Εθνικό Μετσόβιο Πολυτεχνείο
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

# Διπλωματική Εργασία: Περιβάλλοντα ανοικτού κώδικα για ανάπτυξη υλικού σε υψηλό επίπεδο

Λαντζουράκης Γεώργιος

*Επιβλέπων :*
Επίκουρος Καθηγητής Κουκούτσης Ηλίας

Αθήνα, Ιούνιος 2013

Εθνικό Μετσόβιο Πολυτεχνείο
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

# Διπλωματική Εργασία:
# Περιβάλλοντα ανοικτού κώδικα για ανάπτυξη υλικού σε υψηλό επίπεδο

Λαντζουράκης Γεώργιος

*Επιβλέπων :*
Επίκουρος Καθηγητής Κουκούτσης Ηλίας

Αθήνα, Ιούνιος 2013

Υπογραφές τριμελούς επιτροπής

| | | |
|---|---|---|
| Κουκούτσης Ηλίας | Παπαοδυσσεύς Κων/νος | Καγιάφας Ελευθέριος |
| Επίκουρος Καθηγητής | Αναπληρωτής Καθηγητής | Καθηγητής |

Περιβάλλοντα ανοικτού κώδικα για ανάπτυξη υλικού σε υψηλό επίπεδο

.....
Λαντζουράκης Γεώργιος
Διπλωματούχος Ηλεκτρόγος Μηχανικός και Μηχανικός Υπολογιστών

## Περίληψη

Στην παρούσα διπλωματική, το ζητούμενο ήταν να εκπονηθεί μια μελέτη/σχεδίαση ενός στοιχείου ενός επεξεργαστή, το οποίο θα τον καθιστούσε ικανό να εκτελεί αριθμητικές πράξεις με τελεστές μεταβλητού μήκους. Μέρος του ζητούμενου είναι αυτή η σχεδίαση να γίνει αποκλειστικά με εργαλεία Ανοικτού Κώδικα, υιοθετώντας την καινούρια τάση στο χώρο του Υλικού η οποία μιμείται το Ανοικτό Λογισμικό.

Λέξεις κλειδιά: Υλικό ανοιχτού κώδικα, Μεταβλητή ακρίβεια πράξεων, OpenRISC, OpenSPARC, Verilog, VLSI, Ροή Σχεδίασης.

## Abstract

In this thesis, it was requested to study and design a processor component which would be able to execute arbitrary precision arithmetic operations. It was also requested to carry out this task solely based on Open Source tools, which follows the trend of Hardware design that mimicks Open Source Software.

Keywords: Open Source Hardware, Arbitrary Precision, OpenRISC, OpenSPARC, Verilog, VLSI, Design Flow.

## Ευχαριστίες

Οφείλω τις θερμότερες ευχαριστίες μου στην οικογένεια μου για τη στήριξη, υλική και συναισθηματική, που μου παρείχε όπως επίσης και στον επιβλέποντα Καθηγητή κ.Ηλία Κουκούτση για τις συμβουλές και τις διορθώσεις του, την εμπιστοσύνη που έδειξε στην προσπάθεια μου καθώς και τη γενικότερη συμπαράστασή του.

*Αφιερωμένο στη μνήμη της Μητέρας μου, Κυριακής*

# ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

Δώς μοι πᾶ στῶ καὶ τὰν γᾶν κινάσω
(Απόδοση: Δώσε μου σημείο να στηριχθώ και θα κινήσω τη Γη)

*Αρχιμήδης, Αρχαίος Έλληνας φιλόσοφος, 285 - 212 π.Χ.*

# Μέρος Ι
# Εισαγωγικά

## 1 Περιγραφές Υλικού (Hardware)

### 1.1 ΑΣΗ (Αυτοματοποίηση Σχεδίασης Ηλεκτρονικών)

Η ΑΣΗ (EDA, Electronic Design Automation )είναι μια θεαματική επιτυχία της μηχανικής. Το τελευταίο τέταρτο του αιώνα, βελτιωμένα εργαλεία αύξησαν την παραγωγικότητα των σχεδιαστών με ένα συντελεστή μεγαλύτερο του χίλια. Χωρίς την ΑΣΗ, ο νόμος του Μουρ θα ήταν μια μη ρεαλιστική ευχή. Κανένα chip του ενός δις transistors δε θα είχε σχεδιαστεί ή αποσφαλματωθεί χωρίς τα εργαλεία αυτά, γεγονός που σημαίνει ότι δε θα είχαμε σήμερα ούτε λαπτοπ, ούτε έξυπνα κινητά ούτε βιντεοπαιχνίδια ούτε οποιαδήποτε άλλη περίπλοκη ηλεκτρονική συσκευή. Με αυτά τα εργαλεία και τις ανάλογες ικανότητες χειρισμού τους, οι σχεδιαστές ΑΣΗ κατάφεραν να σχεδιάσουν,να αποσφαλματώσουν και να τεστάρουν τεράστια τσιπ με με μειωμένο χρόνο προς την τελική παραγωγή. Η ΑΣΗ είναι πολύ πιο σύνθετη από την πρόοδο στον τομέα των φυσικών διαστάσεων των ΟΚ. Αυτό είναι περισσότερο αντικείμενο της επιστήμης των ηλεκτρονικών υλικών.

**Ορισμός 1** *Ως ΑΣΗ ορίζουμε την κατηγορία των προγραμμάτων και εργαλείων λογισμικού τα οποία χρησιμοποιούνται για τη σχεδίαση ηλεκτρονικών συστημάτων, όπως τυπωμένων πλακετών και ολοκληρωμένων κυκλωμάτων.*

**Ορισμός 2** *Οι συνδυασμοί αυτών των εργαλείων λογισμικού μας επιτρέπουν να έχουμε το layout (πχ σε μορφή GDSII) ξεκινώντας από την περιγραφή RTL μέσα από τα στάδια των synthesis, placement και routing (σύνθεση, τοποθέτηση και δρομολόγηση). Η ακολουθία αυτή και τα εργαλεία που χρησιμοποιούνται ονομάζονται Design Flow (Ροή Σχεδίασης)*

### 1.2 Top Down και Bottom Up Methodology

Top Down και Bottom Up ονομάζουμε γενικά τις στρατηγικές με τις οποίες ιεραρχούμε γνώση και επεξεργαζόμαστε πληροφορία σε πεδία όπως το λογισμικό, τις ανθρωπιστικές επιστήμες και τη διοίκηση.

Η Top Down (αντίστοιχη του παραγωγικού συλλογισμού) είναι η ανάλυση του συστήματος στα υποσυστήματα από τα οποία αποτελείται. Ξεκινάμε δηλαδή από μια γενική σύνοψη του συτήματος χωρίς πολλές λεπτομέρειες(με τη βοήθεια μαύρων κουτιών) και προοδευτικά προσθέτουμε πληροφορία και αναλύουμε τα επί μέρους συστήματα

Η Bottom UP (αντίστοιχη του επαγωγικού συλλογισμού) ξεκινάει μελετώντας και συνδέοντας τα επί μέρους κομμάτια που συνθέτουν τα μεγαλύτερα συστήματα. Μια τέτοια προσέγγιση αρχικά δίνει στα επιμέρους συστήματα τη μέγιστη δυνατή λεπτομέρεια.

Στη συγκεκριμένη περίπτωση των VLSI και της σχεδίασης Hardware, με την Top Down ξεκινάμε βελτιστοποιώντας τα σχεδιαστικά τμήματα τα οποία βρίσκονται στο υψηλό επίπεδο της σχεδιαστικής ιεραρχίας, τις μεταξύ τους συνδέσεις και τα σήματα. Για παράδειγμα αν θέλουμε να σχεδιάσουμε ένα επεξεργαστή, ξεκινάμε από τις δομικές του μονάδες, ALU's, μνήμες κτλ. Το πλεονέκτημα είναι ότι με αυτόν τον τρόπο σχεδίασης είναι εύκολη η μελέτη του υψηλού επιπέδου αλλά το μειονέκτημα ότι δεν μπορούμε να εκτιμήσουμε ακριβώς το πώς θα είναι το επίπεδο του Layout

Αντίστοιχα, στην Bottom Up, ξεκινάμε από το χαμηλό ιεραρχικό επίπεδο της ιεραρχίας (πχ από τις λογικές πύλες) χρησιμοποιούμε αυτές τις λειτουργικές ενότητες ως μπλοκ για να χτίσουμε τις πιο σύνθετες (περιγράφουμε μία και την καλούμε ως στιγμιότυπο ξανά και ξανά, instantiate). Το πλεονέκτημα είναι είμαι σίγουροι για το επίπεδο του layout αλλά ο σχεδιαστικός φόρτος είναι αυξημένος ειδικά όσο πιο σύνθετη συσκευή απαιτείται να σχεδιάσουμε

## 1.3 Περιγραφή σε Συμπεριφερικό,Δομικό και Φυσικό Επίπεδο

Στη συμπεριφερική θεώρηση μας ενδιαφέρει τι ακριβώς κάνει ένα κύκλωμα (πώς συμπεριφέρεται δηλαδή). Για να το θέσουμε διαφορετικά, το σχέδιο θεωρείται ώς ένα «μαύρο κουτί» το οποίο επεξεργάζεται πληροφορίες με το να παράγαγει συγκεκριμένες εξόδους σε αντίδραση συγκεκριμένων εισόδων. Αυτό που μας απασχολεί είναι η εξάρτηση των εξόδων από τις παρούσες και τις παρελθούσες εισόδους όπως επίσης και θέματα χρονικής σχέσης μεταξύ αυτών και του ρολογιού.

Στη δομική θεώρηση, τα ηλεκτρονικά κυκλώματα μελετώνται από τη σκοπιά της συνδεσιμότητας, δηλαδή από ποια δομικά μπλοκ απότελείται το κύκλωμα και πώς αυτά συνδέονται μεταξύ τους. Με δεδομένη μια συμπεριφερική περιγραφή ειναι δυνατόν να σκεφτούμε περισσότερα του ενός κυκλώματα για να την υλοποιήσουμε. Αυτά θα διαφέρουν μεταξύ τους ως προς την πολυπλοκότητα, την απόδοση, την ενεργειακή κατανάλωση και σε επιμέρους χαρακτηριστικά όπως την τεχνολογία κατασκευής, τη λίστα των μερών που χρειάζονται κλπ.

Τέλος, για τη φυσική θεώρηση, σημειώνουμε ότι αφορά στο πώς τα επιμέρους τμήματα και καλώδια διατάσσονται στο χώρο που διατίθεται (είται είναι κουτί, πλακέτα ή τσιπ ημιαγωγών). Και πάλι σημειώνεται ότι δεν υπάρχει 1-1 προς σχέση μεταξύ της συμπεριφερική περιγραφής και της φυσικής διάταξης

## 1.4 Περιγραφή στο Επίπεδο Καταχωρητών (RTL)

Τα μοντέλα συνδυαστικής λογικής που βασίζονται στην περιγραφή RTL ( Register Transer Level) προδιαγράφουν τις διαδοχικές πράξεις στα σήματα μια σύγχρονης διάταξης, στην οποία οι υπολογισμοί ξεκινούν στη ενεργή ακμή(θετική ή αρνητική) και έχουν ολοκληρωθεί για να αποθηκευτούν σε έναν καταχωρητή στην επόμενη ακμή. Δηλαδή, σε κάθε ακμή οι καταχωρητές διαβάζουν και αποθηκεύουν τα δεδομένα που δημιουργήθηκαν ως αποτελέσματα στην προηγούμενη ακμή. Παραδείγματα μιας τέτοιου τύπου δραστηριότητας είναι οι λειτουργίες shif,count,clear και load. Τα μοντέλα αυτά έχουν το χαρακτηρισμό RTL ακριβώς γιατί περιγράφουν τη δραστηριότητα των καταχωρητών σε μια σύγχρονη μηχανή. Αποτελούν δε μια ιδιαίτερη κατηγορία περιγραφής στο συμπεριφερικό επίπεδο διότι περιέχουν πληροφορίες για το χρονισμό. Επίσης, οι περισσότεροι κατασκευαστές λογισμικού για σύνθεση υπόσχονται (αν και κάτι τέτοιο δεν είναιθεμελιωμένο επιστημονικά με τυπικό τρόπο) ότι όλες οι περιγραφές σε αυτό το επίπεδο μπορεί να συντεθούν επιτυχώς.

**Ορισμός 3** *Ως αναπαράσταση ενός συστήματος στο επίπεδο RTL θεωρούμε την περιγραφή η οποία καθορίζει τα ακόλουθα υποσυστήματα:*

1. *Το σύνολο καταχωρητών του συστήματος*
2. *Τις λειτουργίες που μπορούν να εκτελεστούν στα δεδομένα που είαι αποθηκευμένα σε αυτούς τους καταχωρητές*
3. *Τον έλεγχο που επιβλέπει τις διαδικασίες αυτές στο σύστημα*

## 1.5 ASM-ASMD charts

Το διάγραμμα ASM αντιστοιχεί στο συμβατικό flowchart ενός αλγορίμου αλλά ερμηνεύεται διαφορετικά. Το πρώτο μεν ένα περιγράφει διαδοχικά βήματα και μονοπάτια σειριακά χωρίς να λαμβάνει υπόψιν του το χρονισμό ενώ το άλλο περιγράφει την ακολουθία των γεγογνότων (τη σειρά τους στο χρόνο) όπως επίσης και τις σχέσεις χρονισμού και τα γεγονότα που συμβαίνουν κατά τη μετάβαση.Τα βασικά του στοιχεία είναι το κουτί κατάστασης, το κουτί απόφασης και το κουτί υπό συνθήκη.

Ένα μπλοκ ASM (αλγοριθμικής μηχανής καταστάσεων) είναι μια δομή που αποτελείται από ένα πίνακα καταστάσεων και όλους τους πίνακες αποφάσης και συνθηκών που συνδέονται στο μονοπάτι εξόδου. Διαθέτει μία είσοδο/σημείο εκκίνησης και οποινδήποτε αριθμό σημείων εξόδου.

Τα διαγραμματα αλγοριθμικών μηχανών καταστάσεων και διαδρόμου δεδομένων αναπτύχθηκαν για να διασαφηνίζουν την πληροφορία που απεικονίζεται στα ASM charts και για να παρέχουν ένα αποτελεσμάτικό εργαλείο για τη σχεδίαση μίας δεδομένης μονάδας διαδρόμου δεδομένων. Οι βασικές διαφορές του ASMD διαγράμματος από το ASM: (1) Το ASMD διάγραμμα δεν αριθμεί τις λειτουργίες μέσα σε ένα πίνακα καταστάσεων ,(2) οι ακμές του ASMD διαγράμματος επεξηγούνται με λειτουργίες καταχωρητών που είναι ταυτόχρονονος με τις μεταβάσεις που υποδεικνύονται σε κάθε ακμή και (3) ένα ASMD διάγραμμα

περιλαμβάνει κουτιά υπό συνθήκη που ταυτοποιούν τα σήματα τα οποία ελέγοχυν τις λειτουργίες των καταχωρητών

## 1.6 Λογισμικό Ανοικτού Κώδικα-Free and Open Source Software

Από τη σκοπικά της παραγωγής και ανάπτυξης το Λογισμικό Ανοικτού Κώδικα είναι μια φιλοσοφία ή μεθοδολογία που προωθεί την ελεύθερη διανομή προγραμμάτων και την πρόσβαση στις λεπτομέρειες του σχεδίου και της υλοποιήσης κάθε προγράμματος. Στην προώθηση αυτής της φιλοσοφίας σαφώς έπαιξε σοβαρό ρόλο η ανάπτυξη του διαδικτύου επιτρέποντας πρώτα τη διαθεσιμότητα του ανοικτού κώδικα σε όλους και συνεπακόλουθα την ανάπτυξη ποικίλων μοντέλων ανάπυξης και κοινοτήτων ακόμα και πάνω σε παραλλαγές του ίδιου αρχικού κώδικα. Το πρότυπο της ανάπτυξης κατ'αυτόν τον τρόπο είναι η «αγορά» των αρχαίων λαών όπου κάποιοι πρότειναν ιδέες και ακολουθούσαν προτάσεις και διάλογος με τους υπόλοιπους. Οι βασικοί άξονες αυτής της ελευθερίας που αναφέραμε στην αρχή είναι:

1. Την ελευθερία να τρέξει κανείς το πρόγραμμα για οποιδήποτε σκοπό, ερευνητικό, χόμπυ, εκπαίδευτικό ή ακόμα και εμπορικό
2. Την ελευθερία να γνωρίζει κανείς το πώς λειτουργεί το πρόγραμμα, την ακριβή εσωτερική του δομή και να μπορεί να το τροποιεί για τις ανάγκες του. Εδώ πρέπει να σημειώσουμε ότι η ελευθερία αυτή δεν αφορά μόνο το να γίνει ανοικτός ο κώδικας όσο και αυτός να είναι ευανάγνωστος και να συνοδεύεται από κατάλληλα εγχειρίδια και σχόλια.
3. Την ελευθερία να διανέμει αντίγραφα του προγράμματος στον πλησίον του, οποιοσδήποτε και αν είναι αυτός.
4. Την ελευθερία να τροποποιεί το πρόγραμμα βελτιώνοντας το και να δίνει στο κοινό τη νέα βελτιωμένη έκδοση ώστε να επωφελείται όλη η κοινότητα από αυτό

Μια επισήμανση που πρέπει να κάνουμε είναι ότι, όπως προκύπτει και από τις ελευθερίες που αναφέραμε, το πρόγραμμα δεν είναι κατ'ανάγκη «δωρεάν». Ο διανομέας μπορεί να χρεώνει πράγματα όπως εκπαίδευση, υποστήριξη, συντήρηση ή απλώς hosting. Για αυτό και η ακριβέστερη μετάφραση του όρου free δεν είναι δωρεάν αλλά ελεύθερο (διάκριση στα αγγλικά:free beer-free speech).

Στον αντίποδα του ελεύθερου λογισμικού έχουμε φυσικά το κλειστό λογισμικό ιδιοκτησίας για το οποίο συνοπτικά να αναφέρουμε ότι ακολουθεί το μοντέλο ανάπτυξης του «καθεδρικού». Κλειστές και συχνά μικρές ομάδες με αυστηρή ιεραρχία και προθεσμίες παράγει λογισμικό.

### 1.6.1 Η άδεια GNU

Η άδεια γενικού κοινού GNU (General Public Licence) είναι η πιο διαδεδομένη άδεια για χρήση ανοικτού λογισμικού και γράφτηκε αρχικά από τον Richard Stallman για το GNU Project. Τα αρχικά GNU είναι διατυπωμένα με αναδρομικό τρόπο (ενδεικτικό του χιούμορ αλλά και των γνώσεων που υπάρχουν στην κοινότητα του ανοιχτού λογισμικού). Συγκεκριμένα GNU σημαίνει «GNU is Not Unix». Η άδεια αυτή είναι διατυπωμένη με τέτοιο τρόπο ώστε να ισχύει και για εκδόσεις ενός προγράμματος οι οποίες έχουν τροποποιηθεί ή τους έχουν γίνει προσθήκες.

Ο σκοπός αυτής της άδειας είναι να καταστήσει το λογισμικό (αλλά και οτιδήποτε άλλο πχ ένα βιβλίο ή άλλο έγγραφο) χρήσιμο και ελεύθερο και όχι απλώς δωρεάν: να διασφαλίσει αποτελεσματικά στον καθένα την ελευθερία να το αντιγράφει και να το αναδιανέμει, ως έχει ή με τροποποιήσεις, είτε εμπορικά είτε μη εμπορικά. Δευτερευόντως αυτή η άδεια διασφαλίζει στον συντάκτη και στον εκδότη έναν τρόπο να αναγνωρίζεται η πατρότητα της εργασίας τους, χωρίς να θεωρούνται όμως υπεύθυνοι για τις τροποποιήσεις που γίνονται από άλλους. Αυτή η Άδεια είναι τύπου «copyleft», το οποίο σημαίνει ότι τα παράγωγα έργα του έργου πρέπει και αυτά να είναι ελεύθερα υπό την ίδια έννοια. Μια παραλλαγή της GPL GNU είναι η Lesser General Public Licence η οποία επιτρέπει τη χρήση του ανοικτού λογισμικού σε συνδυασμό με λογισμικό ιδιοκτησίας όπως επίση και με κάπως χαλαρότερες απαιτήσεις copyleft.

### 1.6.2 Άλλες άδειες ανοικτού κώδικα

Επιγραμματικά και ενδεικτικά αναφέρουμε τις ακόλουθες:apache licence, BSD license, Common Public license, fair license, LaTeX Project Public license, Mozilla Public license, Python Software Foundation license

### 1.7 Υλικό Ανοικτού Κώδικα - Open Source Hardware

Εδώ θα πρέπει να κάνουμε μια σύντομη αναφορά στο Υλικό Ανοιχτού Κώδικα (Open Source Hardware). Περισσότερο γνωστό είναι το Λογισμικό Ανοιχτού Κώδικα, για παράδειγμα OpenSolaris, Java, Linux, Apache, Perl, σε ποικίλες εφαρμογές όπως Internet ή αναπαραγωγή Blue Ray. Μικρά κομμάτια από υλικό υπολογιστών πνευματικής ιδιοκτησίας (IP, intellectual property) είναι διαθέσιμα εδώ και πολλά χρόνια σε μορφή ανοιχτού κώδικα, τυπικά ως περιγραφές σε επίπεδο RTL σε γλώσσες όπως η Verilog ή η VHDL. Ώστοσο, μέχρι πρόσφατα, λίγα σε αριθμό σχέδια μεγάλου μεγέθους ήταν διαθέσιμα σε μορφή ανοιχτού κώδικα. Ένα παράδειγμα τα πιο περίπλοκα σχέδια που μπορεί να φανταστεί κάποιος είναι αυτό ενός πλήρους μικρο-επεξεργαστή. Ο πρώτος επεξεργαστής που έγινε διαθέσιμος με αυτόν τον τρόπο είναι ο LEON-32 SPARC. Αντί να σχεδιαστούν εκ νέου άδειες ιδιοκτησίας, χρησιμοποιούνται κατά κανόνα οι ήδη υπάρχουσες από το λογισμικό

**Ορισμός 4** *Ως Υλικό Ανοικτού Κώδικα καλούμε το σύνολο που αποτελείται τόσο από τις συσκευές που έχουν σχεδιαστή με το πνεύμα του Ανοικτού σχεδίου όσο και τα σχέδια που αφορούν αυτές (ηλεκτρομηχανολογικά, PCB, RTL σε κάποια HDL, layouts), τα οποία είναι επίσης ανοικτά*

## 2 Γλώσσες Περιγραφής Υλικού Hardware Description Languages

### 2.1 Ιστορική Αναδρομή

Η σχεδίαση ψηφιακών συστημάτων έχει εξελιχθεί ραγδαία τα τελευταία 25 χρόνια. Τα πρώτα κυκλώματα σχεδιάζονταν «με το χέρι» και αποτελούνταν από λυχνίες κενού και αργότερα από τρανζίστορ. Τα ολοκληρωμένα κυκλώματα (ΟΚ) αποτελούνταν από λογικές πύλες συγκεντρωμένες σε ένα τσιπ. Προδευτικά, η ικανότητα να τοποθετούμε λογικές πύλες στα ΟΚ γινόταν όλο και μεγαλύτερη. Ο βαθμός ολοκλήρωσης (η τάξη μεγέθους του αριθμού των πυλών μέσα σε ένα ΟΚ) εκφραζεται με βάση τους ακόλουθους χαρακτηρισμούς:

| Ονομασία | αριθμός πυλών |
|---|---|
| SSI | <100 |
| MSI | 100 |
| LSI | 1000 |
| VLSI | >10000 |

Πίνακας 1: Χαρακτηρισμός Ολοκληρωμένων Κυκλωμάτων με βάση το βαθμό ολοκλήρωσης

Ενδεικτικά για να περιγράψουμε το φαινόμενο της προδευτικής αύξησης της πολυπλοκότητας αναφέρουμε τον περίφημο νόμο του Moore. Το 1965 ο συνιδρυτής της εταιρείας INTEL, Gordon E. Moore, διατύπωσε τη θέση ότι ο αριθμός των τρανζίστορ που θα μπαίνουν σε ένα τσίπ(και ευθέως ανάλογα οι επιδόσεις)θα διπλασιάζονται κάθε 18 μήνες. Όμως η μεγαλύτερη δυνατότητα «συμπύκνωσης» λογικών πυλών δημιούργησε το πρόβλημα της αποτελεσματικής και γρήγορης σχεδίασης. Οι πρώτες μέθοδοι, εμπνευσμένες από την κλασσική ηλεκτρονική δεν επαρκούσαν για να σχεδιάσουν τόσες πολλές πύλες μέσα σε εύλογο χρονικό διάστημα.Η λύση που υιοθετήθηκε από τη βιομηχανία ήταν η χρήση Γλωσσών Περιγραφής Υλικού (ΓΠΥ).

**Ορισμός 5** *Ως γλώσσες περιγραφής υλικού ονομάζουμε τις γλώσσες υπολογιστών, περιγραφής προδιαγραφών ή μοντελοποίησης για την τυποποιημένη περιγραφή και σχεδίαση ηλεκτρονικών κυκλωμάτων και κυρίως ψηφιακής λογικής καθώς και της προσομοίωσης αυτών. Με άλλα λόγια είναι σε θέση να περιγράψουν τη λειτουργία ενός κυκλώματος, το σχέδιο και την οργάνωσή του καθώς και να τα επιβεβαιώσουν μέσω προσομοίωσης.*

Οι γλώσσσες αυτές αντιμετωπίζονται συχνά και κακώς ως γλώσσες προγραμματισμού ενώ όπως προκύπτει από τον ορισμό τους είναι στην ουσία ένας τρόπος να περιγράψουμε hardware ψηφιακών συστήματων.

Οι δύο γνωστότερες και ισχυρότερες ΓΠΥ είναι η Verilog και η VHDL, οι οποίες εμφανίστηκαν τη δεκαετία του 1980. Συγκεκριμένα, η VHDL ξεκίνησε από το Υπουργείο Άμυνας των ΗΠΑ ως γλώσσα προδιαγραφών για ηλεκτρονικά συστήματα αντικαθιστώντας ιδιαίτερα ογκώδη και πολύπλοκα εγχειρίδια και εκθέσεις. Αντιθέτως, η Verilog ξεκίνησε από το χώρο των σχεδιαστών της βιομηχανίας. Αμφότερες τροποποιήθηκαν και προτυποποιήθηκαν από το διεθνή οργανισμό IEEE σε διάφορα έτη.

## 2.2 VHDL

Η VHDL (VHSIC Hardware Description Language, VHSIC=Very High Speed Intergrated Circuits) είναι μια γλώσσα περιγραφής υλικού που χρησιμοποιείται για να περιγράψει ψηφιακά και μικτών σημάτων συστήματα όπως επίσης και για την προγραμματισμό FPGA και τη σχεδίαση ΟΚ.

Η πρώτη ιδέα ήταν να αντικαταστήσει τα ογκώδη και περίπλοκα εγχειρίδια τα οποία περιείχαν λεπτομέρειες σχετιζόμενες αποκλειστικά με την εκάστοτε εφαρμογή. Αυτό σημαίνει ότι η αρχική της χρήση ήταν για να περιγράφει τις προδιαγραφές αντί για το εγχειρίδιο. Το αμέσως επόμενο βήμα ήταν να τα χρησιποιήσουν την περιγραφή αυτή για προσομοίωση του σχεδίου. Η γλώσσα στην οποία βασίζεται η VHDL είναι η γλώσσα προγραμματισμού Ada, από την οποία δανείζεται τις βασικές ιδέες και το συντακτικό. Οι εκδόσεις τις με την υποστήριξη του οργανισμού IEEE είναι η 1076-1987, ακολουθούμενη από την 1076-1993 η οποία περιλάμβανε βελτιώσεις στο ζήτημα της λογική 9 τιμών, την πρόσθεση του τελεστή xnor, ευκολίες στην ονοματολογία και τους χαρακτήρες που μπορούν να χρησιμοποιηθούν.Μικρές βελτιώσεις έλαβαν χώρα το 2002 και 2003. Το 2006 συγκροτήθηκε από τον IEEE η επιτροπή Accellera η οποία παρέδωσε το 2009 την έκδοση VHDL 4.0 ή αλλιώς VHDL 1076-2008 μετά από μια 3ετία δοκιμής της VHDL 3.0 και έχοντας επιλύσει έναν αριθμό προβλημάτων και υλοποιήσει μια σειρά βελτιώσεων.

### 2.2.1 Βασικά χαρακτηριστικά

Ως γλώσσα η VHDL χαρακτηρίζεται ως strongly typed (δηλαδή οι τύποι δεδομένων περιγράφουν αυστηρά και σωστά τα χειριζόμενα δεδομένα) και παράλληλης εκτέλεσης. Επίσης δεν είναι case sensitive (δεν υπάρχει διάκριση μεταξύ κεφαλαίων και μικρών). Επίσης διαθέτει τη δυνατότητα δεικτοδότησης σε δομές δεδομένων και βασικές δυνατότητες εισόδου εξόδου (κυρίως για τις ανάγκες των προσομοιώσεων).

Η βασική οντότητα για τα προγράμματα VHDL είναι το entity.

**Ορισμός 6** *Αυτό αποτελείται από 2 τμήματα, τα οποία αμφότερα αποκαλούνται σχεδιαστική μονάδα. Το ένα, entity declaration, απεικονίζει και δηλώνει την εξεωτερική διεπαφή του entity. Το άλλο, architecture body, απεικονίζει την περιγραφή της εσωτερικής δομής και την προδιαγραφόμενη συμπεριφορά.*

Εν συντομία η VHDL διαθέτει:

1. μεταβλητές bit, boolean, integer πχ για ports

```
a: in bit;
b: out boolean;
K: out integer rang -2**4 to 2**4-1;
```

ή διανυσματικού τύπου δηλαδή bit_vector ή signed

```
E: in bit_vector (7 downto 0);
M: in signed(0 to 4);
```

2. σήματα, που εκφράζουν τη λογική της δρομολόγησης γεγονότων(γεγονός σημαίνει αλλαγή μιας τιμής σε ένα συγκεκριμένο χρονικό σημείο). Για παράδειγμα ανάθεση τιμής ενός σήματος σε άλλο

```
x<= S_temp;
```

3. λογικές πύλες για τη δομική περιγραφή (structural modeling)

```
Gate2: and port map( Output, Input1, Input2)
```

Σε αυτήν την περιγραφή ουσιαστικά ορίζουμε πύλες οι οποίες εκτελούν πράξεις άλγεβρας Boole.

4. process για την συμπεριφερική περιγραφή (behavioral modeling)

```
[process_label:] process [ (sensitivity_list) ] [is]
                      [ process_declarations]
          begin
               list of sequential statements such as:
                            signal assignments
                            variable assignments
                            case statement
                            exit statement
                            if statement
                            loop statement
                            next statement
                            null statement
                            procedure call
                            wait statement
          end process [process_label];
```

5. Δυνατότητα γραφής RTL κώδικα πχ

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity alu is
port (
      A, B :in unsigned(7 downto 0);
      ADD :in std_logic;
      RES :out unsigned(7 downto 0));
end alu;

architecture imp of alu is
begin
   RES <= A + B when ADD = '1' else
   A - B;
end imp;
```

Τέλος να σημειώσουμε τις δεσμευμένες λέξεις και τις αντίστοιχες δομές που δεν μπορούν να μετατραπούν σε layout (Non-synthesizable Constructs)

```
access, after, alias, assert, bus, disconnect, file, guarded,
inertial, impure, label, linkage, new, on, open, postponed,
pure, reject, report, severity, shared, transport, units,
with.
```

## 2.3 Verilog

Η Verilog είναι μια HDL της οποίας η προγραμματιστική λογική είναι επιρεασμένη από τη C. Χρησιμοποιείται για για 4 επίπεδα σχεδίασης, ξεκινώντας από το χαμηλότερο:το επίπεδο transistor, το επίπεδο πυλών, το επίπεδο RTL και το συμπεριφερικό(behavioral).

### 2.3.1 Βασικά Χαρακτηριστικά

Η Verilog διαθέτει ένα ρεπερτόριο εντολών που τις επιτρέπει να μοντελοποιήσει και να προσομοιώσει ψηφιακά συστήματα. Πρέπει όμως να τονιστεί ότι μόνο ένα υποσύνολο αυτών των εντολών είναι συνθέσιμο από τα προγράμματα σύνθεσης.

Η θεμελιώδης οντότητα στη Verilog είναι το άρθρωμα(module).

**Ορισμός 7** *Ως άρθρωμα ορίζουμε το δομικό στοιχείο των προγραμμάτων Verilog, το οποίο διαθέτει θύρες (εισόδους,εξόδου,εισόδου-εξόδου) οι οποίες δηλώνονται στην αρχή του προγράμματος μαζί με το όνομα και μια εσωτερική δομή η οποία περιγράφεται στο «σώμα» του προγράμματος. Για κάθε άρθρωμα ισχύει η σχεδιαστική ιεραχία, καθώς μπορεί να αποτελείται από άλλα απλούστερα αλλά και το ίδιο να καλείται ως στιγμιότυπο (instantiate)*

Η έννοια του module είναι παράλληλη με την έννοια της συνάρτησης στη C,όπως και η απαίτητηση να υπάρχει τουλάχιστον μία συνάρτηση σε κάθε πρόγραμμα (έστω και αυτή είναι η main() ). Αναλυτικά η δομή του module είναι:

```
module module_name (port_name, port_name, ... );
   module_items
endmodule
```

Με τις δηλώσεις θυρών δηλωμένες με ρητό τρόπο (explicit) ενώ αν θέλουμε να γίνει με έμμεσο τρόπο (implicit)

```
module module_name (.port_name (signal_name ), .port_name (signal_name ), ... );
  module_items
endmodule
```

Τα module_items μπορεί να είναι:

1. δηλώσεις θυρών, δηλώνουμε αν είναι εισόδου,εξόδου ή και τα 2 (input, outpu, inout), το μέγεθός τους (από προδιαγραφή η τιμή αυτή πρέπει να έχει μέγιστη τιμή τουλάχιστον 256 bit) με τη λογική [msb:lsb] δηλαδή:

   ```
   port_direction [port_size] port_name, port_name, ... ;
   ```

2. δηλώσεις τύπων δεδομένων

   ```
   register_type [size] variable_name , variable_name , ... ;
   register_type [size] memory_name [array_size];
   net_type [size] #(delay) net_name , net_name , ... ;
   net_type (drive_strength) [size] #(delay) net_name = continuous_assignment ;
   trireg (capacitance_strength) [size] #(delay, decay_time) net_name, net_name, ... ;
   parameter constant_name = value, constant_name = value, ... ;
   specparam constant_name = value, constant_name = value, ... ;
   event event_name, event_name, ... ;
   ```

   Σε αυτά δηλώνουμε την καθυστέρηση (προαιρετικά)
   το μέγεθος πίνακα
   το μέγεθος των σημάτων
   την ισχύ των σημάτων (προαιρετικά)
   το χρόνο αποκατάστασης

3. κλήσεις άλλων αρθρωμάτων

4. κλήσεις από primitives (στοιχειώδη αρθρώματα,δηλαδή πύλες και transistor)

5. procedural_blocks

```
type_of_block @(sensitivity_list)
  statement_group: group_name;
  local_variable_declarations;
  timing_control procedural_statements;
end_of_statement_group
```

τύπος ρολογιού initial ή always, θα εκτελείται δηλαδή μία φορά ή επαναληπτικά λίστα των μεταβλητών στις οποίες είναι ευαίσθητο το block (εκτελεί τη διαδικασία στην ακμή ή στο μέτωπο)

6. continuous_assignements ρητή(explicit) ανάθεση

```
net_type [size] net_name;
assign #(delay) net_name=expression;
```

σιωπηρή(implicit) ανάθεση

```
net_type (strength) [size] net_name=expression;
```

Αυτές οι αναθέσεις δηλώνονται έξω από τα procedural blocks. Ενεργοποιούνται αυτόματα στο χρόνο μηδέν και λαμβάνουν τιμή παράλληλα με όλα τα υπόλοιπα στιγμιότυπα και μπλοκ.

7. δηλώσεις συναρτήσεων

```
function [size or type] function_name;
    input declarations;
    output declarations;
    procedural_statement or statement_group;
endfunction
```

Οι συναρτήσεις επιστρέφουν μία μόνο τιμή και δεν μπορούν να έχουν χρονικούς ελέγχους

8. δηλώσεις tasks

```
task task_name;
    input, outpout, inout declarations;
    local variable declarations;
    procedural_statement or statement_group;
endtask
```

Τα tasks παρέχουν έναν τρόπο να χωριστεί ο κώδικας σε μικρά κομμάτια και συνήθως περιέχουν δυνατότητες που χρησιμοποιούνται συχνά. Μπρορούν να περιέχουν πληροφορία για το χρονισμό

**Παρατηρήσεις:** Η λειτουργία κάθε αρθρώματος μπορεί να είναι συμπεριφερική, δομική ή συνδυασμός των 2. Τα αρθρώματα δεν μπορούν να «εμφωλεύονται» (nesting), μπορούμε να χρησιποιήσουμε σταθερές(constants) και τα module_items μπορούν να είναι σε οποιαδήποτε σειρά μεταξύ τους αρκεί να προηγούνται οι δηλώσεις μεταβλητών

### 2.3.2 Σχεδίαση RTL στη Verilog

Τα ψηφιακά συστήματα μπορούν να σχεδιαστούν στο RTL επίπεδο σε μια HDL. Στη Verilog μπορούμε να το πετύχουμε αυτό με ένα συνδυασμό συμπεριφερικών δομών και δομών ελέγχου ροής δεδομένων. Το βασικό χαρακτηριστικό της RTL σχεδίασης είναι να περιγραφούν οι λειτουργίες των καταχωρητών μαζί με πληροφορία για το ρολόι (χρονισμό), δηλαδή το τι θα συμβαίνει σε κάθε ακμή ή μέτωπο της τάσης. Η πληροφορία για την καθυστέρηση (στις εντολές assign, πχ #3) θεωρείται λανθασμένη και κατά κανόνα αγνοείται από το πρόγραμμα σύνθεσης. Η καθυστέρηση αυτή έχει νόημα μόνο στην προσομοίωση ενώ μια εκτίμηση για την πραγματική της τιμή στο κύκλωμα θα γίνει στο κύκλωμα μετά την σύνθεση.

**Παραδείγματα περιγραφών RTL**

1. `assign S=A+B;`

   συνεχής ανάθεση τιμής για λειτουργία πρόσθεσης

2. ```
   always @(A,B)
       S=A+B;
   ```

   συνδυαστική λογική για πρόσθεση με ευαισθησία στο μέτωπο της τάσης (ο πιθανός τρόπος σύνθεσης είναι αποκλειστικά και μόνο με συνδυαστική λογική αλλά υπάρχει πιθανότητα το πρόγραμμα σύνθεσης να χρησιμοποιήσει και ένα στοιχείο μνήμης ευαίσθητο στο μέτωπο της τάσης,ένα μανδαλωτή)

3. ```
   always @(negedge clock)
   begin
   RA = RA + RB;
   RD = RA;
   end
   ```

   blocking ανάθεση τιμής μεα ευαισθησία στην αρνητική ακμή (αν θέλαμε ευαισθησία στην θετική θα χρησιμοποιούσαμε το posedge). Αυτή η περιγραφή θα περιέχει οπωσδήποτε στοιχεία ευαίσθητα στην ακμή της τάσης του ρολογιού, δηλαδή registers.

4. ```
   always @ (negedge clock)
   begin
   RA <= RA + RB;
   RD <= RA;
   end
   ```

   non-blocking ανάθεση τιμής με ευαισθησία στην αρνητική ακμή

**Διαφορά μεταξύ Blocking και non-Blocking** Σε πρώτη φάση οι διαδικασίες αυτές διαχωρίζονται από τα σύμβολα τους. Η πρώτη χρησιμοποιεί το σύμβολο (=) και η δεύτερο το σύμβολο (<=) ως τελεστές. Οι blocking εκτελούνται ακολουθιακά με την σειρά που έχουν καταγραφεί στο μπλοκ, όταν εκτελούνται έχουν άμεση συνέπεια στα περιεχόμενα της μνήμης πριν εκτελεστεί η επόμενη διαδικασία. Οι non-Blocking εκτελούνται παράλληλα και ταυτόχρονα (concurrently). Ως εκτούτου καμία τέτοια διαδικασία που προηγείται ή που έπεται στο μπλοκ δεν μπορεί να αλλοιώσει το αποτέλεσμα των υπολοίπων, δεν υπάρχει καμία αλληλεπίδραση μεταξύ τους.

Για παράδειγμα ο παρακάτω κώδικας μας δίνει το πρώτο σχήμα ενώ αν αλλάξουμε το

```
RightShift = RightShift & Strobe;
```

σε

```
RightShift <= RightShift & Strobe;
```

παίρνουμε το δεύτερο.

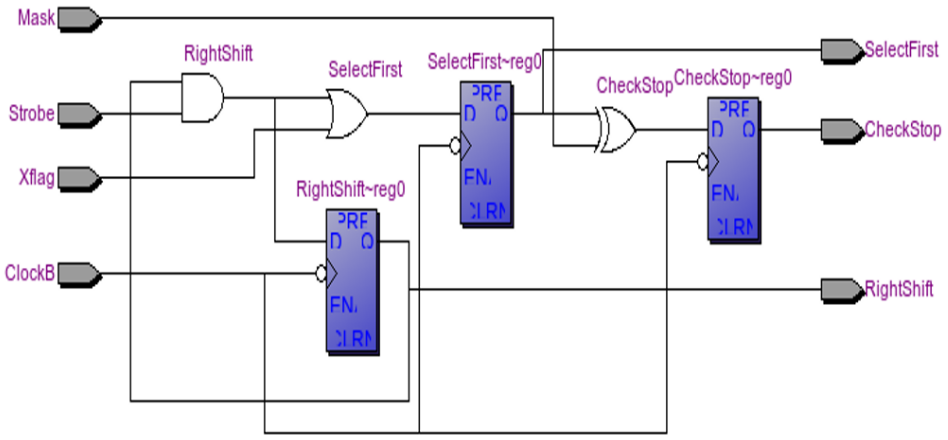
```
module FlabBits (ClockB, Strobe, Xflag, Mask, RightShift,
SelectFirst, CheckStop);


input ClockB, Strobe, Xflag, Mask;
output RightShift, SelectFirst, CheckStop;
reg RightShift, SelectFirst, CheckStop;

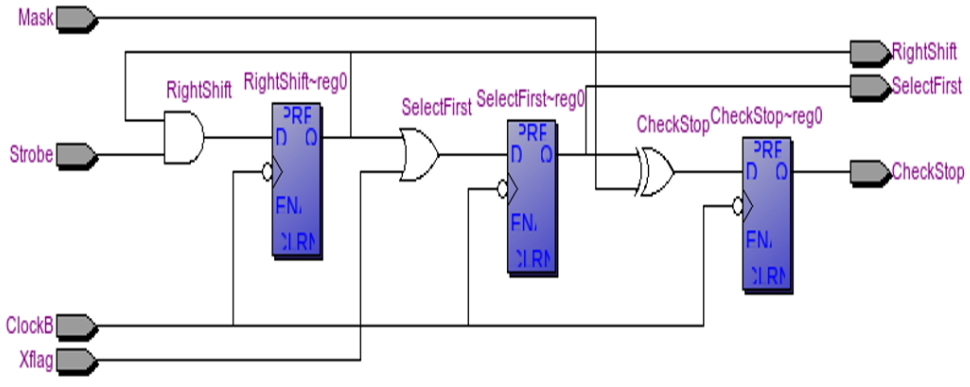
always @(negedge ClockB)
begin
```
```

```
RightShift = RightShift & Strobe;
SelectFirst <= RightShift | Xflag;
CheckStop <= SelectFirst ^ Mask;

end
endmodule
```




(a) Non-Blocking




(b) Blocking

Σχήμα 1: Διαφορετικά κυκλώματα

## 2.4 Σύγκριση των VHDL και VERILOG

| Γλώσσα | Verilog | VHDL |
|---|---|---|
| Βασίζεται-Εμπνέεται | C | Ada,Pascal |
| Κατάληξη αρχείων | .v | .vhd |
| Case Sensitive | Ναι | Όχι |
| Διαχείριση και χρήση Βιβλιοθηκών | Όχι | Ναι |
| Τύποι δεδομένων | Απλοί | Δυνατότητα ορισμού σύνθετων |
| Strongly typed | Όχι | Ναι |
| Περιγραφή χαμηλότερου επιπέδου(διακοπτών) | Ναι | Όχι(μόνο πύλες) |

Πίνακας 2: Σύγκριση βασικών χαρακτηριστικών μεταξύ Verilog και VHDL

Ένα τελευταίο σχόλιο που μπορούμε να κάνουμε είναι ότι η Verilog θεωρείται γενικά ευκολότερη στην εκμάθηση, ειδικά σε μηχανικούς που έχουν ήδη ένα υπόβαθρο σε C.

## 2.5 Tesbenches- Προγράμματα ελέγχου

Η κατασκευή ενός testbench είναι όσο περίπλοκη είναι η διαδικασία περιγραφής του ίδιου του κυκλώματος και με στις μέρες μας γίνται ακόμα πιο σύνθετη. Τυπικά το 60-70% του χρόνου που χρειάζεται για κάθε ASIC δαπανείται για verification-validation-testing (επιβεβαίωση-έλεγχος). Ο σκοπός του προγράμματος ελέγχου (αν και δεν υπάρχει αυστηρός ορισμός αυτού στις προδιαγραφές των γλωσσών) είναι να προσομοιώσει τη λειτουργία του κυκλώματος παρέχοντας διεγέρσεις στις εισόδους του και παροκολουθώντας τις κυματομορφές εξόδου.

# 3 Προγραμματισμός, Σχεδιασμός, Έλεγχος, Ανασκόπηση

Η παρούσα διπλωματική αντιμετωπίστηκε ως ένα ένα προγραμματιστικό/αναπτυξιακό project με μεθόδους και τακτικές παρόμοιες με αυτές που εφαρμόζονται στη βιομηχανία ανάπτυξης λογισμικού. Θα ήταν άτοπο να μην αναφέρουμε έστω και συνοπτικά τη μεθοδολογία αυτή.

Σε πρώτη φάση έρχεται η δήλωση της ανάγκης. Ένα project δεν παράγεται χωρίς σκοπό αλλά με έναν συγκεκριμένο προορισμό, να ικανοποιήσει μια ανάγκη, να λύσει ένα πρόβλημα. Αυτός είναι και ο κατ'ουσίαν σκοπός του μηχανικού, να λύνει προβλήματα. Ο τρόπος που τίθεται το πρόβλημα ή η ανάγκη μπορεί να μην είναι απολύτως πλήρης και τυπικός. Είναι καθηκόν αυτού που προσπαθεί να βρει τη λύση (αλλά και μέρος της ίδιας της λύσης) να διασαφηνίσει το ζητούμενο. Το επόμενο βήμα είναι να «οραματιστεί» τη λύση και να σκεφτεί τα εργαλεία και τα μέσα που μπορεί να χρειαστούν στην πορεία. Για να γίνουμε πιο συγκεκριμένοι, η δήλωση του προβλήματος, αφού την επεξεργαστεί ο μηχανικός, πρέπει να περιέχει τα εξής:Υπόβαθρο του έργου, πόστα και καθήκοντα, χρήστες, ρίσκα και υποθέσεις. Αντίστοια το όραμα της λύσης θα πρέπει να έχει: δήλωση του οράματος, λίστα των τεχνικών χαρακτηριστικών, ορίζοντα παράδοσης και χαρακτηριστικά που ΔΕΝ θα υλοποιηθούν.

Ακολούθως μετά τη δήλωση του προβλήματος πρέπει να καταρτιστεί η δήλωση του έργου. Δηλαδή, η δήλωση όλων των επιμέρους μικρών έργων που θα μοιραστούν στους εμπλεκόμενους, η εκτίμηση των διαθέσιμων πόρων (εργαλείων, εργατοωρών, χρημάτων κτλ), ένας πρόχειρος υπολογισμός του απαιτούμενου χρόνου και προγραμματισμός του έργου, ένα πλάνο για την αντιμετώπιση του ρίσκου και τέλος ένας τρόπος επιβεβαίωσης της ορθής κατασκευής και γενικά του ελέγχου του έργου. Τέλος, κατά τη διάρκεια του έργου μπορεί να χρειαστεί να γίνουν διορθώσεις, υποχωρήσεις ή αλλαγές κατεύθυνσης ανάλογα με τα πορίσματα του προαναφερθέντος ελέγχου. Συμπερασματικά και διαγραμματικά μπορούμε να πούμε ότι η πορεία είναι:

Ζητούμενο => Ανάπτυξη <=> Ανασκόπηση => Προϊόν

# 4 Παρουσίαση των Επεξεργαστών

Στο παρόν κεφάλαιο θα γίνει παρουσίαση των δύο επεξεργαστών. Οι περιγραφές Verilog των επεξεργαστών OpenSPARC και OpenRISC είναι διαθέσιμες, ελεύθερες και ανοικτές μέσω του διαδικτύου σε όλους με την άδεια χρήσης GPL. Οι αντίστοιχοι ιστότοποι είναι οι www.opensparc.net και www.opencores.org

## 4.1 OpenSPARC

OpenSparc είναι το όνομα του πρότζεκτ για την παρουσίαση ενός επεξεργαστή ανοικτού κώδικα που ξεκίνησε το Δεκέμβριο του 2005. Η αρχική συνεισφορά πραγματοποιήθηκε από την Sun Microsystems, και ήταν ο κώδικας Verilog σε επίπεδο RTL για τον επεξεργαστή UltraSPARC T1. Στις 21 Μαρτίου του 2006 η Sun έδωσε στη δημοσιότητα την περιγραφή του κάτω από την άδεια GNU, General Public Licence. Μετά από αυτό στις 11 Δεκεμβρίου του 2007 δημοσιεύτηκε και η περιγραφή του UltraSPARC T2.

Τα ιδιαίτερα χαρακτηριστικά των 2 αυτών επεξαργαστών είναι ότι αποτελούν τους μοναδικούς επεξεργαστές ανοικτού κώδικα στον κόσμο που λειτουργούν στα 64 bit και υποστηρίζουν πολλαπλά νήματα (multi-threaded). Μπορούν να τρέξουν πραγματικού κόσμου γνωστά λειτουργικά (Linux, FreeBSD, OpenBSD) και εμπορικές εφαρμογές. Επίσης, η κοινότητα του ανοιχτού λογισμικού υποστηρίζει θερμά αυτή την προσπάθεια. Τέλος το σχέδιο είναι βιομηχανικού επιπέδου και είναι 14 000 000 γραμμές κώδικα για τον T1 και 41 000 000 για τον T2

### 4.1.1 Ποιοι χρησιμοποιούν τους OpenSPARC

Για την ώρα χρησιμοποιείται σε Πανεπιστήμια για ερευνητικούς σκοπούς. Ο σκοπός που εξυπηρετεί μέχρι τώρα είναι: Ως μέρος εισαγωγικού μαθήματος, ως τρόπος ελέγχου της ρωμαλεότητας των εργαλείων σύνθεσης και CAD. Ακόμα, τους φορτώνουν σε FPGA για να επιτάχυνση της διαδικασίας σχεδίασης και προσομοίωση και τέλος, ως εναρκτήρια σημεία για εταιρείες startups. Ενδεικτικά αναφέρουμε τα εξής:Carnegie Mellon, University of Callifornia-Santa Cruz, University of Michigan, Peking University, Tsinghua University, University of Illinois.

### 4.1.2 OpenSPARC T1/ UltraSPARC T1

Ο επεξεργαστής αυτός, κατά την αναπτυξή του είχε το κωδικό όνομα Niagara, κα είναι μια κεντρική μονάδα επεξεργασίας πολλαπλών νημάτων, πολλαπλών πυρήνων με προορισμό συστήματα σέρβερ χαμηλής κατανάλωσης. Τυπικά οι υλοποιοήσεις του έχουν ταχύτητα 1.4 GHz και κατανάλωση 72 Watt. Διαθέτει εως 8 4-δρόμων πολυνηματικούς πυρήνες που μπορούν να διαχειριστούν εως 32 νήματα ταυτόχρονα. Όλοι οι πυρήνες συνδέονται μέσω διακόπτη crossbar ταχύτητας 134.4 GB\s. Επίσης μοιράζονται υψηλής διαμεταγωγής 12-δρόμων κρυφή μνήμη level-2, συσχέτισης και μεγέθους 3 MB. 4 κανάλια DDR2 με ταχύτητα 23 GB\s. Μια συνήθης υλοποίηση έχει 300 εκατομμύρια transistors και το τσιπ έχει συνολικό μέγεθος 378 $mm^2$

**Βασικά χαρακτηριστικά του T1**

1. Το pipeline του T1 είναι 6 σταδίων:fetch, switch, decode, execute, memory και writeback. Η πρόσβαση στους καταχωρητές γίνεται στο στάδιο decode. Οι σημαίες εμφανίζονται στο στάδιο writeback

2. μονάδα πράξεων κινητής υποδιαστολής, κρίνεται επαρκής για τις περισσότερες εμπορικές εφαρμογές (από τι οποίες λιγότερο από 1% περιέχει πράξεις κινητής υποδιαστολής)

3. κρυφή μνήμη L2 3 MB, 12-way set associative με ψευδο-LRU πολιτική αντικατάστασης(βασίζεται σε τακτική χρησιμοποιημένου bit), με μέγεθος γραμμής 64 bytes.

4. ελεγκτής DRAM

5. γέφυρα I/O

6. J-Bus Interface

7. SSI ROM interface
8. Clock and test Unit
9. EFuse

### 4.1.3 OpenSPARC T2/ UltraSPARC T2

Ο OpenSPARC T2 είναι η συνέχεια του επιτυχημένου πολυνηματικού επεργαστή T1. Το προϊόν αυτό υλοποιεί την τεχνική της Sun για αύξηση της ικανότητας διεκπεραίωσης των servers της (Sun's Throughput Computing Plan). Αυτή η τεχνική χρησιμοποιεί αποτελεσμάτικα τον παραλληλισμό των νημάτων που αντιστοιχούν σε πραγματικά εμπορικά φορτία τα οποία διαφέρουν σημαντικά σε αριθμό νημάτων από τα οικιακά. Έτσι αντί για άλλης μορφής λύσεις (βαθειά διοχεύτευση(pipeline) ή πολυνηματικό μονό πυρήνα), η τεχνική αυτή χρησιμοποιεί πολλαπλούς πυρήνες.

**Βασικά χαρακτηριστικά του T2** Ο T2 αποτελεί μια βελτιωμένη εκδοχή του T1 ως προς τα εξής:

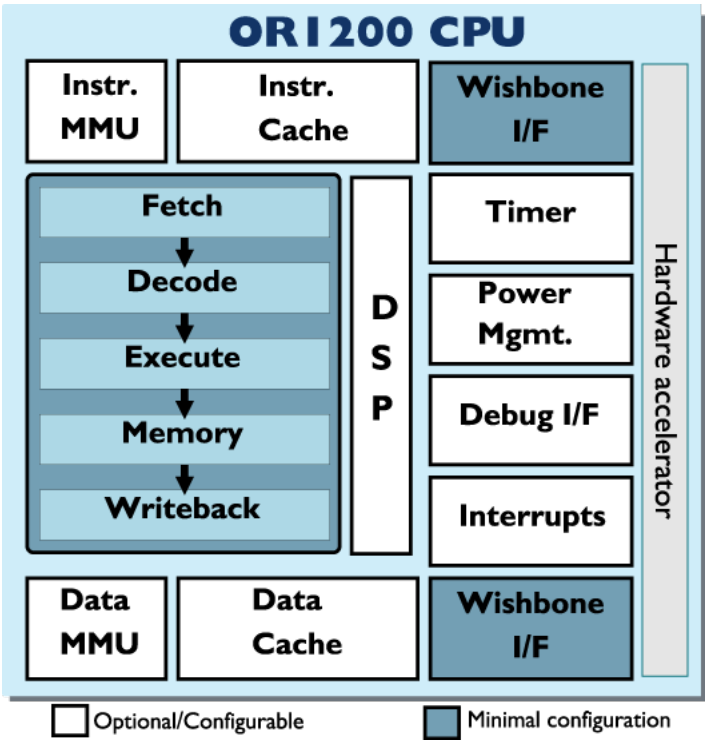
- Αυξημένη ταχύτητα στο 1.6 GHz
- PCI Express 8x θύρα αντί για JBUS
- Ενσωματωμένες Ethernet θύρες
- 4 MB κρυφής μνήμης
- 2 αντί για 1 ALU για πράξεις ακεραίων σε κάθε πυρήνα και 1 FPU σε κάθε πυρήνα αντί 1 και για τους 8
- γεννήτρια τυχαίων αριθμών σε υλικό

## 4.2 Η μονάδα πράξεων κινητής υποδιαστολής του OpenSPARC

Η ΜΠΚΥ της αρχιτεκτονικής OpenSPARC (και στους δύο επεξεργαστές) έχει 32 καταχωρητές μήκους 32 bit, 32 καταχωρητές μήκους 64 bit και 16 καταχωρητές μήκους 128 bit, για μονή,διπλή και τετραπλή ακρίβεια αντίστοιχα. Σύμφωνα με τον κατασκευαστή θεωρείται επαρκές για τις περισσότερες εμπορικές εφαρμογές, καθώς μόλις το 1% των εντολών τους περιλαμβάνουν πράξεις κινητής υποδιαστολής.

Βασικά χαρακτηριστικά της υλοποίησης:

1. Υλοποιεί το σετ εντολών SPARC v9 με την εξαίρεση της εντολής SQRT και των πράξεων τετραπλή ακρίβειας.
2. Εντολές φόρτωσης και αποθήκευσης
3. Δεν υποστηρίζει το σύνολο εντολών VIS
4. Η φυσική τοποθέτηση του register file κινητής υποδιαστολής δεν είναι μέσα στη ΜΠΚΥ. Τη διαχείριση του την αναλαμβάνει η μονάδα FFU. Αυτή η μονάδα επίσης αναλαμβάνει τη διαχείριση μονών/ζυγών διευθύνσεων
5. Η ΜΠΚΥ είναι σύμφωνη ( complies) με το πρότυπο IEEE 754
6. Η ΜΠΚΥ περιλαμβάνει 3 ανεξάρτητα pipeline: FPA: Αθροιστής κινητής υποδιαστολής που αναλαμβάνει προσθέσεις αφαιρέσεις και συγκρίσεις FPM: Πολλαπλασιαστής για πολλαπλασιασμούς FPD: Διαιρέτης Τα οποία μπορούν να περιέχουν παράλληλα εντολές φορτωμένες και να τις εκτελούν.
7. Μόνο μία εντολή ανά κύκλο μπορεί να μπαίνει σε κάθε pipeline. Μόνο μια εντολή σε κάθε κύκλο μπορεί να ολοκληρώνεται σε κάθε pipeline
8. Όλοι οι ιδιαίτεροι τύποι αριθμητικών του προτύπου IEEE-754 υποστηρίζονται (NaN,κανονικοποιημένο, μη κανονικοποιημένο, άπειρο). Ένα μη κανονικοποιημένο δεδομένο δεν ενεργοποιεί trap. Η μονάδα παρέχει πλήρη υποστήριξη για μη κανονικοποιημένα δεδομένα




Σχήμα 2: Υψηλού επιπέδου μπλοκ διάγραμμα του OpenRISC

9. Η κατάσταση λειτουργίας IEEE nonstandard mode αγνοείται

10. Όλες οι ακόλουθες εντολές add, subtract, compare, convert between floating-point formats, convert floating point to integer, convert integer to floating point είναι pipelined και με συγκεκριμένη καθυστέρηση

11. ι ακόλουθες εντολές ΔΕΝ είναι pipelined multiply (σταθερή καθυστέρηση ανεξάρτητη των τιμών των τελεστέων), divide (μεταβλητή καθυστέρηση που εξαρτάται από την τιμή των τελεστεών).

12. Οι εντολές divide έχουν δικό τους datapath και είναι non-blocking

13. Το underflow (μικρότερη απόλυτη τιμή από την ελάχιστη που μπορεί να απεικονίσει το IEEE 754) εντοπίζεται πριν τη στρογγυλοποίηση. Η μη ακριβής συνθήκη ενεργοποιείται όταν οι τιμές εκθέτη και συντελεστή διαφέρουν από τις τιμές που θα είχαν αν δεν ήταν φραγμένες ως προς την ακρίβεια και το εύρος

14. Οι εντολές divide έχουν δικό τους datapath και είναι non-blocking

## 4.3 OpenRISC

Η αρχιτεκτονική OpenRISC περιγράφει τις αρχιτεκτονικές μιας οικογένειας συνθέσιμων επεξεργαστών τύπου RISC, η οποία είναι ανοικτού και ελεύθερου κώδικα. Η αρχιτεκτονική OpenRISC 1000 επίτρεπει ένα ευρύ φάσμα υλοποιήσων για διάφορες εφαρμογές και διαφορετικές σχέσεις κόστους-απόδοσης. Η έμφαση δίνεται στην απόδοση, την απλότητα, τη χαμηλή απαίτηση ισχύος και τη scalability. Οι στόχοι της αρχιτεκτονικής είναι περιβάλλοντα μέσης και υψηλής απόδοσης δικτύων και ενσωματωμένων συστημάτων. Θα καλύψουμε περιληπτικά το σύνολο εντολών, το σύνολο καταχωρητών, τη διαχείριση και τη συνοχή της μνήμης cache, το μοντέλο μνήμης, το μοντέλο εξαιρέσεων, τους τρόπους διευθυνσιοδότητσης, τις συμβάσεις τελεστών και τη δυαδική διεπαφή εφαρμογών.

Σχήμα 3: Λογικό διάγραμμα του OpenRISC

### 4.3.1 Βασικά Χαρακτηριστικά

1. Απολύτως δωρεάν και ανοιχτή αρχιτεκτονική
2. Γραμμικός 32-bit ή 64-bit λογικός χώρος διευθυνσέων με φυσικό χώρο διευθύνσεων σχετικό με την εφαρμογή
3. Απλές και ομοιόμορφες σε μήκος εντολές που χωρίζονται σε 3 βασικές κατηγορίες :
   (a) Βασικό σύνολο εντολών ORBIS32\64
   (b) Επεκτάσεις για διανυσματικές\DSP λειτουργίες ORVDX65
   (c) Επεκτάσεις για πράξεις κινητής υποδιαστολής ORFPX32\64

Η πλατφόρμα αυτή προσφέρει τη δυνατότητα να δημιουργηθεί ένα πλήρες Σύστημα πάνω σε Τσιπ (System on chip) το οποίο είναι ανταγωνιστικό με εμπορικές λύσεις για τους ακόλουθους λόγους:

1. Είναι αποδοτική σε σχέση με το κόστος καθώς δε χρειάζεται να πληρωθούν άδειες αλλά η μείωση του κόστους προκύπτει και από τους επόμενους λόγους
2. Δεν υπάρχουν προβλήματα τέλους χρόνου ζωής καθώς ο πηγαίος κώδικας αναπτύσσεται με σύγχρονη μεθοδολογία και μπορεί εύκολα να μετατραπεί ή να ενσωματωθεί σε άλλες διατάξεις καθιστώντας εύκολη τη συνέχιση της υποστήριξης του προϊόντος
3. Η πλατφόρμα δεν είναι δεμένη με κάποια συγκεκριμένη τεχνολογία και η μετάβαση σε μια άλλη είναι επιτρεπτή, επίσης χωρίς κόστος
4. Απεριόριστης πρόσβαση για χρήση και τροποποιήσεις στον κώδικα, άρα μπορούμε να είμαστε ευέλικτοι και να επαναχρησιμποιούμε τον κώδικα
5. Οι χρόνοι μετάβασης στην αγορά είναι μικρότεροι λόγω της πλήρους διαθεσιμότητας του πηγαίου κώδικα αλλά της ενεργής κοινότητας χρηστών.
6. Η πλήρης διαθεσιμότητα του κώδικα βοηθάει επίσης το σχεδιαστή να αντιληφθεί όλες τις λειτουργίες του σχεδίου ώστε να σκεφτεί λύσεις και να βελτιώσει τη λειτουγικότητα.
7. Οι διαθέσιμες λύσεις για αποσφαλμάτωση και επιβεβαίωση είναι πολύ ισχυρές μέσω της χρήσης του εργαλείου Verilator.

### 4.3.2 Ποιοι τον χρησιμοποιούν

Αρκετοί οργανισμοί έχουν αναπτύξει παράγωγες αρχικτονικές από την αρχιτεκτονική OpenRISC 1000, μεταξύ αυτό το ORC32-1208 της ORSoC και τα BA12, BA14 και BA22 της Beyond Semiconductor.Η Dynalith Systems παρέχει τη πλακέτα iNCITE FPGA, η οποία μπορεί να τρέξει τον OpenRISC 1000 και BA12.Η Flextronics International και η Jennic Limited κατασκεύασαν τον OpenRISC ως μέρος ενός ASIC. Η Samsung χρησιμοποιεί το OpenRISC 1000 στα DTV system-on-chips (SDP83 B-Series, SDP92 C-Series, SDP1001/SDP1002 D-Series, SDP1103/SDP1106 E-Serie). Η Cadence Design Systems άρχισε να τον χρησιμοποιεί ως σχέδιο αναφοράς για να παρουσιάσει design flows. Επίσης, ο TechEdSat της NASA είναι βασισμένος σε OpenRISC και λειτουγεί με linux.

Από την πλευρά της ακαδημαϊκής και μη εμπορικής χρήσης εν γένει, μπορούμε να αναφέρουμε την ομάδα του καθηγητή Stefan Wallentowitz στο Technische Universität München η οποία τον χρησιμοποιεί για να μελετήσει πολυεπεξεργαστικές αρχιτεκτονικές (multicore architectures).

### 4.3.3 Στοιχεία Αρχιτεκτονικής

Τα βασικά μπλοκ αρχιτεκτονικής του πυρήνα OR1200:

1. κεντρικό μπλοκ κεντρική μονάδας,μοναδά πράξεων κινητής υποδιαστολής ή μονάδα ΨΕΣ(ψηφιακής επεξεργασίας σήματος)
2. Κρυφή μνήμη δεδομένων απευθείας απεικόνισης (direct mapped data cache)
3. Κρυφή μνήμη εντολών απευθείας απεικόνισης(direct mapped instruction cache)
4. Χειριστής διακοπών
5. Μονάδα διαχείρισης ισχύος
6. Μονάδα διαχείρισης μνήμης

**CPU-FPU-DSP**
Το κεντρικό τμήμα του επεξεργαστή or1200 RISC.

**Μονάδα εντολών- Instruction Unit**
Η μονάδα εντολών υλοποιεί τη βασική σωλήνωση εντολών, φέρνει τις εντολές από το υποσύστημα της μνήμης και τις αποδίδει στις αντίστοιχες μόνάδες, διαθέτει ιστορικό καταστάσεων για να εξασφαλίζει ακριβές μοντέλο εξαιρέσεων και ότι οι εντολές εκτελούνται με τη σωστή σειρά. Επιτρέπει άλματα με ή χωρίς συνθήκη.

**Load Store Unit-Μονάδα φόρτωσης αποθήκευσης**
Η μονάδα αυτή αναλαμβάνει τη μεταφορά δεδομένων(τελεστέων, δεδομένων, εντολών, διευθύνσεων προορισμού) μεταξύ των καταχωρητών γενικού σκοπού και του εσωτερικού διαύλου της CPU. Η λειτουργία του είναι υλοποιημένη στο υλικό, διαθετεί buffer διευθύνσεων εισόδου, επιτρέπει τη λειτουργία σωλήνωσης και είναι ευθυγραμμισμένο με τη μνήμη για γρήγορη πρόσβαση

**Σετ Καταχωρητών:**

1. 32 ή 16 καταχωρητές γενικού σκοπού ανάλογα με τη χρήση και το περιβάλλον υλοποίησης. Αυτοί οι καταχωρήτες έχουν τις ονομασίες R0-R31 και έχουν μήκος 64 32 bit ανάλογα την υλοποίηση. Μπορούν να αποθηκεύουν δεδομένα ακεραίων, αριθμών κινητής υποδιαστολής, διανύσματα, ή δείκτες μνήμης. Οι καταχωρητές αυτοί μπορούν να αποτελέσουν και πηγή και προορισμό των εντολών ORBIS, ORVDX και ORFPX. Ο R0 είναι σταθερά 0 και συνίσταται (αν και δεν είναι συνδεδεμένος μόνιμα στο λογικό μηδέν)
2. Καταχωρητές ειδικού σκοπού που ορίζονται για κάθε μονάδα ξεχωριστά και είναι προσβάσιμοι μέσω των εντολών l.mtspr/lmfspr.

**Integer Execution Pipeline- Σωλήνωση εκτέλεσης πράξεων με ακέραιους**
Εκτελεί: αριθμητικές πράξεις, πράξεις σύγκρισης, λογικές πράξεις, εντολές περιστροφής και κύλισης, τις περισσότερες σε έναν μόνο κύκλο.

**MAC unit**
Εκτελεί πράξεις επεεργασίας σημάτων με ρυθμό 32x32 και συσσωρευτή 48-bit, με πλήρη σωλήνωση

**Floating Point Unit- Μονάδα πράξεων κινητής υποιαστολής**
Βασίζεται σε προϋπάρχονται σχέδια του OpenCores.org και υλοποιεί τις εντολές της κατηγορίας ORFPX32

**System Unit-Μονάδα συστήματος**
Συνδέει όλα τα υπόλοιπα σήματα που δεν είναι συνδεδεμένα με εντολές και διαπροσωπείες δεδομένων. Επίσης περιέχει όλους τους καταχωρητές ειδικού σκοπού

**Εξαιρέσεις**

- Εξωτερική εντολή διακοπής
- Συγκεκριμένες συνθήκες πρόσβασης της μνήμης
- Κλήση συστήματος
- Εσωτερικά σφάλματα, όπως η προσπάθεια να εκτελεστεί εντολή που δεν έχει υλοποιηθεί
- εσωτερικές εξαίρεσεις,όπως breakpoints
- αριθμητική υπερχείλιση

**Data Cache**

| | Direct mapped |
|---|---|
| 16B/line, 256 lines, 1 way | 4KB |
| 16B/line, 512 lines, 1 way | 8KB (default) |
| 16B/line, 1024 lines, 1 way | 16KB |
| 32B/line, 1024 lines, 1 way | 32KB |

Πίνακας 3: Διαφορετικές εκδοχές διάταξης της κρυφής μνήμης δεδομένων

Σημειώνεται ότι

- Η κρυφή μνήμη δεδομένων είναι διαχωρισμένη από την κρυφή μνήμη εντολών (αρχιτεκτονική Harvard)
- ο αλγόριθμος αντικατάστασης είναι LRU
- write through ή write back
- φυσική διευθυνσιοδότηση του cache directory
- ολόκληρη η μνήμη μπορεί να απεργοποιηθεί, να ακυρωθούν γραμμές, να αδειάσει το περιεχόμενό της ή να αντιγραφεί στην κύρια μνήμη μέσω αλλαγής τιμών στους ειδικού σκοπού καταχωρητές

**Κρυφή μνήμη εντολών-Instruction Cache**

Αντίστοιχη λειτουργία με της κρυφής μνήμης δεδομένων

| | Direct mapped |
|---|---|
| 16B/line, 32 lines, 1way | 512KB |
| 16B/line, 256 lines, 1 way | 4KB |
| 16B/line, 512 lines, 1 way | 8KB (default) |
| 32B/line, 1024 lines, 1 way | 16KB |
| 32B/line, 1024 lines, 1 way | 32KB |

Πίνακας 4: Διαφορετικές εκδοχές διάταξης της κρυφής μνήμης εντολών

**Data MMU-Μονάδα διαχείρισης μνήμης δεδομένων**

Ο OR1200 υλοποιεί ένα σχέδιο διαχείρισης της εικονικής μνήμης που παρέρχει προστασία και αποτελεσματική μετάφραση διευθύνσεων στη φυσική μνήμη.

**Tick Timer**

Ο OR1200 διαθέτει τη δυνατότητα Tick Timer. Βασικά αυτό είναι ένας χρονομετρητής που συγχρονίζεται με ρολόι RISC και χρησιμοποιείται από το λειτουργικό σύστημα για να μετρήσει με ακρίβεια χρονικά διαστήματα και να δρομολογήσει διεργασίες του συστήματος

- Μέγιστος αριθμός μετρήσεων 2 στην 32 κύκλοι ρολογιού
- Μέγιστη χρονική περίδος 2 στην 28 κύκλοι μεταξύ διακοπών
- Δυνατότητα να μετατραπεί σε μάσκα η διακοπή του Tick Timer
- Ο χρονομετρητής μπορεί να τρέξει μία φορά ή διαρκώς με δυνατότητα επανεκκίνησης
- Λειτουργεί με ανεξάρτητη πηγή ρολογιού ώστε να μπορεί να υλοποιηθεί διαχείριση ισχύος τύπου doze (ύπνου)

**Υποστήριξη Διαχείρισης Ισχύος**

Για τη βελτιστοποίηση της κατανάλωσης ισχύος ο OR1200 παρέχει καταστάσεις λειτουργίας χαμηλής ισχύος οι οποίες μπορούν να χρησιμοποιηθούν για να ενεργοποιούν και να απενεργοποιούν δυναμικά συγκεκριμένα εσωτερικά αρθρώματα. Συγκεκριμένα, διαθέτει τα εξής βασικά χαρακτηριστικά για να μειώσει την κατανάλωση: Κατάσταση λειτουγίας Slow και Idle (μείωση της συχνότητας ρολογιού από το λογισμικό), Doze και Sleep, με μειώσεις κατανάλωσης 2χ-10χ, 100χ και 200χ αντίστοιχα

**Debug Unit- Μονάδα αποσφαλμάτωσης**

Η μονάδα αυτή βοηθάει τους προγραμματιστές να αποσφαλματώσουν τα συστήματά τους. Παρέχει υποστήριξει για βασική αποσφαλμάτωση και όχι προωθημένα χαρακτηριστικά όπως breakpoints/watchpoints ή καταχωρητές ελέγχου ροής προγράμματος. Η λειτουργία των Breakpoints/Watchpoints ενεργοποιείται από τους καταχωρητές αποσφαλμάτωσης.

**Clocks and Reset- Ρολόι και Επαναφορά**

Ο επεξεργαστής διαθέτει είσοδο ρολογιού για κάθε εντολή και δεδομένα από τη διάταξη λογικής του διαύλου Wishbone και για για τον πυρήνα της CPU. Το clk_cpu χρονίζει οτιδήποτε μέσα στο δίαυλο. Τα δεδομένα του Wishbone χρονίζονται με το dwb_clk_i και οι εντολές με το iwb_clk_i. Η δυνατότητα Reset είναι ασύγχρονη μέσω του σήματος rst, το οποίο όταν τεθεί σε υψηλό δυναμικό αμέσως επαναφέρει όλα τα φλιπ φλοπς.

**Οι δίαυλοι WISHBONE**

Αυτές οι δύο διατάξεις του πυρήνα OR1200 προσφέρουν συνδεσιμότητα με εξωτερικά περιφεριακά και το υποσύστημα εξωτερικής μνήμης. Η συμβατότητα είναι με τον τύπο: «WISHBONE SoC Interconnection specifications Rev. B3». Για την ώρα η υλοποιήση υποστηρίζει μόνο 32-bit δεδομένα όχι άλλα μήκη.

### 4.3.4 Ρεπερτόριο Εντολών

Έχουμε 5 υποσύνολα εντολών:

- ORBIS32 Βασικό σύνολο εντολών στα 32 bit που μπορεί να λειτουργήσει με δεδομένα 32,16 και 8 bit.
- ORBIS64 Αντίστοιχο σύνολο εντολών στα 64 bit
- ORFPX32 Σύνολο εντολών για πράξεις κινητής υποδιαστολής συμβατό με IEEE-754 στα 32 bit
- ORPFPX64 Αντίστοιχο σύνολο εντολών στα 64 bit
- ORVDX64 Σύνολο εντολών για εφαρμογές διανυσμάτων/επεξεργασίας σήματος

| 31..26 | 25..0 |
|---|---|
| 6 bits | 26 bits |
| Opcode | Immediate |

Πίνακας 5: Φορμάτ των άμεσων εντολών

| 31..26 | 25..21 | 20..16 | 15..11 | 10..0 |
|---|---|---|---|---|
| 6-bits | 5-bits | 5-bits | 5-bits | 11-bits |
| Opcode | rD | rA | rB | Opcode |

Πίνακας 6: Φορμάτ των εντολών από καταχωρητή σε καταχωρητή

| Opc. | Mnemonic | Opc. | Fmt. | Λειτουργία |
|---|---|---|---|---|
| 0x00 | l.j | - | I | PC <- exts(Immediate < < 2) + PC |
| 0x01 | l.jal | - | I | PC <- exts(Immediate < < 2) + PC; LR <- PC + 8 |
| 0x03 | l.bnf | - | I | PC <- SR[F] ? PC + 4 : exts(Immediate < < 2) + PC |
| 0x04 | l.bf | - | I | PC <- SR[F] ? exts(Immediate < < 2) + PC : PC + 4 |
| 0x05 | l.nop | - | I | |
| 0x06 | l.movhi | - | RI | rD <- Immediate « 16 |
| 0x08 | l.sys | - | I | PC <- system-call exception |
| 0x09 | l.rfe | - | I | PC <- EPCR; SR <- ES |
| 0x11 | l.jr | - | R | PC <- rB |
| 0x12 | l.jalr | - | R | PC <- rB; LR <- PC + 8 |
| 0x21 | l.lwz | - | RI | rD <- [rA + Immediate][31:0] |
| 0x22 | l.lws | - | RI | rD <- [rA + Immediate][31:0] |
| 0x23 | l.lbz | - | RI | rD <- extz([rA + Immediate][7:0]) |
| 0x24 | l.lbs | - | RI | rD <- exts([rA + Immediate][7:0]) |
| 0x25 | l.lhz | - | RI | rD <- extz([rA + Immediate][15:0]) |
| 0x26 | l.lhs | - | RI | rD <- exts([rA + Immediate][15:0]) |

Πίνακας 7: Συνοπτικός πίνακας εντολών ORBIS32

| | | | | |
|---|---|---|---|---|
| 0x27 | l.addi | - | RI | rD <- rA + Immediate |
| 0x28 | l.addic | - | RI | rD <- rA + Immediate + SR[CY] |
| 0x29 | l.andi | - | RI | rD <- rA AND Immediate |
| 0x2A | l.ori | - | RI | rD <- rA OR Immediate |
| 0x2B | l.xori | - | RI | rD <- rA XOR Immediate |
| 0x2C | l.muli | - | RI | rD <- rA * Immediate |
| 0x2D | l.mfspr | - | RI | rD <- SPR[rA OR Immediate] |
| 0x2E | l.slli | 0x0 | R | rD <- rA < < Immediate |
| 0x2E | l.srli | 0x1 | R | rD <- rA » Immediate |
| 0x2E | l.srai | 0x2 | R | rD <- rA »> Immediate |
| 0x30 | l.mtspr | - | RI2 | SPR[rA OR Immediate] <- rB |
| 0x35 | l.sw | - | RI2 | rA + Immediate][31:0] <- rB |
| 0x36 | l.sb | - | RI2 | [rA + Immediate][7:0] <- rB |
| 0x37 | l.sh | - | RI2 | [rA + Immediate][15:0] <- rB |
| 0x38 | l.add | 0x0 | R | rD <- rA + rB |
| 0x38 | l.addc | 0x1 | R | rD <- rA + rB + SR[CY] |
| 0x38 | l.sub | 0x2 | R | rD <- rA - rB |
| 0x38 | l.and | 0x3 | R | rD <- rA AND rB |
| 0x38 | l.or | 0x4 | R | rD <- rA OR rB |
| 0x38 | l.xor | 0x5 | R | rD <- rA XOR rB |
| 0x38 | l.mul | 0x6 | R | rD <- rA * rB |
| 0x38 | l.sll | 0x08 | R | rD <- rA < < rB |
| 0x38 | lsrl | 0x18 | R | rD <- rA » rB |
| 0x38 | l.sra | 0x28 | R | rD <- rA »> rB |
| 0x38 | l.mulu | 0xb | R | rD <- rA * rB |
| 0x39 | l.sfeq | 0x0 | RSF | SR[F] <- rA == rB ? 1 : 0 |
| 0x39 | l.sfne | 0x1 | RSF | SR[F] <- rA != rB ? 1 : 0 |
| 0x39 | l.sfgtu | 0x2 | RSF | SR[F] <- rA > rB ? 1 : 0 |
| 0x39 | l.sfgeu | 0x3 | RSF | SR[F] <- rA >= rB ? 1 : 0 |
| 0x39 | l.sfltu | 0x4 | RSF | SR[F] <- rA < rB ? 1 : 0 |
| 0x39 | l.sflue | 0x5 | RSF | SR[F] <- rA =< rB ? 1 : 0 |
| 0x39 | l.sfgts | 0xa | RSF | SR[F] <- rA > rB ? 1 : 0 |
| 0x39 | l.sfges | 0xb | RSF | SR[F] <- rA >= rB ? 1 : 0 |
| 0x39 | l.sflts | 0xc | RSF | SR[F] <- rA > rB ? 1 : 0 |
| 0x39 | l.sfles | 0xd | RSF | SR[F] <- rA =< rB ? 1 : 0 |

Πίνακας 8: Συνοπτικός πίνακας εντολών ORBIS32 (συνέχεια)

# 5 Αναλυτική περιγραφή των εργαλείων software που χρησιμοποιήθηκαν

## 5.1 Icarus Verilog

### 5.1.1 Γενικές πληροφορίες

Η Icarus Verilog (iverilog) είναι ένα εργαλείο(compiler-μεταγλωτιστής) για τη σύνθεση και την προσομοίωση προγραμμάτων Verilog. Παρέχει πλήρη υποστήριξη των προδιαγραφών για τη γλώσσα Verilog που έχουν τεθεί από το διεθνή οργανισμό IEEE τα έτη 1995, 2001 και 2005 (IEEE-1364) και μερική υποστήριξη της προδιαγραφής Systemverilog (2009). Η icarus verilog υποστηρίζεται από τα περισσότερα λειτουργικά περιβάλλοντα: MS Windows, Mac OS, Solaris, FreeBSD και πολλαπλές διανομές linux. Για την παρούσα διπλωματική η Icarus Verilog εγκαταστάθηκε σε Windows XP και Ubuntu 10.04 LTS Linux. Η πιο πρόσφατη έκδοση είναι 0.9.5 (stable) με ημερομηνία έκδοσης 1η Νοεμβρίου 2011. Ο κύριος συγγραφέας πηγαίου κώδικα καθώς και συντηρητής του είναι ο προγραμματιστής Stephen Williams, από την αρχή της το 1998

μέχρι και σήμερα. Το λογότυπο της γλώσσας απεικονίζει το μυθολογικό Έλληνα ήρωα Ίκαρο, γιο του Δαίδαλου, και φιλοτεχνήθηκε από το θείο του Williams ο οποίος είναι γραφίστας. Όπως κάθε άλλο πρότζεκτ του FOSS η Icarus Verilog είναι ένα έργο σε διαρκή εξέλιξη

### 5.1.2 Τεχνικές Λεπτομέρειες

Η Icarus Verilog είναι γραμμένη κατά κύριο λόγο σε C++. Η εκτέλεση της γίνεται μέσω του τερματικού (terminal) στα linux συστήματα ή του αντίστοίχου command prompt (γραμμή εργαλείων) σε Windows.

## 5.2 Gtkwave

Το GTKWave είναι ένα εργαλείο ανάλυσης για την εκτέλεση αποσφαλμάτωσης (debugging) σε μοντέλα προσομοίωσης Verilog και VHDL. Με την εξαίρεση της διαδραστικής (interactive) παρακολούθησης VCD, δεν προδιαγράφεται να τρέχει διαδραστικά με την προσομοίωση αλλά αντιθέτως βασίζεται σε μια προσέγγιση "νεκροψίας" μέσα από τη χρήση dumpfiles.

### 5.2.1 Υποστηριζόμενα φορμάτ

Οι τύποι dumpfile που υποστηρίζονται είναι:

VCD: Value Change Dump Κλασσικός τύπος αρχείου εξόδου κυματομορφών (dumpfile) της βιομηχανίας (industry standard) που γεννάται από τους περισσότερους προσομοιωτές Verilog και προτυποιείται από τον το IEEE-1364. Είναι ο πιο αργός αργός τύπος και απαιτεί την περισσότερη μνήμη αλλά είναι «πανταχού παρών» και σχεδόν όλα τα εργαλεία το υποστηρίζουν.

LXT: InterLaced eXtensible Trace. Αυτό είναι ένα αριστοποιημένο φορμάτ που χρησιμοποιεί πίσω-συνδεδεμένους δείκτης και μεταβολές τιμών. Η επεξεργασία του είναι ταχύτερη από αυτή του VCD. Η Icarus Verilog υποστηρίζει εγγενώς το φορμάτ αυτό

LXT2: Version 2. Εκδοχή του LXT βασισμένη σε μπλοκς και καλύτερη συμπίεση και ταχύτητα πρόσβασης από το LXT. Επιτρέπει τυχαία προσπέλαση σε επίπεδο μπλοκ και κατ' επιλογήν μερική φόρτωση μπλοκς για ταχύτερη λειτουργία. Η Icarus Verilog επίσης υποστηρίζει το φορματ.

VZT: Verilog Zipped Trace. Αποτελεί μια ανάπτυξη του LXT2 καθώς είναι και αυτό βασισμένο σε μπλοκς παρόλα αυτά χρησιμοποιεί διαφορετικούς ευρηστικούς αλγόριθμους για συμπίεση ώστε να πετύχει ακόμα μικρότερα μεγέθη αρχείων. Η ταχύτητα γραφής σε αυτόν τον τύπο αρχείου είναι η πιο αργή από όλους τους τύπους, ωστόσο η ανάγνωση είναι ταχύτητα σε πολυεπεξεργαστικές μηχανές καθώς παρέχει τη δυνατότητα παραλληλισμού.

GHW: GHDL Wave file. Φορμάτ 9 καταστάσεων ("01XZHUWL-") που γράφτηκε για τον προσομοιωτής VHDL, τον GHDL.

AET2: All Events Trace Version 2. Φορμάτ που χρησιμοποιείται από τα εργαλεία EDA της IBM. Το μέγεθος αρχείων είναι πολύ μικρό και η προσπέλαση πολύ γρήγορη. Η υποστήριξη για αυτό καθορίζεται την ώρα της μεταγλώτισσης.

IDX: VCD Recorder Index File. Αυτό το φορμάτ γράφεται από το ίδιο το GTKWave όταν του δίνεται η εντολή να δημιουργήσει αρχεία ταχείας φόρτωσης.

FST: Fast Signal Trace. Διαφορετική εκδοχή του IDX, βασισμένη σε μπλοκς σχεδιασμένη για γρήγορη ακολουθιακή και τυχαία προσπέλαση.

Μετατροπές

Μέσα από το πρόγραμμα είναι εφικτή η μετατροπή VCD αρχείων σε LXT, LXT2, VZT ή FST αρχεία. Επίσης, μετατροπή από LXT2, VZT και FST σε VCD είναι εφικτή. Μετατροπή από LXT σε VCD δεν είναι δυνατή, ωστόσο υπάρχει η δυνατοτήτα αποθήκευσης των ορατών κυματομορφών στο κύριο παράθυρο του GTKWave σε μορφή VCD.

### 5.2.2 Γιατί χρησιμοποιήθηκε;

Το GTKWave αναπτύχθηκε για να εκτελεί διαδικασίες αποσφαλμάτωσης σε μεγάλα SoC(Systems on Chip, Συστήματα πάνω σε τσιπ) και έχει χρησιμοποιηθεί για αυτό το σκοπό ως εκτός σύνδεσης αντικαταστάτης για άλλα παρόμοια εργαλεία. Λειτουργεί ορθά στα 64 bit και είναι έτοιμο για τα μεγαλύτερα σχέδια με δεδομένο ότι τρέχει σε σύστημα με επαρκείς ποσότητες φυσικής μνήμης. Τα φορματ LXT2 και VZT σχεδιάστηκαν ειδικά για να μπορέσουν να ανταπεξέλθουν με τα μεγάλα σχέδια πραγματικών συνθηκών και τα AET2(ειδικά για τους χρήστες των προγραμμάτων της IBM) και το FST έχουν σχεδιαστεί για να αντιμετωπίσουν αποτελεσματικά εξαιρετικά μεγάλα σχέδια.

Για τη Verilog, το GTKWave επιτρέπει στους χρήστες την αποσφαλμάτωση και σε επίπεδο ψηφιακού δικτύου παρέχοντας όψεις πολλαπλών τιμών σημάτων σε διαφορετικές χρονικές περιόδους και επίσης στο RTL επίπεδο μέσα από την ανάδειξη των τιμών των σημάτων για συγκεκριμένες χρονικές στιγμές. Επίσης, παρέχει τη δυνατότητα απεικόνισης του δέντρου ιεραρχίας των αρθρωμάτων καθώς και της αναζήτησης μέσα σε αυτό. Η απεικόνιση δέντρου αυτή είναι ακριβής αναπαράσταση του πραγματικού σχεδίου

## 5.3 Python-MyHDL

Η Python είναι μια ισχυρή γλώσσα προγραμματισμού γρήγορη στην εκμάθηση και αποδοτική στη χρήση. Διαθέτει υψηλού επιπέδου δομές δεδομένων και απλή αλλά αποδοτική προσέγγιση στον αντικειμενοστραφή προγραμματισμό. Βασικό κομμάτι της προγραμματιστικής της φιλοσοφίας (που χιουμοριστικά ονομάζεται "the Zen of Python") είναι η παραγωγή κατά το δυνατόν κομψότερου και πιο ευανάγνωστου κώδικα, καθιστώντας την ιδανική για scripts και rapid development. Να σημειώσουμε επίσης ότι η Python ως εργαλείο ανοιχτού κώδικα (FOSS) υποστηρίζεται πολύ καλά από μια μεγάλη και ένθερμη κοινότητα χρηστών-προγραμματιστών.

Με επιστημονικούς όρους η Python είναι μια διερμηνευόμενη (interpreted), υψηλού επιπέδου γλώσσα με δυναμική σημασιολογία (semantics).

Τα βασικά της χαρακτηριστικά είναι:

- Εύκολη στην εκμάθηση, την αναγνωσιμότητα και τη συντήρηση
- Γρήγορη ανάπτυξη εφαρμογών
- Διερμηνευόμενη
- Πολύ υψηλού επιπέδου δομές δεδομένω
- Ανοιχτού Κώδικα
- Λειτουργεί σε πολυάριθμες πλατφόρμες και περιβάλλοντα (σχεδόν παντού, Linux/Unix, Windows, OS/2, Mac ακόμα και Amiga ή γλώσσες προγραμματισμού Java (Jython) ή .NET)
- Ώριμη (σήμερα βρισκόμαστε στην 3η κύρια έκδοση ενώ η λεγόμενη «άλφα» έκδοση αναπτύχθηκε τη δεκαετία του 1980)
- Αυτόματη διαχείριση μνήμης, (μέτρηση αναφορών-reference counting, η γλώσσσα αντιλαμβάνεται πότε το ίδιο αντικείμενο αναφέρεται πάνω από μία φορά και δεν το αποθηκεύει στη μνήμη αν δε χρειάζεται)
- Από την έκδοση 3 και μετά δεν έχουμε πλέον σφάλματα τμηματοποίησης (segmentation faults)
- Δυναμική τυποποίηση (Dynaming Typing),στην Python δε δηλώνουμε τι τύπος μεαβλητών χρησιμοποιείται

Ένα ακόμα σημείο στο οποίο αξίζει να δοθεί προσοχή είναι η συνάφεια που παρουσιάζει η λογική του αντικειμενοστραφούς προγραμματισμού με τα αφαιρετικά επίπεδα σχεδίασης μιας γλώσσας περιγραφής Hardware. Συγκεκριμένα, η σχέση κλάσης-αντικειμένου (class-object) μπορεί να εκφράζει την ιεραρχία των αρθρωμάτων καθώς και τα σχέση μεταξύ των διαφορετικών επιπέδων σχεδίασης. Επίσης η έννοια του στιγμιότυπου (instance) είναι παρόμοια και στους δύο τρόπους σκέψης. Ένα τρίτο ενδιαφέρον κοινό σημείο είναι η έννοια της κληρονομικότητας (inheritance), όταν μια κλάση αποτελεί εξειδίκευση της άλλης

και έχει όλα τα χαρακτηριστικά που περιγράφουν την κλάση από την οποία κληρονομεί συν κάποιο άλλα ιδιαίτερα που τη διαφοροποιούν.

Η προγραμματιστική λογική και η φιλοσοφία της Python συνοψίζεται από το Zen της Python το οποίο είναι ορατό μέσα από το διαδραστικό περιβάλλον της με την εντολή: import this.
Συγκεκριμένα:

The zen of Python

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

Μετάφραση
(πηγή: Οδηγός Python με παραδείγματα, Δημήτρης Λεβεντέας,TasPython)

Όμορφο είναι καλύτερο από άσχημο.
Άμεσο είναι καλύτερο από έμμεσο.
Απλό είναι καλύτερο από σύνθετο.
Σύνθετο είναι καλύτερο από περίπλοκο.
Επίπεδο είναι καλύτερο από εμφωλευμένο.
Αραιό είναι καλύτερο από πυκνό.
Η αναγνωσιμότητα μετράει.
Οι ειδικές περιπτώσεις δεν είναι αρκετά ειδικές ώστε να σπάνε τους κανόνες.
Ωστόσο η πρακτικότητα υπερτερεί της αγνότητας.
Τα λάθη δεν θα πρέπει ποτέ να αποσιωπούνται.
Εκτός αν αποσιωπούνται ρητά.
΄Οταν αντιμετωπίζεις την αμφιβολία, αρνήσου τον πειρασμό να μαντέψεις.
Θα πρέπει να υπάρχει ένας– και προτιμητέα μόνο ένας –προφανής τρόπος
να το κάνεις.
Αν και αυτός ο τρόπος μπορείς να μην είναι προφανής εκτός αν είσαι Ολλανδός.
Τώρα είναι καλύτερα από ποτέ.
Αν και ποτέ είναι συχνά καλύτερα από ακριβώς τώρα.
Αν η υλοποίηση είναι δύσκολο να εξηγηθεί, τότε είναι κακή ιδέα.
Αν η υλοποίηση είναι εύκολο να εξηγηθεί, τότε ίσως είναι καλή ιδέα.

Αν κάποιος δεν έχει πειστεί ακόμα για τη χρησιμότητα της Python, μπορεί να αναθεωρήσει αν διαβάσει τη λίστα των χρηστών της:

- Google(πχ, Google App Engine)

- Yahoo!
- MIT
- REDHAT και άλλες Linux

Οι τομείς στους οποίους οι αναφερθέντες χρησιμοποίησαν την Python είναι:

- Γρήγορη Προτυποποίηση (Rapid Prototyping)
- Προγραμματισμός στον Παγκόσμιο Ιστό
- Εκπαίδευση
- Εφαρμογές με γραφική διεπαφή
- Επιστήμη
- Προγράμματα Δέσμης Ενεργειών (Scripting)

### 5.3.1 Βασικές λειτουργίες της Python

Συνοψίζοντας την επισκόπηση της Python ας αναφέρουμε εν συντομία κάποιες λειτουργίες και εν γένει τον τρόπο σκέψης:

Στοίχιση και κενά
Στην Python σε αντίθεση με πολλές άλλες γλώσσες προγραμματισμού μεγάλυ βαρύτητα και σημασία στη μεταγλώτισση έχουν τα κενά και οι εσοχές, identation (=ενδοπαρα- γραφοποίηση). Αυτό βοηθάει στην αναγνωσιμότητα του κώδικα. Έτσι για παράδειγμα τα ακόλουθα προγράμματα ΔΕΝ είναι ισοδύναμα:

```
""PROG1 ""
if var1==0:
   print "It is zero"
print "I compared them"
```

```
""PROG2""
if var==0:
   print "It is zero"
   print "I compared them"
```

import
Με την εντολή import μπορούμε να καλέσουμε βιβλιοθήκες εντολών και συναρτήσεων που επιθυμούμε να χρησιμοποιήσουμε στο προγραμμά μας. Τέτοιες είναι, η math για μαθηματικά, η datetime για ημερομηνίες, η urllib για εργασίες στο internet και λοιπά.

έλεγχος ροής και βρόγχοι
if .. elif .. else .., while .. , for .. in range .. όπως στις περισσότερες γλώσσες προγραμματισμού

μεταβλητές
Υποστηρίζονται οι γνωστοί τύποι μεταβλητών, ακέραιοι(απεριόριστου μήκους), κινητής υποδιαστολής, πίνακες, ακολουθίες χαρακτήρων, λίστες κλπ όπως και οι αντίστοιχες πράξεις μεταξύ τους

Συναρτήσεις

```
def function_name():
    statement1
    ...
    statementn
```

Αντικειμενοστραφής προγραμματισμός
Στον αντικειμενοστραφή προγραμματισμό σχεδό όλα είναι αντικείμενα,που περιγράφονται από (1) εσωτερική κατάσταση και (2) δυνατότητα αλληλεπίδρασης με το περιβάλλον,δηλαδή συμπερφιφορά. Οι περιγραφές των αντικειμένων, ονομάζονται classes. Παρέχονται οι δυνατότητες προγραμματισμού με τη βοήθεια

κλάσεων(classes) και αντικειμένων με την επιδίωξη να χρειάζεται όσο το δυνατόν λιγότερο συντακτικό και σημασιολογία. Έννοιες όπως η κληρονομικότητα των κλάσσεων, η αλλαγή των μεθόδων μια πατρικής κλάσσης από τη θυγατρική ή κλήσης της. Δημιουργούνται την ώρα που τρέχει το πρόγραμμα και μπορούν να τροποποιηθούν μετά τη δημιουργία τους (δυναμικά).

### 5.3.2 Πακέτο MyHDL

Ιδιαίτερη έμφαση δόθηκε στη χρήση ενός πακέτου το οποίο δε βρίσκεται στη βασική κύρια βιβλιοθήκη (standard library). Το πακέτο αυτό περιλαμβάνει έτοιμες συναρτήσεις και δεσμευμένες λέξεις ειδικά κατασκευασμένες για σχεδίαση hardware, διατηρώντας την απλότητα και την κομψότητα της Python.

Αναλυτικά,με τη βοήθεια του πακέτου MyHDL ο σχεδιαστής μπορεί να χρησιμοποιήσει μια υψηλού επιπέδου γλώσσα για να προσομοιώσει και και να μοντελοποιήσει τα σχέδιά του. Η βασική ιδέα του πακέτου είναι η χρήση της Python για να μοντελοποιήσει την επαλληλία του hardware. Οι γεννήτορες της MyHDL περιγράφονται ως συναρτήσεις με δυνατότητα συνέχειας και ως τέτοιοι είναι παρόμοιοι με τα always-blocks της Verilog και τις processes της VHDL.

Ένα άρθρωμα (module) μοντελοποιείται ως μια συνάρτηση που επιστρέφει γεννήτορες. Με αυτό τον τρόπο μπορούν να υποστηριχθούν χαρακτηριστικά όπως η πολλαπλή ιεραρχία,η συσχέτιση θυρών, οι πίνακες στιγμιοτύπων και η υπο συνθήκη στιγμιοτυποποίηση. Τέλος, παρέχεται η δυνατότητα κλάσης σημάτων για την επικοινωνία των γεννητόρων, κλάσης για τις πράξεις bit και κλάσης για τους τύπου απαρίθμησης.

Οι γεννήτορες (generators) δημιουργούνται όταν χρησιμοποιούμε την λέξη κλειδί yield αντί της return. Μπορούμε να τις αντιμετωπίσουμε σαν τις κλασι- κές συναρτήσεις με μόνο μια διαφορά. Ένας γεννήτορας, επιστρέφει μια τιμή με την yield.΄Οταν ξανακλειθεί, συνεχίζει από την κατάσταση που ήταν μόλις κλήθηκε η yield, μέχρι να φθάσει ξανά σε κάποιο άλλο yield.΄Ετσι, μπορεί να χρησιμοποιηθεί για να παράγονται δυναμικά τιμές, καταλαμβάνοντας έτσι μικρότερο χώρο στην μνήμη, αφού δεν χρειάζεται να παραχθούν όλες και να επιστραφούν.

### 5.3.3 Reference-Αναφορά

Η myhdl ως βιβλιοθήκη/πακέτο εισάγει έναν αριθμό αντικειμένων, συναρτήσεων και δεσμευμένων λέξεων σε κάθε πρόγραμμα. Αυτά είναι

#### Προσομοίωση και σήματα

1. `class Simulation(arg,< arg ... >)`

   Με αυτή την κλάσση καταασκευάζουμε καινούρια προσομοίωση, τα ορίσματα πρέπει να είναι αντικείμενα κατασκευασμένα με τη βοήθεια της βιβλιοθήκης. Η μέθοδος

   ```
   Simulation.run(<duration>)
   ```

   εκτελεί την προσομοίωση διαρκώς ή μέχρι το καθορισμένο χρονικό διάστημα

2. Συναρτήσεις σχετιζόμενες με τη Simulation: now(), η οποία επιστρέφει τον τρέχοντα χρόνο προσομοίωσης και exception StopSimulation, η οποία σταματάει την προσομοίωση όταν προκύψει μια εξαίρεση

3. `traceSignals(func<, *args ><, **kwargs >)`

   επιτρέπει τη θέαση κυματομορφών. Η func είναι μια συνάρτηση που επιστρέφει instance, η traceSignals() την καλεί και της περνάει τα ορίσματα, με αυτόν τρόπο βρίσκει την ιεραρχία και τα σήματα που επιθυμούμε να παρακολουθήσουμε.

4. `class SignalType`

   η κλάσση αυτή υπάρχει για να ελέγχει αν ένα αντικείμενο είναι Signal

5. `class Signal(<val=None><,delay=0>)`

   με αυτήν την κλάση κατασκευάζουμε ένα καινούριο signal του οποίου την αρχική τιμή και την καθυστέρηση μπορούμε να καθορίσουμε. Ένα αντικείμενο Signal έχει τα attributes (χαρακτηριστικά):
   posedge, ευαισθησία στη θετική ακμη
   negedge, ευσιασθησία στην αρνητική ακμή
   next, επόμενη τιμή (μπορούμε είτε να το διαβάσουμε είτε να το γράψουμε)
   val, παρούσα τιμή (μπορούμε μόνο να το διαβάσουμε)
   min, ελάχιστη τιμή ενός αριθμητικού signal, μονο για ανάγνωση
   max, αντίστοιχα μέγιστη τιμή
   driven, καθορίζει τον τρόπο οδήγησης του σήματος και πιθανώς και τον τρόπο με τον οποίο θα δηλωθεί στη Veriog, τιμές: 'reg', 'wire', True, False
   read, boolean χαρακτηριστικό που δείχνει αν έχουμε διαβάσει ένα Signal

6. `class _SliceSignal (sig, left <,right=None>)`

   δημιουργεί ένα σήμα που είναι δείκτης ή τμήμα ενός «πατρικού» σήματος

7. `class ConcatSignal(*args)`

   κατασκευάζει «ένωση» 2 σημάτων

8. `class TristateSignal (val)`

   κατασκευάζει τρισταθές σήμα

**Γεννήτορες και μοντελοποίηση**

Η συνάρτηση yield clause <,clause...> . Η yield σχετίζεται με τους γεννήτορες και είναι ανάλογη της λίστας ευαισθησίας. Όταν ένας γεννήτορας εκτελεί μια εντολή yield, η εκτέλεσή του αναστέλλεται σε αυτό το σημείο. Ακριβώς την ίδια στιγμή, κάθε clause (συνθήκη) γίνεται αντικείμενο trigger (σκανδάλης) και περιγράφη την κατάσταση από την οποία ο γεννήτορας θα συνεχίσει. Ο τρόπος που θα συνεχίσει θα είναι ακριβώς μία φορά ανεξάρτητα του αριθμού των συνθηκών (θα εκκινήσει στην πρώτη σκανδάλη). Αντικείμενα που μπορούν να προκαλέσουν σκανδαλισμό είναι τα Signals όπως επίσης και οι συναρτήσεις delay(t) και η join.

1. instance(), η πιο γενική συνάρτηση που μπορεί να δημιουργήσει ένα γέννητορα

```
 def top(...):
...
@instance
def inst():
    <generator body>
     ...
    return inst, ..
```

2. always(arg <,*arg>)

3. always_comb(), συνάρτηση που περιγράφει συνδυαστική λογική

4. class intbv

5. λοιπές συναρτήσεις: bit(num <,width>), επιστρέφει μια αναπαράσταση bit σε μορφή string,

**Co-simulation (Συν-προσομοίωση)**

```
class Cosimulation (exe, **kwargs)
```

Το όρισμα exe είναι η εντολή να εκτελέσει η HDL προσομοίωση. Το όρισμα kwargs είναι λέξεις που συσχετίζουν τα σήματα της HDL με τα σήματα της MyHDL. Κάθε λέξη πρέπει να μπαίνει στη λίστα των κλήσεων $to_myhdl και στο $from_myhdl στον κώδικα HDL. Κάθε όρισμα πρέπει να ανήκει στην κλάση Signal.

```
$to_myhdl(arg,<,arg \dots>)
```

διαδικασία που ορίζει ποια σήματα θα αναγνωστούν από τον προσοιμοιωτή της MyHDL, πρέπει να γίνει κλήση του, προφανώς στην αρχή της προσομοίωσης

```
$from_myhdl(arg,<,arg ldots>)
```

διαδικασία που ορίζει ποια σήματα θα οδηγηθούν από τον προσομοιωτή της MyHDL.

**Μετατροπή σε Verilog και VHDL**

```
toVerilog(func <,*args><,**kwargs>)
```

Μετατρέπει ένα σχέδιο σε MyHDL σε αντίστοιχο κώδικα Verilog και επίσης δημιουργεί και testbench για να το επιβεβαιώσει/προσομοιώσει. func είναι μια συνάρτηση που επιστρέφει στιγμιότυπο. Η toVerilog() καλή τη func κάτω από τον έλεγχο της και της περνάει τα *args και τα **kwargs. Το στιγμιότυπο του ανώτατου επιπέδου (top level instance), αυτό δηλαδή που περιέχει τα υπόλοιπα θα δώσει και το όνομα του στο αρχείο Verilog εξόδου.

```
toVHDL
```

Ακριβώς αντίστοιχη συνάρτηση με την toVerilog με τη διαφορά ότι παράγει αρχείο εξόδου σε VHDL

### 5.3.4 Απλά παραδείγματα

Ένα D flip-flop

```
from myhdl import *

def dff(q, d, clk):
    @always(clk.posedge)
    def logic():
        q.next = d

    return logic
```

και το testbench του

```
from random import randrange

def test_dff():

    q, d, clk = [Signal(bool(0)) for i in range(3)]

    dff_inst = dff(q, d, clk)

    @always(delay(10))
    def clkgen():
        clk.next = not clk
```

```
    @always(clk.negedge)
    def stimulus():
        d.next = randrange(2)

    return dff_inst, clkgen, stimulus


def simulate(timesteps):
    tb = traceSignals(test_dff)
    sim = Simulation(tb)
    sim.run(timesteps)

simulate(2000)
```

Ένα επιθυμούμε να μετατρέψουμε την περιγραφή αυτή σε Verilog τότε έχουμε να κάνουμε το εξής:

```
def convert():
    q, d, clk = [Signal(bool(0)) for i in range(3)]
    toVerilog(dff, q, d, clk)

convert()
```

## 5.4 Αναπτυξιακή Πλατφόρμα Eclipse

Η πλατφόρμα Eclipse είναι μια ανοικτού κώδικα και επεκτάσιμη πλατφόρμα για "οτιδήποτε και ταυτόχρονα τίποτα συγκεκριμένο", σύμφωνα με το εγχειρίδιο χρήσης της. Παρέχει τα θεμέλια για την κατασκευή και την εκτέλεση ενσωματωμένων εργαλείων ανάπτυξης λογισμικού. Παρέχει τη δυνατότητα στους κατασκευαστές εργαλείων να αναπτύσσουν ανεξάρτητα τα εργαλεία τους τα οποία ενσωματώνονται σε άλλα εργαλεία κατά τέτοιο τρόπο ώστε είναι αδύνατο να αντιληφθεί κανείς πού το ένα εργαλείο τελειώνει και πού αρχίζει το επόμενο.

Η ίδια η πλατφόρμα δομείται ως ένα σύνολο υποσυστημάτων τα οποία υλοποιούνται ως 1 ή περισσότερα εξωτερικά πρόσθετα (plug-ins). Κεντρικό ρόλο στη λειτουργία της πλατφόρμας παίζει ο πάγκος λειτουργίας (workbench) ο οποίος ουσιαστικά αναφέρεται στο περιβάλλον ανάπτυξης. Ο στόχος είναι να ενσωματώνονται όλα τα εργαλεία χωρίς εμφανείς διαχωριστικές γραμμές και με ελεγχόμενη ελευθερία παρέχονταις μια κοινή νοοτροπία για τη δημιουγία, διαχείριση και πλοήγηση των πόρων του χώρου εργασίας.

Τα πρόσθετα (plug-ins) που χρησιμοποιήθηκαν είναι τα εξής.

- VEditor : Επεξεργαστής κειμένου για τη Verilog. Παρέχει τη δυνατότητα της παρακολούθησης της ιεραχίας των αρθρωμάτων, της παρουσίας κάθε μεταβλητής σε κάθε πρόγραμμα καθώς και βοήθεια στο συνακτικό, τα συμφραζόμενα και ορθογραφικό έλεγχο(βοήθεια περιεχομένου). Επίσης, για βοήθεια στην αναγνωσιμότητα των γραφόμενων προγραμμάτων διαθέτει και χρωματικό κώδικα (οι δεσμευμένες λέξεις, οι συναρτήσεις, οι μεταβλητές κτλ έχουν διαφορετικά χρώματα μεταξύ τους)

- PyDev for Eclipse : Αντίστοιχο εργαλείιο με το προηγουμένο με τη διαφορά ότι προορίζεται για τη γλώσσα προγραμματισμού

- Texclipse : Εργαλείο που επιτρέπει τη συγγραφή αρχείων κειμένου pdf μέσω του επαγγελματικού τυπογραφικού συστήματος XeTeX και LaTeX για τη συγγραφή

## 5.5 Altera Quartus

Το Quartus της εταιρίας Altera είναι λογισμικό προγραμματισμού και σχεδίασης FPG και PLA το οποίο επιτρέπει τη χρήση των VHDL και Verilog για την περιγραφή του υλικού και παρέχει τη δυνατότητα της οπτικής εκδοχής του λογικού κυκλώματος καθώς την προσομοίωση. Χρησιμοποιήθηκε η ΔΩΡΕΑΝ WEB EDITION η οποία μπορεί να προγραμματίσει μόνο ορισμένα από τα προϊόντα FPGA της εταιρίας αλλά

είναι επαρκές για ακαημαϊκή χρήση (ακριβώς ο σκοπός για τον οποίο διατίθεται δωρεάν). Είναι συμβατό τόσο με Windows όσο και με Linux.

## 5.6 OrpSoC, OpenRISC platform System on Chip

Αυτή η πλατφόρμα προορίζεται για διπλό σκοπό. Από την μία να λειτουργήσει ως ανατπυξιακή πλατφόρμα για επεξεργαστές OpenRISC εν γένει και από την ως πλατφόρμα ανάπτυξης για SoC που βασίζονται στον OpenRISC και στοχεύουν σε συγκεκριμένες εφορμογές.

## 5.7 Magic VLSI και Electric VLSI design system

Το **Magic VLSI** είναι ένα αξιοπρεπές εργαλείο για VLSI layout, που γράφτηκε το 1980 στο Berkeley από τον John Ousterhout, γνωστού για την ανάπτυξη της γλώσσας Tcl. Χάρη στο γεγονός ότι είναι ανοικτού και ελεύθερου κώδικα, παραμένει δημοφιλές στα πανεπιστήμια αλλά και σε μικρές εταιρίες. Η άδεια αυτή επέτρεψε στους μηχανικούς VLSI που είχαν ικανότητες προγραμματισμού να το υλοποιήσουν έξυπνες ιδέες και να βοηθήσουν το Magic να μείνει ενημερωμένο με τις σύγχρονες τεχνολογίες κατασκευής κυκλωμάτων. Ωστόσο, είναι οι καλά σχεδιάσμενοι αλγόριθμοι του κορμού του που το καθιστούν τόσο δημοφιλές. Θεωρείται το ευκολότερο εργαλείο για σχεδιασμό layout ακόμα και από ανθρώπους που τελικά χρησιμοποιούν εμπορικά εργαλεία.

Η βασική διαφορά του Magic από τα υπόλοιπα εργαλεία είναι η χρήση της γεωμετρίας των «ραμμένων γωνιών» στην οποία κάθε layout είναι μία στίβα από επίπεδα και κάθα επίπεδο απότελείται από τετράγωνα «πλακάκια», τα οποία καλύπτουν όλη την επιφάνεια του. Κάθε τέτοιο πλακάκι έχει συντεταγμένες (του σήμειου κάτω αριστερά) και συνδέεται με άλλα τα 4 πλακάκια. Το δεξί πιο γειτονικό στην κορυφή, το πάνω πιο γειτονικό στα δεξιά, το κάτω πιο γειτονικό στα αριστέρά και το αριστερό πιο γειτονικό στη βάση. Με την προσθήκη του τύπου του υλικού που απεικονίζεται από το πλακάκι η γεωμετρία του κάθε επιπέδου περιγράφεται ακριβώς. Με αυτή τη λογική ο χρήστης το αντιμετωπίζει ως ένα πρόγραμμα ζωγραφικής όπου η μπογιά εφαρμόζεται ή σβήνεται σε αντιδιαστολή με άλλα προγράμματα όπου κάθε layout έχει «αντικείμενα» που πρέπει να αντιμετωπιστούν διαφορετικά. Και οι 2 προσεγγίσεις έχουν τα πλεονεκτήματα και τις αδυναμίες τους. Η πρώτη προσέγγιση υπερισχύει σε σχέδια ενός επιπέδου αλλά αντιμετωπίζει προβλήματα σε πολύ μεγάλες βάσεις δεδομένων, καθώς η ανάγκη να τηρούνται αρχεία 4 σημείων για κάθε γειτονικό πλακάκι όπως και η απαίτηση να αποθηκεύονται πλακάκι που απεικονίζουν ο χώρο μεταξύ περιοχών του υλικού απαιτούν μεγάλες ποσότητες μνήμης.

Το Magic επί του παρόντος συντηρείται από προγραμματιστική από τον Dr Tim Edwards και τρέχει σε συστήματα Linux όσο και Windows.

Το **Electric VLSI Design System** από την άλλη είναι εργαλείο γραμμένο σε JAVA από τον Steve M. Rubin. Χρησιμοποιείται τόσο για σχεδιαγράμματα όσο και για layout ολοκληρωμένων κυκλωμάτων. Μπορεί να χειριστεί σε ένα βαθμό τις γλώσσες VHDL και Verilog. Για πολλά χρόνια διανέμεται ως ελεύθερο λογισμικό.

Μπορεί να διαχειριστεί με ισχυρό τρόπο layouts, θεωρώντας ως σύνολα κόμβων και ακμών. Οι κόμβοι είναι τα στοιχεία του κυκλώματος (transistors, επαφές) και οι ακμές συνδέουν κόμβους. Αυτό το σχεδιαστικό στυλ διαφέρει από το Magic ή το Cadence γιατί δεν αντιμτωπίζει το layout ως πολύγωνα σε διαφορετικά στρώματα του wafer. Αυτό επιτρέπει τη σύγκριση σχεδιγράμματος και layout πολύ νωρίς στη σχειδαστική διαδικασία και πολύ γρήγορα. Έχει παρατηρηθεί ότι άνθρωποι με μηδενική εμπειρία νιώθουν άνεση με το ασυνήθιστο αυτό στυλ ενώ άλλοι που έχουν εμπειρία layout με άλλα προγράμματα προβληματίζονται κατά τη χρήση του.

Ένα ακόμα πλεονέκτημα αυτής της θεώρησης είναι ότι είναι εύκολο να προστεθούν περιορισμοί στις ακμές. Συνοπτικά αναφέρουμε τις δυνατότητες:

- έλεγχος σχεδιαστικών κανόνων
- έλεγχος ηλεκτρικών κανόνων
- προσομοίωση layout
- δρομολόγηση (routing)

- υποστήριξη των format CIF, GDS, EDIF, DXF, και VHDL.
- εκτίμηση λογική προσπάθειας
- υποστήριξη των τεχνολογιών
  - CMOS
  - NMOS
  - διπολικού transistor
  - σχεδιαγραμμάτα (Schematics)

Λόγω της γλώσσας στηνο οποία είναι γραμμένο το Electric είναι συμβατό με οποιαδήποτε πλατφόρμα μπορεί να έχει Java Virtual Machine, δηλαδή όλα τα γνωστά λειτουργικά και συντηρείται από το Dr Steve Rubin, ο οποίος μάλιστα εργάζεται για λογαριασμό της Oracle με αντικείμενο ακριβώς αυτό το πρόγραμμα.

# Μέρος II
# Πεπραγμένα

## 6 Εγκατάσταση και Δοκιμή των Εργαλείων Λογισμικού

### 6.1 Icarus Verilog

Η εγκατάσταση της Icarus Verilog είναι απλή στο περιβάλλον Linux καθώς είναι διαθέσιμη στα περισσότερα αποθετήρια δεδομένων (repositories) των αντίστοιχων διανομών (distros). Έτσι, με μια απλή εντολή (είτε yum install είτε apt-get install) μπορούμε να εγκαταστήσουμε τον μεταγλωτιστή (compiler) στο σύστημα μάς. Αν η διανομή που έχουμε στη διάθεσή μας δεν περιέχει την icarus στο αποθετήριο της τότε μπορούμε να κατεβάσουμε το αρχείο .rpm και να κάνουμε εγκατάσταση από εκεί.Αναλυτικά, οι επιλογές μας είναι οι εξής, ανάλογα το σύστημα μας:

1. Για Ubuntu linux:

```
sudo apt-get update
sudo apt-get install iverilog
```

2. άλλες διανομές:

```
sudo gedit /etc/apt/sources.lst
```

Επεξεργαζόμαστε το αρχείο sources.lst με κάποιο text editor, προσθέτοντας τα αποθετήρια:

```
## PPA for Icarus Verilog
deb http://ppa.launchpad.net/team-electronics/ppa/ubuntu jaunty main
deb-src http://ppa.launchpad.net/team-electronics/ppa/ubuntu jaunty main
```

Προσθέτουμε το κλειδί αποθετηρίου στο σύστημα για να αποφύγουμε τυχόν μηνύματα σφάλματος, κάνουμε ανανέωση και δίνουμε την εντολή εγκατάστασης

```
sudo apt-key adv --recv-keys --keyserver keyserver.ubuntu.com 7FE97A0D3D7F2EA1
sudo apt-get update
sudo apt-get install verilog
```

3. διανομή fedora:

   ```
   yum install iverilog
   ```

   Αν η διανομή που έχουμε στη διαθεσή μας δεν έχει την Icarus Verilog στα αποθετήρια της τότε θα πρέπει να κατεβάσουμε ή τον πηγαίο κώδικα ή το αρχείο rpm το οποίο είναι διαθέσιμο στον ftp server στη διεύθυνση: ftp://ftp.icarus.com/pub/eda/verilog/v0.9/ και ακολουθούμε τις εξής οδηγίες:

4. εγκατάσταση από το αρχείο .rpm: αρχικά κατασκευάζουμε το αρχείο .rpm από το αρχείο src.rpm

   ```
   rpmbuild --rebuild verilog-version.src.rpm
   ```

   και μετά το εγκαθιστούμε

   ```
   rpm -Uvh verilog-version.type.rpm
   ```

   ή αν το σύστημά μας διαθέτει την εντολή yum

   ```
   yum -y localinstall verilog-version.type.rpm
   ```

5. εγκατάσταση απευθείας από τον πηγαίο κώδικα:

   ```
   % ./configure
   % make
   (su to root)
   # make install
   ```

   που θα εγκαταστήσει το αρχείο εξ ορισμού στο φάκελο /bin ή αν θέλουμε κάποιον άλλον φάκελο αλλάζουμε την πρώτη εντολή:

   ```
   /configure --prefix=/my/special/directory
   ```

Για την εγκατάσταση σε Windows έχουμε 2 επιλογές. Είτε θα την εγκαταστήσουμε στο περιβάλλον Cygwin (που στην ουσία είναι μια προσομοίωση του τερματικού των Linux για περιβάλλον windows και επομένως θα πρέπει να κάνουμε εγκατάσταση απευθείας από το αρχείο πηγαίου κώδικα) είτε κατεβάζουμε το εκτελέσιμο αρχείο εγκατάστασης από το site http://bleyer.org/icarus/. Στη δεύτερη περίπτωση πρέπει να ενημερώσουμε της μεταβλητές περιβάλλοντος του command prompt.

### 6.1.1 Βασικές οδηγίες χρήσης

Η χρήση της Icarus Verilog συνίσταται από 2 βασικά βήματα. Πρώτον τη δημιουργία ενός ενδιάμεσου αρχείου το οποίο χαρακτηρίζεται ως "vvp assembly". Το ενδιάμεσο αυτό αρχείο μπορεί να έχει 2 διαφορετικές μορφές ανάλογα με τον αν ο σκοπός είναι να γίνει προσομοίωση ή σύνθεση(ανάλογα με τις επιλογές,options, που χρησιμοποιούμε) . Η δημιουργία του αρχείου αυτού επιτυγχάνεται με τη χρήση της εντολής iverilog -o xxx yyy.v , όπου xxx το όνομα του ενδιάμεσου αρχείου (έξοδος) και yyy.v το όνομα του αρχείου πηγαίου κώδικα (είσοδος). Δεύτερο βήμα είναι η εκτέλεση του αρχείου εξόδου με τη βοήθεια της εντολής vvp xxx στην περίπτωση της επιλογής μας να κάνουμε προσωμοίωση. Για να ελέγξουμε αν όλα είναι καλά μπορούμε να δοκιμάσουμε το ακόλουθο παράδειγμα:

1. Φτιάχνουμε ένα αρχείο με το όνομα που μας ενδιαφέρει και κατάληξη .v, για παράδειγμα my_first_program.v. Το αρχείο αυτό μπορούμε να το επεξεργαστούμε με οποιονδήποτε επεξεργαστή κειμένου (gedit, notepad, notepad++, eclipse με plugin για verilog κ.ο.κ.).

2. Γράφουμε το απλό πρόγραμμα

```
module my_firs_program;

initial
   begin
     $display ("hello world!");
   end

endmodule
```

Και το αποθηκεύουμε.

3. Μεταγλωτίζουμε το προγραμμα με την εντολή

   ```
   iverilog -o my_first_program my_first_program.v
   ```

   Το όρισμα -o και η επόμενη λέξη καθορίζουν το ότι όνομα του εκτελέσιμου αρχείου εξόδου

4. Αν δεν είχαμε σφάλμα κατά τη μεταγλώτισση τότε μπορούμε να τρέξουμε το αρχείο με την εντολή vvp

   ```
   vvp my_first_program
   ```

   Και η έξοδος πρέπει να είναι:

   ```
   hello world!
   ```

## 6.2 Eclipse IDE

Η εγκατάσταση του ενσωματωμένου περιβάλλοντος Eclipse είναι απλή και εφικτή σε όλα τα συστήματα λόγω της φορητότητας που παρουσιάζει η γλώσσα Java πάνω στην οποία στηρίζεται.

Για την εγκατάσταση του στα Windows η διαδικασία είναι:

1. Προαπαιτούμενο για την σωστή λειτουργία του περιβάλλοντος είναι η ύπαρξη Java και Java Runtime Environment στο σύστημά μας.
2. Κατεβάζουμε από την επίσημη ιστοσελίδα www.eclipse.org την πιο πρόσφατη έκδοση που αντιστοιχεί στο λειτουργικό που εργαζόμαστε (Windows) είτε μέσω απευθείας συνδέσμου είτε μέσω torrent. Η διαδικασία είναι απολύτως νόμιμη τονίζουμε.
3. Αποσυμπιέζουμε το αρχείο zip στο μέρος του δίσκου που θέλουμε να βρίσκεται το πρόγραμμα και βρίσκουμε το εκτελέσιμο αρχείο eclipse.exe (Φρόνιμο είναι να δημιουργήσουμε συντόμευση για αυτό στην επιφάνεια εργασίας)
4. Στην πρώτη εκτέλεση (με διπλό κλικ)θα ερωτηθούμε από το πρόγραμμα και θα διαλέξουμε το χώρο εργασίας (workspace), εκεί που θα αποθηκεύονται τα αρχεία και ο κώδικας που θα γράφουμε.

Για την εγκατάσταση σε Linux η διαδικασία είναι ίδια με τη διαφορά ότι προτιμούμε να εγκαταστήσουμε το εκτελέσιμο αρχείο στο φάκελο bin ώστε να μπορούμε να καλούμε το eclipse με την εντολή eclipse από το τερματικό. Εναλλακτικά μπορούμε να μην το κατεβάσουμε αλλά να το εγκαταστήσουμε μέσω terminal από τα αποθετήρια της διανομής που χρησιμοποιούμε. Η μόνη ουσιώδης διαφορά είναι ότι για την τη χρήση της JAVA σε περιβάλλον linux θα χρειαστούμε το OpenJDK το οποίο είναι το αντίστοιχο opensource του Java Runtime Environment

Το πιο σημαντικό κομμάτι αφού εγκαταστήσουμε το περιβάλλον Eclipse είναι να εγκαταστήσουμε τα πρόσθετα (plugins) που είναι απαραίτητα για γράψουμε κώδικα στη γλώσσα ή στην εφαρμογή που μας

ενδιαφέρει αφού το eclipse υποστηρίζει εγγενώς μόνο Java. Για να εγκαταστήσουμε πρόσθετα ψάχνουμε την επιλογή «Install New Software» στο μενού Help. Στο παράθυρο που μας ανοίγει γράφουμε τη διεύθυνση του site στην οποία είναι διαθέσιμο το πρόσθετο που μας ενδιαφέρει. Επιλέγουμε να εγκατασταθεί και ακολουθούμε τις οδηγίες (συνήθως χρειάζεται να αποδεχτούμε μια δήλωση περί πνευματικών δικαιωμάτων). Τις διευθύνσεις αυτές τις βρίσκουμε ψάχνοντας για το αντίστοιχο πρόσθετο είτε στην επίστημη ιστοσελίδα www.marketplace.eclipse.org είτε σε άλλες ιστοσελίδες όπως το www.sourceforge.net

### 6.2.1 Βασικές Οδηγίες χρήσης

Το Eclipse έχει δύο βασικά συστατικά στοιχεία, τους επεξεργαστές(editors) και τις οπτικές (views). Οι επεξεργαστές μπορούν να περιέχουν κείμενο ή αντικείμενα που μπορούν να τροποποιηθούν αλλά συνήθως περιέχουν τον πηγαίο κώδικα. Ανάλογα με το είδος του κειμένου ή του αντικειμένου μπορεί να ανοίγει άλλος επεξεργαστής. Οι οπτικές αντιπροσωπεύουν μία και μόνο διαδικασία που μπορεί να γίνει μέσα στην πλατφόρμα. Πχ, μια οπτική μπορεί να απεικονίζει την ιεραρχία μιας περιγραφής κυκλώματος σε Verilog, να περιέχει τις μεταβλητές και τις συναρτήσεις κώδικα σε C ή τις κλάσεις ενός προγράμματος JAVA

## 6.3 Python-Python myhdl

Η Python είναι προεγκατεστημένη σε όλες τις διανομές Linux, ενώ η διαδικασία εγκατάστασης σε windows είναι απλή: αρκεί να κατεβάσουμε την έκδοση (2.7+ ή 3.2+) από το σάιτ http://www.python.org/ download/releases/ και να την τρέξουμε. Συστήνεται η χρήση της πρώτης καθώς είναι συμβατή με τη βιβλιοθήκη myhdl.

Για την εγκατάσταση του πακέτου myhdl έχουμε να κάνουμε τα εξής:

Πρώτα ελέγχουμε αν έχουμε την Python εγκατεστημένη στον υπολογιστή μας (καθώς και την έκδοσή της) με την εντολή

```
python --version
```

Θα πρέπει να μας επιστρέψει το αριθμό έκδοσης. Φρόνιμο είναι να φροντίσουμε ώστε να έχουμε την πιο πρόσφατη έκδοση εγακτεστημένη στο σύστημά μας. Ακολούθως κατεβάζουμε την πιο πρόσφατη έκδοση από το σάιτ

```
http://sourceforge.net/project/showfiles.php?group_id=91207
```

Αποσυμπιέζουμε το αρχείο:

```
tar xvf myhdl-0.x.tar.gz
gunzip myhdl-0.x.tar
```

όπου x ο αριθμός της έκδοσης που κατεβάσαμε. Μπαίνουμε στο φάκελο που αποσυμπιέσαμε

```
cd myhdl-0.x
```

Δίνουμε την εντολή εγκατάστασης

```
python setup.py install
```

Και τέλος, αν θέλουμε, τρέχουμε και ένα τεστ που βρίσκεται στο φάκελο myhdl/test/core για να δούμε αν έγινε σωστά η όλη διαδικασία

```
cd myhdl/test/core
python test_all.py
```

## 6.4 Εγκατάσταση αλυσίδας εργαλείων OpenRISC και της πλατφόρμας ανάπτυξης OrpoSoC

Η χρήση της αλυσίδας εργαλείων GNU (gnu toolchain) είναι κατά το πλείστον εφικτή σε Linux και όχι σε windows. Ο πιο απλός τρόπος είναι να κατεβάσουμε το αρχείο εγκατάστασης και να το εγκαταστήσουμε στο σύστημα μας.

Αφού πάμε στο φάκελο που θέλουμε να γίνει η εγκατάσταση, τρέχουμε τις εντολές

```
git clone git://openrisc.net/jonas/toolchain
cd Toolchain
git submodule update --init
make -j3 PREFIX=<absolute install path>
export PATH=<PREFIX>/bin;$PATH
```

Το όρισμα j χρησιμοποιείται για να δώσουμε τον αριθμό των πυρήνων που θέλουμε να χρησιμοποιηθούν κατά τη μεταγλώτισση. Ο αλγόριθμος είναι j=2χαριθμός πυρήνων+1, πχ για ένα τετραπύρηνο επεξεργαστή ο αριθμός αυτός μπορεί να είναι το πολύ 9.

Το PREFIX καθορίζει το σημείο εγκατάστασης, συνιστάται PREFIX= /openrisc/toolchain

Η πλατφόρμα ανάπτυξης χρειάζεται απλώς να την κατεβάσουμε στο σύστημα μας

```
svn export http://opencores.org/ocsvn/openrisc/openrisc/trunk/orpsocv2
```

### 6.4.1 Απλές Εφαρμογές και Χρήσεις

Η πλατφόρμα ανάπτυξης ORPSoC περιέχει τα απολύτως ελάχιστα για ένα SoC που υλοποιεί τον OpenRISC με σκοπό τη μελέτη του για αναπτυξιακούς σκοπούς. Το σχέδιο μπορεί να προσομοιωθεί με 2 τρόπους. Είτε με τους κλασσικούς «event-driven» προσομοιωτές Icarus Verilog ή Mentor Graphics' Modelsim είτε με τη δεύτερη μέθοδο «cyle-accurate» με το εραγαλείο Verilator.

Τα σχέδια βρίσκονται στο φάκελο rtl/verilog και κάθε module έχει το δικό του φάκελο. Ένα κοινό μονοπάτι με οδηγίες προς τον compiler υπάρχει στο φάκελο rtl/verilog/include. Από άποψη λογισμικού προσομοίωσης αυτά βρίσκονται στο φάκελο sw/tests/or1200/sim. Λογισμικό οδηγών θα βρούμε στο φάκελο sw/tests/drivers και κάποιες χρήσιμες βιβλιοθήκες της C στο sw/lib.

Στο φάκελο orpsocv2/sim/run υπάρχει η δυνατότητα να τρέξουμε scripts, γραμμένα σε C++ τα οποία εκτελούν αυτοματοποιημένα προσομοίωση του επεξεργαστή όπως και διάφορα τεστ καλής λειτουργίας (πχ έλεγχος ορθής λειτουργίας των πράξεων κινητής υποδιαστολής)

Για παράδειγμα με την εντολή:

```
make rtl-tests TEST=or1200-basic VCD=1
```

τρέχουμε τεστ και προσομοίωση για τη βασική μονάδα του or1200 και ορίζουμε ότι πρέπει να έχουμε έξοδο με μορφή αρχείου VCD. Τα αρχεία εξόδου βρίσκονται στο φάκελο sim/out

Επίσης μπορούμε να τρέξουμε σε πραγματικό χρόνο διαδικασία αποσφαλμάτωσης ως εξής:

```
make rtl-tests TEST=or1200-basic VPI=1 VCD=1
```

σε ένα άλλο τερματικό τρέχουμε την εντολή:

```
ddd --debugger or32-elf-gdb
ή
ddd --debugger or32-uclinux-gdb --gdb
```

ενώ σε μέσα στο debugger δίνουμε τη διεύθυνση του τοπικού, εικονικού χειριστή εισόδου:

target remote localhost:50002

Επίσης αν θέλουμε να διαγραφούν τα αρχεία εξόδου δίνουμε την εντολή

```
make clean
```

Εαν επιθυμούμε μπροούμε να φορτώσουμε μαι εικόνα της μνήμης με την εντολή:

```
make rtl-stes USER_VMEM=/path/to/myapp.vmem
```

## 6.5 Ανάπτυξη με τη βοήθεια Virtual Machine

Αν μας προβληματίζει το ενδεχόμενο της εγκατάστασης τόσων πολλών εφαρμογών τοπικά στο σύστημα μας (natively) ή χρησιμοποιούμε ένα λειτουργικό σύστημα με μειωμένη ή καθόλου συμβατότητα (MS Windows ή Mac OS) μπορούμε να εγκαταστήσουμε όλα σε μια εικόνα (virtual image) η οποία να τρέχε σε μια ψηφιακή ψηχανή (πχ Virtual Box ). Μια καλή λύση για αυτόν τον προβληματισμό είναι να κατεβάσει κανείς έτοιμη την εικόνα από εδώ: http://opencores.org/or1k/Ubuntu_VirtualBox-image_updates_and_information
η οποία διαθέτει:

- διανομή Ubuntu Linux 11.10
- αλυσίδα εργαλείων Openrisc-GCC, GDB κτλ
- OR1KSIM προσομειωτής για OpenRISC
- OR_DEBUG_PROXY για αποσφαλμάτωση υλικού, προγραμματισμό FPGA και SPI-flash
- Icarus Verilog
- ORPSoCv2 πλατφόρμα αναφοράς για ανάπτυξη Soc
- Εκπαιδευτικό υλικό για την πλακέτα ordb2a-ep4ce22 board
- Εκδοση των Linux 3.1 συμβατή με τον OpenRISC
- Γενικό εκπαιδευτικό υλικό για τα εργαλεία και τις προσομοιώσεις
- εργαλείο για προγραμματισμό SPI-flash

Η διαδικασία είναι απλή, αφού κατεβάσουμε την εικόνα και εγκαταστήσουμε το virtual box απλώς φορτώνουμε την εικόνα. Επιπροσθέτως αν επιθουμε μπορούμε να προμηθευτούμε την πλακέτα και να χρησιμοποιήσουμε και το εργαλείο

# 7 Δοκιμές/Εφαρμογές/Ανάπτυξη

## 7.1 Αυστηρή διατύπωση του προβλήματος και οραματισμός της λύσης

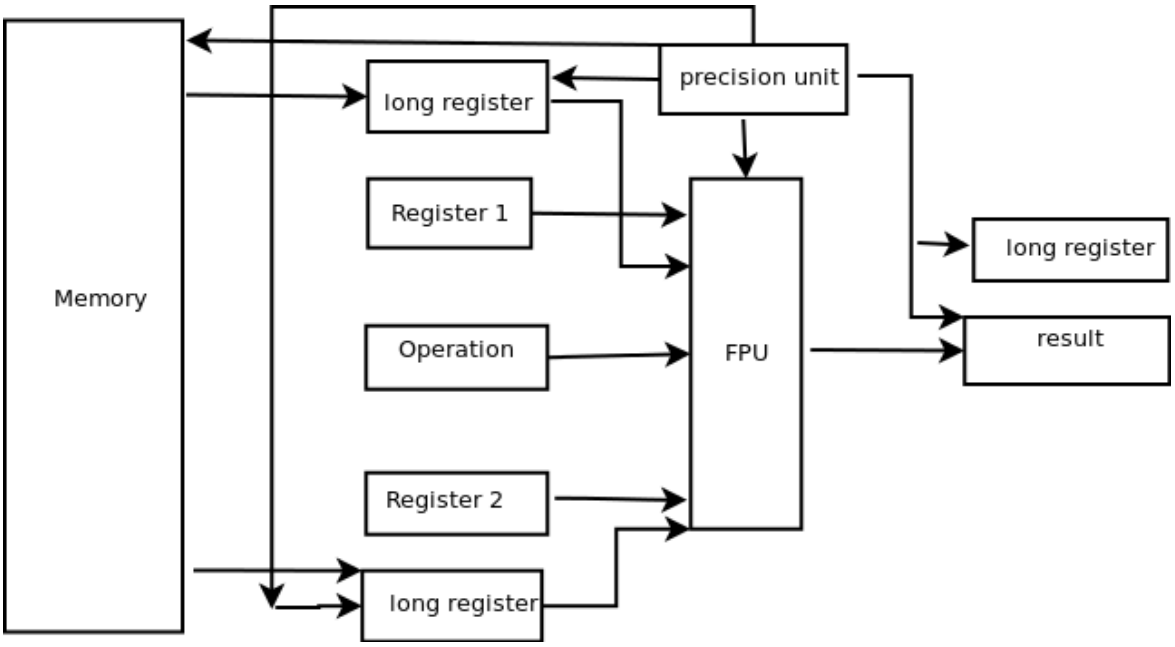
Το ζητούμενο είναι να κατασκευαστεί μια μονάδα πράξεων κινητής υποδιαστολής με τη δυνατότητα αύξησης της ακρίβειας. Δηλαδή να υπάρχει η δυνατότητα να χρησιμοποιεί περισσότερα bits από τα 32 που είναι το κανονικό αντλώντας τα απευθείας από τη μνήμη. Για την ακρίβεια περισσότερα bits θα χρησιμοποιούνται από τη mantissa και όχι το exponent ή το sign bit όπως είναι προφανές. Με αυτή ακριβώς τη σκέψη στο μυαλό θα τροποποιήσουμε το σχέδιο ενός γνωστού και μελετημένου από τα προηγούμενα επεξεργαστή. Επιλέγουμε για αυτό το σκοπό τον επεξεργαστή OpenRISC καθώς αυτός είναι αρκετά απλούστερος από τους OpenSPARC T1/T2 (που περιγράφονται από εκατομύρια γραμμές κώδικα) χωρίς αυτό να σημαίνει ότι δε θα μπορούσαμε να χρησιμοποιήσουμε και αυτούς.

Ο οραματισμός της λύσης έχει ως εξής: Θα προσθέσουμε όσο το δυνατόν λιγότερο hardware το οποίο θα επιτρέπει την αύξηση της ακρίβειας των πράξεων. Αυτό το υλικό με βάση τη θεωρία του ASM-ASMD chart θα πρέπει να διαθέτει και μονάδα ελέγχου.

Ιδιαίτερα βολικό είναι το γεγονός ότι για την πράξη του πολλαπλασιασμού χρησιμοποιείται σειριακός αθροιστής και επομένως μπορεί να δεχτεί ως είσοδο δεδομένα απεριόριστου μήκους. Αντιστρόφως, αρκούντως προβληματική είναι η σχεδίαση της μονάδες για διαίρεση καθώς η πράξη της διαίρεσης δεν μπορεί να επιμεριστεί, δηλαδή

$$\frac{A+Ad}{B+Bd} \neq \frac{A}{B} + \frac{Ad}{Bd} \tag{1}$$

## 7.2 Απλή διαγραμματική επίλυση του προβλήματος




Σχήμα 4: Λύση του προβλήματος σχηματικά (σε υψηλό επίπεδο)

**Επεξήγηση** Η βασική ιδέα είναι να χρησιμοποιήσουμε 2 μεγάλου μήκους καταχωρητές που θα παίρνουν δεδομένα από τη μνήμη, τα οποία θα χρησιμοποιούνται στις πράξεις αυξημένης ακρίβειας. Το αποτέλεσμα θα αποθηκεύεται επίσης σε ένα μεγάλου μήκους καταχωρητή.

## 7.3 Κώδικας σε Verilog- Κώδικας σε Python-MyHDL

Περιγραφή του FSM του Precision Control Unit σε Python

### 7.3.1 Προσθέτης αφαιρέτης

```
module or1200_fpu_addsub(
 clk_i,
 fpu_op_i,
 fracta_i,
 fractb_i,
 signa_i,
 signb_i,
 fract_o,
 sign_o);
```

```
   parameter FP_WIDTH = 32;
   parameter MUL_SERIAL = 0; // 0 for parallel multiplier, 1 for serial
   parameter MUL_COUNT = 11; //11 for parallel multiplier, 34 for serial
   parameter FRAC_WIDTH = 23;
   parameter EXP_WIDTH = 8;
   parameter ZERO_VECTOR = 31'd0;
   parameter INF = 31'b1111111100000000000000000000000;
   parameter QNAN = 31'b1111111110000000000000000000000;
```

```
   parameter SNAN = 31'b11111111000000000000000000000001;

   input clk_i;
   input fpu_op_i;
   input [FRAC_WIDTH+4:0] fracta_i;
   input [FRAC_WIDTH+4:0] fractb_i;
   input    signa_i;
   input    signb_i;
   output reg [FRAC_WIDTH+4:0] fract_o;
   output reg         sign_o;

   wire [FRAC_WIDTH+4:0]       s_fracta_i;
   wire [FRAC_WIDTH+4:0]       s_fractb_i;
   wire [FRAC_WIDTH+4:0]       s_fract_o;
   wire         s_signa_i, s_signb_i, s_sign_o;
   wire         s_fpu_op_i;

   wire         fracta_gt_fractb;
   wire         s_addop;

   assign s_fracta_i = fracta_i;
   assign s_fractb_i = fractb_i;
   assign s_signa_i  = signa_i;
   assign s_signb_i  = signb_i;
   assign s_fpu_op_i = fpu_op_i;

   always @(posedge clk_i)
     begin
fract_o <= s_fract_o;
sign_o <= s_sign_o;
     end

   assign fracta_gt_fractb = s_fracta_i > s_fractb_i;

   // check if its a subtraction or an addition operation
   assign s_addop = ((s_signa_i ^ s_signb_i) & !s_fpu_op_i) |
    ((s_signa_i ^~ s_signb_i) & s_fpu_op_i);

   // sign of result
  assign s_sign_o = ((s_fract_o == 28'd0) & !(s_signa_i & s_signb_i)) ? 0 :
     (!s_signa_i & (!fracta_gt_fractb & (fpu_op_i^s_signb_i)))|
     (s_signa_i & (fracta_gt_fractb | (fpu_op_i^s_signb_i)));

   // add/substract
   assign s_fract_o = s_addop ?
      (fracta_gt_fractb ? s_fracta_i - s_fractb_i :
       s_fractb_i - s_fracta_i) :
      s_fracta_i + s_fractb_i;


endmodule // or1200_fpu_addsub
```

### 7.3.2 Πολλαπλασιαστής

```
////////////////////////////////////////////////////////////////////////
////                                                                ////
////  or1200_fpu_mul                                                ////
////                                                                ////
////  This file is part of the OpenRISC 1200 project                ////
////  http://opencores.org/project,or1k                             ////
////                                                                ////
////  Description                                                   ////
////  Serial multiplication entity for the multiplication unit      ////
////                                                                ////
////  To Do:                                                        ////
////                                                                ////
////                                                                ////
////  Author(s):                                                    ////
////      - Original design (FPU100) -                              ////
////        Jidan Al-eryani, jidan@gmx.net                          ////
////      - Conv. to Verilog and inclusion in OR1200 -              ////
////        Julius Baxter, julius@opencores.org                     ////
////                                                                ////
////////////////////////////////////////////////////////////////////////
//
//  Copyright (C) 2006, 2010
//
// This source file may be used and distributed without
// restriction provided that this copyright statement is not
// removed from the file and that any derivative work contains
// the original copyright notice and the associated disclaimer.
//
// THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY
// EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
// TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
// FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL THE AUTHOR
// OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
// INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
// (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
// GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
// BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
// LIABILITY, WHETHER IN  CONTRACT, STRICT LIABILITY, OR TORT
// (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
// OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
// POSSIBILITY OF SUCH DAMAGE.
//

module or1200_fpu_mul
(
   clk_i,
   fracta_i,
   fractb_i,
   signa_i,
   signb_i,
   start_i,
   fract_o,
   sign_o,
```

```
   ready_o
   );

   parameter FP_WIDTH = 32;
   parameter MUL_SERIAL = 0; // 0 for parallel multiplier, 1 for serial
   parameter MUL_COUNT = 11; //11 for parallel multiplier, 34 for serial
   parameter FRAC_WIDTH = 23;
   parameter EXP_WIDTH = 8;
   parameter ZERO_VECTOR = 31'd0;
   parameter INF = 31'b1111111100000000000000000000000;
   parameter QNAN = 31'b1111111110000000000000000000000;
   parameter SNAN = 31'b1111111100000000000000000000001;

   input clk_i;
   input [FRAC_WIDTH:0] fracta_i;
   input [FRAC_WIDTH:0] fractb_i;
   input  signa_i;
   input  signb_i;
   input  start_i;
   output reg [2*FRAC_WIDTH+1:0] fract_o;
   output reg       sign_o;
   output reg       ready_o;

   parameter t_state_waiting = 1'b0,
       t_state_busy = 1'b1;

   reg [47:0]       s_fract_o;
   reg [23:0]       s_fracta_i;
   reg [23:0]       s_fractb_i;
   reg       s_signa_i, s_signb_i;
   wire       s_sign_o;
   reg       s_start_i;
   reg       s_ready_o;
   reg       s_state;
   reg [4:0]       s_count;
   wire [23:0]       s_tem_prod;

   // Input Register
   always @(posedge clk_i)
     begin
s_fracta_i <= fracta_i;
s_fractb_i <= fractb_i;
s_signa_i<= signa_i;
s_signb_i<= signb_i;
s_start_i <= start_i;
     end

   // Output Register
   always @(posedge clk_i)
     begin
fract_o <= s_fract_o;
sign_o <= s_sign_o;
ready_o <= s_ready_o;
     end
```

```
 assign s_sign_o = signa_i ^ signb_i;

 // FSM
 always @(posedge clk_i)
   if (s_start_i)
     begin
s_state <= t_state_busy;
s_count <= 0;
     end
   else if (s_count==23)
     begin
s_state <= t_state_waiting;
s_ready_o <= 1;
s_count <=0;
     end
   else if (s_state==t_state_busy)
     s_count <= s_count + 1;
   else
     begin
s_state <= t_state_waiting;
s_ready_o <= 0;
     end

 assign s_tem_prod[0] = s_fracta_i[0] & s_fractb_i[s_count];
 assign s_tem_prod[1] = s_fracta_i[1] & s_fractb_i[s_count];
 assign s_tem_prod[2] = s_fracta_i[2] & s_fractb_i[s_count];
 assign s_tem_prod[3] = s_fracta_i[3] & s_fractb_i[s_count];
 assign s_tem_prod[4] = s_fracta_i[4] & s_fractb_i[s_count];
 assign s_tem_prod[5] = s_fracta_i[5] & s_fractb_i[s_count];
 assign s_tem_prod[6] = s_fracta_i[6] & s_fractb_i[s_count];
 assign s_tem_prod[7] = s_fracta_i[7] & s_fractb_i[s_count];
 assign s_tem_prod[8] = s_fracta_i[8] & s_fractb_i[s_count];
 assign s_tem_prod[9] = s_fracta_i[9] & s_fractb_i[s_count];
 assign s_tem_prod[10] = s_fracta_i[10] & s_fractb_i[s_count];
 assign s_tem_prod[11] = s_fracta_i[11] & s_fractb_i[s_count];
 assign s_tem_prod[12] = s_fracta_i[12] & s_fractb_i[s_count];
 assign s_tem_prod[13] = s_fracta_i[13] & s_fractb_i[s_count];
 assign s_tem_prod[14] = s_fracta_i[14] & s_fractb_i[s_count];
 assign s_tem_prod[15] = s_fracta_i[15] & s_fractb_i[s_count];
 assign s_tem_prod[16] = s_fracta_i[16] & s_fractb_i[s_count];
 assign s_tem_prod[17] = s_fracta_i[17] & s_fractb_i[s_count];
 assign s_tem_prod[18] = s_fracta_i[18] & s_fractb_i[s_count];
 assign s_tem_prod[19] = s_fracta_i[19] & s_fractb_i[s_count];
 assign s_tem_prod[20] = s_fracta_i[20] & s_fractb_i[s_count];
 assign s_tem_prod[21] = s_fracta_i[21] & s_fractb_i[s_count];
 assign s_tem_prod[22] = s_fracta_i[22] & s_fractb_i[s_count];
 assign s_tem_prod[23] = s_fracta_i[23] & s_fractb_i[s_count];

 wire [47:0] v_prod_shl;
 assign v_prod_shl = {24'd0,s_tem_prod} << s_count[4:0];

 always @(posedge clk_i)
   if (s_state==t_state_busy)
```

```
      begin
  if (|s_count)
    s_fract_o <= v_prod_shl + s_fract_o;
  else
    s_fract_o <= v_prod_shl;
      end

endmodule // or1200_fpu_mul
```

### 7.3.3 Διαιρέτης

```
//////////////////////////////////////////////////////////////////////
////                                                              ////
////  or1200_fpu_div                                              ////
////                                                              ////
////  This file is part of the OpenRISC 1200 project              ////
////  http://opencores.org/project,or1k                           ////
////                                                              ////
////  Description                                                 ////
////  division entity for the division unit                       ////
////                                                              ////
////  To Do:                                                      ////
////                                                              ////
////                                                              ////
////  Author(s):                                                  ////
////      - Original design (FPU100) -                            ////
////        Jidan Al-eryani, jidan@gmx.net                        ////
////      - Conv. to Verilog and inclusion in OR1200 -            ////
////        Julius Baxter, julius@opencores.org                   ////
////                                                              ////
//////////////////////////////////////////////////////////////////////
//
//  Copyright (C) 2006, 2010
//
// This source file may be used and distributed without
// restriction provided that this copyright statement is not
// removed from the file and that any derivative work contains
// the original copyright notice and the associated disclaimer.
//
// THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY
// EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
// TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
// FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL THE AUTHOR
// OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
// INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
// (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
// GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
// BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
// LIABILITY, WHETHER IN  CONTRACT, STRICT LIABILITY, OR TORT
// (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
// OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
// POSSIBILITY OF SUCH DAMAGE.
```

```
//

module or1200_fpu_div
  (
   clk_i,
   dvdnd_i,
   dvsor_i,
   sign_dvd_i,
   sign_div_i,
   start_i,
   ready_o,
   qutnt_o,
   rmndr_o,
   sign_o,
   div_zero_o
   );

   parameter FP_WIDTH = 32;
   parameter MUL_SERIAL = 0; // 0 for parallel multiplier, 1 for serial
   parameter MUL_COUNT = 11; //11 for parallel multiplier, 34 for serial
   parameter FRAC_WIDTH = 23;
   parameter EXP_WIDTH = 8;
   parameter ZERO_VECTOR = 31'd0;
   parameter INF = 31'b1111111100000000000000000000000;
   parameter QNAN = 31'b1111111110000000000000000000000;
   parameter SNAN = 31'b1111111100000000000000000000001;


   input clk_i;
   input [2*(FRAC_WIDTH+2)-1:0] dvdnd_i;
   input [FRAC_WIDTH+3:0]  dvsor_i;
   input  sign_dvd_i;
   input  sign_div_i;
   input  start_i;
   output  ready_o;
   output [FRAC_WIDTH+3:0]  qutnt_o;
   output [FRAC_WIDTH+3:0]  rmndr_o;
   output  sign_o;
   output  div_zero_o;

   parameter t_state_waiting = 1'b0,
       t_state_busy = 1'b1;

   reg [FRAC_WIDTH+3:0]  s_qutnt_o;
   reg [FRAC_WIDTH+3:0]  s_rmndr_o;
   reg [2*(FRAC_WIDTH+2)-1:0]  s_dvdnd_i;
   reg [FRAC_WIDTH+3:0]  s_dvsor_i;
   reg  s_sign_dvd_i, s_sign_div_i;
   wire  s_sign_o;
   wire  s_div_zero_o;
   reg  s_start_i;
   reg  s_ready_o;
   reg  s_state;
   reg [4:0]  s_count;
```

```
   reg [FRAC_WIDTH+3:0]  s_dvd;

   // Input Register
   always @(posedge clk_i)
     begin
s_dvdnd_i <= dvdnd_i;
s_dvsor_i <= dvsor_i;
s_sign_dvd_i<= sign_dvd_i;
s_sign_div_i<= sign_div_i;
s_start_i <= start_i;
     end

   assign qutnt_o = s_qutnt_o;
   assign rmndr_o = s_rmndr_o;
   assign sign_o = s_sign_o;
   assign ready_o = s_ready_o;
   assign div_zero_o = s_div_zero_o;

   assign s_sign_o = sign_dvd_i ^ sign_div_i;
   assign s_div_zero_o = !(|s_dvsor_i) & (|s_dvdnd_i);


   always @(posedge clk_i)
     if (s_start_i)
       begin
  s_state <= t_state_busy;
  s_count <= 26;
       end
     else if (!(|s_count) & s_state==t_state_busy)
       begin
  s_state <= t_state_waiting;
  s_ready_o <= 1;
  s_count <=26;
       end
     else if (s_state==t_state_busy)
       s_count <= s_count - 1;
     else
       begin
  s_state <= t_state_waiting;
  s_ready_o <= 0;
       end

   wire [26:0] v_div;
   assign v_div = (s_count==26) ? {3'd0,s_dvdnd_i[49:26]} : s_dvd;
   wire [26:0] v_div_minus_s_dvsor_i;
   assign v_div_minus_s_dvsor_i = v_div - s_dvsor_i;


   always @(posedge clk_i)
     begin
//Reset
if (s_start_i)
  begin
     s_qutnt_o <= 0;
```

```
     s_rmndr_o <= 0;
  end
else if (s_state==t_state_busy)
  begin

     if (v_div < s_dvsor_i)
       begin
  s_qutnt_o[s_count] <= 1'b0;
  s_dvd <= {v_div[25:0],1'b0};
       end
     else
       begin
  s_qutnt_o[s_count] <= 1'b1;
  s_dvd <= {v_div_minus_s_dvsor_i[25:0],1'b0};
       end

     s_rmndr_o <= v_div;

  end // if (s_state==t_state_busy)
     end // always @ (posedge clk_i)

endmodule // or1200_fpu_div
```

## 7.4 Κώδικας για τη μονάδα ελέγχου του ASM

```
from myhdl import *

ACTIVE_LOW = bool(0)
FRAME_SIZE = 8
t_State = enum('SEARCH', 'CONFIRM', 'SYNC', encoding="one_hot")

def FramerCtrl(SOF, state, syncFlag, clk, reset_n):

    """ Framing control FSM.

    SOF -- start-of-frame output bit
    state -- FramerState output
    syncFlag -- sync pattern found indication input
    clk -- clock input
    reset_n -- active low reset

    """

    index = Signal(intbv(0)[8:]) # position in frame

    @always(clk.posedge, reset_n.negedge)
    def FSM():
        if reset_n == ACTIVE_LOW:
            SOF.next = 0
            index.next = 0
            state.next = t_State.SEARCH
```

```
        else:
            index.next = (index + 1) % FRAME_SIZE
            SOF.next = 0
            if state == t_State.SEARCH:
                index.next = 1
                if syncFlag:
                    state.next = t_State.CONFIRM
            elif state == t_State.CONFIRM:
                if index == 0:
                    if syncFlag:
                        state.next = t_State.SYNC
                    else:
                        state.next = t_State.SEARCH
            elif state == t_State.SYNC:
                if index == 0:
                    if not syncFlag:
                        state.next = t_State.SEARCH
                SOF.next = (index == FRAME_SIZE-1)
            else:
                raise ValueError("Undefined state")

    return FSM

SOF = Signal(bool(0))
syncFlag = Signal(bool(0))
clk = Signal(bool(0))
reset_n = Signal(bool(1))
state = Signal(t_State.SEARCH)
toVerilog(FramerCtrl, SOF, state, syncFlag, clk, reset_n)
toVHDL(FramerCtrl, SOF, state, syncFlag, clk, reset_n)
```

Verilog που παρήχθει αυτόματα από την περιγραφή της Python MyHDL.

```
// File: FramerCtrl.v
// Generated by MyHDL 0.7
// Date: Wed May 29 05:24:18 2013


`timescale 1ns/10ps

module FramerCtrl (
    SOF,
    state,
    syncFlag,
    clk,
    reset_n
);
// Framing control FSM.
//
// SOF -- start-of-frame output bit
// state -- FramerState output
// syncFlag -- sync pattern found indication input
// clk -- clock input
// reset_n -- active low reset
```

```
output SOF;
reg SOF;
output [2:0] state;
reg [2:0] state;
input syncFlag;
input clk;
input reset_n;

reg [7:0] index;




always @(posedge clk, negedge reset_n) begin: FRAMERCTRL_FSM
    if ((reset_n == 0)) begin
        SOF <= 0;
        index <= 0;
        state <= 3'b001;
    end
    else begin
        index <= ((index + 1) % 8);
        SOF <= 0;
        casez (state)
            3'b??1: begin
                index <= 1;
                if (syncFlag) begin
                    state <= 3'b010;
                end
            end
            3'b?1?: begin
                if ((index == 0)) begin
                    if (syncFlag) begin
                        state <= 3'b100;
                    end
                    else begin
                        state <= 3'b001;
                    end
                end
            end
            3'b1??: begin
                if ((index == 0)) begin
                    if ((!syncFlag)) begin
                        state <= 3'b001;
                    end
                end
                SOF <= (index == (8 - 1));
            end
            default: begin
                $finish;
            end
        endcase
    end
end
```

```
endmodule
```

## 7.5 Testbench

```
`include "or1200_fpu.v"


module or1200_fpu_tb();

   parameter width = `OR1200_OPERAND_WIDTH;


 //
   // I/O
   //

   //
   // Clock and reset
   //
   reg clk;
   reg rst;

   //
   // FPU interface
   //
   reg ex_freeze;
   reg [width-1:0]  a;
   reg [width-1:0]  b;
   reg [`OR1200_FPUOP_WIDTH-1:0]  fpu_op;
   wire [width-1:0]  result;
   wire  done;

   //
   // Flag signals
   //
   wire  flagforw;
   wire  flag_we;

   //
   // FPCSR interface
   //
   reg  fpcsr_we;
   wire [`OR1200_FPCSR_WIDTH-1:0]  fpcsr;

   //
   // Exception signal
   //
   wire  sig_fp;
   reg  except_started;


   //
   // SPR interface
```

```
   //
   reg spr_cs;
   reg spr_write;
   reg [31:0]  spr_addr;
   reg [31:0]  spr_dat_i;
   wire [31:0]  spr_dat_o;

or1200_fpu or1200_fpu_dut (// Clock and reset
  clk, rst,

  // FPU interface
  ex_freeze, a, b, fpu_op, result, done,

  // Flag controls
  flagforw, flag_we,

  // Exception signal
  sig_fp, except_started,

  // FPCSR system register
  fpcsr_we, fpcsr,

  // SPR interface -- currently unused
  spr_cs, spr_write, spr_addr, spr_dat_i, spr_dat_o);

always
#5 clk=~clk;

initial
begin
#0 clk=0;
end

initial  begin
     $dumpfile ("or1200_fpu_arith_tb_wave.vcd");
     $dumpvars;
     $display;
  end

endmodule
```

## 7.6 Έλεγχοι και δοκιμές

Με την εντολή:

```
make rtl-tests
```

λαμβάνουμε τα εξής logs εξόδου:

| | | |
|---|---|---|
| or1200-basic-executed.log | or1200-dsxinsn-general.log | or1200-mac-lookup.log |
| or1200-basic-general.log | or1200-dsxinsn-lookup.log | or1200-mac-sprs.log |
| or1200-basic-lookup.log | or1200-dsxinsn-sprs.log | or1200-mmu-executed.log |
| or1200-basic-sprs.log | or1200-dsx-lookup.log | or1200-mmu-general.log |
| or1200-cbasic-executed.log | or1200-dsx-sprs.log | or1200-mmu-lookup.log |
| or1200-cbasic-general.log | or1200-except-executed.log | or1200-mmu-sprs.log |
| or1200-cbasic-lookup.log | or1200-except-general.log | or1200-ov-executed.log |
| or1200-cbasic-sprs.log | or1200-except-lookup.log | or1200-ov-general.log |
| or1200-cy-executed.log | or1200-except-sprs.log | or1200-ov-lookup.log |
| or1200-cy-general.log | or1200-ext-executed.log | or1200-ov-sprs.log |
| or1200-cy-lookup.log | or1200-ext-general.log | or1200-sf-executed.log |
| or1200-cy-sprs.log | or1200-ext-lookup.log | or1200-sf-general.log |
| or1200-dctest-executed.log | or1200-ext-sprs.log | or1200-sf-lookup.log |
| or1200-dctest-general.log | or1200-float-executed.log | or1200-sf-sprs.log |
| or1200-dctest-lookup.log | or1200-float-general.log | or1200-simple-executed.log |
| or1200-dctest-sprs.log | or1200-float-lookup.log | or1200-simple-general.log |
| or1200-dsx-executed.log | or1200-float-sprs.log | or1200-simple-lookup.log |
| or1200-dsx-general.log | or1200-mac-executed.log | or1200-simple-sprs.log |
| or1200-dsxinsn-executed.log | or1200-mac-general.log | vvp.log |

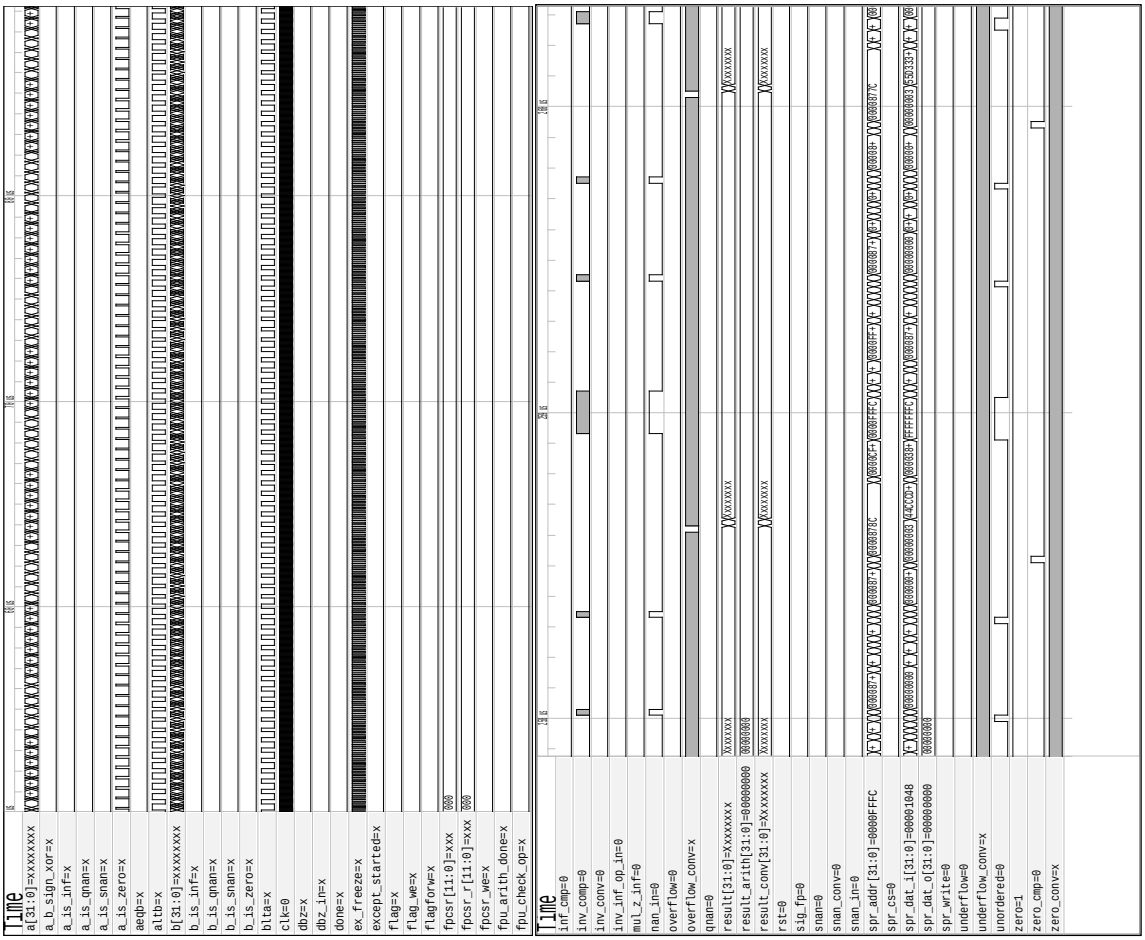
Μέρος του περιεχομένου του or1200-float-executed.log, περιεχόμενα των καταχωρητών στους κύκλους 2152, 2153 και 2154:

```
EXECUTED(       2152): 00001778:  13fffffe
GPR 0: 00000000  GPR 1: 000087cc  GPR 2: fffffffd  GPR 3: 00002640
GPR 4: 00000040  GPR 5: 00001000  GPR 6: 00001000  GPR 7: 00000100
GPR 8: 00000000  GPR 9: 00001114  GPR10: 00000000  GPR11: 00000000
GPR12: 00000000  GPR13: 00000000  GPR14: 00000010  GPR15: 00000000
GPR16: 00000000  GPR17: 00000000  GPR18: 00000000  GPR19: 00000000
GPR20: 00000000  GPR21: 00000000  GPR22: 00000000  GPR23: 00000000
GPR24: 00000000  GPR25: 00000000  GPR26: 00000000  GPR27: 00000000
GPR28: 00000000  GPR29: 00000000  GPR30: 00000000  GPR31: 00000000
SR   : 00008011  EPCR0: 00000000  EEAR0: 00000000  ESR0 : 00008001

EXECUTED(       2153): 0000177c:  e0c67000
GPR 0: 00000000  GPR 1: 000087cc  GPR 2: fffffffd  GPR 3: 00002640
GPR 4: 00000040  GPR 5: 00001000  GPR 6: 00001010  GPR 7: 00000100
GPR 8: 00000000  GPR 9: 00001114  GPR10: 00000000  GPR11: 00000000
GPR12: 00000000  GPR13: 00000000  GPR14: 00000010  GPR15: 00000000
GPR16: 00000000  GPR17: 00000000  GPR18: 00000000  GPR19: 00000000
GPR20: 00000000  GPR21: 00000000  GPR22: 00000000  GPR23: 00000000
GPR24: 00000000  GPR25: 00000000  GPR26: 00000000  GPR27: 00000000
GPR28: 00000000  GPR29: 00000000  GPR30: 00000000  GPR31: 00000000
SR   : 00008011  EPCR0: 00000000  EEAR0: 00000000  ESR0 : 00008001

EXECUTED(       2154): 00001780:  b4c00011
GPR 0: 00000000  GPR 1: 000087cc  GPR 2: fffffffd  GPR 3: 00002640
GPR 4: 00000040  GPR 5: 00001000  GPR 6: 00008011  GPR 7: 00000100
GPR 8: 00000000  GPR 9: 00001114  GPR10: 00000000  GPR11: 00000000
GPR12: 00000000  GPR13: 00000000  GPR14: 00000010  GPR15: 00000000
GPR16: 00000000  GPR17: 00000000  GPR18: 00000000  GPR19: 00000000
GPR20: 00000000  GPR21: 00000000  GPR22: 00000000  GPR23: 00000000
GPR24: 00000000  GPR25: 00000000  GPR26: 00000000  GPR27: 00000000
GPR28: 00000000  GPR29: 00000000  GPR30: 00000000  GPR31: 00000000
SR   : 00008011  EPCR0: 00000000  EEAR0: 00000000  ESR0 : 00008001
```

και ενδεικτικά τις ακόλουθες κυματομορφές:




Σχήμα 5: Κυματομορφές

Αναφορά σύνθεσης από το Quartus

```
Flow Status Successful - Mon Jun 17 11:38:46 2013
Quartus II 32-bit Version 12.1 Build 243 01/31/2013 SP 1 SJ Web Edition
Revision Name orpsoc_top
Top-level Entity Name orpsoc_top
Family Cyclone IV E
Device EP4CE22F17C6
Timing Models Final
Total logic elements 17,337 / 22,320 ( 78 % )
Total combinational functions 15,111 / 22,320 ( 68 % )
Dedicated logic registers 8,501 / 22,320 ( 38 % )
Total registers 8541
Total pins 77 / 154 ( 50 % )
Total virtual pins 0
Total memory bits 147,840 / 608,256 ( 24 % )
Embedded Multiplier 9-bit elements 6 / 132 ( 5 % )
Total PLLs 1 / 4 ( 25 % )
```

# 8 Πρόταση ενός εργαστηριακού/αναπτυξιακού περιβάλλοντος για φοιτητές και καθητές στα VLSI

## 8.1 Πρόταση ενός ολοκληρωμένου WorkFlow

Σε ακαδημαϊκό επίπεδο, τόσο για διδασκαλία όσο και για πειραματισμό, είναι εφικτό να χρησιμοποιηθεί η αλληλουχία των εργαλείων που μελετήθηκαν για τη μελέτη και την εκπόνηση έργου στα VLSI, αλλά και η χρήση κάποιων από αυτά ξεχωριστά για μελέτη ειδικευμένων κομματιών.

Συγκεκριμένα στη διάθεση οποιουδήποτε υπάρχει:

- Περιγραφή σε Verilog RTL δύο επαγγελματικών επεξεργαστών του OpenRISC και του OpenSPARC, τα οποία μπορούν να χρησιμοποιηθούν αυτούσια ή με τροποποιήσεις. Με αυτά ο ενδιαφερόμενος μπορεί να εξοικειωθεί με μεγάλα κυκλωματικά σχέδια και να κάνει δοκιμές σε μια πλατφόρμα η οποία δεν είναι εισαγωγικού επιπέδου αλλά βιομηχανικού.

- Ένα προσομοιωτής και compiler για Verilog, η Icarus Verilog, μαζί με ένα εργαλείο επαγγελματικού επίπεδου για την παρακολούθηση των κυματομορφών, το οποίο μπορεί να ανταπεξέλθει στην προσομοίωση διάφορων επιπέδων σχεδίασης και μεγάλων σχεδίων.

- Ένα εργαλείο που του επιτρέπει τη μελέτη των σχεδίων σε υψηλό επίπεδο καθώς και τη δυνατότητα να αναπτύξει σχέδια με rapid prototyping, τη βιβλιοθήκη Python MyHDL.

- Ένα εργαλείο για να κάνει σύνθεση του λογικού κυκλώματος και προγραμματισμό FPGA, το QUARTUS

- Και τέλος 2 εγαλεία για να μελετήσει και να κατασκευάσει layouts, το Magic VLSI και το ELECTRIC VLSI

Συμπερασματικά, με βάση τα ανωτέρω, ο ερευνητής ή ο φοιτητής μπορεί να ασχοληθεί με τα VLSI:

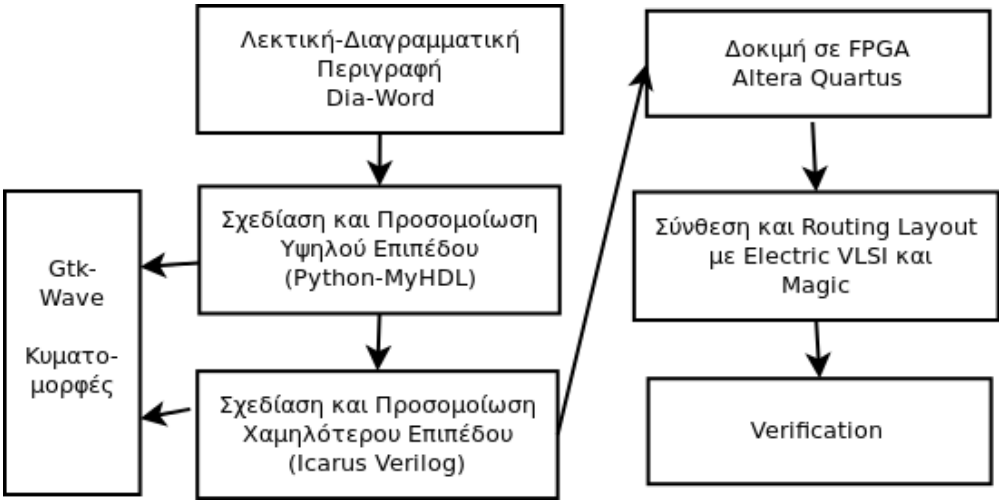
- Να χρησιμοποιήσει ένα έτοιμο σχέδιο βιομηχανικού επιπέδου για μια ειδικευμένη εργασία ή να το τροποποιήσει. Αυτό μπορεί να υλοποιηθεί με FPGA ή με ASIC αν υπάρχει η δυνατότητα

- Να σχεδιάσει και να προσομοιώσει περιγραφές Verilog, μικρές και μεγάλες.

- Να δει το λογικό κύκλωμα που προκύπτει από μια περιγραφή Verilog, σε όλα του τα επίπεδα, ξεκινώντας από λογικές πύλες και φτάνοντας με μεγαλύτερα blocks.

- Να παρακολουθήσει το τι συμβαίνει στο εσωτερικό ενός επεξεργαστή όταν αυτός εκτελεί ένα πρόγραμμα, αποκτώντας αίσθηση των εσωτερικών διεργασιών και της assembly.

Αν ο ενδιαφερόμενος θέλει να εκπονήσει ένα ολοκληρωμένο project, ξεκινώντας από τις προδιαγραφές και την ιδέα και καταλήγοντας στο layout, προτείνουμε το ακόλουθο WorkFlow




Σχήμα 6: Ένα WorkFlow

Να σημειωθεί ότι έχουν παραληφθεί ενδιάμεσα στάδια, όπως μελέτη του χρονισμού ή βελτιστοποίηση παραμέτρων (ενεργειακή κατανάλωση, εμβαδόν κλπ)

# 9 Συμπεράσματα - Προτάσεις για μελλοντική ενασχόληση

## 9.1 Συμπεράσματα

Κατόπιν αυτών, μπορούμε να σημειώσουμε τα εξής: Η παρουσία του ανοικτού λογισμικού στο χώρο της πληροφορικής είναι ισχυρή και προβλέπεται να γίνει ακόμα πιο εμφανής στο μέλλον. Με την ίδια λογική και σε συνδυασμό με τη μείωση του κόστους παραγγελίας ASIC αλλά και τη διάδοση των συσκευών FPGA όπως και την ανάγκη για σχεδίαση και παραγωγή προϊόντων ειδικής παραγγέλιας για συγκεκριμένες εφαρμογές ανοίγεται ο δρόμος στις εφαρμογές και στις διατάξεις ανοικτού υλικού.

Συγκεκριμένα, δείξαμε ότι είναι εφικτή η ανάπτυξη υλικού, με μεγάλο αριθμό πυλών και σύνθετο σχέδιο χρησιμόποιώντας αποκλειστικά εργαλεία ανοικτού λογισμικού. Με τον όρο ανάπτυξη εννοούμε όλα τα στάδια, ξεκινώντας από την αφαιρετική σχεδίαση και την περιγραφή των προδιαγραφών, την προκαταρκτική προσομοίωση σε συμπεριφερικό επίπεδο, την προσομοίωση σε επίπεδο RTL-διακοπτών, την παρακολούθηση της συμπεριφοράς του κυκλώματος (μέσω των κυματομορφών),τη σύνθεση του λογικού κυκλώματος,την επέμβαση στο layout για βελτιστοποίηση και τη μελέτη θεμάτων χρονισμού. Με λίγα λόγια, όλα τα στάδια που υπάρχουν και στην βιομηχανία ανάπτυξης Υλικού.

Τέτοιου είδους ανάπτυξη είναι εφικτή τόσο «from scratch» (εκ του μηδένος) όσο και κατόπιν επέμβασης σε ήδη διαθέσιμες περιγραφές υλικού, οι οποίες είναι πλήρεις, βιομηχανικού επίπεδου και έχουν ήδη χρησιμοποιηθεί σε επαγγελματικές εφαρμογές. Με αυτόν τον τρόπο δε χρειάζεται να σπαταληθεί χρόνος για να λυθούν προβλήματα των οποίων η λύση είναι ήδη γνωστή αλλά ο σχεδιαστής να επικεντρωθεί στην ανάπτυξη της διάταξης η οποία τον ενδιαφέρει. Είναι προφανές ότι ακόμα και η απλή εξοικείωση και ενασχόληση με σύνθετα, μεγάλα σχέδια του κάθε ενδιαφερόμενου συνεισφέρει στη γνώση και στη δημιουργία «κοινοτήτων» από χομπύστες και διάφορους ερευνητές οι οποίοι αποκτούν με αυτόν τον τρόπο εμπειρία σε τέτοιου είδους projects. Η διαθεσιμότητα τέτοιου είδους πληροφορίας διαμορφώνει καινούριες βάσεις σε αυτό το πεδίο ενδιαφέροντος.

Ως παράδειγμα ανάπτυξης τέτοιου επιπέδου αλλά και ως αυτούσιο πρόβλημα ανάπτυξης υλικού, σχεδιάστηκε και συντέθηκε ένα σύνολο λογικών κυκλωμάτων που ικανοποιούν ορισμένες προδιαγραφές τα οποία είναι συμβατά με μία από τις ανοικτές περιγραφές επεξεργαστών βιομηχανικού επιπέδου. Με αυτόν τον τρόπο, αναδείξαμε καλύτερα τις δυνατότητες των εργαλείων ανοικτού κώδικά αλλά και απαντήσαμε σε ένα αίτημα ανάπτυξης.

Συγκκεκριμένα, προδιαγράψαμε και σχεδιάσαμε έναν αριθμό βελτιώσεων πάνω σε ένα ήδη υπάρχον σχέδιο, το προσομοιώσαμε και το συνθέσαμε και από είμαστε σε θέση να εξάγουμε το συμπέρασμα ότι αν

και η υλοποίηση είναι αργή, έχουμε στη διάθεσή μας μια βελτιωμένη περιγραφή που μπορεί να εκτελέσει πράξεις με μεταβλητή ακρίβεια.

## 9.2 Προτάσεις

Ως επέκταση και σύσταση για περαιτέρω μελέτη μπορούμε να αναφέρουμε τα εξής:

- Μελέτη εργαλείων και δυνατοτήτων verification
- Εξέταση των δυνατοτήων βελτιστοποίησης στο επίπεδο του layout
- Ανάπτυξη λύσεων σε άλλου είδους προδιαγραφές
- Μελέτη και ανάπτυξη λύσεων χρησιμοποιώντας ως πλατφόρμα τον OpenSPARC
- Μελέτη και ανάπτυξη λύσεων σε υλικό που υποστηρίζει παραλλήλες πολυπεξεργαστικές αρχιτεκτονικές (πχ CUDA)

# ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] M. Morris Mano, Michael D. Ciletti, *Digital Design, With an Introduction to the Verilog HDL* , Fifth Edition, Pearson, 2013

[2] Michael D. Ciletti, *Advanced Digital Design with the Verilog HDL*, Prentice-Hall of India, 2005

[3] Pong P. Chu, *FPGA PROTOTYPING BY VERILOG EXAMPLES*, John Wiley and Sons, 2008

[4] J. Bhasker, *VERILOG HDL SYNTHESIS, A Practical Primer*, Star Galaxy Publishing, 1998

[5] Bob Zeidman, *Verilog Designer's Library*, Prentice Hall, 1999

[6] Douglas J. Smith, *HDL Chip Design, A practical guide for designing, synthesizing and simulating ASICs and FPGAs with VHDL or Verilog*, Doone Publications, 1996

[7] Samir Palnitkar, *Verilog HDL guide to Digital Design and Synthesis,*, SunSoft Press, 1996

[8] IEEE Computer Society, *IEEE Standard Verilog Hardware Description Language, IEEE Std 1364-2001 (Revision of IEEE Std 1364-1995)*, The Institute of Electrical and Electronics Engineers, Inc., 28 September 2001

[9] Pong P. Chu, *RTL HARDWARE DESIGN USING VHDL,Coding for Efficiency, Portability, and Scalability*, John Wiley and Sons, 2006

[10] Neil Weste, David Harris, *CMOS VLSI Design: A Circuits and Systems Perspective (4th Edition)* , Addison-Wesley, 2010

[11] Hubert Kaeslin, *Digital Integrated Circuit Design From VLSI Architectures to CMOS Fabrication*, Cambridge University Press, 2008

[12] Edited by Louis Scheffer, Luciano Lavagno, and Grant Martin, *Electronic Design Automation for Integrated Circuits Handbook*, Taylor & Francis Group, LLC, 2006

[13] Noor Mahammad Sk, *Icarus Verilog Installation and Usage Manual*, Indian Institute of Technology Madras, Chennai, 2007

[14] Richard Petersen, *Linux: The Complete Reference,Sixth Edition*, McGraw Hill, 2008

[15] Δημήτρης Λεβεντεας, TasPython, *Εκμάθηση Python Βήμα Βήμα Οδηγός Python Μέσω Παραδειγμάτων*

[16] Guido van Rossum, *Python Setup and Usage, Release 2.7.2*, Python Software Foundation, October 14, 2011

[17] *GTKWave 3.3 Wave Analyzer User's Guide*,Updated October 24, 2011, BSI

[18] Jan Decaluwe, *MyHDL manual Release 0.7*, December 19, 2010

[19] Tom Dillon, Jeremy Paatela, Guenter Dannoritzer, Scott Hussong, *Accelerating Algorithm Implementation in FPGA/ASIC Using Python*,Dillon Engineering, Inc.,2007

[20] David L. Weaver, Editor, *OpenSPARCTM™ Internals OpenSPARC T1/T2 CMT Throughput Computing* , Sun Microsystems, Inc, 2008

[21] OPENCORES.ORG and Authors, *OpenRISC 1000 Architecture Manual*,OPENCORES.ORG, 2006

[22] Julius Baxter, *ORPSoC User Guide, OpenCores Issue 3 for ORPSoC*OpenCores, 2010,2011

[23] Jeremy Bennett, Julius Baxter, *OpenRISC 1200 Supplementary Programmer's Reference Manual*, OPENCORES.ORG, Revision 0.2.1 23 Nov 2010

[24] Opencores Authors, *OpenRISC 1200 IP Core Specification (Preliminary Draft)* , OPENCORES.ORG, v0.12 13/9/11

[25] Jeremy Bennett, Embecosm *Howto: Porting the GNU Debugger,Practical Experience with the OpenRISC 1000 Architecture*, Embecosm, Application Note 3. Issue 2 Published November 2008

[26] Jeremy Bennett,Embecosm *The OpenCores OpenRISC 1000 Simulator and Tool Chain Installation Guide*, Embecosm, Application Note 2. Issue 3 Published November 2008

[27] David R. Wilkins, *Getting Started with LATEX*,2nd Edition, Copyright David R. Wilkins, 1995

[28] Andrew Stellman and Jennifer Greene, *Applied Software Project Management*, O'Reilly Media, Inc., 2006

[29] GNU Authors, The GNU Manifesto, http://www.gnu.org/gnu/manifesto.html, retrieved June 2013

[30] Freedomdefined Authors, Open Source Hardware Definition http://freedomdefined.org/OSHW, retrieved June 2013

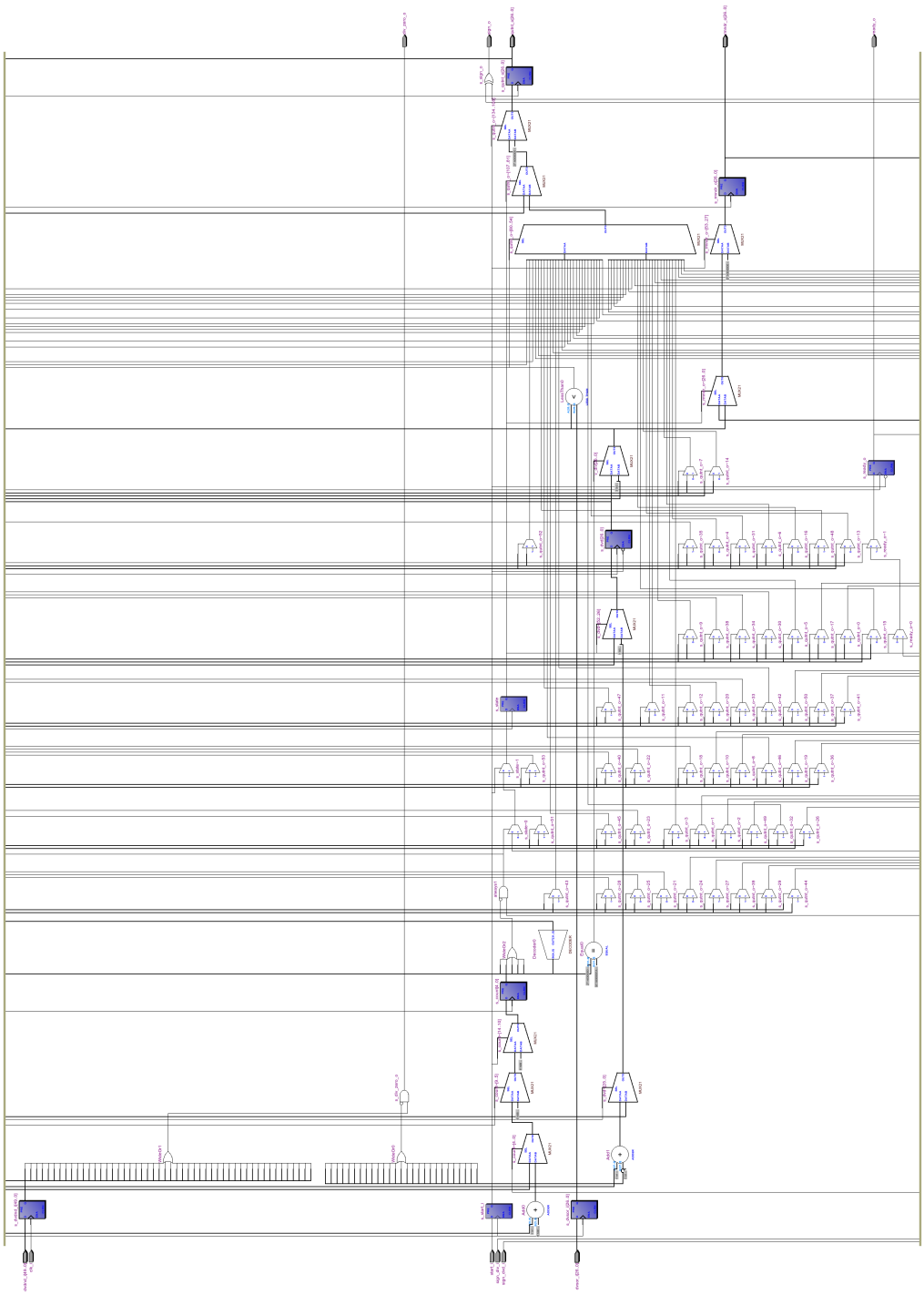
# A Σύνθεση των σχεδίων

## A.1 Σύνθεση των σχεδίων

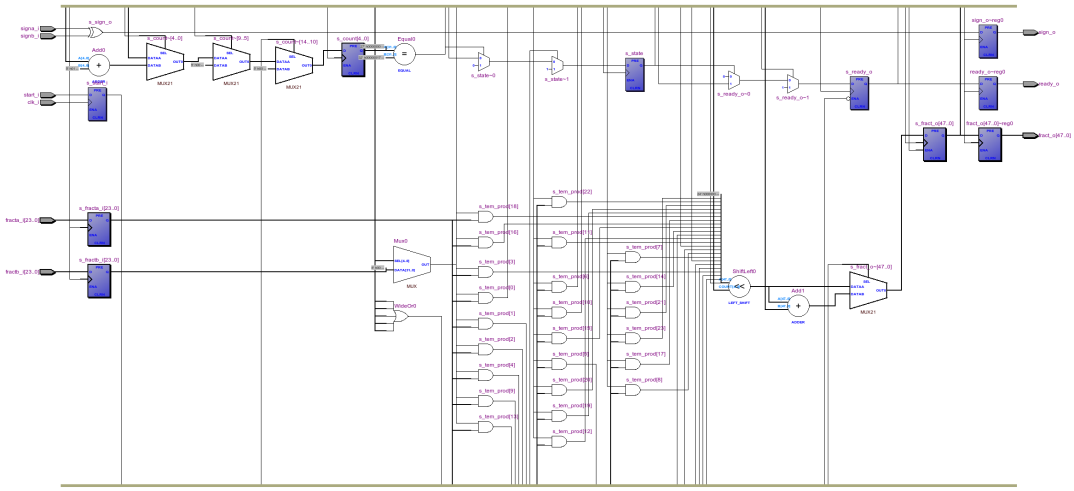
Synthesis Report

```
Flow Status Successful - Fri Jun 21 13:39:30 2013
Quartus II 32-bit Version 12.1 Build 243 01/31/2013 SP 1 SJ Web Edition
Revision Name divider
Top-level Entity Name divider
Family Cyclone IV GX
Total logic elements 217 / 29,440 ( < 1 % )
Total combinational functions 195 / 29,440 ( < 1 % )
Dedicated logic registers 165 / 29,440 ( < 1 % )
Total registers 165
Total pins 138 / 307 ( 45 % )
Total virtual pins 0
Total memory bits 0 / 1,105,920 ( 0 % )
Embedded Multiplier 9-bit elements 0 / 160 ( 0 % )
Total GXB Receiver Channel PCS 0 / 4 ( 0 % )
Total GXB Receiver Channel PMA 0 / 4 ( 0 % )
Total GXB Transmitter Channel PCS 0 / 4 ( 0 % )
Total GXB Transmitter Channel PMA 0 / 4 ( 0 % )
Total PLLs 0 / 6 ( 0 % )
```

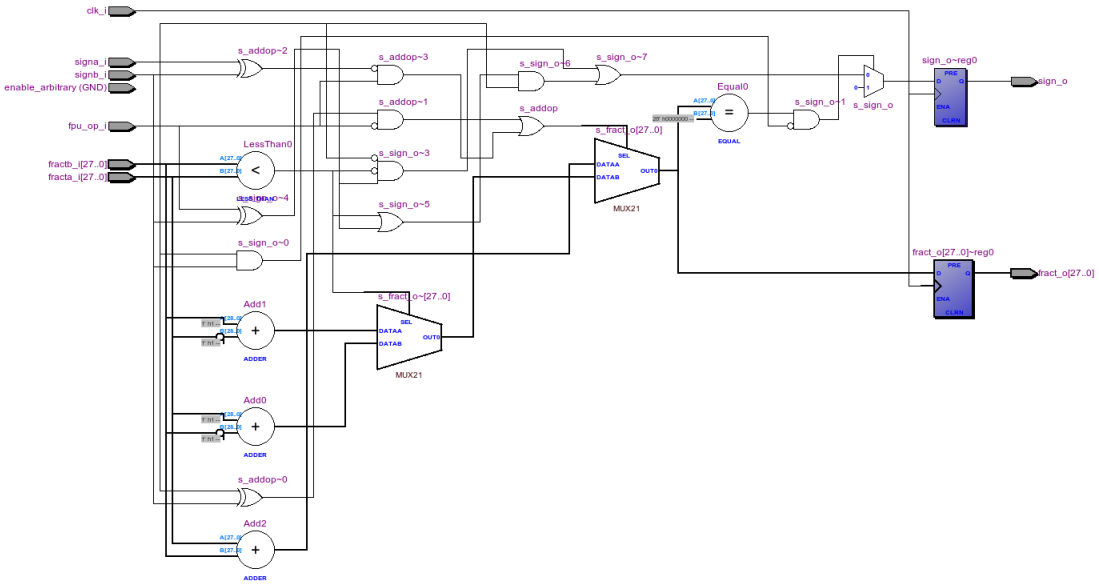
```
Device EP4CGX30CF23C6
Timing Models Final
```



Σχήμα 7: Σύνθεση λογικού κυκλώματος του Διαιρέτη με το Quartus



Σχήμα 8: Σύνθεση λογικού κυκλώματος του Πολλαπλασιαστή (Quartus)




Σχήμα 9: Σύνθεση λογικού κυκλώματος του Αθροιστή (Quartus)