πρώτων γλωσσών προγραμματισμού $3^{ης}$ γενιάς, ή αλλιώς γλωσσών υψηλού επιπέδου, η οποίες κατάφεραν να ξεπεράσουν το πρόβλημα της μεταφερσιμότητας και της συμβατότητας των προγόνων τους. Το 1957 η εταιρία IBM ανέπτυξε την πρώτη γλώσσα υψηλού επιπέδου, τη FORTRAN, το όνομα της οποίας προέρχεται από τις λέξεις FORmula TRANlation, δηλαδή μετάφραση τύπων. Η συγκεκριμένη γλώσσα αναπτύχθηκε για την επίλυση μαθηματικών και επιστημονικών προβλημάτων και μπορούσε να εκτελεστεί από οποιονδήποτε άλλο υπολογιστή, αρκεί να ήταν εξοπλισμένος με την κατάλληλη συσκευή μεταγλώττισης. Η FORTRAN χρησιμοποιείται ακόμα και σήμερα, μετά βέβαια από πολλές αλλαγές, προσθήκες και βελτιώσεις.

Η δεύτερη σε παλαιότητα γλώσσα προγραμματισμού υψηλού επιπέδου, η οποία πρωτοπαρουσιάστηκε το 1958, είναι η LISP. Το όνομά της προέρχεται από τις λέξεις LISt Processing (επεξεργασία λίστας), εξαιτίας του γεγονότος ότι οι βασικές δομές δεδομένων της συγκεκριμένης γλώσσας είναι σε λίστες, όπως επίσης και ο κώδικας της LISP αποτελείται από λίστες. Η LISP πρωτοδημιουργήθηκε ως ένα πρακτικό μαθηματικό σύστημα χαρακτήρων και συμβόλων για υπολογιστικά προγράμματα, στη συνέχεια όμως αποτέλεσε μια από τις πιο δημοφιλείς γλώσσες προγραμματισμού στον τομέα της έρευνας στην τεχνητή νοημοσύνη.

Άλλη μια σημαντική γλώσσα στην ιστορία του προγραμματισμού είναι η COBOL, η οποία αναπτύχθηκε το 1960. Το όνομά της προέρχεται από τις λέξεις COmmon Business-Oriented Language (κοινή γλώσσα προσανατολισμένη για επιχειρήσεις), κάτι που

φανερώνει ότι η συγκεκριμένη γλώσσα είναι κατάλληλη για την ανάπτυξη εμπορικών εφαρμογών και γενικότερα διαχειριστικών εφαρμογών, τομέας όπου η FORTRAN και η LISP υστερούσαν.

Μια από τις σημαντικότερες γλώσσες προγραμματισμού με ελάχιστη πρακτική εφαρμογή, η οποία όμως επηρέασε σημαντικά τον προγραμματισμό και τις μεταγενέστερες γλώσσες, ιδιαίτερα εκείνες που κατατάσσονται στην κατηγορία των αλγοριθμικών, είναι η ALGOL (ALGOrithmic Language – αλγοριθμική γλώσσα). Αναπτύχθηκε από μία επιτροπή Αμερικάνων και Ευρωπαίων επιστημόνων με σκοπό τη δημιουργία προγραμμάτων γενικής φύσης που να μην συνδέονται με συγκεκριμένες εφαρμογές.

Στα μέσα της δεκαετίας του 1960 αναπτύχθηκε η γλώσσα PL/1 (Programming Language One), η οποία προσπάθησε, ανεπιτυχώς, να ενσωματώσει τις καλύτερες ιδέες των γλωσσών FORTRAN και COBOL, με στόχο να καλύψει όλους τους τομείς προγραμματισμού, επιστημονικούς και εμπορικούς. Άλλες δύο σημαντικές γλώσσες γενικής χρήσεως που αναπτύχθηκαν τη δεκαετία του 1960, οι οποίες όμως χρησιμοποιούνται ακόμα και στις μέρες μας, είναι οι BASIC και PASCAL.

Η BASIC (Beginner's All Purpose Symbolic Code - συμβολικός κώδικας εντολών γενικής χρήσης για αρχάριους), όπως άλλωστε αποκαλύπτει και το όνομα της, αρχικά αναπτύχθηκε ως μια γλώσσα για την εκπαίδευση αρχαρίων στον προγραμματισμό για τον σχεδιασμό σύντομων προγραμμάτων. Στη συνέχεια, η ανάπτυξη των μικροϋπολογιστών και των προσωπικών υπολογιστών σε συνδυασμό με τις συνεχείς βελτιωμένες εκδόσεις της BASIC, την έχουν καταστήσει μια από τις δημοφιλέστερες γλώσσες στο συγκεκριμένο πεδίο, με χαρακτηριστικό σύγχρονο παράδειγμα την γνωστή Microsoft Visual Basic.

Η PASCAL, η οποία οφείλει το όνομά της στον διάσημο Γάλλο μαθηματικό του 17^{ου} αιώνα, έφερε μεγάλες αλλαγές στον προγραμματισμό και στηρίχθηκε πάνω στην ALGOL. Είναι μια γλώσσα γενικής χρήσης, η οποία είναι κατάλληλη τόσο για την εκπαίδευση όσο και τη δημιουργία ισχυρών προγραμμάτων κάθε τύπου. Χαρακτηριστικό της συγκεκριμένης γλώσσας είναι η καταλληλότητά της για τη δημιουργία δομημένων προγραμμάτων. Η PASCAL γνώρισε και συνεχίζει να γνωρίζει τεράστια εξάπλωση, ειδικά στον χώρο των μικροϋπολογιστών, και αποτέλεσε τη βάση για την ανάπτυξη άλλων ισχυρότερων γλωσσών όπως η ADA και η Modula-2.

Το 1972 εμφανίστηκε η PROLOG (PROgramming in LOGic), η οποία ήταν η πρώτη λογικοκεντρική γλώσσα και ήταν εξειδικευμένη για χρήση στον τομέα της τεχνητής νοημοσύνης. Μία ακόμη γλώσσα που γνώρισε μεγάλη διάδοση και πρωτοπαρουσιάστηκε την ίδια χρονιά είναι η C. Η C χρησιμοποιήθηκε για την ανάπτυξη του λειτουργικού

συστήματος Unix και πρόκειται για μια γλώσσα με ισχυρά χαρακτηριστικά, μερικά από αυτά κοινά με την PASCAL. Η συγκεκριμένη γλώσσα είναι κατάλληλη για την ανάπτυξη δομημένων εφαρμογών, ενώ επίσης έχει με πολλές δυνατότητες γλώσσας χαμηλού επιπέδου. Η C εξελίχτηκε στη C++ και πιο πρόσφατα στην C#, που είναι αντικειμενοστραφείς γλώσσες.

Η ραγδαία ανάπτυξη του διαδικτύου στα μέσα της δεκαετίας του 1990 δημιούργησε ευκαιρίες για την ανάπτυξη νέων γλωσσών προγραμματισμού, όπως η Perl και η JAVA, οι οποίες είναι αντικειμενοστραφείς γλώσσες, οι οποίες όμως δεν θεωρούνται εξολοκλήρου καινούργιες γλώσσες αλλά μάλλον βελτιωμένες εκδόσεις ήδη υπάρχουσων γλωσσών προγραμματισμού, μια και βασίζονται σημαντικά στις γλώσσες προγραμματισμού της οικογένειας C. Το βασικό πλεονέκτημα των συγκεκριμένων γλωσσών είναι ότι τα προγράμματά τους μπορούν να εκτελούνται από διαφορετικούς υπολογιστές, είτε προσωπικούς είτε μεγάλα συστήματα, με διαφορετικά λειτουργικά συστήματα χωρίς να απαιτούνται αλλαγές.

Επιπλέον, η εμφάνιση γραφικών περιβάλλοντων εργασίας δημιούργησε την ανάγκη για ανάπτυξη αντίστοιχου λογισμικού. Αυτό είχε σαν αποτέλεσμα την εμφάνιση καινούργιων γλωσσών ή νέων εκδόσεων προϋπάρχουσων που βασίζονταν στις έννοιες του λεγόμενου οπτικού προγραμματισμού (visual programming), και του προγραμματισμού καθοδηγούμενου από κάποιο γεγονός (object driven programming).

Με τον πρώτο όρο εννοείται η δυνατότητα δημιουργίας γραφικών περιβάλλοντων, ενώ με τον δεύτερο η δυνατότητα να ενεργοποιούνται λειτουργίες του προγράμματος με την εκτέλεση ενός γεγονότος, όπως για παράδειγμα η επιλογή μιας εντολής από ένα μενού ή το κλικ του ποντικιού. Η ανάπτυξη των συγκεκριμένων κατηγοριών προγραμματισμού είναι υπεύθυνη για το γεγονός ότι οι προσωπικοί υπολογιστές έχουν γίνει πλέον εξαιρετικά εύκολοι στη χρήση και φιλικοί για τον απλό χρήστη. Οι πιο διαδεδομένες γλώσσες προγραμματισμού σε γραφικό περιβάλλον για προσωπικούς υπολογιστές είναι η Visual Basic, η Visual C++ και η JAVA.

Οι γλώσσες προγραμματισμού υψηλού επιπέδου εμφανίζουν πολλά πλεονεκτήματα σε σχέση με τις προγενέστερές τους. Ο τρόπος έκφρασης των προβλημάτων από αυτές είναι πιο φυσικός και «ανθρώπινος». Επίσης είναι ανεξάρτητες από τον τύπο και την αρχιτεκτονική του εκάστοτε υπολογιστή με αποτέλεσμα να υπάρχει δυνατότητα μεταφερσιμότητας των προγραμμάτων. Άμεση συνέπεια των παραπάνω είναι η ευκολία στην εκμάθηση τους, όπως επίσης και στη διόρθωση των λαθών και στη συντήρηση των

προγραμμάτων τους. Όπως γίνεται εύκολα κατανοητό, αυτό βοήθησε και στην ελάττωση του χρόνου αλλά και του κόστους παραγωγής νέων προγραμμάτων.

Η τελευταία γενιά γλωσσών προγραμματισμού, η τέταρτη γενιά, αποτελείται από γλώσσες ειδικής χρήσης, στις οποίες ο χρήστης ενός υπολογιστή έχει τη δυνατότητα σχετικά εύκολα να υποβάλει ερωτήσεις στο σύστημα ή να αναπτύσσει εφαρμογές που ανακτούν πληροφορίες από βάσεις δεδομένων και να καθορίζει τον ακριβή τρόπο εμφάνισης αυτών των πληροφοριών. Κλασικό παράδειγμα γλώσσας τέταρτης γενιάς είναι η γλώσσα προγραμματισμού SQL.

1.4 Τελικά ποια είναι η καλύτερη γλώσσα;

Ένα σαφές ερώτημα που προκύπτει μετά την ταξινόμηση και την ιστορική αναδρομή στην εξέλιξη των γλωσσών προγραμματισμού, είναι η απορία σχετικά με το ποια είναι τελικά η καλύτερη γλώσσα προγραμματισμού και ποια είναι τα κριτήρια για την επιλογή της. Το συγκεκριμένο θέμα εγείρει συχνά διαμάχες. Για κάθε γλώσσα προγραμματισμού μπορεί να βρει κανείς οπαδούς της, πρόθυμους να την υπερασπιστούν ενάντια σε οποιαδήποτε άλλη. Επίσης, υπάρχουν αντιρρήσεις όσον αφορά τόσο το στυλ προγραμματισμού, όσο και την ιστορικά σπουδαιότερη και σημαντικότερη γλώσσα. Ακόμα, διαφωνίες ανακύπτουν ακόμα και ανάμεσα στους οπαδούς της ίδιας γλώσσας προγραμματισμού, σχετικά με τις προδιαγραφές, τις βελτιώσεις και τις προσθήκες για κάθε επόμενη επίσημη έκδοση της αγαπημένης τους γλώσσας.

Τα κριτήρια για την επιλογή της καταλληλότερης γλώσσας διαμορφώνονται ανάλογα με την περίσταση. Σημαντικός παράγοντας είναι σίγουρα το πεδίο χρήσης της επιθυμούμενης εφαρμογής, καθώς επίσης και οι στόχοι του εκάστοτε project. Όπως προαναφέρθηκε, υπάρχουν γλώσσες κατάλληλες για ανάπτυξη εξειδικευμένων εφαρμογών και άλλες κατάλληλες για γενική χρήση. Επίσης άλλες γλώσσες είναι καταλληλότερες για τον τομέα της επιστήμης, άλλες για τον τομέα της εκπαίδευσης και άλλες για την ανάπτυξη εμπορικών εφαρμογών.

Εξίσου σημαντικό κριτήριο είναι και οι δυνατότητες της εκάστοτε γλώσσας, αναφορικά με τη συνέπεια, την αξιοπιστία, τη γρήγορη μετάφραση και τη μεταφερσιμότητα της εφαρμογής που πρόκειται να χρησιμοποιηθεί. Υπάρχουν γλώσσες ισχυρές, καθώς επίσης και γλώσσες χωρίς μεγάλες δυνατότητες αλλά εύκολες στην εκμάθηση. Οι γνώσεις του προγραμματιστή είναι και αυτές σπουδαίος παράγοντας. Αν σε όλα αυτά προσθέσουμε και το κόστος παραγωγής, είτε αυτό έχει να κάνει με την εκτέλεση,

μετάφραση, δημιουργία και συντήρηση του προγράμματος, είτε με τους διαθέσιμους πόρους και μέσα για την υλοποίηση του εκάστοτε project (για παράδειγμα, υπάρχουν γλώσσες που επιτρέπουν την εύκολη ανάπτυξη εφαρμογών σε γραφικό περιβάλλον και άλλες που εκμεταλλεύονται τα παράλληλα συστήματα), γίνεται κατανοητό ότι η επιλογή της καταλληλότερης γλώσσας προγραμματισμού δεν είναι εύκολη υπόθεση.

Μετά από όλα αυτά, καταλήγουμε στο συμπέρασμα ότι μπορούμε να ισχυριστούμε με βεβαιότητα ότι δεν υπάρχει μια γλώσσα προγραμματισμού που να είναι αντικειμενικά καλύτερη από τις υπόλοιπες και ούτε πρόκειται να υπάρξει στο μέλλον.

1.5 Τεχνικές σχεδίασης προγραμμάτων

Στον «αγώνα» για την, όσο το δυνατόν, πιο αξιόπιστη και ταυτόχρονα εύκολη από όλες τις απόψεις επίλυση προβλημάτων με τη χρήση ηλεκτρονικών υπολογιστών, γίνονται συνεχείς προσπάθειες για ανάπτυξη μεθοδολογιών και τεχνικών προγραμματισμού που θα εξασφαλίζουν τη δημιουργία απλών, κομψών και συνάμα λειτουργικών προγραμμάτων και εφαρμογών.

Μια δημοφιλής στρατηγική είναι η τεχνική της ιεραρχικής σχεδίασης και επίλυσης ή αλλιώς η διαδικασία σχεδίασης «από πάνω προς τα κάτω» (top-down program design). Σκοπός της συγκεκριμένης τεχνικής είναι η διάσπαση του προβλήματος σε απλούστερα υποπροβλήματα, των οποίων η επίλυση είναι ευκολότερη. Έτσι, η διαδικασία σχεδίασης περιλαμβάνει τον καθορισμό των βασικών λειτουργιών ενός προγράμματος σε ανώτερο επίπεδο, και στη συνέχεια τη διάσπαση των λειτουργιών αυτών σε όλο και μικρότερες, μέχρι το τελευταίο στάδιο όπου οι λειτουργίες είναι πολύ απλές και επιλύονται ευκολότερα.

Η ιεραρχική σχεδίαση προγράμματος υλοποιείται με τον τμηματικό προγραμματισμό. Μετά την ανάλυση του προβλήματος σε αντίστοιχα υποπροβλήματα, κάθε υποπρόβλημα αποτελεί ανεξάρτητη ενότητα (module), που γράφεται ξεχωριστά από τα υπόλοιπα τμήματα προγράμματος. Με αυτόν τον τρόπο ο τμηματικός προγραμματισμός διευκολύνει τη δημιουργία του προγράμματος, μειώνει τα λάθη και επιτρέπει την ευκολότερη παρακολούθηση, κατανόηση και συντήρηση του προγράμματος από τρίτους.

Η μεθοδολογία που έχει επικρατήσει σήμερα απόλυτα και την υποστηρίζουν σχεδόν όλες οι σύγχρονες γλώσσες προγραμματισμού, είναι ο δομημένος προγραμματισμός (structured programming) που πρωτοπαρουσιάστηκε στα μέσα της δεκαετίας του 1960. Ο δομημένος προγραμματισμός δεν είναι απλώς ένα είδος προγραμματισμού, είναι μια

μεθοδολογία σύνταξης προγραμμάτων που έχει σκοπό να βοηθήσει τον προγραμματιστή στην ανάπτυξη σύνθετων προγραμμάτων, να μειώσει τα λάθη, να εξασφαλίσει την εύκολη κατανόηση των προγραμμάτων και να διευκολύνει τις διορθώσεις και τις αλλαγές σ' αυτά.

Ο δομημένος προγραμματισμός στηρίζεται στη χρήση τριών και μόνο στοιχειωδών λογικών δομών, τη δομή της ακολουθίας, τη δομή της επιλογής και τη δομή της επανάληψης. Όλα τα προγράμματα μπορούν να γραφούν χρησιμοποιώντας μόνο αυτές τις τρεις δομές καθώς και συνδυασμό τους. Ο δομημένος προγραμματισμός ενθαρρύνει και βοηθάει την ανάλυση του προγράμματος σε επί μέρους τμήματα, έτσι ώστε σήμερα ο όρος δομημένος προγραμματισμός περιέχει τόσο την ιεραρχική σχεδίαση όσο και τον τμηματικό προγραμματισμό.

Η συγκεκριμένη μεθοδολογία εμφανίζει πλήθος πλεονεκτημάτων. Τέτοια είναι η δημιουργία απλούστερων προγραμμάτων, η άμεση μεταφορά των αλγορίθμων σε προγράμματα, η διευκόλυνση της ανάλυσης του προγράμματος σε τμήματα, ο περιορισμός των λαθών κατά την ανάπτυξη του προγράμματος, η ευκολία στην ανάγνωση και κατανόηση του προγράμματος από τρίτους και, τέλος, η ευκολότερη διόρθωση και συντήρηση.

Μια σχετικά πιο σύγχρονη τάση αντιμετώπισης προγραμματιστικών αντιλήψεων και δομών είναι ο αντικειμενοστραφής (object-oriented) προγραμματισμός. Η αντικειμενοστραφής σχεδίαση εκλαμβάνει ως πρωτεύοντα δομικά στοιχεία για τον σχεδιασμό εφαρμογών και προγραμμάτων τα δεδομένα, από τα οποία δημιουργούνται με κατάλληλη μορφοποίηση τα αντικείμενα (objects). Τα αντικείμενα είναι δομές δεδομένων που αποτελούνται από πεδία δεδομένων και μεθόδους σε συνδυασμό με τις αλληλεπιδράσεις τους, ή με πιο απλά λόγια μικρές δομές δεδομένων που «γνωρίζουν» πώς να χειρίζονται τους εαυτούς τους. Η συγκεκριμένη τεχνική χρησιμοποιεί την ιεραρχική σχεδίαση, τον τμηματικό προγραμματισμό και ακολουθεί τις αρχές του δομημένου προγραμματισμού

Σε αυτό το σημείο αξίζει να σημειωθεί ότι σχετικά πρόσφατα εμφανίστηκαν υπολογιστές που ξεφεύγουν από την κλασική αρχιτεκτονική και διαθέτουν περισσότερους από έναν επεξεργαστές. Οι επεξεργαστές αυτοί μοιράζονται την ίδια μνήμη και λειτουργούν παράλληλα εκτελώντας διαφορετικές εντολές του ιδίου προγράμματος. Οι υπολογιστές αυτοί εμφανίζονται θεωρητικά να πετυχαίνουν ταχύτητες, που είναι ασύλληπτες για τους τυπικούς υπολογιστές με έναν επεξεργαστή. Για να εκμεταλλευτούμε όμως την ταχύτητα που προσφέρει η αρχιτεκτονική αυτή, πρέπει το πρόβλημα να διαιρεθεί σε τμήματα που εκτελούνται παράλληλα και στη συνέχεια να προγραμματιστεί σε ένα περιβάλλον που να επιτρέπει τον παράλληλο προγραμματισμό.

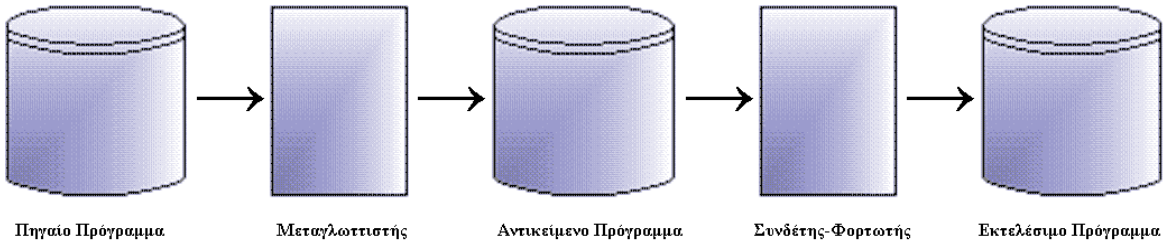
Εικόνα 4: *Υπολογιστής στον οποίο πραγματοποιείται παράλληλος προγραμματισμός.*

1.6 Προγραμματιστικά περιβάλλοντα

Όπως αναφέρθηκε και παραπάνω, κάθε πρόγραμμα που γράφτηκε σε οποιαδήποτε γλώσσα προγραμματισμού, για να εκτελεσθεί σε έναν ηλεκτρονικό υπολογιστή πρέπει να μετατραπεί σε μορφή αναγνωρίσιμη και εκτελέσιμη από τον υπολογιστή, δηλαδή σε εντολές γλώσσας μηχανής. Η μετατροπή αυτή επιτυγχάνεται με τη χρήση ειδικών μεταφραστικών προγραμμάτων.

Υπάρχουν δύο μεγάλες κατηγορίες τέτοιων προγραμμάτων οι μεταγλωττιστές (compilers) και οι διερμηνευτές (interpreters). Ο μεταγλωττιστής δέχεται στην είσοδο ένα πρόγραμμα γραμμένο σε μια γλώσσα υψηλού επιπέδου και παράγει ένα ισοδύναμο πρόγραμμα σε γλώσσα μηχανής. Το τελευταίο μπορεί να εκτελείται οποτεδήποτε από τον υπολογιστή και είναι τελείως ανεξάρτητο από το αρχικό πρόγραμμα. Αντίθετα ο διερμηνευτής διαβάζει μία προς μία τις εντολές του αρχικού προγράμματος και για κάθε μια εκτελεί αμέσως μια ισοδύναμη ακολουθία εντολών μηχανής.

Εικόνα 5: *Μεταγλώττιση και σύνδεση του προγράμματος.*



Η χρήση μεταγλωττιστή έχει το μειονέκτημα, ότι προτού χρησιμοποιηθεί ένα πρόγραμμα, πρέπει να περάσει από τη διαδικασία της μεταγλώττισης και σύνδεσης. Από την άλλη μεριά, η χρήση διερμηνευτή έχει το πλεονέκτημα της άμεσης εκτέλεσης και συνεπώς και της άμεσης διόρθωσης, όμως η εκτέλεση του προγράμματος καθίσταται πιο αργή, μερικές φορές σημαντικά πιο αργή, από εκείνη του ισοδύναμου εκτελέσιμου προγράμματος που παράγει ο μεταγλωττιστής. Τα σύγχρονα προγραμματιστικά περιβάλλοντα παρουσιάζονται συνήθως με μεικτές υλοποιήσεις, όπου χρησιμοποιείται διερμηνευτής κατά τη φάση δημιουργίας του προγράμματος και μεταγλωττιστής για την τελική έκδοση και εκμετάλλευση του προγράμματος

Το αρχικό πρόγραμμα λέγεται πηγαίο πρόγραμμα (source) ενώ το πρόγραμμα που παράγεται από τον μεταγλωττιστή λέγεται αντικείμενο πρόγραμμα (object). Το αντικείμενο πρόγραμμα είναι μεν σε μορφή κατανοητή από τον υπολογιστή, αλλά συνήθως δεν είναι σε θέση να εκτελεστεί. Χρειάζεται να συμπληρωθεί και να συνδεθεί με κάποια άλλα τμήματα προγράμματος που είναι απαραίτητα για την εκτέλεση του, τμήματα που είτε τα γράφει ο προγραμματιστής είτε βρίσκονται στις βιβλιοθήκες της γλώσσας. Το πρόγραμμα που επιτρέπει αυτή τη σύνδεση ονομάζεται συνδέτης-φορτωτής (linker-loader). Το αποτέλεσμα του συνδέτη είναι η παραγωγή του εκτελέσιμου προγράμματος, το οποίο είναι το τελικό

πρόγραμμα που εκτελείται από τον υπολογιστή. Για τον λόγο αυτό η συνολική διαδικασία αποκαλείται μεταγλώττιση και σύνδεση.

Η δημιουργία του εκτελέσιμου προγράμματος γίνεται μόνο στην περίπτωση, που το αρχικό πρόγραμμα δεν περιέχει λάθη, κάτι που δυστυχώς τις περισσότερες φορές αρχικά είναι δύσκολο να αποφευχθεί. Τα λάθη κάθε προγράμματος είναι γενικά δύο ειδών, λογικά και συντακτικά. Τα λογικά λάθη εμφανίζονται μόνο στην εκτέλεση, ενώ τα συντακτικά λάθη στο στάδιο της μεταγλώττισης. Τα λογικά λάθη που είναι τα πλέον σοβαρά και δύσκολα στη διόρθωση τους, οφείλονται σε σφάλματα κατά την υλοποίηση του αλγορίθμου. Τα συντακτικά λάθη οφείλονται σε αναγραμματισμούς ονομάτων εντολών,

παράληψη δήλωσης δεδομένων και πρέπει πάντα να διορθωθούν, ώστε να παραχθεί το τελικό εκτελέσιμο πρόγραμμα.

Ο μεταγλωττιστής ή ο διερμηνευτής ανιχνεύει λοιπόν τα λάθη και εμφανίζει κατάλληλα διαγνωστικά μηνύματα. Το στάδιο που ακολουθεί είναι η διόρθωση των λαθών. Το διορθωμένο πρόγραμμα επαναϋποβάλλεται για μεταγλώττιση και η διαδικασία αυτή επαναλαμβάνεται, μέχρι να εξαλειφθούν πλήρως όλα τα λάθη. Για την αρχική σύνταξη των προγραμμάτων και τη διόρθωση τους στη συνέχεια χρησιμοποιείται ένα ειδικό πρόγραμμα που ονομάζεται συντάκτης (editor). Ο συντάκτης είναι ουσιαστικά ένας "μικρός" επεξεργαστής κειμένου με μειωμένες δυνατότητες , οι οποίες όμως διευκολύνουν τη γρήγορη εγγραφή των εντολών των προγραμμάτων.

Βλέπουμε λοιπόν ότι για τη δημιουργία, τη μετάφραση και την εκτέλεση της εφαρμογής που θα δημιουργηθεί χρησιμοποιώντας μια γλώσσα προγραμματισμού, είναι απαραίτητο ένα πακέτο προγραμμάτων, το οποίο, ευτυχώς για τον προγραμματιστή, τα σύγχρονα προγραμματιστικά περιβάλλοντα παρέχουν με ενιαίο τρόπο.

1.7 Η γλώσσα Handel-C

Η Handel-C είναι μια γλώσσα προγραμματισμού, η οποία όμως χρησιμοποιείται κυρίως ως γλώσσα περιγραφής υλικού (Hardware Description Language και εν συντομία HDL). Οι γλώσσες αυτού του είδους χρησιμοποιούνται για την περιγραφή ηλεκτρονικών κυκλωμάτων και πιο συγκεκριμένα ψηφιακών λογικών κυκλωμάτων ηλεκτρονικών υπολογιστών. Αυτό που επιτυγχάνεται με τη χρήση των συγκεκριμένων γλωσσών είναι η περιγραφή, ο έλεγχος και η επαλήθευση του σχεδιασμού, της οργάνωσης και της λειτουργίας των ηλεκτρονικών κυκλωμάτων. Οι HDLs χρησιμοποιούνται για τη δημιουργία εφαρμογών ελέγχου των προδιαγραφών κάποιου ηλεκτρονικού υλικού. Ένα τέτοιο πρόγραμμα εξομοίωσης λειτουργίας, επιδόσεων και χρόνου, δίνει την ευκαιρία στον σχεδιαστή του εκάστοτε υλικού να εξετάσει ένα μοντέλο του υλικού πριν την τελική κατασκευή του.

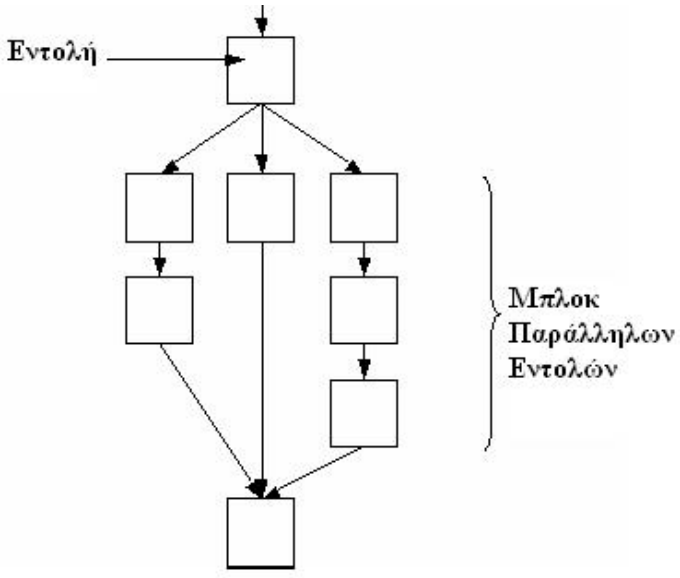
Συγκεκριμένα, η Handel-C χρησιμοποιείται για την εξομοίωση της λειτουργίας ολοκληρωμένων κυκλωμάτων FPGAs (κυκλώματα που σχεδιάζονται έτσι ώστε να μπορούν να διαμορφωθούν από τον πελάτη ή τον σχεδιαστή μετά την κατασκευή τους) και ASICs (όπως για παράδειγμα ένα τσιπ που έχει σχεδιαστεί για αποκλειστική χρήση σε συσκευές κινητών τηλεφώνων). Η Handel-C είναι ένα πλούσιο υποσύνολο της γλώσσας

προγραμματισμού C, το οποίο περιέχει μεταβλητές επεκτάσεις για να επιτυγχάνεται έλεγχος και παραλληλισμός υλικού.

Οι ιστορικές ρίζες της Handel-C εντοπίζονται σε μια σειρά από HDLs που αναπτύχθηκαν στο Πανεπιστήμιο της Οξφόρδης. Η Handel-HDL αναπτύχθηκε σε Handel-C το 1996 και «ωρίμασε» ως προϊόν της εταιρίας Embedded Solutions Limited (ESL). Το 2000 η ESL μετονομάσθηκε σε Celoxica, το 2006 συγχωνεύθηκε με την εταιρία Catalytic με αποτέλεσμα τον σχηματισμό της εταιρίας Agility, η οποία το 2009 σταμάτησε τη λειτουργία της λόγω αδυναμίας να εξασφαλίσει επιπλέον κεφάλαιο ή πίστωση. Αργότερα την ίδια χρονιά η εταιρία Mentor Graphics εξαγόρασε τα περιουσιακά στοιχεία της Agility που σχετίζονταν με την γλώσσα προγραμματισμού C, ανάμεσα στα οποία ήταν και η Handle-C.

Δεδομένου ότι η Handel-C βασίζεται στη σύνταξη της γλώσσας C, τα προγράμματα που είναι γραμμένα σε Handle-C είναι εν δυνάμει σειριακά, δηλαδή το γράψιμο της μιας εντολής μετά την άλλη υποδηλώνει ότι αυτές οι οδηγίες πρέπει να εκτελεσθούν με ακριβώς την ίδια σειρά. Από τα δυνατά σημεία της συγκεκριμένης γλώσσας είναι ότι μπορεί να εκτελέσει παράλληλα εντολές, να χρησιμοποιεί κανάλια επικοινωνίας μεταξύ των παράλληλων εντολών, όπως και να επιτρέπει την ύπαρξη κοινών πεδίων και μεταβλητών μεταξύ των παράλληλων εντολών. Η Handel-C παρέχει εργαλεία που ελέγχουν τη ροή του προγράμματος, όπως εκτέλεση του κώδικα ανάλογα με την τιμή μιας έκφρασης ή επανάληψη του κώδικα με τη χρήση ενός εργαλείου βρόγχου.

Για τη δημιουργία αποτελεσματικών προγραμμάτων με την Handle-C είναι σημαντικό να δίνεται εντολή στον μεταγλωττιστή της να χρησιμοποιεί κατάλληλα το υλικό ώστε να εκτελούνται οι εντολές παράλληλα, κάτι που επιτυγχάνεται με χρήση της λέξης κλειδιού par. Αυτό σημαίνει ότι όταν δίνεται εντολή να εκτελεσθούν δύο ενέργειες παράλληλα, αυτές θα εκτελεσθούν ακριβώς την ίδια χρονική στιγμή από δύο διαφορετικά τμήματα του υλικού. Όταν αντιμετωπίζεται ένα παράλληλο μπλοκ εντολών, η



Εικόνα 6: *Εκτέλεση μπλοκ παράλληλων εντολών.*

ροή εκτέλεσης χωρίζεται στην αρχή του παράλληλου μπλοκ εντολών και κάθε κλάδος του μπλοκ εκτελείται ταυτόχρονα. Στη συνέχεια, η ροή εκτέλεσης επανασυνδέεται στο τέλος του μπλοκ αφού ολοκληρωθούν όλοι οι κλάδοι. Όποιοι κλάδοι ολοκληρωθούν νωρίτερα αναγκάζονται να περιμένουν μέχρι να ολοκληρωθεί και ο τελευταίος κλάδος πριν συνεχιστεί η ροή των εντολών.

Όσον αφορά τα κανάλια επικοινωνίας, αυτά παρέχουν έναν σύνδεσμο μεταξύ των κλάδων εντολών που εκτελούνται παράλληλα. Ένας παράλληλος κλάδος αποδίδει δεδομένα στο κανάλι και κάποιος άλλος κλάδος διαβάζει δεδομένα από το κανάλι. Επιπλέον, για να μην υπάρχουν προβλήματα με τη χρήση κοινών μεταβλητών σε περισσότερες από μία επεξεργασίες δεδομένων, η Handle-C περιέχει την εντολή prialt, η οποία δίνει στον προγραμματιστή τη δυνατότητα επιλογής σχετικά με τη σειρά ανάγνωσης και χρησιμοποίησης της κοινής μεταβλητής από μια ομάδα καναλιών.

2. ΕΙΣΑΓΩΓΗ ΣΤΑ FPGA

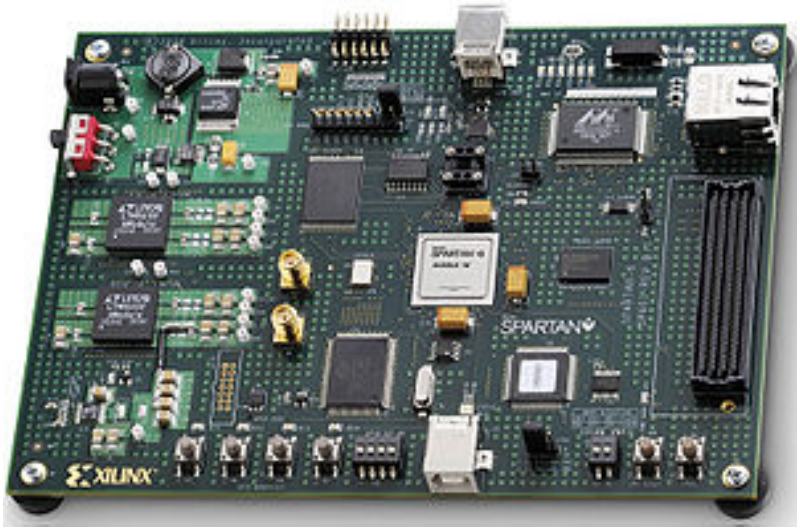
2.1 Γενική εισαγωγή

Το FPGA ή Field Programmable Gate Array ή "συστοιχία επιτόπια προγραμματιζόμενων πυλών" είναι τύπος ολοκληρωμένου κυκλώματος ικανό να επαναπροσδιορίζει την εσωτερική του δομή ανάλογα με τις ανάγκες του σχεδιαστή, Εξ ου και ο όρος "προγραμματιζόμενων πυλών". Η τροποποίηση του FPGA γίνεται με περιγραφικές γλώσσες (Hardware Description Languages-HDL), όπως VHDL, Verilog, Handle-C κ.ο.κ. Τα FPGAs μπορούν να χρησιμοποιηθούν στην υλοποίηση οποιουδήποτε λογικού κυκλώματος όπως καταχωρητές, απαριθμητές ακόμη και μικρών επεξεργαστών, αντικαθιστώντας πολλά από τα ήδη υπάρχοντα ολοκληρωμένα, αυξάνοντας αντίστοιχα τις επιδόσεις και μειώνοντας το κόστος. Η σημαντικότερη καινοτομία που εισήγαγαν είναι η ικανότητα επαναπροσδιορισμού του εσωτερικού τους κάτι που φάνταζε αδύνατο πριν την εμφάνιση τους. Αυτό το χαρακτηριστικό τα έκανε να επικρατήσουν έναντι των ASIC (application-specific integrated circuit ή ολοκληρωμένα κυκλώματα για συγκεκριμένη εφαρμογή), προσφέροντας πλεονεκτήματα για πολλές εφαρμογές.

Εικόνα 1.1: FPGA της οικογένειας Stratix IV της εταιρίας Altera.

Τα FPGAs περιέχουν ομάδες από λογικές πύλες διασυνδεδεμένες μεταξύ τους ώστε να φέρνουν εις πέρας μια συγκεκριμένη λειτουργία. Αυτές οι ομάδες ονομάζονται "Λογικά μπλοκ" και δεν μπορούμε να επέμβουμε στην δομή τους. Μέσω κάποιων ιεραρχικών κανόνων και δυνατών συνδέσεων τα "Λογικά μπλοκ" συνδέονται μεταξύ τους (επαναπροσδιορίζονται) και φέρνουν εις πέρας το επιθυμητό αποτέλεσμα είτε αυτό είναι απλές λογικές πύλες ή πολύπλοκες λογικές συναρτήσεις. Τα περισσότερα FPGAs περιέχουν μέσα στα λογικά μπλοκ και στοιχεία από μνήμες, τα οποία είτε είναι μερικά Flip-Flops ή συστάδες από κύτταρα μνήμης (DRAM, SRAM). Εκτός από τις ψηφιακές λειτουργίες, μερικές FPGAs έχουν και αναλογικά χαρακτηριστικά. Ένα από αυτά είναι η ομαδοποίηση των ακροδεκτών ανάλογα με τις δυνατότητες τους τόσο σε ικανότητα οδήγησης απαιτητικών στοιχείων από πλευράς ισχύος, όσο και από πλευράς ταχύτητας. Ένα ακόμη χαρακτηριστικό τους το οποίο είναι δημιούργημα της εξελιγμένης μικροηλεκτρονικής είναι η ενσωμάτωση σε αυτά διαφορικών ενισχυτών, μετατροπέων σήματος από αναλογικό σε ψηφιακό και το αντίστροφο και τέλος ακόμη και PLL (Phase Locked-Loop). Τέτοια χαρακτηριστικά ξενίζουν τους σχεδιαστές οι οποίοι είχαν να κάνουν με αμιγώς ψηφιακά η

αναλογικά ολοκληρωμένα, αλλά ταυτόχρονα χαράσσουν το μέλλον και τους εφοδιάζουν με δυνατότητες που απογειώνουν την παραγωγικότητα τους χωρίς αντίκτυπο στην ποιότητα του τελικού προϊόντος.



Εικόνα 1.2: Αναπτυξιακή πλακέτα της εταιρίας Xilinx.

2.2 Ιστορία των FPGA

Η βιομηχανία των FPGAs δημιουργήθηκε από την ένωση δύο προϊόντων, της προγραμματιζόμενης μνήμης μόνο για ανάγνωση (programmable read-only memory ή PROM) και των προγραμματιζόμενων λογικών διατάξεων (programmable logic devices ή PLDs). Και οι δύο είχαν την δυνατότητα να προγραμματιστούν είτε από το εργοστάσιο είτε μετέπειτα.

Στα τέλη της δεκαετίας του 1980 το υπουργείο άμυνας των ΗΠΑ και το Surface Warfare Department χρηματοδοτούν ένα πείραμα το οποίο προτάθηκε από τον Steve Casselman, με σκοπό την ανάπτυξη ενός υπολογιστή ο οποίος θα περιείχε 600.000 επαναπρογραμματιζόμενες πύλες. Ο Casselman ολοκλήρωσε τον υπολογιστή και ένα δίπλωμα ευρεσιτεχνίας που σχετίζονταν με το σύστημα εκδόθηκε το 1992.

Μερικές από τις θεμελιώδεις γνώσεις και τεχνολογίες που ενστερνίστηκε η βιομηχανία των επαναπροσδιοριζόμενων λογικών πυλών βασίζονται σε διπλώματα ευρεσιτεχνίας που κατέθεσαν οι David W. Page και Luverne R. Peterson το 1985.

Οι ιδρυτές της Xilinx Co., Ross Freeman και Bernard Vonderschmitt , εφηύραν το πρώτο εμπορικά βιώσιμο FPGA (XC2064), το 1985. Το XC2064 είχε προγραμματιζόμενες πύλες και προγραμματιζόμενες διασυνδέσεις μεταξύ των πυλών και ήταν η αρχή μιας νέας τεχνολογίας στην αγορά. Το XC2064 καυχιόταν για 64 διαμορφώσιμα μπλοκ λογικής (CLBs), με δύο πίνακες αναζήτησης 3-εισροών (LUTs). Περισσότερα από 20 χρόνια αργότερα, ο Freeman εντάχθηκε στο Inventor Hall of Fame για την εφεύρεσή του.

Η Xilinx συνέχισε χωρίς ανταγωνισμό και αναπτυσσόταν γοργά από το 1985 μέχρι τα μέσα της δεκαετίας του 1990, όταν οι ανταγωνιστές της πλέον πολλαπλασιάστηκαν, κερδίζοντας σημαντικά μερίδια της αγοράς. Από το 1993, η Actel εξυπηρετούσε περίπου το 18% της αγοράς.

Η δεκαετία του 1990 ήταν ένα εκρηκτικό διάστημα για τα FPGAs, τόσο στην πολυπλοκότητα όσο και στον όγκο της παραγωγής. Στις αρχές της δεκαετίας του 1990, τα FPGAs χρησιμοποιήθηκαν κυρίως στις τηλεπικοινωνίες και τη δικτύωση. Μέχρι το τέλος

της δεκαετίας, τα FPGAs βρήκαν τον δρόμο τους σε τελικούς καταναλωτές, αυτοκινητιστικές και βιομηχανικές εφαρμογές.

Τα FPGAs έγιναν ευρέως γνωστά το 1997, όταν ο Adrian Thompson, ένας ερευνητής που εργαζόταν στο Πανεπιστήμιο του Sussex, στην Αγγλία, συγχώνευσε γενετικούς αλγορίθμους και FPGAs για να δημιουργήσει μια συσκευή αναγνώρισης ήχου. Ο αλγόριθμος του Thompson διαμορφώνει μια σειρά από 10 x 10 κελιά σε ένα FPGA της Xilinx που κάνει διάκριση ανάμεσα σε δύο τόνους, χρησιμοποιώντας αναλογικά χαρακτηριστικά του FPGA. Η εφαρμογή των γενετικών αλγορίθμων για την διαμόρφωση συσκευών FPGA αναφέρεται πλέον ως Evolvable hardware.

2.3 Σύγχρονη εξέλιξη των FPGA

Η σύγχρονη τάση η όποια απογειώνει σε εφαρμογές την χρήση των FPGAs δημιουργώντας μια καινοτόμα αρχιτεκτονική είναι ο συνδυασμός των παραδοσιακών λογικών μπλοκ τα οποία συναντώνται στα FPGAs με ενσωματωμένους μικροεπεξεργαστές και συναφή περιφερειακά ώστε να δημιουργηθεί ένα πλήρες σύστημα ενσωματωμένο σε ένα και μοναδικό ολοκληρωμένο. Το έργο αυτό αντανακλά την προτεινομένη αρχιτεκτονική από τον Ron Perlof και Hana Potash της εταιρίας Advanced Systems Group οι οποίοι δημιούργησαν μια επαναπροσδιοριζόμενη αρχιτεκτονική επεξεργαστή σε ένα και μόνο ολοκληρωμένο το οποίο και ονόμασαν SB24. Το συγκεκριμένο εγχείρημα ολοκληρώθηκε με επιτυχία το 1982. Παραδείγματα αυτής της υβριδικής τεχνολογίας μπορούμε σήμερα να τα δούμε σε FPGA της εταιρίας Xilinx με ονομασία Virtex-II Pro και Virtex-4, τα οποία περιέχουν έναν η περισσότερους επεξεργαστές με αρχιτεκτονική PowerPC ενσωματωμένους σε συσκευές FPGA. Ένα άλλο παράδειγμα τέτοιας συσκευής είναι της εταιρίας Atmel με κωδική ονομασία FPSLIC, το οποίο συνδυάζει FPGA και μικροεπεξεργαστή AVR. Αντίστοιχα οι συσκευές SmartFusion της Actel ενσωματώνουν μικροεπεξεργαστές βασισμένους στην αρχιτεκτονική ARM. Χαρακτηριστικό παράδειγμα της τελευταίας εταιρίας είναι το μέγεθος του επεξεργαστή το οποίο είναι ενσωματωμένο στο ολοκληρωμένο και περιλαμβάνει πυρήνα επεξεργαστή τύπου Cortex-M3 με 512KB Flash μνήμης και 64KB RAM μαζί με περιφερειακά, όπως πολυκάναλο ADC και DACs ενσωματωμένα στο FPGA.

Μια εναλλακτική προσέγγιση χρήσης επεξεργαστών είναι η χρήση των πυρήνων επεξεργαστών που υλοποιούνται στο FPGA.

Όπως αναφέραμε παραπάνω, πολλά σύγχρονα FPGAs έχουν την δυνατότητα να επαναπρογραμματιστούν σε "πραγματικό χρόνο", και αυτό είναι καθοριστικής σημασίας για την υλοποίηση της ιδέας της επαναπροσδιοριζόμενης υπολογιστικής ή των επαναπροσδιοριζόμενων συστημάτων. Όπως επεξεργαστές οι οποίοι αναπροσαρμόζουν την εσωτερική τους δομή ώστε να βελτιστοποιήσουν την απόδοση τους σε κάποια συγκεκριμένη εργασία. Χαρακτηριστικό παράδειγμα είναι ο Mitrion Virtual Processor της εταιρίας Mitrionics, υλοποιημένος σε FPGAs. Μπορεί να μην υποστηρίζει δυναμικό επαναπροσδιορισμό σε πραγματικό χρόνο, ωστόσο μπορεί να προσαρμοστεί σε ένα συγκεκριμένο πρόγραμμα.

Παρόλα ταύτα, καινούργιες αρχιτεκτονικές στηριζόμενες σε υλοποιήσεις με FPGA αρχίζουν να εμφανίζονται. Επαναπροσδιοριζόμενοι μικροεπεξεργαστές με βάση το λογισμικό τους όπως ο Stretch S5000, υιοθετώντας μια υβριδική προσέγγιση και χρησιμοποιώντας ένα πλήθος από κανονικούς επεξεργαστές και επεξεργαστές υλοποιημένους σε FPGA στο ίδιο ολοκληρωμένο.

2.4 Πύλες των FPGA

- 1987: 9.000 πύλες, Xilinx
- 1992: 600.000, Naval Surface Warfare Department
- Αρχές του 2000: Εκατομμύρια

2.5 Το μέγεθος της αγοράς των FPGA (MARKET SIZE)

- 1985: Πρώτη εμπορική τεχνολογία FPGA που εφευρέθηκε από την Xilinx
- 1987: 14 εκατομμύρια δολάρια
- 1993: 385 εκατομμύρια δολάρια
- 2005: 1,9 δισεκατομμύρια δολάρια
- 2010 οι εκτιμήσεις: 2,75 δισεκατομμύρια δολάρια

2.6 Ζητήματα ασφαλείας

Σε θέματα που αφορούν την ασφάλεια, τα FPGAs έχουν τόσο πλεονεκτήματα όσο και μειονεκτήματα σε σχέση με τους ανταγωνιστές τους όπως τα ολοκληρωμένα τα οποία σχεδιάστηκαν και υλοποιήθηκαν για συγκεκριμένο σκοπό και τους μικροεπεξεργαστές. Εύκολα μπορούμε να καταλάβουμε ότι είναι αδύνατον να αλλάξουμε την εσωτερική δομή ενός ολοκληρωμένου ή ενός μικροεπεξεργαστή. Ωστόσο αυτή η σταθερότητα δεν υπάρχει στα FPGAs και κατά την διάρκεια του προγραμματισμού του FPGA και μετά ο κώδικας μπορεί να είναι εκτεθειμένος. Ως λύση σε αυτό το πρόβλημα, ορισμένα FPGAs υποστηρίζουν κρυπτογράφηση.

2.7 Εφαρμογές των FPGA

Τα FPGAs συναντώνται σε πληθώρα εφαρμογών όπως ψηφιακή επεξεργασία σήματος, ψηφιακό ραδιόφωνο, αεροδιαστημική και αμυντική τεχνολογία, ιατρική απεικόνιση, υπολογιστική όραση, αναγνώριση ομιλίας, κρυπτογραφία, εξομοίωση πειραματικών μονάδων υπολογιστών, ραδιοαστρονομία, ανίχνευση μετάλλων και σε πολλές άλλες γοργά αναπτυσσόμενες εφαρμογές.

Τα FPGAs υλοποιήθηκαν και παρουσιάστηκαν στην αγορά σαν τον κυριότερο ανταγωνιστή των CPLDs, με δυνατότητες πολύ μεγαλύτερες από αυτές των προκατόχων τους τόσο σε ταχύτητα όσο και σε μέγεθος ίσως και μικρότερο από αυτό των CPLDs. Ποιο συγκεκριμένα μετά την εισαγωγή κυκλωμάτων πολλαπλασιαστών στην αρχιτεκτονική των FPGA στα τέλη της δεκαετίας του '90, εφαρμογές όπου παραδοσιακά κυριαρχούσαν τα DSPs άρχισαν να υλοποιούνται από FPGAs.

Συγκεκριμένα τα FPGAs βρίσκουν εφαρμογές στην επεξεργασία αλγορίθμων οι οποίοι κάνουν χρήση της παράλληλης επεξεργασίας που προσφέρει η αρχιτεκτονική τους. Ένα τέτοιο πεδίο εφαρμογής είναι η αποκρυπτογράφηση, η βίαιη προσπάθεια εύρεσης του κλειδιού ή του αλγορίθμου που υλοποιεί την κρυπτογράφηση.

Ο εγγενής παραλληλισμός των λογικών πόρων σε ένα FPGA επιτρέπει σημαντική υπολογιστική απόδοση ακόμα και σε χαμηλές τιμές MHz. Η ευελιξία του FPGA επιτρέπει την επίτευξη ακόμα υψηλότερων επιδόσεων, όμως η ανάπτυξη των FPGAs σε υπολογιστές υψηλών επιδόσεων περιορίζεται σήμερα από την πολυπλοκότητά τους, σε σύγκριση με τα συμβατικά λογισμικά και τη συνεχώς αυξανόμενη βελτίωση των εργαλείων σχεδιασμού.

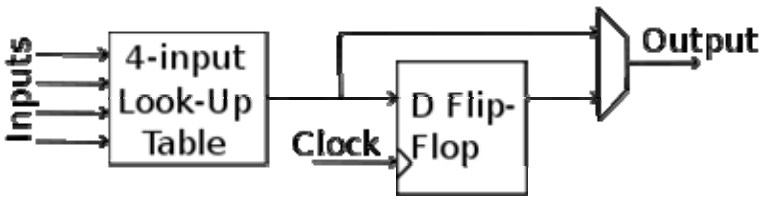
Αν και το κόστος δημιουργίας των FPGAs είναι αρκετά μεγάλο παρόλα αυτά δαπανώνται αρκετοί πόροι από τις εταιρείες για την ανάπτυξή τους. Ο λόγος είναι η υψηλή δυναμική των επιδόσεών τους σε πολλές εφαρμογές.

2.8 Αρχιτεκτονική των FPGA

Η πιο κοινή αρχιτεκτονική FPGA αποτελείται από ένα πλέγμα ρύθμισης μπλοκ λογικής (CLBs), I / O pads, και διαύλων δρομολόγησης. Σε γενικές γραμμές, όλα τα κανάλια δρομολόγησης έχουν το ίδιο πλάτος (αριθμού αγωγών). Πολλαπλά I / O pads μπορούν να ενταχθούν στο ύψος μιας γραμμής ή το πλάτος μιας στήλης του πίνακα.

Ένα εφαρμοσμένο κύκλωμα πρέπει να αντιστοιχιστεί σε ένα FPGA με επαρκείς πόρους. Ενώ ο αριθμός των CLBs και I/Os που απαιτείται είναι εύκολο να προσδιοριστεί από το σχεδιασμό, ο αριθμός των κομματιών που απαιτούνται μπορεί να ποικίλλει σημαντικά ακόμη και μεταξύ των σχεδίων με την ίδια λογική. (Για παράδειγμα, ένα crossbar switch απαιτεί πολύ περισσότερη δρομολόγηση από μια συστοιχία με τον ίδιο αριθμό πυλών.) Δεδομένου ότι αχρησιμοποίητα κομμάτια δρομολόγησης αυξάνουν το κόστος (και μείωση της απόδοσης) του τμήματος χωρίς να προσφέρουν κανένα όφελος, κατασκευαστές των FPGAs προσπαθούν να παρέχουν τόσα κομμάτια έτσι ώστε τα περισσότερα σχέδια που θα ενταχθούν θα ικανοποιούν τους όρους των LUTs και τα IOs θα μπορούν να δρομολογηθούν. Αυτό καθορίζεται από τις εκτιμήσεις όπως αυτές που προέρχονται από Rent's rules ή από πειράματα με υπάρχοντα σχέδια.

Ένα κλασικό μπλοκ λογικής FPGA αποτελείται από ένα 4-εισροών πίνακα αναζήτησης (LUT), και ένα flip-flop , όπως φαίνεται παρακάτω. Κατά τα τελευταία έτη, οι κατασκευαστές έχουν αρχίσει να κινούνται προς αρχιτεκτονικές 6-εισροών LUTs στα μέρη όπου απαιτούνται υψηλές επιδόσεις , υποστηρίζοντας την αυξανόμενη απόδοση.

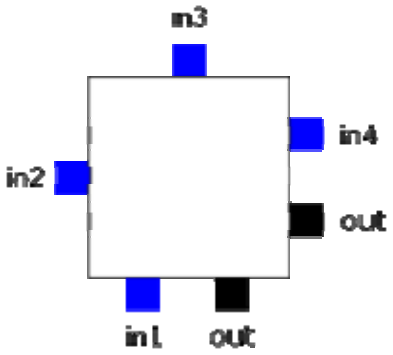


Typical logic block

Υπάρχει μόνο μία έξοδος, η οποία μπορεί να είναι εγγεγραμμένη ή μη εγγεγραμμένη έξοδος LUT. Το μπλοκ λογικής έχει τέσσερις εισόδους για την LUT και ένα clock εισόδου. Από τότε που τα clock σήματα (υψηλού fanout σήματα),δρομολογούνται κατά κανόνα

μέσω ειδικών δικτύων δρομολόγησης σε εμπορικά FPGAs, αυτά (όπως και άλλα σήματα) έχουν χωριστή διαχείριση.

Για τη συγκεκριμένη αρχιτεκτονική, οι θέσεις των pins του μπλοκ διαγράμματος λογικής του FPGA φαίνονται παρακάτω.



Logic Block Pin Locations

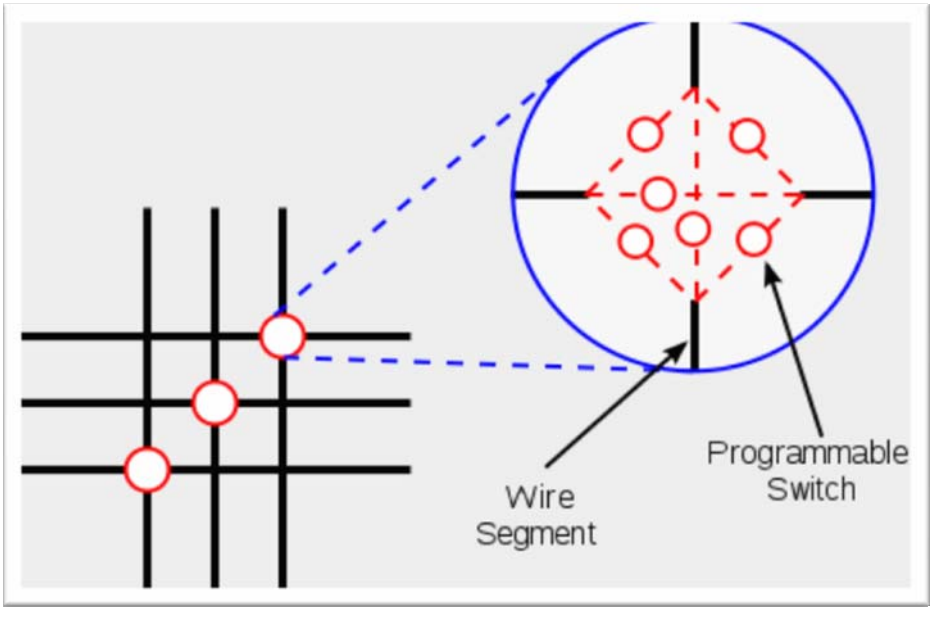
Κάθε είσοδος είναι προσβάσιμη από τη μια πλευρά του μπλοκ λογικής, ενώ το pin της εξόδου μπορεί να συνδεθεί με τα καλώδια δρομολόγησης, τόσο το κανάλι στα δεξιά και το κανάλι κάτω από το μπλοκ λογικής.

Κάθε pin εξόδου του μπλοκ μπορεί να συνδεθεί με οποιοδήποτε από τα τμήματα καλωδίωσης στα κανάλια που το πλαισιώνουν.

Ομοίως, ένα I / O pad μπορεί να συνδεθεί με οποιοδήποτε από τα τμήματα καλωδίωσης στο κανάλι δίπλα σε αυτό. Για παράδειγμα, ένα I / O pad στην κορυφή του chip μπορεί να συνδεθεί σε κάποιο από τα καλώδια W (όπου W είναι το πλάτος καναλιού) στο οριζόντιο κανάλι αμέσως κάτω από αυτό.

Σε γενικές γραμμές, η δρομολόγηση του FPGA είναι κατακερματισμένη. Δηλαδή, κάθε καλωδιωμένο τμήμα εκτείνεται για ένα μόνο τετράγωνο λογικής πριν τερματίσει σε έναν διακόπτη. Ενεργοποιώντας ορισμένους από τους προγραμματιζόμενους διακόπτες, μεγαλύτερες διαδρομές μπορούν να κατασκευαστούν. Για υψηλότερες ταχύτητες διασύνδεσης, κάποιες αρχιτεκτονικές FPGA χρησιμοποιούν πλέον μεγαλύτερες γραμμές δρομολόγηση ς που επεκτείνονται σε πολλαπλά μπλοκ λογικής.

Κάθε φορά που ένα κάθετο και ένα οριζόντιο κανάλι, τέμνονται, υπάρχει ένα κουτί διακοπτών. Σε αυτήν την αρχιτεκτονική, όταν ένα καλώδιο μπαίνει σε ένα κουτί διακόπτη, υπάρχουν τρεις προγραμματιζόμενοι διακόπτες που του επιτρέπουν να συνδεθεί με άλλα τρία καλώδια σε παρακείμενα τμήματα του καναλιού.. Το μοτίβο ή η τοπολογία των



διακοπτών που χρησιμοποιούνται σε αυτήν την αρχιτεκτονική είναι βασισμένη στον τομέα της τοπολογίας των διακοπτών. Σε αυτή την τοπολογία του κουτιού διακοπτών, ένα καλώδιο του πρώτου κομματιού συνδέεται μόνο με καλώδια του ίδιου κομματιού σε παρακείμενα τμήματα, κανάλια και σύρματα, ενώ καλώδια του δεύτερου κομματιού μπορούν να συνδεθούν μόνο με καλώδια του ίδιου κομματιού και ούτω καθεξής. Το σχήμα που ακολουθεί επιδεικνύει τις συνδέσεις μέσα σε ένα κουτί διακοπτών (switch box).

Σύγχρονες οικογένειες FPGA επεκτείνουν τις παραπάνω δυνατότητες ώστε να επιτευχθεί υψηλότερο επίπεδο λειτουργικότητας. Έχοντας ενσωματώσει αυτές τις κοινές λειτουργίες σε πυρίτιο μειώνεται η απαιτούμενη έκταση και δίνει στις λειτουργίες αυτές αυξημένη ταχύτητα σε σύγκριση με τα πρωτότυπα. Παραδείγματα τέτοιων λειτουργιών είναι πολλαπλασιαστές, γενικά μπλοκ DSP, ενσωματωμένοι επεξεργαστές, μνήμες υψηλής λογικής IO ταχύτητας και ενσωματωμένες μνήμες.

Τα FPGAs επίσης χρησιμοποιούνται ευρέως για συστήματα που περιλαμβάνουν πυρίτιο και firmware ανάπτυξη. Αυτό επιτρέπει σε εταιρείες κατασκευής chip την επικύρωση του σχεδιασμού τους, πριν το τσιπ παραχθεί στο εργοστάσιο μειώνοντας έτσι το χρόνο διάθεσης στην αγορά.

2.9 FPGA σχεδιασμός και προγραμματισμός

Για να προγραμματίσουμε το FPGA, κάνουμε χρήση μιας γλώσσας περιγραφής υλικού (HDL) ή σχηματικό. Το έντυπο της HDL είναι πιο κατάλληλο για την εργασία με μεγάλες δομές, διότι είναι δυνατό να προσδιοριστούν ακριβώς αριθμητικά, αντί να κατασκευάσει κάθε κομμάτι με το χέρι. Ωστόσο, η σχηματική είσοδος μπορεί να επιτρέψει την ευκολότερη απεικόνιση ενός σχεδίου ή υποδείγματος.

Στη συνέχεια, χρησιμοποιώντας ένα ηλεκτρονικό αυτόματο εργαλείο σχεδιασμού, μια εικονική διασύνδεση δημιουργείται. Η netlist μπορεί στη συνέχεια να τοποθετηθεί στην πραγματική αρχιτεκτονική FPGA χρησιμοποιώντας μια διαδικασία που ονομάζεται place-and-route , που συνήθως εκτελούνται από τις διαδρομές και λογισμικό της εταιρείας FPGA. Ο χρήστης θα επικυρώσει το σχέδιο, τον τόπο και τα αποτελέσματα διαδρομής μέσω της ανάλυσης χρονισμού, προσομοίωση, και άλλες επαληθεύσεις. Μόλις η διαδικασία σχεδιασμού ολοκληρωθεί, το δυαδικό αρχείο δημιουργείται (χρησιμοποιώντας ιδιόκτητο λογισμικό της εταιρείας του FPGA) και χρησιμοποιείται για να ρυθμίσουμε το FPGA.

Πηγαίνοντας από το σχηματικό / HDL αρχείο προέλευσης σε πραγματική διαμόρφωση: Τα πηγαία αρχεία τροφοδοτούνται σε μια σουίτα λογισμικού από το FPGA / CPLD που μέσα από διάφορα στάδια, θα παράγει ένα αρχείο. Αυτό το αρχείο στη συνέχεια μεταφέρεται στο FPGA / CPLD μέσω μιας σειριακής θύρας (JTAG) ή σε εξωτερική μνήμη της συσκευής όπως ένα EEPROM .

Οι πιο συχνές HDLs είναι η VHDL και η Verilog, αν και σε μια προσπάθεια να μειωθεί η πολυπλοκότητα του σχεδιασμού σε HDLs, οι οποίες έχουν σχέση με το αντίστοιχο των γλωσσών συναρμολόγησης , έγιναν κινήσεις για να αυξηθεί το επίπεδο αφαίρεσης μέσω της εισαγωγής εναλλακτικών γλωσσών .

Για να απλοποιηθεί ο σχεδιασμός των πολύπλοκων συστημάτων σε FPGAs, υπάρχουν βιβλιοθήκες προκαθορισμένων λειτουργιών και κυκλώματα που έχουν δοκιμαστεί και βελτιστοποιηθεί ώστε να επιταχυνθεί η διαδικασία σχεδιασμού. Αυτά τα προκαθορισμένα κυκλώματα ονομάζονται κοινώς IP πυρήνες , και είναι διαθέσιμα από FPGA προμηθευτές και τρίτους προμηθευτές IP (σπάνια ελεύθερο, και τυπικά, που διατίθεται βάσει αποκλειστικής άδειας). Άλλα προκαθορισμένα κυκλώματα είναι διαθέσιμα από τις αναπτυξιακές κοινότητες, όπως η OpenCores (τυπικά, που διατίθεται βάσει ελεύθερου και ανοικτού κώδικα αδειών, όπως η GPL , BSD ή παρόμοιες άδειες), και από άλλες πηγές.

Σε ένα τυπικό σχέδιο ροής, η ανάπτυξη εφαρμογών για FPGA θα προσομοιώνουν το σχεδιασμό σε πολλαπλά στάδια σε όλη τη διαδικασία σχεδιασμού. Αρχικά η RTL περιγραφή σε VHDL ή Verilog προσομοιώνεται με τη δημιουργία δοκιμών για την προσομοίωση του συστήματος και παρατηρούνται τα αποτελέσματα. Στη συνέχεια, μετά τη σύνθεση του πυρήνα έχει χαρτογραφηθεί το σχέδιο στη netlist. Η netlist μεταφράζεται σε μια επίπεδη πύλη όπου η προσομοίωση επαναλαμβάνεται για να επιβεβαιωθεί η σύνθεση προχωρώντας χωρίς λάθη. Τέλος, η σχεδίαση που αναφέρεται στο FPGA δείχνει σε ποιο σημείο υπάρχουν καθυστερήσεις διάδοσης και η προσομοίωση τρέχει και πάλι με τις τιμές αυτές επάνω στο netlist.

Παράδειγμα VHDL

```
library ieee;
use ieee.std_logic_1164.all;
-----------------------------------------------
entity OR_GATE is
        port (a,b:      in std_logic;
                c:      out std_logic);
end OR_GATE;
-----------------------------------------------
architecture algorithm of OR_GATE is
        begin
-----------------------------------------------
        process(a,b)
                begin
                        if a = '0' and b = '0' then
                                c <= '0';
                        elsif (a = '1') or (b= '1') then
                                c <= '1';
                        else c<='X';
```

```
                    end if;
        end process;
-----------------------------------------------
end algorithm;
```

(Πρόκειται για πρόγραμμα στο οποίο διευκρινίζεται η οντότητα μιας πύλης OR)

Παράδειγμα Verilog

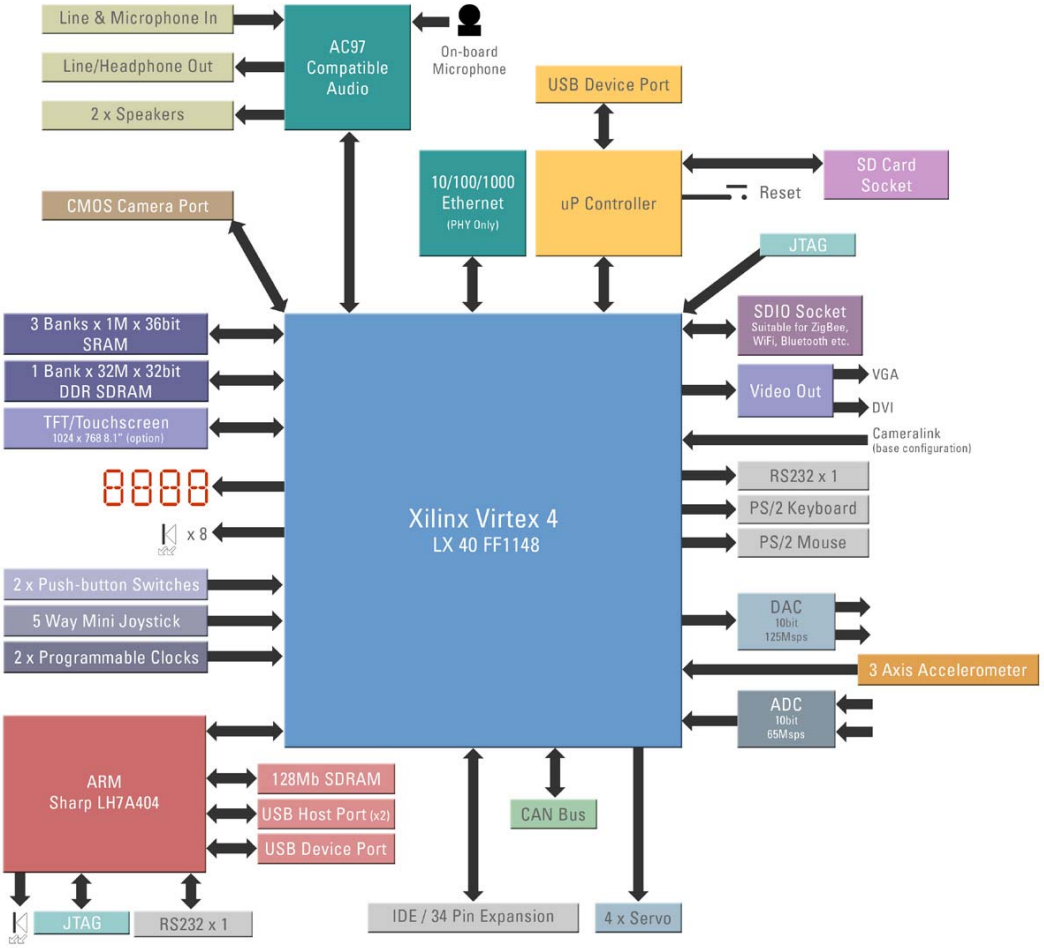
```
module toplevel(clock,reset);
 input clock;
 input reset;

 reg flop1;
 reg flop2;

 always @ (posedge reset or posedge clock)
 if (reset)
   begin
     flop1 <= 0;
     flop2 <= 1;
   end
 else
   begin
     flop1 <= flop2;
     flop2 <= flop1;
   end
endmodule
```

(Πρόκειται για ένα πρόγραμμα που χρησιμοποιεί 2 flip=flops)

3. ΠΑΡΟΥΣΙΑΣΗ ΤΩΝ ΠΛΑΚΕΤΩΝ RC10 ΚΑΙ RC240

3.1 Βασικά χαρακτηριστικά της RC240

Το πακέτο RC240 παρέχει ένα περιβάλλον επιφάνειας εργασίας για τους σχεδιαστές της προηγμένης απεικόνισης και συστήματα επεξεργασίας σημάτων με τη χρήση συνδυασμού της προγραμματιζόμενης λογικής και ARM μικροεπεξεργαστή για εφαρμογές όπως η ψηφιακή ψυχαγωγία και επικοινωνιών, στο αυτοκίνητο , η ρομποτική και μηχανικής όρασης.



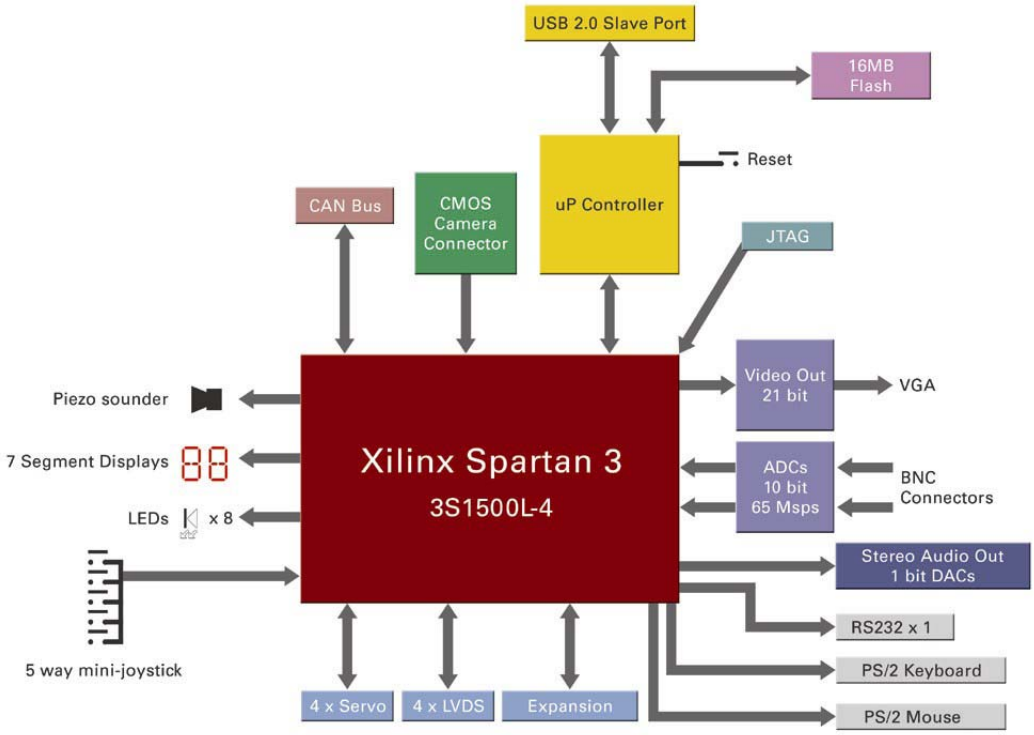
Το RC240 είναι ειδικά σχεδιασμένο για την κατανόηση και την αξιολόγηση των βίντεο υψηλής ταχύτητας και εφαρμογές δεδομένων χρησιμοποιώντας ένα μεγάλο υψηλής απόδοσης FPGA

Το RC240 περιλαμβάνει 4M Gate Xilinx virtex-4 FPGA LX40 με άμεση πρόσβαση σε 12M bytes της ZBT SRAM και 128M bytes της DDR SDRAM. Η Sharp LH7A404 είναι χαρακτηρίζει ένα ARM922T CPU, συνδεδεμένο με το FPGA μέσω του EBI. Η ARM είναι εξοπλισμένη με 128MB του SDRAM για ιδιωτική χρήση και USB υποδοχής και θύρες συσκευών. Οι βίντεο δυνατότητες παρέχονται από την είσοδο βίντεο CameraLink και προαιρετικά επί του CMOS κάμερα. Και οι δύο ψηφιακές και αναλογικές έξοδοι βίντεο, ικανών pixel συχνοτήτων έως 165MHz παρέχονται μέσω DVI-I σύνδεσης. Μια προαιρετική 1024x768 8,1 ιντσών TFT / οθόνη αφής είναι διαθέσιμη. Οι επικοινωνίες παρέχονται από μία Gigabit Ethernet PHY και 10/100/1000baseT πρίζα προσφέροντας δυνατότητα για τα δεδομένα συνεχούς ροής, τα στοιχεία της διανομής και της υψηλής απόδοσης δικτυακές εφαρμογές. Το RC240 προσφέρει επίσης μια σύνδεση USB 2.0 για υψηλής ταχύτητας μεταφορά αρχείων και υποδοχής FPGA για εφαρμογή επικοινωνιών. Ο σκελετός του RC240 είναι εξοπλισμένο με RS232 που συνδέονται άμεσα με τις δύο FPGA και CPU για να βοηθήσει με την ανάπτυξη. Για τον χρήστη προσφέρονται, οκτώ LED , ένα τετραψήφιο display, δύο κουμπιά, ένα ψηφιακό χειριστήριο, μια προαιρετική οθόνη αφής και PS2 ποντίκι και το πληκτρολόγιο.

Διαμόρφωση αρχείων και δεδομένα του χρήστη μπορούν να αποθηκευτούν μαζί σε ένα FAT διαμορφωμένης κάρτας SD, προσβάσιμη μέσω του ειδικού μικροελεγκτή,

δίνοντας στο FPGA τη δυνατότητα να επαναρυθμιστεί. Μια κάρτα SD 1GB προγραμματισμένη εκ των προτέρων με πολλά αρχεία επίδειξης παρέχεται με το RC240. Για να καταστεί δυνατή η ανάπτυξη καινοτόμων εφαρμογών του RC240, διαθέτει επίσης ένα επιταχυνσιόμετρο άξονα, dual channel 65Msps AD μετατροπέα εισροών και 125Msps dual channel Δ Α μετατροπέα εξόδων, CAN bus, 4 εξόδους ελέγχου servo control, μια ATA συμβατή με την επέκταση των ports και μια υποδοχή κάρτας SDIO για να φιλοξενήσει και άλλες I / O απαιτήσεις, όπως WiFi ή Bluetooth.

3.2 Βασικά χαρακτηριστικά της RC10



Στις 17/5/2005,η Celoxica Ltd,ο μεγαλύτερος πάροχος της C-based electronic system level (ESL), του σχεδιασμού και σύνθεσης λύσεων, ανακοίνωσε τη διαθεσιμότητα του RC10, ένα χαμηλού κόστους, υψηλής πυκνότητας, FPGA βασισμένο στο ενσωματωμένο σύστημα διδασκαλίας και της πλατφόρμας αξιολόγησης. Δημιουργήθηκε για τους σχεδιαστές, τους εκπαιδευτικούς και τους μαθητές. Η RC10 αντιπροσωπεύει μια σημαντική καινοτομία στα FPGA λόγω του κόστους της και της απόδοσης.
Το RC10 είναι εξοπλισμένο με 1.5 εκατομμύρια – gate Spartan 3 FPGA και είναι κατασκευασμένο με μια σειρά από κατανοητές βιβλιοθήκες υποστήριξης που προορίζεται για χρήση της Celoxica suite of ESL, εργαλείων σχεδιασμού, συμπεριλαμβανομένης της DK Design Suite και PixelStreams εικόνας και επεξεργασίας βίντεο-βιβλιοθήκης. Το σύνολο των δυνατοτήτων της RC10 το καθιστά κατάλληλο για προτυποποίηση και ανάπτυξη των διαφόρων εφαρμογών, όπως στη ρομποτική, την κρυπτογράφηση, στα αυτοκίνητα διάγνωσης, ήχου, βίντεο και επεξεργασία εικόνας.

Το RC10 είναι εφοδιασμένο με ένα ισχυρό σύνολο των I / O λειτουργιών, 2 υψηλής ταχύτητας κανάλια ADC, έξοδο VGA video, 1-bit DAC έξοδο ήχου, CAN bus και σύνδεση

με κάμερα CMOS. Ένα υψηλής ταχύτητας USB 2.0 επιτρέπει την επικοινωνία υψηλού ρυθμού δεδομένων μεταξύ των προγραμμάτων υποδοχής Η/Υ και FPGA εφαρμογές.

Το RC10 είναι επίσης ιδανικό για τη γρήγορη εκμάθηση των χρηστών που σχετίζονται με C-based ενσωματωμένων συστημάτων και σύνθεσης. Το πακέτο περιλαμβάνει παραδείγματα σχεδιασμού και tutorials-διδασκαλίες που δείχνει την ένταξη MicroBlaze soft-core επεξεργαστών και PicoBlaze ενσωματωμένων μικροελεγκτών. Το σύστημα APIs που παρέχεται με το RC10 επιτρέπει τον HW / SW συν-σχεδιασμό και την αρχιτεκτονική διερεύνηση των τμημάτων του συστήματος.

Το RC10 με βάση το C-based σχεδιασμό προσελκύει νέους χρήστες λόγω της ευκολίας χρήσης, του πολύ χαμηλού κόστους καθώς περιλαμβάνει μια προηγμένη σειρά χαρακτηριστικών γνωρισμάτων extending beyond raw υλικού και την υποστήριξη ενός πολύ ευρύτερου φάσματος εφαρμογών."

Εκτός από την Xilinx Spartan 3 FPGA και I / O, το RC10 περιλαμβάνει 8 LEDs, 2 οθόνες επτά-τομέων, PS / 2 πληκτρολόγιο και ποντίκι και ένα 5-way mini joystick. Ένα βοηθητικό πρόγραμμα που παρέχεται από τη Celoxica χρησιμοποιεί τη σύνδεση USB 2.0 για προγράμματα αρχείων στο FPGA και η θύρα USB χρησιμοποιείται επίσης από τον κεντρικό υπολογιστή για ευκολία στη χρήση και ευελιξία στην τάξη. Οι δυνατότητες επέκτασης της πλακέτας περιλαμβάνουν 50-way επέκταση για παροχή στοιχείων, 33 εισόδους / εξόδους, 2 clock pins, καθώς και +12 V, +5 V και 3,3 V τροφοδοτικά, ένα 4-way LVDS connector επίσης κατάλληλο για τη σύνδεση μιας TFT οθόνης, καθώς και συνδέσεις για 4 standard servos.

3.3 Οικογένειες FPGA της εταιρίας XILINX

Οι βασικές οικογένειες των κυκλωμάτων FPGA της εταιρίας Xilinx είναι οι εξής:

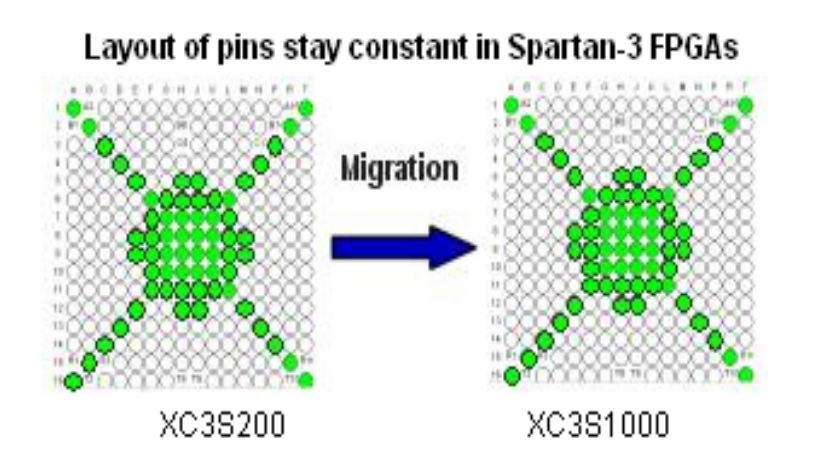
- Spartan II/ IIE
- Spartan 3/ 3L/ 3E
- Spartan XL
- Virtex II/ II Pro/ II ProX
- Virtex E/ EM
- Virtex 4

Στο παρόν κεφάλαιο ακολουθεί αναλυτική περιγραφή των χαρακτηριστικών της σειράς Spartan 3 και Virtex 4,τις οποίες οικογένειες χρησιμοποιούμε στις πλακέτες τις οποίες περιγράψαμε παραπάνω και που θα χρησιμοποιήσουμε παρακάτω για την παρουσίαση των προγραμμάτων μας.

3.3.1 Spartan 3

Η οικογένεια αυτή αποτελείται από 8 συσκευές που αποτελούνται από 50K έως 5M πύλες συστήματος και έως 1,8 Mbits μνήμες τύπου Block RAM που μπορούν να

χρησιμοποιηθούν είτε ως buffer είτε ως cache memory.Τα FPGA αυτά χρησιμοποιούν και distributed μνήμη και 16 bit Shift Register Logic (SRL 16). Επίσης περιλαμβάνει από 124 έως 784 I/O pins, 8 ανεξάρτητες I/ O banks που υποστηρίζουν 24 διαφορετικά I/ O standards και ενσωματωμένο ψηφιακό ρολόι (DCM) που μειώνει την ανάγκη για χρησιμοποίηση εξωτερικών ρολογιών. Τα FPGA της οικογένειας Spartan 3 είναι ιδανικά για χαμηλότερο κόστος ανά I/O και επιτρέπουν την εύκολη αλλαγή από μια συσκευή σε μια άλλη μεγαλύτερης ή μικρότερης πυκνότητας χωρίς την ανάγκη να γίνει Relayout της πλακέτας όπως φαίνεται χαρακτηριστικά στο σχήμα για δύο συσκευές της οικογένειας (XC3S200 και XC3S1000):



Σχήμα 5.3 Σύγκριση του layout για τα FPGA Spartan-3

Αυτό γίνεται εξαιτίας του γεγονότος ότι η σχετική θέση της VCC και της GND παραμένουν σταθερές για όλες τις συσκευές της οικογένειας. Η σημαντικότερη όμως διαφορά της οικογένειας αυτής σε σχέση με τις Spartan II και IIE είναι ότι χρησιμοποιεί επιπλέον πολλαπλασιαστές 18x 18 έτσι ώστε να υποστηρίζει DSP κυκλώματα υψηλών επιδόσεων. Οι συχνότητες λειτουργίας αυτής της οικογένειας φθάνουν τα 326 MHz. Στον παρακάτω πίνακα συνοψίζονται τα βασικά χαρακτηριστικά των FPGA της οικογένειας Spartan-3:

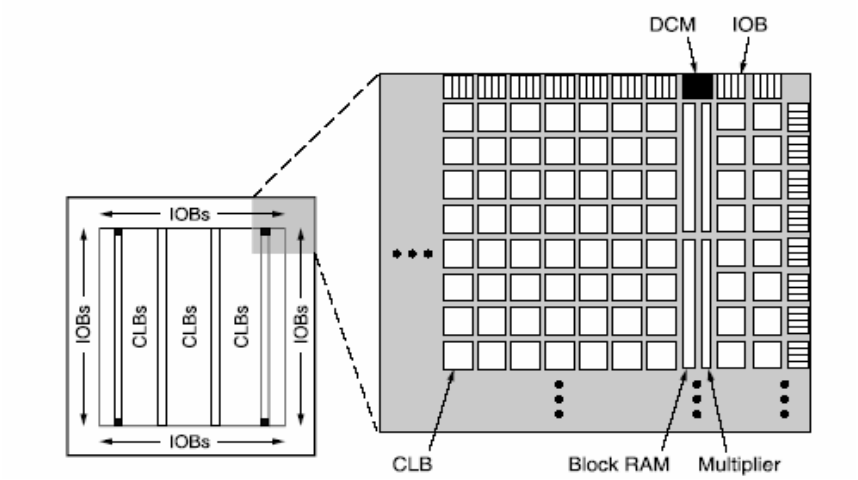
Spartan-3 FPGA Family								
Spartan-3	XC3S50	XC3S200	XC3S400	XC3S1000	XC3S1500	XC3S2000	XC3S4000	XC3S5000
Spartan-3L	—	—	—	XC3S1000L	XC3S1500L	—	XC3S4000L	—
Spartan-3 EasyPath	—	—	—	—	XCE3S1500	XCE3S2000	XCE3S4000	XCE3S5000
System Gates	50K	200K	400K	1000K	1500K	2000K	4000K	5000K
Logic Cells	1,728	4,320	8,064	17,280	29,952	46,080	62,208	74,880
Block RAM Bits	72K	216K	288K	432K	576K	720K	1,728K	1,872K
Distributed RAM Bits	12K	30K	56K	120K	208K	320K	432K	520K
DCMs	2	4	4	4	4	4	4	4
Multipliers	4	12	16	24	32	40	96	104
I/O Standards	24	24	24	24	24	24	24	24
Max Single Ended I/O**	124	173	264	391	487	565	712	784
Package and I/O Offerings								
	XC3S50	XC3S200	XC3S400	XC3S1000	XC3S1500	XC3S2000	XC3S4000	XC3S5000
VQ100 14 x 14 mm	63	63						
TQ144 20 x 20 mm	97	97	97					
PQ208 28 x 28 mm	124	141	141					
FT256 17 x 17 mm		173	173	173*				
FG320 23 x 23 mm			221	221*	221*			
FG456 23 x 23 mm			264	333*	333*			
FG676 27 x 27 mm				391	487*	489		
FG900 31 x 31 mm						565	633*	633
FG1156 35 x 35 mm							712	784

Πίνακας 5.3 Χαρακτηριστικά της οικογένειας Spartan 3

Η αρχιτεκτονική της οικογένειας των FPGA αυτών αποτελείται από 5 θεμελιώδη προγραμματιζόμενα λειτουργικά στοιχεία:

- Τα Configural Logic Blocks (CLBs) που περιέχουν Look-Up Tables ώστε να εφαρμοστούν σε λογικά και αποθηκευτικά στοιχεία τα οποία θα χρησιμοποιηθούν σαν flip-flops ή ως latches.Τα CLBs μπορούν να προγραμματιστούν ώστε να εκτελέσουν μια μεγάλη γκάμα από λογικές λειτουργίες όπως και για την αποθήκευση δεδομένων.
- Τα Input/ Output Blocks (IOBs) ελέγχουν την μεταφορά των δεδομένων ανάμεσα στα I/O pins και στην εσωτερική λογική της συσκευής. Κάθε IOB υποστηρίζει την αμφίδρομη μεταφορά δεδομένων.
- Η Block RAM παρέχει αποθήκευση των δεδομένων σε μορφή block των 18-bit.
- Οι πολλαπλασιαστές δέχονται 2 δυαδικούς αριθμούς των 18-bit ως εισόδους και υπολογίζουν το γινόμενό τους.
- Τα blocks των Digital Clock Managers (DCMs) παρέχουν πλήρεις ψηφιακές λύσεις για καθυστέρηση, πολλαπλασιασμό, διαίρεση και φασική μετατόπιση των σημάτων των ρολογιών.

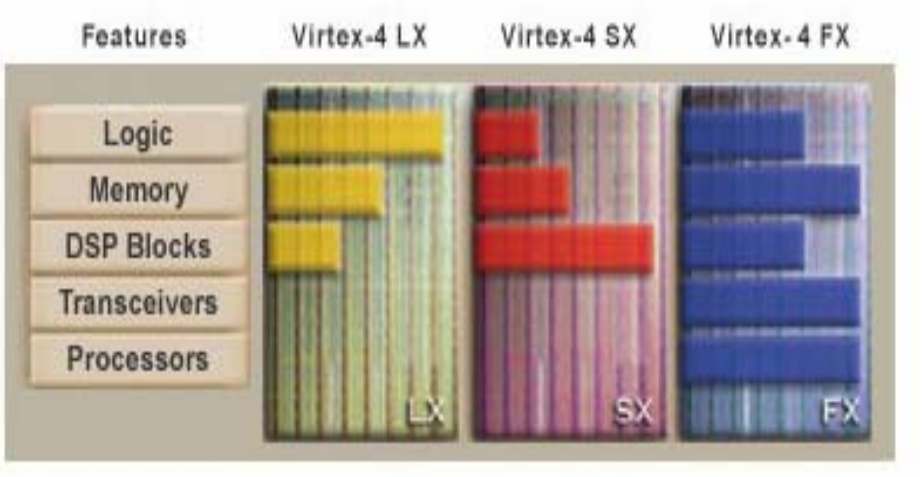
Αυτά τα στοιχεία οργανώνονται όπως φαίνεται στο σχήμα:



Σχήμα 5.4 Block διάγραμμα του Spartan 3

3.3.2 Virtex-4

Η οικογένεια Virtex-4 προσφέρει την πιο εξελιγμένη λογική, υψηλότερη επίδοση, υψηλότερη πυκνότητα και μεγαλύτερη χωρητικότητα μνήμης από κάθε οικογένεια FPGA. Με 200000 λογικές κυψέλες, 500MHz συχνότητα λειτουργίας και μη ανταγωνίσιμες αποτυχίες συστήματος οι συσκευές της Virtex-4 προσφέρουν διπλάσια συχνότητα και επίδοση αλλά και μισή κατανάλωση ισχύος σε σχέση με κάθε άλλη οικογένεια FPGA. Η οικογένεια Virtex-4 περιλαμβάνει 3 πλατφόρμες (LX, SX, FX) όπου η καθεμία έχει μια βέλτιστη ισορροπία από δυνατότητες και κόστος. Η ASMBL (**A**dvanced **S**ilicon **M**odular **BL**ock) αρχιτεκτονική, όπως φαίνεται και στο σχήμα, επιτρέπει στην Xilinx να δημιουργήσει πλατφόρμες FPGA με μια ποικιλία χαρακτηριστικών κατάλληλα για διαφορετικά πεδία εφαρμογής.



Σχήμα 5.10 Σύγκριση οικογένειας Virtex-4

4. Ανάλυση της γλώσσας Handel-C

4.1 Η Σύνταξη της Γλώσσας

Η Handel-C είναι μία πραγματικά καινοτόμος γλώσσα για την μετάφραση αλγορίθμων σε ηλεκτρονικό υλικό (hardware), για την εξερεύνηση αρχιτεκτονικής δομής και για τον ολοκληρωμένο σχεδιασμό συστημάτων υλικού/ λογισμικού. Έχοντας βασισθεί στην ISO/ANSI-C, έχει τις κατάλληλες επεκτάσεις που απαιτούνται για την ανάπτυξη κυκλωμάτων. Γι' αυτό τα προγράμματα που έχουν σχεδιασθεί για την Handel-C, είναι εν δυνάμει σειριακά. Αυτό περιλαμβάνει ευέλικτα εύρη δεδομένων, παράλληλη επεξεργασία και επικοινωνία ανάμεσα σε παράλληλα στοιχεία. Η γλώσσα έχει σχεδιαστεί γύρω από ένα απλό μοντέλο συγχρονισμού που την κάνει προσιτή για αρχιτέκτονες συστημάτων και σχεδιαστές λογισμικού.

Η Handel-C παρέχει ειδικές καινοτομίες, που κάνουν δυνατή την παράλληλη εκτέλεση εντολών. Επίσης, παρέχει τη δυνατότητα καθορισμού του εύρους μιας μεταβλητής δεδομένων.

Σειριακές Εντολές	Παράλληλες Εντολές

{
.....
a = 1;
b = 2;
.....
}
	par {
	a = 1;
	b = 2;
	}

Εκτέλεση δύο εντολών σειριακά η μία μετά την άλλη	Εκτέλεση και των δύο εντολών παράλληλα

Η Handel-C επίσης καθιστά δυνατή τη χρήση μεγεθών μεταβλητών που καθορίζονται από τον χρήστη. Για παράδειγμα:

Int *n* x; Καθορίζει μία μεταβλητή x τύπου int και μεγέθους *n* bits.

Όταν οι εντολές αξιολογούνται παράλληλα, η επικοινωνία μεταξύ των παράλληλων κλάδων γίνεται προβληματική εξαιτίας του συγχρονισμού. Η Handel-C περιλαμβάνει ένα σχεδιαστικό κατασκεύασμα, γνωστό ως κανάλι, για να ξεπερασθεί το συγκεκριμένο πρόβλημα.

4.2 Ομοιότητες με την ANSI-C

Η Handel-C έχει πολλές ομοιότητες με την C. Ταυτόχρονα, η Handel-C έχει πολλά χαρακτηριστικά που δεν υπάρχουν στην C και το αντίστροφο. Η Handel-C δεν υποστηρίζει τόσο μεγάλη ποικιλία τύπων δεδομένων όσο η C, αφού οι μοναδικοί τύποι δεδομένων που υποστηρίζονται από την Handel-C είναι οι ακέραιοι αριθμοί και οι χαρακτήρες. Όμως αντίθετα με την C, οι χρήστες μπορούν να καθορίσουν το εύρος των ακέραιων. Αυτό είναι δυνατό επειδή η υλοποίηση είναι απευθείας σε επίπεδο υλικού.

Η Handel-C επίσης δεν υποστηρίζει δείκτες (pointers), μια και αυτοί δεν μπορούν να υλοποιηθούν στο υλικό. Ο μεταγλωττιστής της Handel-C επιτρέπει τον καθορισμό μακροεντολών. Αυτό καθιστά δυνατή την επαναχρησιμοποίηση του ίδιου υλικού και έτσι αυξάνεται η αποδοτικότητα της χρήσης του υλικού.

4.3 Διαδικασία σχεδιασμού

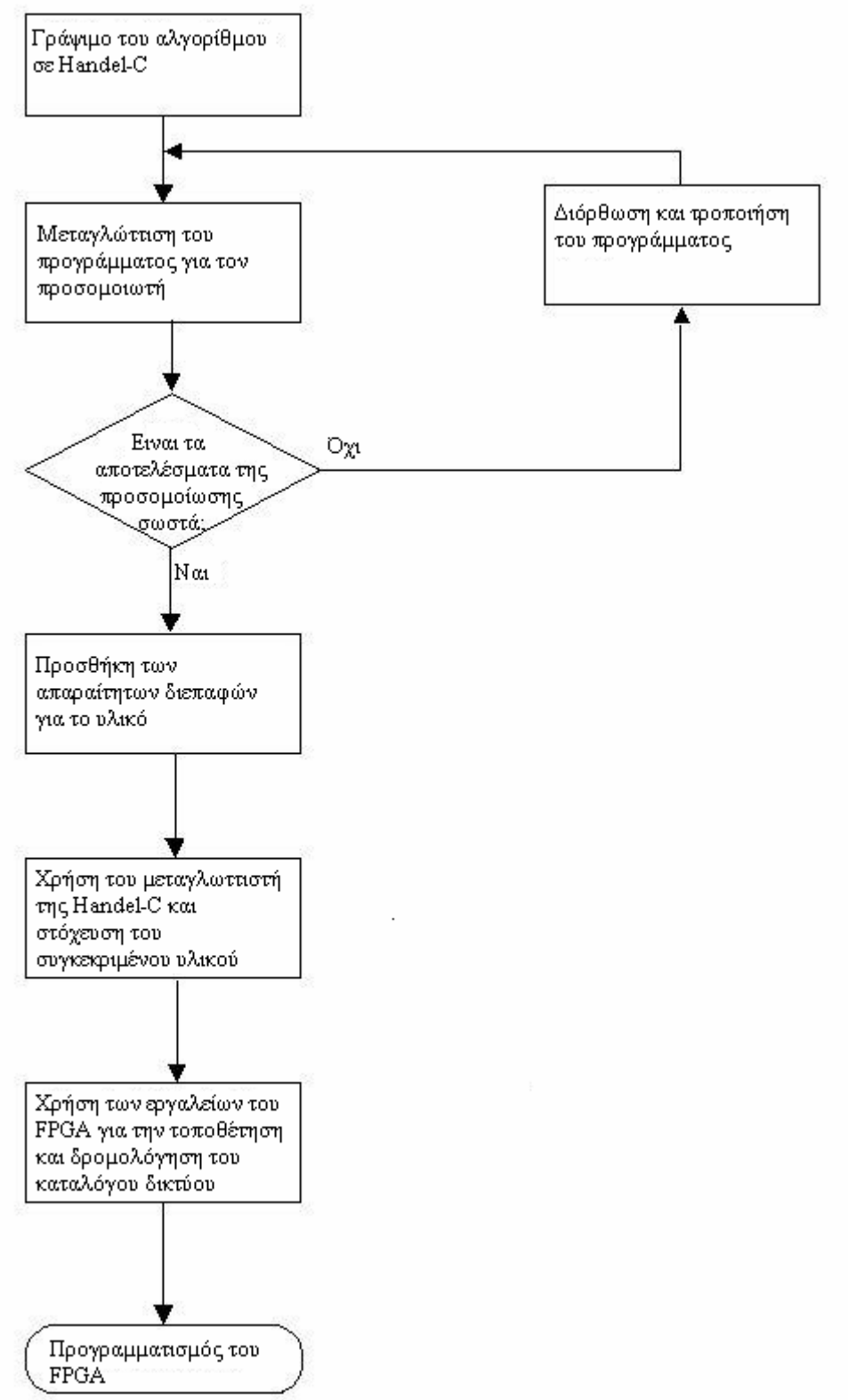
Η Handel-C παρέχει έναν προσομοιωτή για τον έλεγχο της υλοποίησης του προγράμματος πριν την πραγματική υλοποίησή του στο υλικό. Ο προσομοιωτής μπορεί να πραγματοποιεί βήμα βήμα κάθε κύκλο εκτέλεσης και να παρουσιάζει τις τιμές των μεταβλητών μετά από κάθε κύκλο.

Τύποι δεδομένων της Handel-C και της ANSI-C

Handel-C Only	ANSI-C	Both
	double	
	float	
Chan	enum	
Ram	register	Int
Rom	static	unsigned
chanin	extern	Char
chanout	struct	Long
undefined	volatile	Short
interface	void	
	const	
	union	

Όταν πλέον ο σχεδιαστής είναι ικανοποιημένος από το σχέδιό του, τότε μπορεί να μεταγλωττιστεί στο υλικό. Όταν γίνεται η μεταγλώττιση στο υλικό, ο σχεδιαστής μπορεί να στοχεύσει σε μια συγκεκριμένη πλατφόρμα υλικού. Ο μεταγλωττιστής της Handel-C στις μέρες μας παράγει καταλόγους δικτύου για συσκευές Xilinx και Altera.

Η Handel-C ακολουθεί την παρακάτω διαδικασία σχεδιασμού:



4.4 Σύγκριση μεταξύ Handel-C και VHDL

Η πρωτοτυποποίηση νέων ιδεών ή η κατασκευή ηλεκτρονικών συσκευών πρώτης γενιάς είναι χρονοβόρα και ακριβή, ενώ σε ορισμένες περιπτώσεις ενέχει και υψηλό ρίσκο. Οι περισσότεροι αλγόριθμοι γράφονται αρχικά σε C και στη συνέχεια μεταφράζονται σε VHDL ή Verilog, μια διαδικασία που παρουσιάζει ρίσκα και λάθη.

Η Handel-C αποφεύγει αυτό το πρόβλημα επειδή είναι μία γλώσσα που βασίζεται στη C και είναι σχεδιασμένη για να περιγράφει αλγόριθμους, οι οποίοι μεταγλωττίζονται διαδοχικά σε κάποιο υλικό. Αλλαγές στον κώδικα της Handel-C παράγουν προβλέψιμες αλλαγές στο υλικό. Στοχεύοντας απευθείας στα FPGAs, η Handel-C παρέχει μία γρήγορη διαδρομή για την πρωτοτυποποίηση και ανάπτυξη του υλικού ηλεκτρονικών προϊόντων πρώτης γενιάς. Η διαδικασία ανάπτυξης εκτελείται σε ένα απλό λογισμικό περιβάλλον, με τους βρόχους διόρθωσης/ τροποποίησης/ κατασκευής να μετρώνται σε λεπτά παρά σε εβδομάδες και μήνες. Αυτό παρέχει το ιδανικό εργαλείο για την αξιολόγηση της απόδοσης επιλογών ανταλλαγής (trade-off decisions) και τον έλεγχο της αξιοπιστίας των τελικών σχεδίων. Η Handel-C υποστηρίζει μία μεθοδολογία λογισμικού που βασίζεται στην επαναχρησιμοποίηση του σχεδίου. Οι λειτουργίες μπορούν να μεταγλωττιστούν σε βιβλιοθήκες και να χρησιμοποιηθούν και σε άλλες εργασίες, με μια απλή δήλωση που παρέχει τη διεπαφή σε άλλο κώδικα. Οι πυρήνες που έχουν γραφθεί σε Handel-C μπορούν να εξαχθούν ως «μαύρα κουτιά» EDIF ή VHDL για επαναχρησιμοποίηση του σχεδίου.

Χαρακτηριστικά	**Πλεονεκτήματα**
Λύσεις υψηλού επιπέδου γλώσσας	Επιτρέπει τη γρήγορη ανάπτυξη πολλών εκατομμυρίων σχεδίων πυλών FPGA και λύσεων συστημάτων σε ολοκληρωμένα κυκλώματα
Βασίζεται στην ISO/ANSI-C	Επιτρέπει στους σχεδιαστές εφαρμογών να «στέλνουν» έννοιες απευθείας στο υλικό, για γρήγορη πρωτοτυποποίηση και για ηλεκτρονικά προϊόντα πρώτης γενιάς
Καλά καθορισμένος συγχρονισμός	Γρήγορο εξωτερικό I/O Απλοποίηση της διαδικασίας επεξεργασίας της πληροφορίας
Σαφής παραλληλισμός	Εντολή par Ταυτόχρονη αξιολόγηση
Υποστηρίζει τη σύνθετη λειτουργικότητα της C, συμπεριλαμβάνοντας δομές, δείκτες και λειτουργίες	Εύκολη εκμάθηση για τους σχεδιαστές λογισμικού, επιτρέπει τις γρήγορες υλοποιήσεις πολύ σύνθετων αρθρωτών συστημάτων
Περιλαμβάνει εκτεταμένους χειριστές για τροποποίηση των ψηφίων και υψηλού επιπέδου μαθηματικές μακροεντολές (συμπεριλαμβανομένης και της κινητής υποδιαστολής)	Επιτρέπει τη γρήγορη μετάφραση των DSP αλγορίθμων σε αποτελεσματικό και αποδοτικό υλικό
Δεν χρειάζεται να σχεδιαστούν μηχανές κατάστασης (state machines), ο έλεγχος της ροής γίνεται από εντολές της C όπως η if, η case και η while	Απλοποίηση του σχεδιασμού πολύπλοκων σειριακών ελέγχων ροής
Απλές και συνεπείς συντακτικές επεκτάσεις για συγκεκριμένα χαρακτηριστικά του υλικού, όπως οι	Επιτρέπει την αποτελεσματική χρήση του διαθέσιμου υλικού χωρίς να απαιτείται δύσκολο και δύσχρηστο συντακτικό

RAMs/ ROMs, τα σήματα και οι εξωτερικές pin συνδέσεις	
Αντιμετωπίζει αυτόματα τα ρολόγια (clocks), τα clock enables και τις μεταφορές δεδομένων στα όρια των πεδίων των ρολογιών	Παραβλέπει μεγάλο μέρος της πολυπλοκότητας στον σχεδιασμό υλικού

Καθώς τα συστήματα αυξάνονται σε μέγεθος και πολυπλοκότητα, οι σχεδιαστές επωφελούνται από μία νέα γενιά εργαλείων που συμπληρώνουν αυτά που χρησιμοποιούν σήμερα. Αυτά τα νέα εργαλεία απλοποιούν τη διαδικασία της περιγραφής της λειτουργικότητας στο υλικό μέσω της εφαρμογής μιας υψηλού επιπέδου προσέγγισης στην EDA (Electronic Design Automation) που είναι εμπνευσμένη από τον τομέα του λογισμικού. Συγχωνεύοντας μεθοδολογίες υλικού και λογισμικού, αυτά τα εργαλεία εισάγουν στον σχεδιαστή υλικού τρεις πτυχές της ανάπτυξης λογισμικού: μία γλώσσα που βασίζεται στην C για την περιγραφή της λειτουργικότητας, ένα σύστημα σχεδιασμού με συμβολική διόρθωση, και βιβλιοθήκες προκαθορισμένων λειτουργιών που περιλαμβάνουν την πρόσβαση στα περιφερειακά και στους επεξεργαστές του υλικού μέσω κοινών APIs.

4.5 Μία προσέγγιση για μία γλώσσα προγραμματισμού που βασίζεται στη C

Οι μεθοδολογίες σχεδιασμού υλικού, συλλαμβάνουν από τη σχηματική αναπαράσταση ποια λειτουργικότητα υλικού αναπτύσσεται περιγράφοντας τη δομή του κυκλώματος. Για αυτόν τον σκοπό, τέτοιες προσεγγίσεις έχουν εστιάσει στη διατήρηση του χαμηλού επιπέδου ελέγχου του σχεδίου, αλλά υπάρχουν περιορισμοί εάν τέτοιες μέθοδοι χρησιμοποιούνται αποκλειστικά για να εξετάσουν μια μεγάλη περιοχή σχεδίου. Ένα από τα πρώτα προβλήματα προκύπτει επειδή αν και η περισσότερη λειτουργικότητα περιλαμβάνει και σειριακή αλλά και παράλληλη λογική, παρ' όλα αυτά οι HDLs έχουν εξελιχθεί από έναν αποκλειστικά παράλληλο κόσμο για την περιγραφή του υλικού παρά για την περιγραφή της επιθυμητής λειτουργίας. Αυτό που χρειάζεται είναι μία γλώσσα που αυξάνει το επίπεδο αφαίρεσης ικανοποιητικά για να δώσει στον σχεδιαστή την δυνατότητα να περιγράψει με τον συντομότερο δυνατό τρόπο την επιθυμητή λειτουργία παρά την υποκείμενη δομική της λεπτομέρεια.

Ενώ τα υποσύνολα επιπέδων μεταφοράς καταλόγων (RTL subsets) των HDLs, όπως η VHDL και η Verilog, παρέχουν μια λειτουργική ερμηνεία της περιγραφής υλικού για να επιτρέψουν την παραγωγή της δομής του υλικού στον χρόνο μεταγλώττισης, η παράλληλη φύση τους απαιτεί από τον σχεδιαστή να προσθέσει επιπλέον λογική για τη διαδοχική εκτέλεση, παραδείγματος χάριν μία FSM (Finite State Machine) που εκφράζεται ως εντολή περίπτωσης (case statement). Ο κώδικας κόβεται σε κομμάτια και τοποθετείται στην αντίστοιχη εντολή ανάλογα με τον κύκλο ρολογιού που πρόκειται να εκτελεστεί. Η σειρά της εκτέλεσης τότε ελέγχεται από υποκείμενες σε όρους ρυθμίσεις της κατάστασης καθενός από αυτά τα κομμάτια. Αυτός είναι ουσιαστικά προγραμματισμός GOTO. Με τις ανεπάρκειες του προγραμματισμού GOTO, κάτι που έχει εγκαταλειφθεί εδώ και δεκαετίες στον τομέα του λογισμικού, ο κώδικας γίνεται δύσκολα αναγνώσιμος και πιθανότατα επικίνδυνος, για παράδειγμα για τον σχηματισμό ατελείωτων βρόχων. Εισάγοντας στη διαδικασία σχεδιασμού υλικού μία γλώσσα που είναι όμοια με την ANSI-C, οι σχεδιαστές αποκτούν μία γλώσσα που είναι σειριακή εξ ορισμού με μια υψηλού επιπέδου ροή για τον προγραμματισμό της λειτουργικότητας. Όμως το υλικό είναι παράλληλο και αυτό πρέπει να ληφθεί υπόψη αν θέλουμε να πετύχει σε επίπεδο RTL μία μεθοδολογία σχεδιασμού υλικού που έχει επιρροές από τον τομέα του λογισμικού και βασίζεται εξ ολοκλήρου στην C. Υπάρχουν δύο εναλλακτικές υψηλού επιπέδου: μεταγλώττιση της συμπεριφοράς

(behavioral compilation) ή προσθήκη ενός απλού τρόπου για την έκφραση του παραλληλισμού. Οι μεταγλωττιστές συμπεριφοράς μοιάζουν να είναι ιδανικοί, αυτοματοποιώντας τις διαδικασίες μεταξύ εισαγωγής λογισμικού και απόδοσης υλικού. Όμως, οι σημερινοί μεταγλωττιστές προσφέρουν στον χρήστη μικρό έλεγχο στην ποιότητα της απόδοσης του υλικού.

4.6 Περιβάλλον σχεδιασμού

Οι σύγχρονες μεθοδολογίες σχεδιασμού υλικού είναι μεταφορά κάποιων πρώιμων εργαλείων όπως ράστερ (bread-boards) και λογικοί αναλυτές (logic analyzers) σε περιβάλλον ηλεκτρονικών υπολογιστών. Αν και ο σχεδιασμός υλικού έχει προοδεύσει με την χρήση RTL, οι μεθοδολογίες φανερώνουν την προέλευσή τους από τον παλαιό τρόπο κατασκευής. Υπάρχουν σημαντικά πλεονεκτήματα στον σχεδιασμό με γνώμονα την πλευρά του λογισμικού. Μία σημαντική μεθοδολογία παρουσιάζεται εδώ:

4.7 Λογισμικό Celoxica DK1 Design Suite

Το λογισμικό Celoxica DK1 Design Suite είναι μία μοναδική λύση που απευθύνεται άμεσα στο υλικό και δίνει τη δυνατότητα στους ειδικούς εφαρμογών να «στέλνουν» έννοιες απευθείας στο υλικό χωρίς να απαιτείται η δημιουργία, προσομοίωση ή σύνθεση των HDLs. Το λογισμικό Celoxica DK1 Design Suite εστιάζει στον σχεδιασμό, τον έλεγχο της αξιοπιστίας, τη συνεχή «τελειοποίηση» και την υλοποίηση πολύπλοκων αλγορίθμων στο υλικό. Περιλαμβάνει ενσωματωμένη είσοδο σχεδίου, προσομοίωση και σύνθεση που καθοδηγούνται άμεσα από τη Handel-C, μία γλώσσα προγραμματισμού που βασίζεται στην ISO/ANSI-C. Το αποτέλεσμα του μεταγλωττιστή είναι είτε ένας αρχιτεκτονικά βελτιστοποιημένος πίνακας διασυνδέσεων EDIF κατάλληλος για FPGAs, ή ένα RTL VHDL για ήδη υπάρχοντα λογισμικά εργαλεία.

Το λογισμικό DK1 Design Suite έχει την όψη και την αίσθηση ενός λογισμικού περιβάλλοντος. Η εφαρμογή διόρθωσης (debugger) περιέχει σε βάθος χαρακτηριστικά γνωρίσματα που φυσιολογικά βρίσκονται μόνο στην ανάπτυξη λογισμικού. Αυτά περιλαμβάνουν σημεία παύσης (breakpoints), βηματική εκτέλεση (single stepping), παρακολούθηση μεταβλητών και την ικανότητα να ακολουθεί παράλληλες σειρές εκτέλεσης. Ο σχεδιαστής του υλικού μπορεί χρησιμοποιώντας αυτήν την προσέγγιση να κινηθεί μέσα στο σχέδιο όπως ακριβώς σε ένα σύστημα σχεδιασμού λογισμικού. Τα ενσωματωμένα εργαλεία προσομοίωσης και επαλήθευσης διευκολύνουν τον σχεδιασμό με τις οδηγίες καθορισμένων προσομοιωτών, VHDL προσομοιωτών όπως ο ModelSim. Ένα σημαντικό πλεονέκτημα αυτού είναι ότι οι αποφάσεις διαχωρισμού υλικού/ λογισμικού μπορούν να αλλάξουν σε οποιοδήποτε στάδιο της διαδικασίας σχεδιασμού.

Η σύνθεση σε αυτό το σύστημα σχεδιασμού συσχετίζεται με τη σύνταξη λογισμικού στο γεγονός ότι είναι πολύ γρήγορη: οι σχεδιαστές λογισμικού είναι εξοικειωμένοι με τη σύνταξη αλλαγών στα σχέδια τους πολύ γρήγορα και τον έλεγχο των αποτελεσμάτων. Αυτό τους δίνει τη δυνατότητα να κάνουν πολλές αλλαγές, μεγάλες ή μικρότερες και να επανασυνθέτουν πολύ γρήγορα. Η ταχύτητα του λογισμικού σχεδιασμού της Celoxica φέρνει αυτό το πλεονέκτημα και στον σχεδιασμό υλικού.

4.8 Προκαθορισμένες Βιβλιοθήκες Υλικού

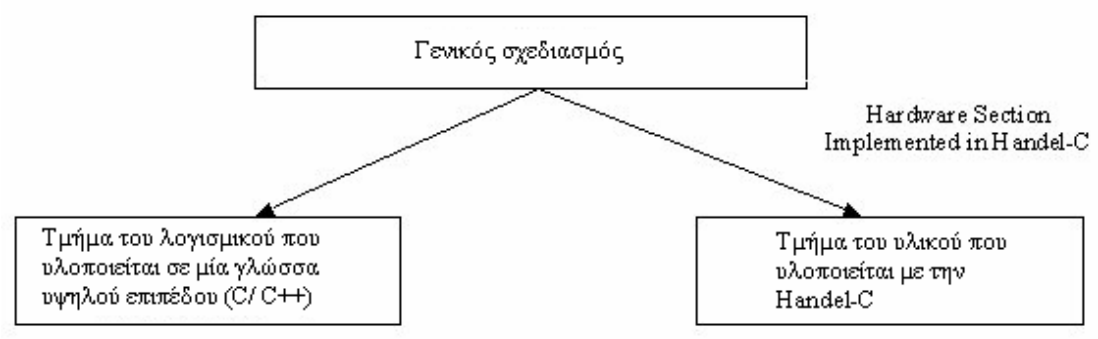
Οι προκαθορισμένες βιβλιοθήκες στον σχεδιασμό υλικού, όπως και οι συνήθεις βιβλιοθήκες της ANSI-C και άλλων λογισμικών περιβάλλοντων, δημιουργούν ευκαιρίες για την απλοποίηση της ανάπτυξης νέας λειτουργικότητας όπως και για την ενθάρρυνση της επαναχρησιμοποίησης των σχεδίων. Η Handel-C έχει ικανότητες για πρόσβαση στην

εσωτερική και εξωτερική μνήμη καθώς και σε καταλόγους. Μέσω βιβλιοθηκών προκαθορισμένων λειτουργιών, κοινοί χρήστες APIs shield από χαμηλού επιπέδου διεπαφές διευκολύνονται στην ενσωμάτωση των FPGAs σε φυσικούς πόρους όπως τα περιφερειακά και οι επεξεργαστές, με τους τελευταίους να καθιστούν δυνατό τον ταυτόχρονο σχεδιασμό υλικού/ λογισμικού. Αυτή είναι μία προσέγγιση που μπορεί να σημαίνει σημαντικό κέρδος σε χρόνο, δίνοντας στον σχεδιαστή περισσότερο χρόνο να συγκεντρωθεί στη λειτουργικότητα του πυρήνα.

4.9 Πλεονεκτήματα της χρήσης της Handel-C

Εκτός από το προφανές πλεονέκτημα της χρήσης μιας μεθόδου σχεδιασμού που βασίζεται σε μία γλώσσα υψηλού επιπέδου σε σχέση με μία που θα βασίζεται σε γλώσσα χαμηλού επιπέδου, η Handel-C παρέχει μία πολύ ισχυρή μέθοδο υλοποίησης υλικού, η οποία είναι πολύ χρήσιμη ιδιαίτερα στις εφαρμογές του προσαρμοσμένου υπολογισμού (Custom Computing applications).

Αφού η Handel-C υποστηρίζει ένα μεγάλο τμήμα κατασκευασμάτων της ANSI-C, είναι δυνατή η εύκολη μεταφορά μεταξύ των δύο γλωσσών.



Στις εφαρμογές του προσαρμοσμένου υπολογισμού, μέρος του σχεδιασμού υλοποιείται στο τμήμα υλικού ενώ ένα άλλο μέρος της υλοποιείται στο τμήμα του λογισμικού. Αφού η Handel-C είναι πολύ όμοια με μία προστακτική γλώσσα, διευκολύνει πάρα πολύ τη διαίρεση του αρχικού σχεδιασμού σε τμήμα υλικού και τμήμα λογισμικού.

Η προστακτική φύση της Handel-C επίσης διευκολύνει τη διόρθωση και την αναβάθμιση του σχεδίου του υλικού. Αυτό σημαίνει ότι το τμήμα του υλικού μπορεί να τροποποιηθεί εύκολα για να ληφθούν υπόψη τροποποιήσεις που έχουν γίνει στο τμήμα του λογισμικού και το αντίστροφο.

Εξαιτίας της φύσης υψηλού επιπέδου της Handel-C είναι δυνατό να μπορεί το ίδιο άτομο να πραγματοποιήσει τόσο την υλοποίηση υλικού αλλά και την υλοποίηση λογισμικού. Αυτό ελαττώνει σημαντικά το κόστος ανάπτυξης καθώς δεν απαιτούνται δύο άτομα για να ασχοληθούν ξεχωριστά με τον σχεδιασμό του υλικού και του λογισμικού.

4.10 Υλοποιήσεις

Ένα πραγματικό παράδειγμα FPGA σχεδιασμού

Ένα πρόγραμμα που χρηματοδοτήθηκε από την Ευρωπαϊκή Επιτροπή και ονομάσθηκε SEHaD (Software Engineering for Hardware Design) έδωσε στη Celoxica την ευκαιρία να επιδείξει την αξία της Handel-C. Σαν μέρος του προγράμματος, δύο σχεδιαστικές ομάδες της Ericsson συμμετείχαν σε μία παράλληλη σχεδιαστική προσπάθεια για την ανάπτυξη μιας λειτουργίας συμπίεσης IPv6 επικεφαλίδων σε ένα FPGA. Η μία

ομάδα χρησιμοποίησε την Handel-C και η άλλη παραδοσιακή μέθοδο και εργαλεία σχεδιασμού υλικού (Verilog/Leonardo).

- **Η Εφαρμογή**

Τα routers (δρομολογητές) είναι σημαντικά στοιχεία των δικτύων Πρωτοκόλλου Διαδικτύου. Για να αντιμετωπισθούν οι αυξανόμενες απαιτήσεις για απόδοση, τα σύγχρονα routers βασίζονται όλο και περισσότερο στις υλοποιήσεις υλικού της λίστας πρωτοκόλλου. Για τη μεταφορά πληροφοριών σε πραγματικό χρόνο όπως η φωνή μέσω Διαδικτύου, η αναλογία μεγέθους φορτίου-πακέτων είναι ανεπαρκής από άποψη χαρακτηριστικών πραγματικού χρόνου αλλά και χρησιμοποίησης εύρους γραμμής (bandwidth). Το IPv6 όξυνε ακόμα περισσότερο το πρόβλημα. Προκειμένου να αντιμετωπισθεί το πρόβλημα για τις συνδέσεις από σημείο σε σημείο, έχουν επινοηθεί σχέδια αντιπροσώπευσης των αμετάβλητων μερών των επικεφαλίδων πακέτων σε μια ροή δεδομένων από έναν οκταψήφιο με δεκαεξαψήφιο αριθμό (CID). Η συμπιεσμένη επικεφαλίδα πακέτων περιέχει αυτόν τον αριθμό και απόλυτες ή δέλτα αντιπροσωπεύσεις των μη σταθερών πεδίων δεδομένων των επικεφαλίδων πακέτων στη ροή δεδομένων. Η αντίστροφη λειτουργία ονομάζεται αποσυμπίεση και εκτελείται στο λαμβανόμενο τέλος της σύνδεσης. Σε κάθε σύνδεση Διαδικτύου, πολλές συνεδρίες (sessions) μεταδίδονται ταυτόχρονα. Κάθε συνεδρία έχει τη δικιά της αναλογία φορτίου-πακέτων.

Η ιδέα πίσω από τη συμπίεση των επικεφαλίδων πακέτων είναι να γίνει δυνατό το μεταδίδον και το λαμβάνον router να συμφωνήσουν σε μία μικρή αντιπροσώπευση των επικεφαλίδων. Οι επικεφαλίδες πακέτων συμπιέζονται όταν μεταδίδονται και αποσυμπιέζονται στην αρχική μορφή όταν λαμβάνονται. Αυτό μειώνει το απαιτούμενο εύρος γραμμής της σύνδεσης και βελτιώνει τα χαρακτηριστικά πραγματικού χρόνου της ροής δεδομένων. Για παράδειγμα, η συμπίεση IPv6 half-rate πακέτων φωνής θα οδηγήσει σε 65% οικονομία στο εύρος γραμμής. Αυτά θα αποτελούνται από μια ροή δεδομένων IP πακέτων με σχεδόν τις ίδιες επικεφαλίδες και συχνά μικρή αναλογία φορτίου-πακέτων.

- **Τα αποτελέσματα**

Δύο σχεδιαστές αποτελούσαν την ομάδα που χρησιμοποίησε τη Handel-C. Ο ένας είχε κάποια εμπειρία στη χρησιμοποίηση μιας παλαιότερης μορφής της Handel-C και κάποια σχεδιαστική εμπειρία στο πεδίο των εφαρμογών, ενώ ο άλλος ήταν ένα απόφοιτος πανεπιστημίου με καθόλου εμπειρία στη Handel-C ή στις εφαρμογές. Ο παρακάτω πίνακας συνοψίζει τα αποτελέσματα της υλοποίησης με Handel-C ή με την εφαρμογή.

	Handel-C/ DK1	Verilog/Leonardo
Design time	4 man-months	12-16 man-months
Program size	40 pages	200 pages
Compile time	3 minutes	1.5 hours
Size (Virtex 1000-4)	35 % logic, 30 % memory*	NA
Size (Virtex 2000E-8)	17 % logic, 15 % memory*	All logic, all memory**
Speed (Virtex 1000-4)	28 – 33 MHz	NA
Speed (Virtex 2000E-8)	44 MHz	49 MHz

*: Κατανεμημένη μνήμη. Δεν χρησιμοποιήθηκαν καθόλου memory blocks.
**: Μία συνειδητή επιλογή. Χρησιμοποιήθηκε όλη η λογική για να αυξηθεί η ταχύτητα. Χρησιμοποιήθηκαν όλα τα memory blocks.

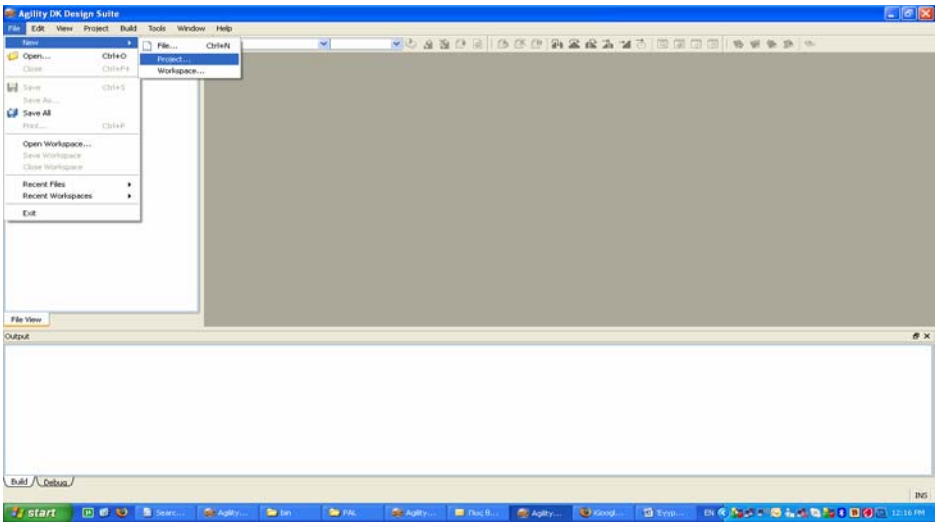
Το πιο εντυπωσιακό αποτέλεσμα είναι η διαφορά στον χρόνο σχεδιασμού, ο οποίος ήταν 3-4 μικρότερος όταν χρησιμοποιήθηκε η Handel-C, με παρόμοια αποτελέσματα από άποψη ταχύτητας και μια πολύ μειωμένη περιοχή. Μια ανάλυση των αποτελεσμάτων του

προγράμματος απέδωσε τη διαφορά στο χρόνο σχεδιασμού σε τρία βασικά πλεονεκτήματα που καταδείχθηκαν από την Handel-C: η υποστήριξή για σειριακή λογική, η συμπαγής αντιπροσώπευση της λειτουργικότητας και η μεθοδολογία σχεδιασμού που μοιάζει με αυτή του λογισμικού που παρείχε γρήγορες αλλαγές στο σχεδιαστικό περιβάλλον. Σύμφωνα με την αναφορά του SEHaD, αυτά τα αποτελέσματα ήταν σύμφωνα με την εμπειρία άλλων χρηστών του προγράμματος, καθώς επίσης και μικρότερων παραδειγμάτων που έγιναν από την Ericsson. «Η διαφορά στο μέγεθος προγράμματος απεικονίζει την περιεκτικότητα της σύνταξης της C και την υψηλότερου επιπέδου αφαιρετικότητα της Handel-C. Ένα παράδειγμα αυτού είναι το στυλ της περιγραφής του σειριακού/ παράλληλου κώδικα στη Handel-C απέναντι στο έλεγχο της αλληλουχίας με χρήση μηχανών πεπερασμένων-καταστάσεων στη Verilog ή τη VHDL. Το μικρό μέγεθος του προγράμματος και η αλγοριθμική εκφραστικότητα της Handel-C το κατέστησαν εύκολο για μας να κάνουμε δραστικές πειραματικές αλλαγές στον κώδικα. Αυτό επέτρεψε να ερευνήσουν ένα μεγαλύτερο εύρος λύσεων.» Άλλη μία βασική διαφορά που σχολιάστηκε στα ευρήματα της αναφοράς αφορούσε το παρόμοιο με του λογισμικού σχεδιαστικό στυλ της DK1 Design Suite εφαρμογής της Celoxica, το οποίο ενσωματώνει τη διαχείριση του προγράμματος, την προσομοίωση, τη σύνθεση και τη βελτιστοποίηση σε ένα σχεδιαστικό περιβάλλον. «Η επαλήθευση πραγματοποιήθηκε αρχικά χρησιμοποιώντας το παράδειγμα του λογισμικού διόρθωσης. Η μεταγλώττιση σε πραγματικό υλικό έγινε πολύ νωρίς και μετά κάθε σημαντική προσθήκη στη λειτουργικότητα χρησιμοποιώντας ένα προσαυξητικό σχεδιαστικό στυλ. Αυτό δημιούργησε αυτοπεποίθηση στη λύση και έδωσε τη δυνατότητα να τρέξουν περισσότερα σετ ελέγχων σε πραγματικό χρόνο.»

Για τον σχεδιασμό της λειτουργικότητας παρόμοιας πολυπλοκότητας, ο κόσμος του σχεδιασμού λογισμικού είναι απλούστερος από το αντίστοιχο του σχεδιασμού υλικού. Η δυνατότητα της χρήσης μεθοδολογιών σχεδιασμού λογισμικού για τη δημιουργία λύσεων υλικού μπορεί να φαίνεται σχεδόν αιρετική. Όμως, όπως φαίνεται και από το παραπάνω παράδειγμα, λύνει πολλά θέματα στην ανάπτυξη σύγχρονων ολοκληρωμένων κυκλωμάτων. Ελαττώνει τον χρόνο σχεδιασμού κατά 3-4 φορές, η γλώσσα και τα συστήματα σχεδιασμού υιοθετούνται εύκολα και ο μικρός και αποτελεσματικός κώδικας κάνει τις βασικές αλλαγές σε επίπεδο συστήματος απλές. Στο παράδειγμα του σχεδιασμού λογισμικού, έχουμε ότι χρειαζόμαστε για να αυξήσουμε το μέγεθος αφαιρετικότητας επαρκώς ώστε να περιγράψουμε με τον συντομότερο δυνατό τρόπο την επιθυμητή λειτουργία, παρά την υποκείμενη δομική λεπτομέρεια και να ξεπεράσουμε πολλές από τις δυσκολίες και τις ανεπάρκειες του σύγχρονου σχεδιασμού υλικού.

5. Λειτουργία του προγράμματος Agility DK Design Suite

Όταν ξεκινάμε το πρόγραμμα Agility DK Design Suite εμφανίζεται το παρακάτω παράθυρο. Στην περίπτωση που πρέπει να ξεκινήσει ο χρήστης ένα καινούριο project τότε ακολουθεί τις επόμενες ενέργειες.

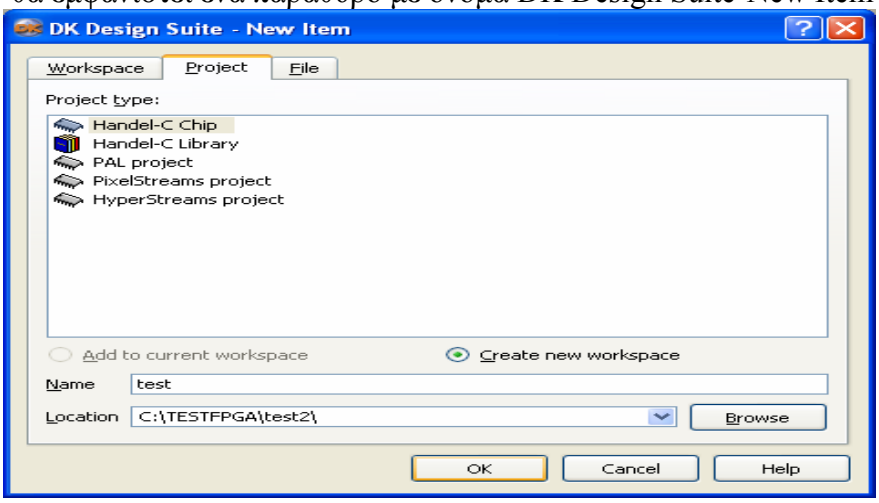


Δημιουργία Καινούριου Project

Ακολουθούμε τις παρακάτω ενέργειες:

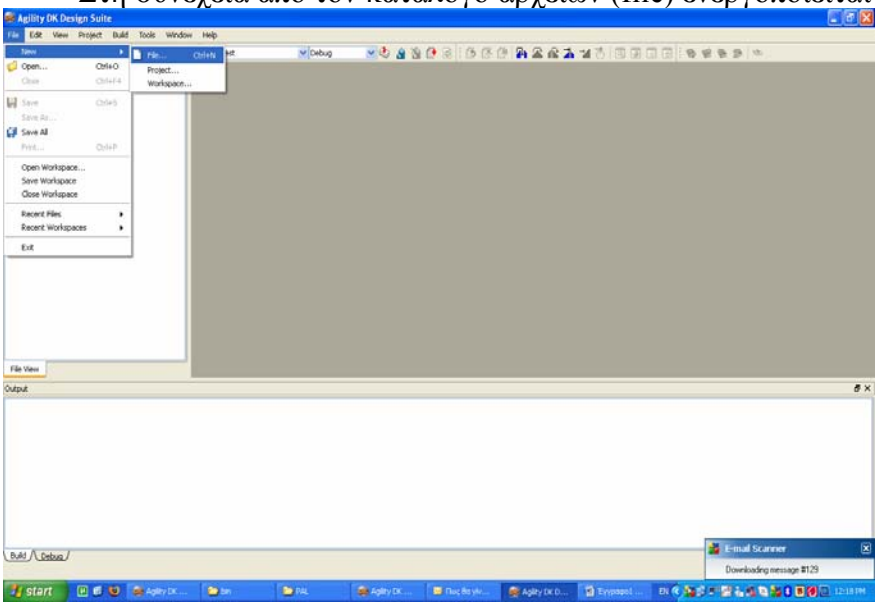
File → New → Project

θα εμφανιστεί ένα παράθυρο με όνομα DK Design Suite-New Item το οποίο φαίνεται.

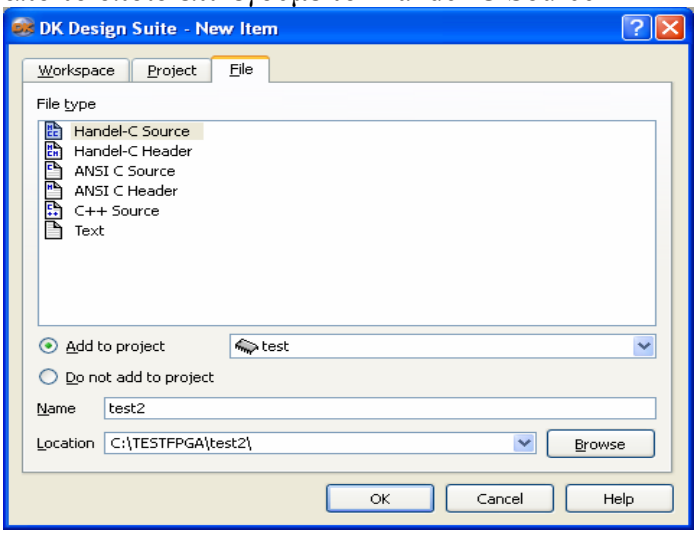


Στο Name δίνουμε το όνομα του project ενώ στο Location δίνουμε το Path στο οποίο θα αποθυκευτεί το project.

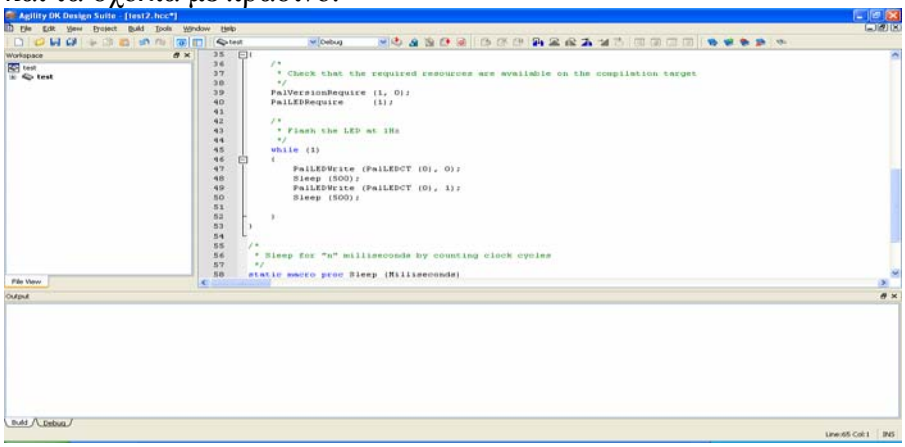
Στη συνέχεια από τον κατάλογο αρχείων (file) ενεργοποιείται η επιλογή new .



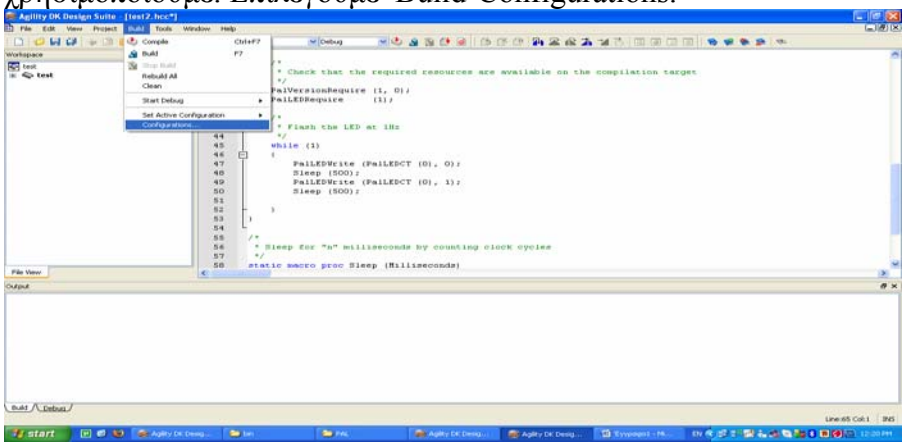
Όταν γίνει αυτή η επιλογή, τότε παρουσιάζεται το πλαίσιο με το όνομα New Item από το οποίο επιλεγούμε το Handel-C Source



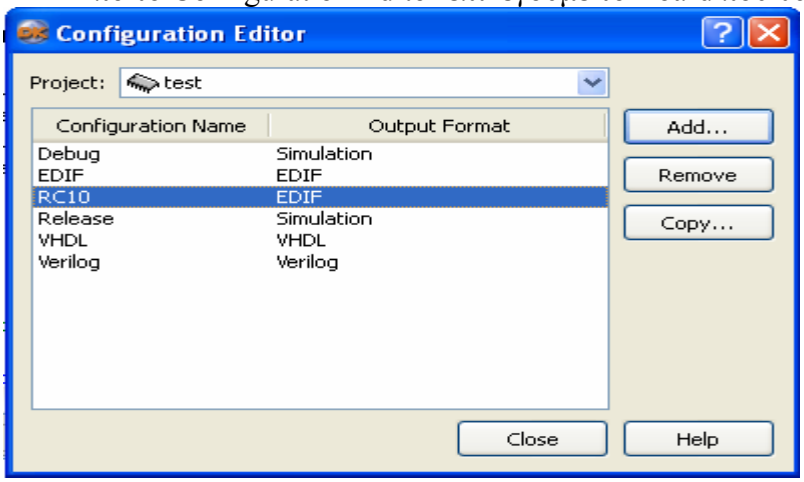
Το άσπρο πλαίσιο που καταλαμβάνει το μεγαλύτερο μέρος του είναι ο επεξεργαστής κειμένου μέσα στον οποίο γράφουμε τον κώδικα. Οι λέξεις κλειδιά εμφανίζονται με μπλε, και τα σχόλια με πράσινο.



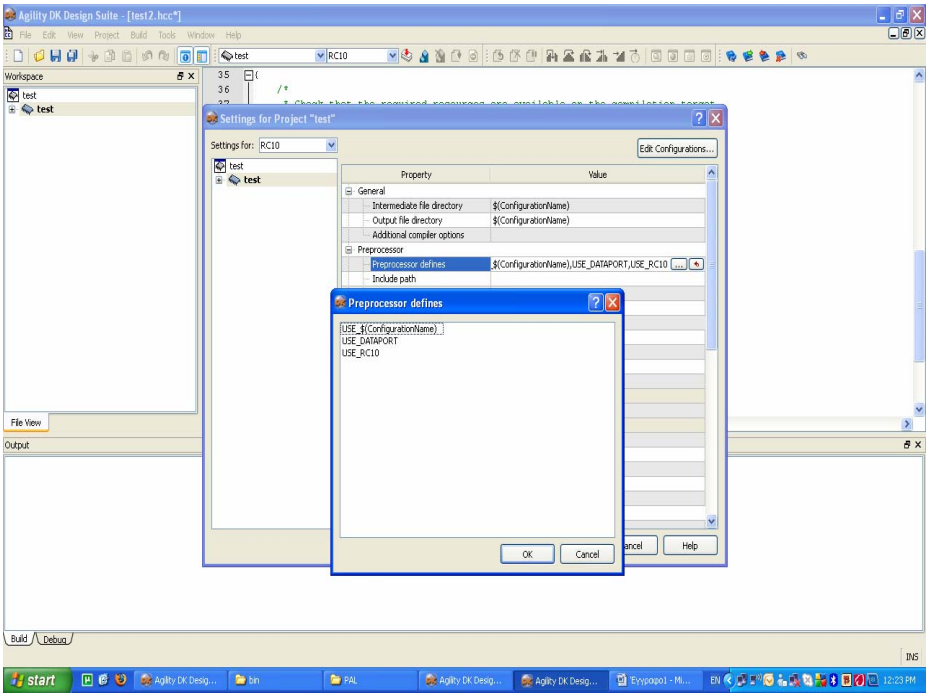
Στην συνέχεια πρέπει να γίνουν οι ρυθμίσεις ανάλογα με ποιο Board χρησιμοποιούμε. Επιλέγουμε Build-Configurations.



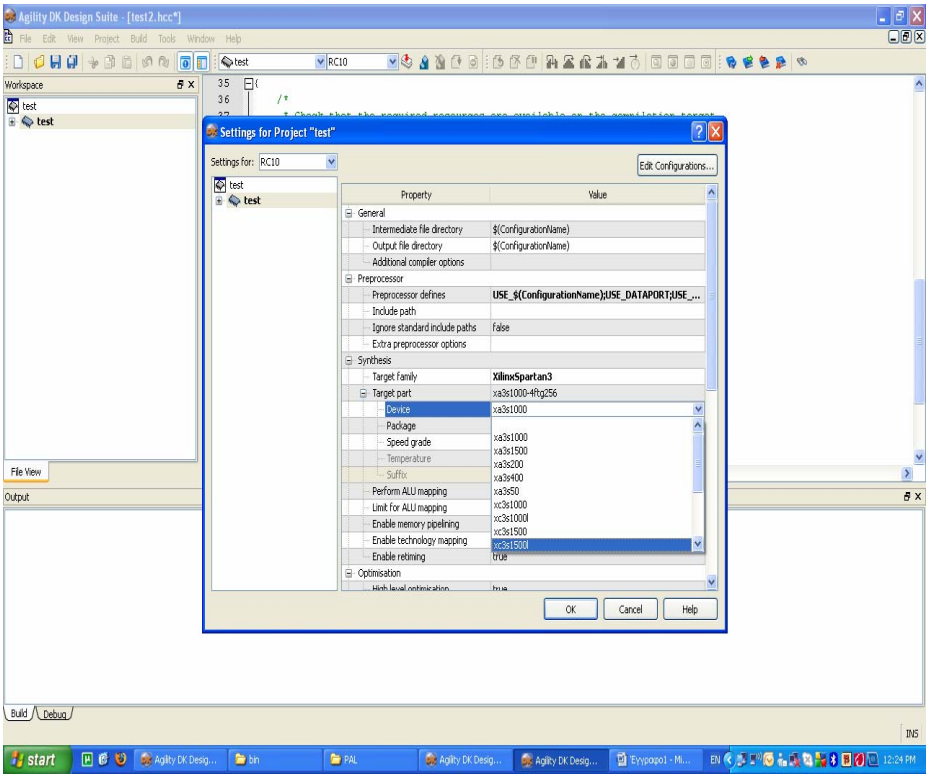
Από το Configuration Editor επιλέγουμε το Board που θα χρησιμοποιήσουμε.



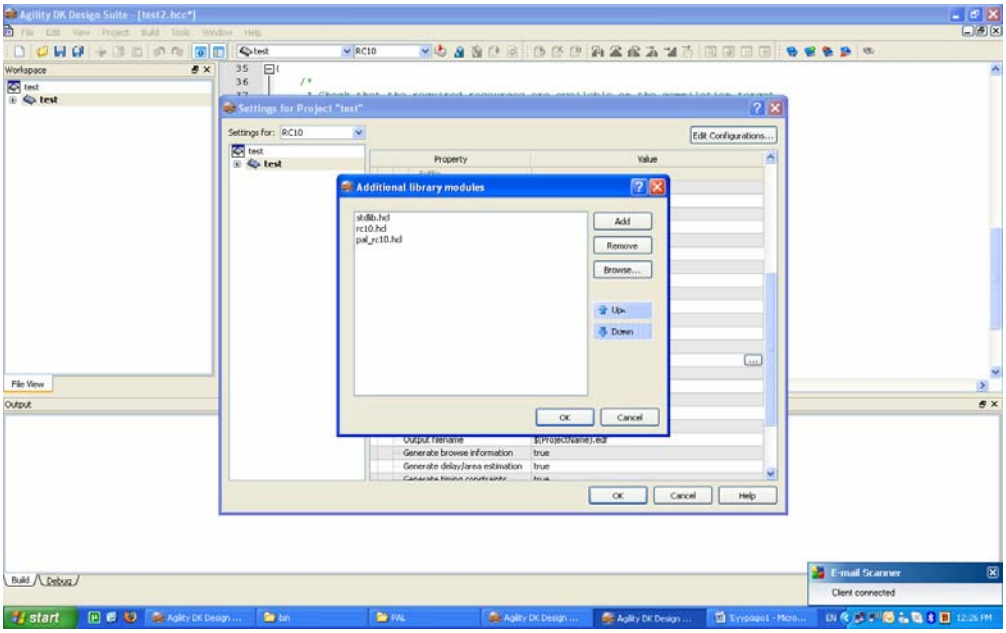
Στην συνέχεια από το Workspace κάνουμε δεξί κλικ στο project και επιλέγουμε settings



Από το Settings-Target family επιλέγουμε XilinxSpart3 ενώ από το Target part-Device XC3S1500.



Από το Linker-Additional library modules κάνουμε add stdlib.hcl rc10.hcl pal_console.hcl pal_rc10.hcl. Ανάλογα ποιες εντολές χρησιμοποιούμε και ποιο board , αλλάζουν τα αρχεία που κάνουμε add.



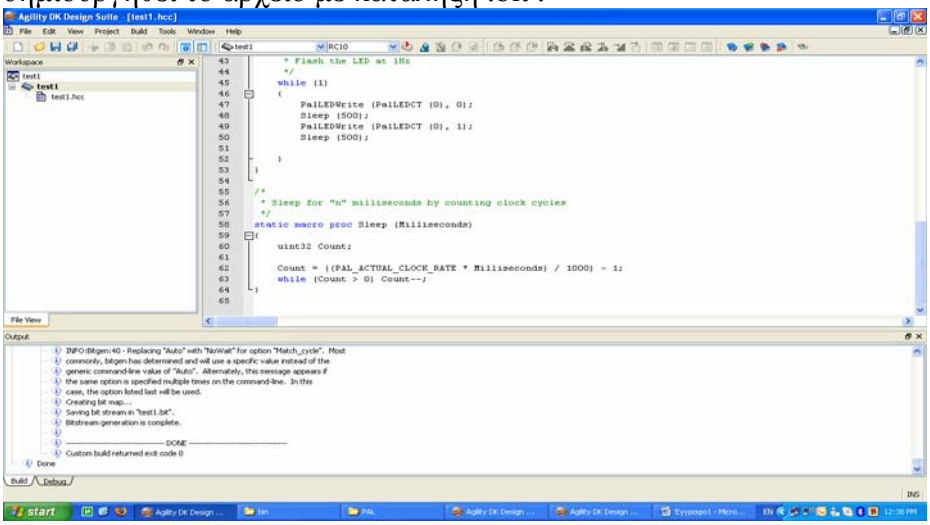
Τέλος από το Custom Build –custom build steps commands γράφουμε:

cd $(ConfigurationName)

call "$(AGILITY_PDK_HOME)\Software\Bin\edifmake_rc10" $(ProjectName)

και από το custom build steps output files : **$(ProjectName).bit**

Αφού έχουμε ολοκληρώσει και το settings επιλέγουμε από Build-build (F7) για να δημιουργηθεί το αρχείο με κατάληξη .bit .



6. Παραδείγματα για εκμάθηση της γλώσσας HANDEL-C

Παράδειγμα 1
Στο παρακάτω πρόγραμμα αναβοσβήνει το δεύτερο Led.

```
#ifdef USE_SIM
#define PAL_TARGET_CLOCK_RATE 1000000
#else
#define PAL_TARGET_CLOCK_RATE PAL_PREFERRED_VIDEO_CLOCK_RATE
#endif
#include "pal_master.hch"
#include "stdlib.hch"

void main (void)
{

   while (1)
   {
      PalLEDWrite (PalLEDCT (2), 1);
   }
}
```

Για να χρησιμοποιήσουμε την while θα πρέπει να κάνουμε χρήση του #include με το αρχείο βιβλιοθήκης stdio.hch (standard input output) πριν από τη main, ενώ την βιβλιοθηκη #include "pal_master.hch" γιατί εδώ περιέχεται η μακροεντολή PalLEDWrite.
PalLEDWrite (PalLEDCT (2), 1);
PalLEDCT (2) ⟶ Το δύο σημαίνει ποιο Led θα ενεργοποιηθεί
PalLEDCT (2), ⟶ Δηλώνει την κατάσταση του Led (1 ενεργοποιημένο, 0 απενεργοποιημένο)
Χρησιμοποιούμε την εντολή while(1) για να εκτελείτε συνέχεια η εντολή PalLEDWrite (PalLEDCT (2), 1).

```
   while (1)
   {
      PalLEDWrite (PalLEDCT (2), 1);
   }
```

Παράδειγμα 2

Στο παρακάτω πρόγραμμα αναβοσβήνει το τρίτο Led.

```
#ifdef USE_SIM
#define PAL_TARGET_CLOCK_RATE 1000000
#else
#define PAL_TARGET_CLOCK_RATE PAL_PREFERRED_VIDEO_CLOCK_RATE
#endif

#include "pal_master.hch"
#include "stdlib.hch"
```

```
void main (void)
{
   while (1)
   {
      PalLEDWrite (PalLEDCT (3), 1);
      PalLEDWrite (PalLEDCT (3), 0);
   }
}
```

Παράδειγμα 3

Στο παρακάτω πρόγραμμα ανάβουν όλα τα Led ένα-ένα και σβήνουν ένα- ένα με καθυστέρηση 500ms .

```
#ifdef USE_SIM
#define PAL_TARGET_CLOCK_RATE 1000000
#else
#define PAL_TARGET_CLOCK_RATE PAL_PREFERRED_VIDEO_CLOCK_RATE
#endif
#include "pal_master.hch"
#include "stdlib.hch"
typedef  unsigned 32 uint32; //ορίζω το είδος της μεταβλητής σε 32-bit
static macro proc Sleep (Milliseconds);
void main (void)
{
   while (1)
   {
      PalLEDWrite (PalLEDCT (0), 1);
      Sleep (500);
      PalLEDWrite (PalLEDCT (1), 1);
      Sleep (500);
      PalLEDWrite (PalLEDCT (2), 1);
      Sleep (500);
      PalLEDWrite (PalLEDCT (3), 1);
      Sleep (500);
      PalLEDWrite (PalLEDCT (4), 1);
      Sleep (500);
      PalLEDWrite (PalLEDCT (5), 1);
      Sleep (500);
      PalLEDWrite (PalLEDCT (6), 1);
      Sleep (500);
      PalLEDWrite (PalLEDCT (7), 1);
      Sleep (500);
      PalLEDWrite (PalLEDCT (0), 0);
      Sleep (500);
      PalLEDWrite (PalLEDCT (1), 0);
      Sleep (500);
      PalLEDWrite (PalLEDCT (2), 0);
      Sleep (500);
      PalLEDWrite (PalLEDCT (3), 0);
```

```
        Sleep (500);
        PalLEDWrite (PalLEDCT (4), 0);
        Sleep (500);
        PalLEDWrite (PalLEDCT (5), 0);
        Sleep (500);
        PalLEDWrite (PalLEDCT (6), 0);
        Sleep (500);
        PalLEDWrite (PalLEDCT (7), 0);
        Sleep (500);
    }
}
static macro proc Sleep (Milliseconds)
{
    uint32 Count;
    Count = ((PAL_ACTUAL_CLOCK_RATE * Milliseconds) / 1000) - 1;
    while (Count > 0) Count--; //θα αφαιρείται ένα μέχρι το count να φτάσει στο μηδέν
}
```

Παράδειγμα 4

Στο παρακάτω πρόγραμμα όταν είναι πατημένος ο διακόπτης 0 αναβοσβήνει το Led 5 ενώ όταν δεν είναι πατημένος ο διακόπτης αναβοσβήνει το Led 3 . Τα led αναβοσβήνουν με καθυστέρηση 500ms.

```
#ifdef USE_SIM
#define PAL_TARGET_CLOCK_RATE 1000000
#else
#define PAL_TARGET_CLOCK_RATE PAL_PREFERRED_VIDEO_CLOCK_RATE
#endif
#include "pal_master.hch"
#include "stdlib.hch"
unsigned 1 Value;
static macro proc Sleep (Milliseconds);
typedef  unsigned 32 uint32;
void main (void)
{   while(1){
    PalSwitchRead (PalSwitch(0),&Value);
    if (Value == 1)
    {
        PalLEDWrite (PalLEDCT (3), 1);
          Sleep (500);
        PalLEDWrite (PalLEDCT (3), 0);
          Sleep (500);
    }
    else {

        PalLEDWrite (PalLEDCT (5), 1);
          Sleep (500);
        PalLEDWrite (PalLEDCT (5), 0);
          Sleep (500);
        }
    }
}
```

```
static macro proc Sleep (Milliseconds)
{
   uint32 Count;

   Count = ((PAL_ACTUAL_CLOCK_RATE * Milliseconds) / 1000) - 1;
   while (Count > 0) Count--;
}
```

Παράδειγμα 5

Στο παρακάτω πρόγραμμα ανάβουν ένα-ένα με την σειρά (0,1,2….6,7)τα Led χρησιμοποιώντας την μακροεντολή PalLEDWrite.

```
#ifdef USE_SIM
#define DELAY_WIDTH 16
#else
#define DELAY_WIDTH 26
#endif
#include "pal_master.hch"
#include "stdlib.hch"
unsigned 1 Value;
static macro proc Sleep (Milliseconds);
typedef  unsigned 32 uint32;
void main (void)
{ unsigned (log2ceil (PalLEDCount () + 1)) Index;
   unsigned DELAY_WIDTH Count;
   while(1)
   {
   for (Index = 0; Index < PalLEDUniqueCount (); Index++)
   {
      macro expr I = Index <- (log2ceil (PalLEDCount ()));
      do
      {
         par
         {
            PalLEDWrite (PalLED (I), Count[DELAY_WIDTH-5]);
            Count++;
         }
      }
      while (Count != 0);
      PalLEDWrite (PalLED (I), 0);
   }
}
   }
```

Παράδειγμα 6

Στο παρακάτω πρόγραμμα εμφανίζεται ο αριθμός 170 στο δυαδικό σύστημα.

```
#define PAL_TARGET_CLOCK_RATE 25175000
#include "pal_master.hch"
#include "stdlib.hch"
#include "rc10.hch"
```

```
void led()
{while(1)
{
   RC10LEDWriteMask (0b10101010);
}
}
void main (void)
{
   led();
   }
```

Παράδειγμα 7

Στο παρακάτω πρόγραμμα όταν είναι πατημένος ο διακόπτης 1 εμφανίζεται ο αριθμός 123 στο δυαδικό σύστημα ,αλλιώς εμφανίζεται ο αριθμός 204.

```
#define PAL_TARGET_CLOCK_RATE 25175000
#include "pal_master.hch"
#include "stdlib.hch"
#include "rc10.hch"

#ifdef USE_SIM
#define PAL_TARGET_CLOCK_RATE 1000000
#else
#define PAL_TARGET_CLOCK_RATE PAL_PREFERRED_VIDEO_CLOCK_RATE
#endif

unsigned 1 Value;

void main (void)
{
   while(1)
   {
   PalSwitchRead (PalSwitch(1),&Value);
   if (Value == 1)
   {
   RC10LEDWriteMask (0b01111011);
   }
   else
   {
   RC10LEDWriteMask (0b11001100);
   }
}
}
```

Παράδειγμα 8

Παρουσιάζεται πρόγραμμα ώστε να ανάβει κάθε φορά ένα Led αρχίζοντας από το 0 έως το 7(χωρίς να επηρεάζεται η κατάσταση του προηγούμενου Led) ,επίσης κάθε φορά που αλλάζουμε Led η χρονοκαθυστέρηση θα αυξάνεται κατά 500 ms. Η αρχική χρονοκαθυστέρηση θα είναι 500ms.

```
#ifdef USE_SIM
```

```
#define PAL_TARGET_CLOCK_RATE 1000000
#else
#define PAL_TARGET_CLOCK_RATE PAL_PREFERRED_VIDEO_CLOCK_RATE
#endif
#include "pal_master.hch"
#include "stdlib.hch"
typedef  unsigned 32 uint32;
static macro proc Sleep (Milliseconds);
void main (void)
{
   while (1)
   {
      PalLEDWrite (PalLEDCT (0), 1);
      Sleep (500);
      PalLEDWrite (PalLEDCT (1), 1);
      Sleep (1000);
      PalLEDWrite (PalLEDCT (2), 1);
      Sleep (1500);
      PalLEDWrite (PalLEDCT (3), 1);
      Sleep (2000);
      PalLEDWrite (PalLEDCT (4), 1);
      Sleep (2500);
      PalLEDWrite (PalLEDCT (5), 1);
      Sleep (3000);
      PalLEDWrite (PalLEDCT (6), 1);
      Sleep (3500);
      PalLEDWrite (PalLEDCT (7), 1);
      Sleep (4000);

   }
}
static macro proc Sleep (Milliseconds)
{
   uint32 Count;
   Count = ((PAL_ACTUAL_CLOCK_RATE * Milliseconds) / 1000) - 1;
   while (Count > 0) Count--;
}
```

Παράδειγμα 9

Παρουσιάζεται πρόγραμμα ώστε να ανάβει κάθε φορά μόνο ένα Led αρχίζοντας από το 0 έως το 7 ,επίσης η χρονοκαθυστέρηση που χρησιμοποιείται είναι διπλάσια από την προηγούμενη. Η αρχική χρονοκαθυστέρηση είναι 500ms.

```
#ifdef USE_SIM
#define PAL_TARGET_CLOCK_RATE 1000000
#else
#define PAL_TARGET_CLOCK_RATE PAL_PREFERRED_VIDEO_CLOCK_RATE
#endif
#include "pal_master.hch"
#include "stdlib.hch"
```

```
typedef  unsigned 32 uint32;
static macro proc Sleep (Milliseconds);
void main (void)
{
   while (1)
   {
      PalLEDWrite (PalLEDCT (0), 1);
      Sleep (500);
      PalLEDWrite (PalLEDCT (0), 0);
      PalLEDWrite (PalLEDCT (1), 1);
      Sleep (1000);
      PalLEDWrite (PalLEDCT (1), 0);
      PalLEDWrite (PalLEDCT (2), 1);
      Sleep (2000);
      PalLEDWrite (PalLEDCT (2), 0);
      PalLEDWrite (PalLEDCT (3), 1);
      Sleep (4000);
      PalLEDWrite (PalLEDCT (3), 0);
      PalLEDWrite (PalLEDCT (4), 1);
      Sleep (8000);
      PalLEDWrite (PalLEDCT (4), 0);
      PalLEDWrite (PalLEDCT (5), 1);
      Sleep (16000);
      PalLEDWrite (PalLEDCT (5), 0);
      PalLEDWrite (PalLEDCT (6), 1);
      Sleep (32000);
      PalLEDWrite (PalLEDCT (6), 0);
      PalLEDWrite (PalLEDCT (7), 1);
      Sleep (64000);
      PalLEDWrite (PalLEDCT (7), 0);
   }
}
static macro proc Sleep (Milliseconds)
{
   uint32 Count;
   Count = ((PAL_ACTUAL_CLOCK_RATE * Milliseconds) / 1000) - 1;
   while (Count > 0) Count--;
}
```

Παράδειγμα 10

Παρουσιάζεται πρόγραμμα που θα προσθέτει δυο αριθμούς και θα εμφανίζει το αποτέλεσμα στα led σε δυαδική μορφή. Με την χρήση συναρτήσεων.

```
#define PAL_TARGET_CLOCK_RATE 25175000
#include "pal_master.hch"
#include "stdlib.hch"
#include "rc10.hch"

void led()
{while(1)
{  int a,b,c;
```

```
   a=0b11010100;
   b=0b10010010;
   c=a+b;
   RC10LEDWriteMask (c);
}
}
void main (void)
{
   led();
   }
```

Παράδειγμα 11
Παρουσιάζεται πρόγραμμα που θα αφαιρεί δυο αριθμούς και θα εμφανίζει το αποτέλεσμα στα led σε δυαδική μορφή. Με την χρήση συναρτήσεων.

```
#define PAL_TARGET_CLOCK_RATE 25175000
#include "pal_master.hch"
#include "stdlib.hch"
#include "rc10.hch"

void led()
{while(1)
{  int a,b,c;

   a=0b11010100;
   b=0b10010010;
   c=b-a;
   RC10LEDWriteMask (c);
}
}
void main (void)
{
   led();
   }
```

Παράδειγμα 12
Παρουσιάζεται πρόγραμμα που θα πολλαπλασιάζει δυο αριθμούς και θα εμφανίζει το αποτέλεσμα στα led σε δυαδική μορφή. Με την χρήση συναρτήσεων.

```
#define PAL_TARGET_CLOCK_RATE 25175000
#include "pal_master.hch"
#include "stdlib.hch"
#include "rc10.hch"

void led()
{while(1)
{  int a,b,c;

   a=0b00001000;
```

```
   b=0b00010010;
   c=b*a;
   RC10LEDWriteMask (c);
}
        }
void main (void)
{
   led();
   }
```

Παράδειγμα 13

Παρουσιάζεται πρόγραμμα που θα διαιρεί δυο αριθμούς και θα εμφανίζει το αποτέλεσμα στα led σε δυαδική μορφή. Με την χρήση συναρτήσεων.

```
#define PAL_TARGET_CLOCK_RATE 25175000
#include "pal_master.hch"
#include "stdlib.hch"
#include "rc10.hch"

void led()
{while(1)
{  int a,b,c;

   a=0b00001000;
   b=0b00010010;
   c=b/a;
   RC10LEDWriteMask (c);
}
}
void main (void)
{
   led();
   }
```

Παραδειγμα 14

Παρουσιάζεται πρόγραμμα που θα προσθέτει δυο αριθμούς , στη συνέχεια θα πολλαπλασιάζεται το αποτέλεσμα της πρόσθεσης με τον πρώτο αριθμό. Στην συνέχεια το αποτέλεσμα της πρόσθεσης θα αφαιρείτε από το αποτέλεσμα του πολλαπλασιασμού αφού διαιρεθεί με το 10 .Τέλος το αποτέλεσμα θα εμαφανιζεται στα led σε δυαδική μορφή. Με την χρήση συναρτήσεων.

```
#define PAL_TARGET_CLOCK_RATE 25175000
#include "pal_master.hch"
#include "stdlib.hch"
#include "rc10.hch"

void led()
{while(1)
{  int a,b,c,d,e,f;
```

```
   a=16;
   b=2;
   c=a+b;
   d=a*c;
   e=10;
   f=(d-c)/e;
   RC10LEDWriteMask (f);
}
}
void main (void)
{
   led();
   }
```

Παράδειγμα 15

Παρουσιάζεται πρόγραμμα που προσθέτει δυο αριθμούς και εμφανίζει το αποτέλεσμα στα led σε δυαδική μορφή αφού πρώτα αυξηθεί το αποτέλεσμα κατά ένα . Με την χρήση συναρτήσεων.

```
#define PAL_TARGET_CLOCK_RATE 25175000
#include "pal_master.hch"
#include "stdlib.hch"
#include "rc10.hch"

void led()
{while(1)
{  int a,b,c;

   a=0b10101010;
   b=0b10110010;
   c=a+b;
   c++;
   RC10LEDWriteMask (c);
}
}
void main (void)
{
   led();
   }
```

Παράδειγμα 16

Στο παρακάτω πρόγραμμα εκτελείται η λογική πράξη OR ανάμεσα σε δυο αριθμούς και εμφανίζεται το αποτέλεσμα στα led σε δυαδική μορφή. Με την χρήση συναρτήσεων.

```
#define PAL_TARGET_CLOCK_RATE 2517500
#include "pal_master.hch"
#include "stdlib.hch"
#include "rc10.hch"

void led()
```

```
{while(1)
{  int a,b,c;

   a=0b00001000;
   b=0b00010010;
   c=a|b;
   RC10LEDWriteMask (c);
}
}
void main (void)
{
   led();
   }
```

Παράδειγμα 17

Στο παρακάτω πρόγραμμα εκτελείται η λογική πράξη XOR ανάμεσα σε δυο αριθμούς και εμφανίζεται το αποτέλεσμα στα led σε δυαδική μορφή. Με την χρήση συναρτήσεων.

```
#define PAL_TARGET_CLOCK_RATE 25175000
#include "pal_master.hch"
#include "stdlib.hch"
#include "rc10.hch"

void led()
{while(1)
{  int a,b,c;

   a=0b10101000;
   b=0b10010010;
   c=a^b;
   RC10LEDWriteMask (c);
}
}
void main (void)
{
   led();
   }
```

Παράδειγμα 18

Παρουσιάζεται πρόγραμμα όπου εκτελείται η λογική πράξη AND ανάμεσα σε δύο αριθμούς και εμφανίζεται το αποτέλεσμα στα led σε δυαδική μορφή. Με την χρήση συναρτήσεων.

```
#define PAL_TARGET_CLOCK_RATE 25175000
#include "pal_master.hch"
#include "stdlib.hch"
#include "rc10.hch"

void led()
{while(1)
```

```
{  int a,b,c;

   a=0b10101010;
   b=0b10110010;
   c=a&b;
   RC10LEDWriteMask (c);
}
}
void main (void)
{
   led();
   }
```

Παράδειγμα 19

Παρουσιάζεται πρόγραμμα που θα προσθέτει δύο αριθμούς στη συνέχεια θα διαιρεί το αποτέλεσμα με τον πρώτο αριθμό και θα βγάζει στην έξοδο το υπόλοιπο της διαίρεσης αυτής.Με την χρήση συναρτήσεων.

```
#define PAL_TARGET_CLOCK_RATE 25175000
#include "pal_master.hch"
#include "stdlib.hch"
#include "rc10.hch"

void led()
{while(1)
{  int a,b,c,d;

   a=0b10101010;
   b=0b10110010;
   c=a+b;
   d=0b00001010;
   d=c % a;
   RC10LEDWriteMask (d);
}
}
void main (void)
{
   led();
   }
```

Παράδειγμα 20

Παρουσιάζεται πρόγραμμα στο οποίο ανάβουν τα led με αριθμητική πρόοδο (1...3…5…7).

```
#define PAL_TARGET_CLOCK_RATE 25175000
#ifdef USE_SIM
#define DELAY_WIDTH 16
#else
#define DELAY_WIDTH 26
#endif
```

```
#include "pal_master.hch"
#include "stdlib.hch"
unsigned 1 Value;
static macro proc Sleep (Milliseconds);
typedef  unsigned 32 uint32;
void main (void)
{   unsigned Index;
   unsigned DELAY_WIDTH Count;
   while(1)
   {

   for (Index = 0; Index <4; Index++)
   {
      macro expr I = 2*Index+1 ;
          do
      { par
         {
            PalLEDWrite (PalLED (I), Count[DELAY_WIDTH-5]);
            Count++;
         }
      }
      while (Count != 0);
      PalLEDWrite (PalLED (I), 0);
   }
}
   }
```

Παράδειγμα 21
Συγκρίνουμε δύο αριθμούς. Στην έξοδο να εμφανίζεται ο μεγαλύτερος.

```
#ifdef USE_SIM
#define PAL_TARGET_CLOCK_RATE 1000000
#else
#define PAL_TARGET_CLOCK_RATE PAL_PREFERRED_VIDEO_CLOCK_RATE
#endif
#include "pal_master.hch"
#include "stdlib.hch"

void main (void)
{
   int A,B,C;
   A = 10;
   B = 8;
if (A>B)
   {
      C = A;
       RC10LEDWriteMask (C);
   }
else if (B>A)
   {
```

```
      C = B;
       RC10LEDWriteMask (C);
   }
else
   {
    C = 0;
       RC10LEDWriteMask (C);
   }

}
```

Παράδειγμα 22
Συγκρίνουμε τρεις αριθμούς. Στην έξοδο να εμφανίζεται ο μικρότερος.

```
#ifdef USE_SIM
#define PAL_TARGET_CLOCK_RATE 1000000
#else
#define PAL_TARGET_CLOCK_RATE PAL_PREFERRED_VIDEO_CLOCK_RATE
#endif
#include "pal_master.hch"
#include "stdlib.hch"
void main(void)
{
int A,B,C,D;

A = 100;
B = 171;
C = 53;
if (A<B && A<C)
   {
      D = A;
      RC10LEDWriteMask (D);
   }
else if (B<A && B<C)
   {
      D = B;
      RC10LEDWriteMask (D);
   }
else if (C<A && C<B)
   {
      D = C;
      RC10LEDWriteMask (D);
   }
else
        {
                D = 0;
                RC10LEDWriteMask (D);
        }
}
```

Παράδειγμα 23
Αύξων και φθίνων μετρητής. Όταν είναι σε λογική κατάσταση 1 το PA0 τότε να αυξάνεται το περιεχόμενο της εξόδου έως την τιμή 255, ενώ όταν το PA0 είναι σε λογική κατάσταση 0 να μειώνεται το περιεχόμενο της εξόδου έως την τιμή 0.

```
#define PAL_TARGET_CLOCK_RATE 25175000
#include "pal_master.hch"
#include "stdlib.hch"
#include "rc10.hch"

#ifdef USE_SIM
#define PAL_TARGET_CLOCK_RATE 1000000
#else
#define PAL_TARGET_CLOCK_RATE PAL_PREFERRED_VIDEO_CLOCK_RATE
#endif
static macro proc Sleep (Milliseconds);
unsigned 1 Value;
typedef  unsigned 32 uint32;
void main(void)
{
int a1;
int A;
int B;
        while(1)
   {
   PalSwitchRead (PalSwitch(1),&Value);
   if (Value == 1)
        {
                do
                        {
                                A ++ ;
                                RC10LEDWriteMask (A);
                                Sleep(150);
                        }
         while (A<255);
        }

   PalSwitchRead (PalSwitch(2),&Value);
   if (Value == 1)
        {
                do
                        {
                                A -- ;
                                RC10LEDWriteMask (A);
                                Sleep(150);
                        }
        while (A>0);
```

```
        }
    }
}
static macro proc Sleep (Milliseconds)
{
    uint32 Count;
    Count = ((PAL_ACTUAL_CLOCK_RATE * Milliseconds) / 1000) - 1;
    while (Count > 0) Count--;
}
```

Παράδειγμα 24

Παρουσιάζεται πρόγραμμα στο οποίο αυξάνεται ένας αριθμός με αριθμητική πρόοδο και το αποτέλεσμα εμφανίζεται στα LED στο δυαδικό σύστημα.

```
#define PAL_TARGET_CLOCK_RATE 25175000
#ifdef USE_SIM
#define DELAY_WIDTH 16
#else
#define DELAY_WIDTH 26
#endif
#include "pal_master.hch"
#include "stdlib.hch"
#include "rc10.hch"

unsigned 1 Value;
static macro proc Sleep (Milliseconds);
typedef  unsigned 32 uint32;

void main (void)
{   unsigned I,Index;
    unsigned DELAY_WIDTH Count;
    while(1)
    {
    for (Index = 0; Index <255; Index++)
    {
        I = 2*Index+1 ;
       RC10LEDWriteMask (I);
       Sleep(500);

    }

        }
}
    static macro proc Sleep (Milliseconds)
{
    uint32 Count;
    Count = ((PAL_ACTUAL_CLOCK_RATE * Milliseconds) / 1000) - 1;
    while (Count > 0) Count--;
}
```

Παράδειγμα 25
Παρουσιάζεται πρόγραμμα που εμφανίζει στους ενδείκτες επτά τομέων τα εξής : 0,1,2….E,F

```
#ifdef USE_SIM
#define PAL_TARGET_CLOCK_RATE 1000000
#else
#define PAL_TARGET_CLOCK_RATE PAL_PREFERRED_VIDEO_CLOCK_RATE
#endif

#include "pal_master.hch"
#include "stdlib.hch"
static macro proc Sleep (Milliseconds);
void main (void)
{
   unsigned 5 Count;
   unsigned 8 Mask;
   PalVersionRequire (1, 0);
   PalSevenSegRequire (1);


   PalSevenSegEnable (PalSevenSegCT (0));
   while (1)
   {
      do
      {
         PalSevenSegWriteDigit (PalSevenSegCT (0), Count[4:1], 0);
         Sleep (500);
         Count++;
      }
      while (Count != 0);
   }
}
static macro proc Sleep (Milliseconds)
{
   macro expr Cycles = (PAL_ACTUAL_CLOCK_RATE * Milliseconds) / 1000;
   unsigned (log2ceil (Cycles)) Count;

   Count = 0;
   do
   {
      Count++;
   }
   while (Count != Cycles - 1);
}
```

Παράδειγμα 26
Παρουσιάζεται πρόγραμμα που εμφανίζει έναν αύξων μετρητή από το 0,1…99.

```
#ifdef USE_SIM
#define PAL_TARGET_CLOCK_RATE 1000000
#else
#define PAL_TARGET_CLOCK_RATE PAL_PREFERRED_VIDEO_CLOCK_RATE
#endif

#include "pal_master.hch"
#include "stdlib.hch"
static macro proc Sleep (Milliseconds);
void main (void)
{  unsigned I,K,ctr;
   unsigned 5 Count;
   unsigned 8 Mask;

   PalVersionRequire (1, 0);
   PalSevenSegRequire (1);

   PalSevenSegEnable (PalSevenSegCT (0));
   PalSevenSegEnable (PalSevenSegCT (1));

   while (1)
   {

      for (I = 0; I <10; I++)
      {
        PalSevenSegWriteDigit (PalSevenSegCT (0),I, 0);
      for (K = 0; K <10; K++)
      {
        PalSevenSegWriteDigit (PalSevenSegCT (1),K, 0);
         Sleep (500);
        PalSevenSegWriteDigit (PalSevenSegCT (1),0,0);
      }
      }
   }
}
static macro proc Sleep (Milliseconds)
{
   macro expr Cycles = (PAL_ACTUAL_CLOCK_RATE * Milliseconds) / 1000;
   unsigned (log2ceil (Cycles)) Count;

   Count = 0;
   do
   {
      Count++;
```

```
    }
    while (Count != Cycles - 1);
}
```

Παράδειγμα 27
Παρουσιάζεται πρόγραμμα που εμφανίζει στον πρώτο ενδείκτη επτά τομέων την λέξη "THESSALONIKI"

```
#ifdef USE_SIM
#define PAL_TARGET_CLOCK_RATE 1000000
#else
#define PAL_TARGET_CLOCK_RATE PAL_PREFERRED_VIDEO_CLOCK_RATE
#endif

#include "pal_master.hch"
#include "stdlib.hch"
static macro proc Sleep (Milliseconds);
void main (void)
{  unsigned I,K,ctr;
   unsigned 5 Count;
   unsigned 8 Mask;
   static unsigned 8 capitals[256]={119,127,57,63,121,113,61,118,6,30,105,56,85,55,
                                    63,115,103,123,109,7,62,74,106,81,73,91};
   static unsigned char name[13]="THESSALONIKI";
   PalVersionRequire (1, 0);
   PalSevenSegRequire (1);

   PalSevenSegEnable (PalSevenSegCT (0));
   PalSevenSegEnable (PalSevenSegCT (1));

   while (1)
   {
     for(ctr=0; ctr<13; ctr++)
     {

       PalSevenSegWriteShape (PalSevenSegCT (0),capitals[name[ctr]-65]);

      Sleep (1000);

     }
   }
}
static macro proc Sleep (Milliseconds)
{
   macro expr Cycles = (PAL_ACTUAL_CLOCK_RATE * Milliseconds) / 1000;
   unsigned (log2ceil (Cycles)) Count;

   Count = 0;
   do
   {
     Count++;
```

```
    }
    while (Count != Cycles - 1);
}
```

Παράδειγμα 28

Παρουσιάζεται πρόγραμμα με το οποίο εμφανίζεται και στους δυο ενδείκτες επτά τομέων το όνομα “FPGA” σαν κυλιόμενο μήνυμα .

```
#ifdef USE_SIM
#define PAL_TARGET_CLOCK_RATE 1000000
#else
#define PAL_TARGET_CLOCK_RATE PAL_PREFERRED_VIDEO_CLOCK_RATE
#endif

#include "pal_master.hch"
#include "stdlib.hch"
static macro proc Sleep (Milliseconds);
void main (void)
{  unsigned I,K,ctr;
   unsigned 5 Count;
   unsigned 8 Mask;
   static unsigned 8 capitals[256]={119,127,57,63,121,113,61,118,6,30,105,56,85,55,
                                    63,115,103,123,109,7,62,74,106,81,73,91};
   static unsigned char name[6]="FPGA";
   PalVersionRequire (1, 0);
   PalSevenSegRequire (1);

   PalSevenSegEnable (PalSevenSegCT (0));
   PalSevenSegEnable (PalSevenSegCT (1));
   while (1)
   {
     for(ctr=0; ctr<=5; ctr++)
     {
       if(ctr>=0 && ctr<5)
          PalSevenSegWriteShape (PalSevenSegCT (1),capitals[name[ctr]-65]);
       else
          PalSevenSegWriteShape (PalSevenSegCT (1),0);
       if(ctr>=1 && ctr<=5)
          PalSevenSegWriteShape (PalSevenSegCT (0),capitals[name[ctr-1]-65]);
       else
          PalSevenSegWriteShape (PalSevenSegCT (0),0);

      Sleep (1000);
     }
   }
}
static macro proc Sleep (Milliseconds)
{
   macro expr Cycles = (PAL_ACTUAL_CLOCK_RATE * Milliseconds) / 1000;
   unsigned (log2ceil (Cycles)) Count;

   Count = 0;
```

```
   do
   {
      Count++;
   }
   while (Count != Cycles - 1);
}
```

Παράδειγμα 29
Παρουσιάζεται πρόγραμμα το οποίο ελέγχει αν ο αριθμός που δώσαμε είναι μικρότερος από το 100 τότε να τον εμφανίσει στο δεκαδικό σύστημα στους ενδείκτες επτά τομέων ενώ αν είναι μεγαλύτερος στο δεκαεξαδικό .

```
#ifdef USE_SIM
#define PAL_TARGET_CLOCK_RATE 1000000
#else
#define PAL_TARGET_CLOCK_RATE PAL_PREFERRED_VIDEO_CLOCK_RATE
#endif

#include "pal_master.hch"
#include "stdlib.hch"
static macro proc Sleep (Milliseconds);
void main (void)
{
   unsigned 8 x;
   unsigned a,a1,y,z;
   unsigned 5 Count;
   unsigned 8 Mask;
  PalVersionRequire (1, 0);
  PalSevenSegRequire (1);


  PalSevenSegEnable (PalSevenSegCT (0));
  PalSevenSegEnable (PalSevenSegCT (1));
   while (1)
   {
        x=189;
        a1=10;
        a=16;
       if (x>=0 && x<=99)
       { y=(x/a1)[3:0];
         PalSevenSegWriteDigit (PalSevenSegCT (0), y, 0);
         z=(x%a1)[3:0];
         PalSevenSegWriteDigit (PalSevenSegCT (1), z, 0);
         Sleep(500);
       }
       if (x>99 && x<=255)
       {y=(x/a)[3:0];
         PalSevenSegWriteDigit (PalSevenSegCT (0), y, 0);
         z=(x%a)[3:0];
         PalSevenSegWriteDigit (PalSevenSegCT (1), z, 0);
```

```
        Sleep(500);
      }

   }
}
static macro proc Sleep (Milliseconds)
{
   macro expr Cycles = (PAL_ACTUAL_CLOCK_RATE * Milliseconds) / 1000;
   unsigned (log2ceil (Cycles)) Count;
   Count = 0;
   do
   {
      Count++;
   }
   while (Count != Cycles - 1);
}
```

Παράδειγμα 30

Παρουσιάζεται πρόγραμμα στο οποίο υπολογίζεται το αποτέλεσμα της συνάρτησης $x = \frac{75*n}{(n+4)}$ **για n=1,2….8 και το αποτέλεσμα εμφανίζεται στους ενδείκτες επτά τομέων.**

```
#ifdef USE_SIM
#define PAL_TARGET_CLOCK_RATE 1000000
#else
#define PAL_TARGET_CLOCK_RATE PAL_PREFERRED_VIDEO_CLOCK_RATE
#endif

#include "pal_master.hch"
#include "stdlib.hch"
static macro proc Sleep (Milliseconds);
void main (void)
{
   unsigned 8 n,x;
   unsigned a,y,z;
   unsigned 5 Count;
   unsigned 8 Mask;
  PalVersionRequire (1, 0);
  PalSevenSegRequire (1);


  PalSevenSegEnable (PalSevenSegCT (0));
  PalSevenSegEnable (PalSevenSegCT (1));
   while (1)
   {
      a=16;
   for (n=1;n<8;n++)
   {  x=75*n/(n+4);
      y=(x/a)[3:0];
      PalSevenSegWriteDigit (PalSevenSegCT (0), y, 0);
```

```
        z=(x%a)[3:0];
        PalSevenSegWriteDigit (PalSevenSegCT (1), z, 0);
          Sleep(500);
    }

    }
}

static macro proc Sleep (Milliseconds)
{
    macro expr Cycles = (PAL_ACTUAL_CLOCK_RATE * Milliseconds) / 1000;
    unsigned (log2ceil (Cycles)) Count;
    Count = 0;
    do
    {
        Count++;
    }
    while (Count != Cycles - 1);
}
```

Παράδειγμα 31

Παρουσιάζεται πρόγραμμα στο οποίο όταν ενεργοποιηθεί ο διακόπτης 1 θα προσθέτει δυο αριθμούς ,όταν ενεργοποιηθεί ο διακόπτης 2 θα αφαιρει δυο αριθμούς ,όταν ενεργοποιηθεί ο διακόπτης 3 θα πολλαπλασιάζει δυο αριθμούς ενώ όταν ενεργοποιηθεί ο διακόπτης 4 θα διαιρεί τους δυο αριθμούς. Το αποτέλεσμα εμφανίζεται στους ενδείκτες επτά τομέων στο δεκαεξαδικό σύστημα.

```
#ifdef USE_SIM
#define PAL_TARGET_CLOCK_RATE 1000000
#else
#define PAL_TARGET_CLOCK_RATE PAL_PREFERRED_VIDEO_CLOCK_RATE
#endif

#include "pal_master.hch"
#include "stdlib.hch"
static macro proc Sleep (Milliseconds);
static macro proc Sevenseg (x);
void main (void)
{
    unsigned 8 a1,a2,n,x;
    unsigned a,y,z;
    unsigned 5 Count;
    unsigned 8 Mask;
    unsigned 1 Value;
   PalVersionRequire (1, 0);
   PalSevenSegRequire (1);


   PalSevenSegEnable (PalSevenSegCT (0));
   PalSevenSegEnable (PalSevenSegCT (1));
    while (1)
    {
```

```
    a1=20;
    a2=10;
    PalSwitchRead (PalSwitch(1),&Value);
    if (Value == 1)
    {x=a1+a2;
       Sevenseg (x);
    }
    PalSwitchRead (PalSwitch(2),&Value);
    if (Value == 1)
    {x=a1-a2;
       Sevenseg (x);
    }
    PalSwitchRead (PalSwitch(3),&Value);
    if (Value == 1)
    {x=a1*a2;
       Sevenseg (x);
    }
    PalSwitchRead (PalSwitch(0),&Value);
    if (Value == 1)
    {x=a1/a2;
       Sevenseg (x);
    }
    }
}
static macro proc Sevenseg (x)
{   unsigned a,y,z;
    a=16;
    y=(x/a)[3:0];
    PalSevenSegWriteDigit (PalSevenSegCT (0), y, 0);
    z=(x%a)[3:0];
    PalSevenSegWriteDigit (PalSevenSegCT (1), z, 0);
    Sleep(500);
}
static macro proc Sleep (Milliseconds)
{
    macro expr Cycles = (PAL_ACTUAL_CLOCK_RATE * Milliseconds) / 1000;
    unsigned (log2ceil (Cycles)) Count;

    Count = 0;
    do
    {
       Count++;
    }
    while (Count != Cycles - 1);
```

Παράδειγμα 32

Παρουσιάζεται πρόγραμμα που υπολογίζει την *Ακολουθία Φιμπονάτσι* X[n]=X[n-1]+X[n-2] με X[0]=0 και X[1]=1.

```
 #ifdef USE_SIM
#define PAL_TARGET_CLOCK_RATE 1000000
```

```
#else
#define PAL_TARGET_CLOCK_RATE PAL_PREFERRED_VIDEO_CLOCK_RATE
#endif

#include "pal_master.hch"
#include "stdlib.hch"
static macro proc Sleep (Milliseconds);


void main (void)
{  unsigned y,z,n,ctr;
   unsigned 8 a;
   unsigned 5 Count;
   unsigned 8 Mask;
   unsigned 8 x[12] ;
   PalVersionRequire (1, 0);
   PalSevenSegRequire (1);

  PalSevenSegEnable (PalSevenSegCT (0));
  PalSevenSegEnable (PalSevenSegCT (1));

   while (1)
   {
      x[0]=0;
       y=(x[0]/a)[3:0];
       PalSevenSegWriteDigit (PalSevenSegCT (0), y, 0);
       z=(x[0]%a)[3:0];
       PalSevenSegWriteDigit (PalSevenSegCT (1), z, 0);
       Sleep(1000);
       x[1]=1;
       y=(x[1]/a)[3:0];
       PalSevenSegWriteDigit (PalSevenSegCT (0), y, 0);
       z=(x[1]%a)[3:0];
       PalSevenSegWriteDigit (PalSevenSegCT (1), z, 0);
       Sleep(1000);
      for (n=2; n<=11; n++)
       {
       x[n]=x[n-1]+x[n-2];
       a=10;
       y=(x[n]/a)[3:0];
       PalSevenSegWriteDigit (PalSevenSegCT (0), y, 0);
       z=(x[n]%a)[3:0];
       PalSevenSegWriteDigit (PalSevenSegCT (1), z, 0);
       Sleep(1000);

       }
   }
}

static macro proc Sleep (Milliseconds)
```

```
{
   macro expr Cycles = (PAL_ACTUAL_CLOCK_RATE * Milliseconds) / 1000;
   unsigned (log2ceil (Cycles)) Count;

   Count = 0;
   do
   {
      Count++;
   }
   while (Count != Cycles - 1);
}
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

- Aaby, A.A. 1996. *Introduction to Programming Languages*. Anthony A. Aaby. url: www.emu.edu.tr/aelci/Courses/D-318/D-318-Files/plbook/intro.htm
- Bergin, T. J. Gibson, R.G. 1996. *History of Programming Languages*: Volume 2. ACM Press. σσ: 27-28, 223-224, 331-332, 671-672.
- Carl A. Gunter, C. A. 1992. *Semantics of Programming Languages: Structures and Techniques*. MIT Press. σσ: 1-9.
- Celoxica. 2005. *Handel-C Language Reference Manual*. Celoxica Limited.
- Dahl, O.J. Dijkstra, E.W. Hoare, C. A. R. 1972. *Structured Programming*. Academic Press. σσ: 39-41.
- Hurst, J. 2007. Types of Programming Languages. *The SANS Institute*. url: www.giac.org/resources/whitepaper/architecture/97.php
- Jackson, M.A. 1975. *Principles of Program Design*. Academic Press. London.
- Mermet, J. 1993. *Fundamentals and Standards in Hardware Description Languages*. Springer Verlag. σσ: ix.
- Null, L. Lobur, J. 2006. *The Essentials of Computer Organization and Architecture*: 2nd Edition. Jones & Bartlett Publishers. σσ: 39-41.
- O' Reilly Media. 2004. *History of Programming Languages*. (PDF) O' Reilly Media.
- Scott, M. 2009. *Πραγματολογία Γλωσσών Προγραμματισμού*. Κλειδάριθμος.
- Stewart, B. 2000. History of the C Programming Language. *Living Internet*. url: www.livinginternet.com/i/iw_unix_c.htm
- Βακάλη, Α. Γιαννόπουλος, Η. Ιωαννίδης, Ν. Μάλαμας, Κ. Μανωλόπουλος Ι. Πολίτης, Π. 2009. *Ανάπτυξη Εφαρμογών σε Προγραμματιστικό Περιβάλλον*, Γ' Τάξης Ενιαίου Λυκείου. ΟΕΔΒ.
- Γιαννέλης, Δ. 2005. *Ανάπτυξη Εφαρμογών σε Προγραμματιστικό Περιβάλλον*. Εκδόσεις Γκιούρδας.
- Θραμπουλίδης, Κ. 2000. *Γλώσσες Προγραμματισμού*. Ελληνικό Ανοικτό Πανεπιστήμιο. σσ: 27-32, 37-41.
- http://en.wikipedia.org/wiki/Fpga
- Clive Maxfield, book, "The Design Warrior's Guide to FPGAs".Published by Elsevier, 2004
- Δελτίο Τύπου, " Xilinx συνιδρυτής Freeman Ross Τιμήθηκε το 2009 ως Εθνικό Μέγαρο Εφευρέτες του νεοσύλλεκτος Fame για την εφεύρεσή του FPGA
- http://www.eecg.toronto.edu/~vaughn/challenge/fpga_arch.html
- http://www.mentor.com/products/fpga/handel-c/rc-series-platforms/upload/rc240.pdf?CFID=107703&CFTOKEN=60782427&jsessionid=5924DFAFDA0EDD7DB03378937C8BF7C0.prod6
- http://www.embeddedstar.com/press/content/2005/5/embedded18313.html