



**ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ  
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ  
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ  
ΠΠΣ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ**

**ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ**

**«Μελέτη και ανάπτυξη RISC-V SoC συστήματος σε FPGAs»**

**Ελευθέριος Δ. Μπατζόλης**

**Επιβλέπων: Δημήτριος Καραμπατζάκης, Επίκουρος Καθηγητής**

**ΚΑΒΑΛΑ**

**ΣΕΠΤΕΜΒΡΙΟΣ 2022**



## **ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ**

«Μελέτη και ανάπτυξη RISC-V SoC συστήματος σε FPGAs»

«Design and development of a RISC-V SoC system on FPGAs»

**Ελευθέριος Δ. Μπατζόλης**  
**A.M.: 4518**

**Επιβλέπων: Δημήτριος Καραμπατζάκης, Επίκουρος Καθηγητής**

**Copyright@ Τμήμα Πληροφορικής, Διεθνές Πανεπιστήμιο της Ελλάδος**

Το περιεχόμενο της συγκεκριμένης Πτυχιακής Εργασίας αποτελεί πνευματική ιδιοκτησία του/των συγγραφέα/ων, του/της επιβλέποντα/ουσας καθηγητή/τριας και του Τμήματος Πληροφορικής του ΔΙΠΑΕ και προστατεύεται από το νόμο περί πνευματικής ιδιοκτησίας (Νόμος 2121/1993 και κανόνες Διεθνούς Δικαίου που ισχύουν στην Ελλάδα).

## **ΔΗΛΩΣΗ ΤΗΡΗΣΗΣ ΑΚΑΔΗΜΑΪΚΗΣ ΔΕΟΝΤΟΛΟΓΙΑΣ**

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ενυπόγραφα ότι είμαι αποκλειστικός συγγραφέας της παρούσας Πτυχιακής Εργασίας, για την ολοκλήρωση της οποίας κάθε βοήθεια είναι πλήρως αναγνωρισμένη και αναφέρεται με λεπτομέρεια στην πτυχιακή εργασία. Έχω αναφέρει πλήρως και με σαφείς αναφορές όλες τις πηγές χρήσης δεδομένων, απόψεων, θέσεων και προτάσεων, ιδεών και λεκτικών αναφορών, είτε κατά κυριολεξία, είτε βάσει επιστημονικής παράφρασης. Αναλαμβάνω την προσωπική και ατομική ευθύνη ότι σε περίπτωση αποτυχίας στην υλοποίηση των ανωτέρω δηλωθέντων στοιχείων, είμαι υπόλογος έναντι λογοκλοπής, γεγονός που σημαίνει αποτυχία στην Πτυχιακή μου Εργασία και κατά συνέπεια αποτυχία απόκτησης Τίτλου Σπουδών, πέραν των λοιπών συνεπειών του νόμου περί πνευματικών δικαιωμάτων. Δηλώνω, συνεπώς, ότι αυτή η Πτυχιακή Εργασία προετοιμάστηκε και ολοκληρώθηκε από εμένα προσωπικά και αποκλειστικά και ότι, αναλαμβάνω πλήρως όλες τις συνέπειες του νόμου στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής άλλης πνευματικής ιδιοκτησίας.

## **ΕΥΧΑΡΙΣΤΙΕΣ**

«Η παρούσα αφιερώνεται πρωτίστως στη μητέρα μου,  
χωρίς την ανιδιοτελή υποστήριξη της οι σπουδές μου θα ήταν αδύνατες.  
Αφιερώνεται επίσης στα παιδιά μου, Δημήτρη και Πέτρο,  
για να τους δείχνει πως με δύναμη ψυχής τίποτα δεν είναι αδύνατο»

Ευχαριστώ τον επιβλέποντα καθηγητή μου  
για την εμπιστοσύνη που μου έδειξε  
και τη βοήθεια που μου παρείχε  
κατά την εκπόνηση της πτυχιακής μου εργασίας.



## ΠΕΡΙΛΗΨΗ

Στο πλαίσιο της εργασίας αυτής θα σχεδιαστεί και αναπτυχθεί ένα SoC με επεξεργαστή αρχιτεκτονικής ανοιχτού κώδικα RISC-V και με χρήση νέων λογισμικών σύγχρονης ροής σχεδίασης.

Ως τεχνολογία στόχος ορίζεται η τεχνολογία FPGA της Intel που διαθέτει το εργαστήριο. Ως σύγχρονη ροή σχεδίασης θα μελετηθεί το LiteX που είναι μια ροή σχεδίασης SoC και ταυτόχρονα μία βιβλιοθήκη HARDWARE IP που φιλοξενείται στο GitHub και παρέχει βοηθητικά προγράμματα που μπορούν να χρησιμοποιηθούν για τη δημιουργία SoC σε FPGA.

Τα στοιχεία IP του LiteX περιγράφονται εξ ολοκλήρου χρησιμοποιώντας Python, το οποίο απλοποιεί τον σχεδιασμό του. Το LiteX υποστηρίζει ήδη διάφορους soft πυρήνες επεξεργαστών καθώς και βασικά περιφερειακά, χωρίς εξαρτήσεις από κλειστού κώδικα IP ή γεννήτριες κώδικα.

**ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ:** Κατασκευή RISC-V SoC σε FPGA

**ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ:** FPGA , LiteX , Python, RISC-V, SoC .

## **ABSTRACT**

This work presents a SoC RISC-V SoC open-source architecture processor which will be designed and developed using new synchronous design flow software.

The target technology is Intel's FPGA technology provided by the university laboratory. LiteX will be considered as a modern design flow, which is a SoC design flow and at the same time a HARDWARE IP library hosted on GitHub and providing utilities that can be used to create SoC in FPGA.

LiteX IP components are described entirely using Python, which simplifies its design. LiteX already supports various soft processor cores as well as basic peripherals, without dependencies on closed source IP or code generators.

**SUBJECT AREA:** Implementation of a RISC-V core with an FPGA.

**KEYWORDS:** FPGA, LiteX, Python, RISC-V, SoC.



## ΠΕΡΙΕΧΟΜΕΝΑ

<b>1<sup>ο</sup> ΚΕΦΑΛΑΙΟ: ΕΙΣΑΓΩΓΗ.....</b>	<b>15</b>
1.1 Η ιστορία των Ηλεκτρονικών Υπολογιστών μέχρι σήμερα.....	15
1.2 Οι Τεχνολογικές εξελίξεις της τελευταίας δεκαετίας .....	16
1.3 Οι σύγχρονες απαιτήσεις Υλικού \ Λογισμικού .....	16
1.4 Οι σύγχρονες απαιτήσεις Αρχιτεκτονικής και Οργάνωσης.....	18
1.5 Γιατί χρειαζόμαστε λογισμικά όπως το LiteX .....	19
<b>2<sup>ο</sup> ΚΕΦΑΛΑΙΟ: ΤΕΧΝΟΛΟΓΙΑ FPGA.....</b>	<b>21</b>
2.1 Τι είναι τα FPGA .....	21
2.2 Αφαιρετικά επίπεδα .....	22
2.3 Gateware .....	23
2.4 Περιγραφή Ψηφιακών Κυκλωμάτων .....	23
2.5 Γλώσσες Περιγραφής Υλικού .....	24
2.6 Οι Διαφορές των HDL μεταξύ τους.....	25
2.7 Migen.....	26
2.8 Ανάπτυξη λογικών κυκλωμάτων σε VHDL, Verilog, Migen.....	27
2.9 Διαδικασία Ανάπτυξης Λογικών Σχεδίων σε FPGA .....	28
2.10 Στρατηγικές ελέγχου αξιοπιστίας λογικών σχεδίων .....	29
2.11 Οι μεγαλύτεροι κατασκευαστές FPGAs .....	30
2.12 Το De10lite της Intel.....	31
<b>3<sup>ο</sup> ΚΕΦΑΛΑΙΟ: RISC-V ISA .....</b>	<b>33</b>
3.1 Γιατί χρειαζόμαστε μία ανοιχτή αρχιτεκτονική.....	33
3.2 Η Ιστορία της RISC-V ISA.....	33
3.3 Η Αρχιτεκτονική και Οργάνωση της RISC-V ISA.....	34
3.4 RISC-V International .....	35
3.5 RISC-V Community .....	35
3.6 RISC-V Technical Organization .....	36
3.7 RISC-V CPUs .....	37
3.8 RISC-V Specifications.....	37
3.9 RISC-V Extension Lifecycle .....	44
3.10 Τα Format Εντολών του βασικού RISC-V .....	45
3.11 Γενικός κύκλος Φόρτωσης/Εκτέλεσης (Fetch/Execute cycle) .....	46
3.12 Περιβάλλοντα εκτέλεσης λογισμικού RISC-V .....	47
3.13 RISC-V vs ARM vs x64.....	47
<b>4<sup>ο</sup> ΚΕΦΑΛΑΙΟ: LITEX &amp; SOC DESIGN FLOW .....</b>	<b>49</b>
4.1 LiteX: SoC builder framework .....	49
4.2 Τί είναι το SoC .....	49
4.3 Ορολογία: Library, Framework, toolkit, API, SDK.....	49
4.4 Εγκατάσταση του LiteX.....	50
4.5 Πυρήνες Λειτουργικότητάς .....	51

4.6	Δίαυλοι Επικοινωνίας.....	51
4.7	Ένα τυπικό SoC Design flow με το LiteX .....	52
4.8	Κατασκευάζοντας συνδυαστικά κυκλώματα με το LiteX .....	52
4.9	Διαμόρφωση Ubuntu 22.04.....	53
4.10	Integrated Development Environments (IDEs) .....	54
4.11	Κύκλος ζωής ανάπτυξης λογισμικού (Software Development Life Cycle - SDLC) .....	55
4.12	Η ανάγκη για Continuous Integration \ Continuous Development pipelines .....	56
4.13	LiteX SoC Design Flow .....	61
4.14	Δημιουργία RISC-V SoC ME LiteX.....	63
4.15	Προγραμματισμός του RISC-V SoC.....	65
4.16	ΣΥΜΠΕΡΑΣΜΑΤΑ .....	71
	<b>ΑΝΑΦΟΡΕΣ.....</b>	<b>73</b>
	<b>ΠΑΡΑΡΤΗΜΑ Ι .....</b>	<b>75</b>

## ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

Εικόνα 1. Το πλήθος των τρανζίστορ της CPU ανά έτος από το 1970 – 2020 [2] .....	15
Εικόνα 2. CPU AMD Epyc 7302P [8] .....	17
Εικόνα 3. Running TensorFlow on Cloud TPU [10] .....	19
Εικόνα 4. Typical CNN architecture [11] .....	19
Εικόνα 5. Αφαιρετική αναπαράσταση των περιεχομένων ενός CLB .....	21
Εικόνα 6. Διάγραμμα Gajski–Kuhn .....	22
Εικόνα 7. Τι είναι το Gateway; .....	23
Εικόνα 8. Στρατηγικές ελέγχου αξιοπιστίας λογικών σχεδίων σε FPGA .....	29
Εικόνα 9. Το DE10-Lite FPGA Board .....	31
Εικόνα 10. Τεχνική Οργάνωση RISC-V .....	36
Εικόνα 11. Δακτύλιοι δικαιωμάτων για αρχιτεκτονικές x86 [30] .....	37
Εικόνα 12. Ο κύκλος επέκτασης ζωής για την RISC-V [32] .....	44
Εικόνα 13. Κύκλος εκτέλεσης .....	46
Εικόνα 14. Τυπικό SoC Design flow με το LiteX [34] .....	52
Εικόνα 16. Ο κύκλος ανάπτυξης λογισμικού .....	55
Εικόνα 17. CI\CD pipeline Template [35] .....	56
Εικόνα 18. Τα είδη της Εικονοποίησης [37] .....	57
Εικόνα 19. Επίδειξη Docker Development Environment .....	58
Εικόνα 20. Δημιουργία Docker Image μέσω zsh shell .....	59
Εικόνα 21. DockerHub στιγμιότυπο με τη φορτωμένη εικόνα .....	59
Εικόνα 22. Git workflow [39] .....	60
Εικόνα 23. Στιγμιότυπο του Jenkins που δείχνει τα Jobs .....	61
Εικόνα 24. LiteX SoC Design Flow .....	62
Εικόνα 25. Αποτέλεσμα προσομοίωσης .....	65
Εικόνα 25 LED Demo με κώδικα Migen .....	67
Εικόνα 26. Demo App, έλεγχος LED του FPGA .....	68
Εικόνα 27. LiteX BIOS με demo προγράμματα για επίδειξη λειτουργίας SoC .....	69
Εικόνα 28. Demo App .....	70
Εικόνα 29. C demo εφαρμογή .....	70

## ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

Πίνακας 1. Επίπεδα προνομίων .....	38
Πίνακας 2. Βασική Αρχιτεκτονική RISC-V .....	39
Πίνακας 3. Standard ISA Extensions. ....	40
Πίνακας 4. RISC-V Instruction formats .....	46

## ΠΙΝΑΚΑΣ ΟΡΟΛΟΓΙΑΣ

Ξενόγλωσσος όρος	Ελληνικός Όρος
Adder	Λογικό Κύκλωμα Αθροιστή
Behavioral Description	Περιγραφή Συμπεριφοράς
Bitstream	Ροή Δεδομένων
Chipselets\tiles	Μικροτσίπ
Combinatorial Logic	Συνδυαστική Λογική
Deep learning	Βαθιά μάθηση
Electronic Design Automation	Λογισμικό αυτοματοποίησης Κατασκευής Ηλεκτρονικών Κυκλωμάτων
Fabbing	Κατασκευή σε εργοστάσιο
Formal Verification	Τυπική επαλήθευση
FPGA	Προγραμματιζόμενη μέσω Πεδίου Διάταξη Πυλών
Functional Simulation	Λειτουργική Προσομοίωση
Hardware Description Language	Γλώσσα Περιγραφής Υλικού
Heterogeneous Integration	Ετερογενής ενσωμάτωση
Instruction set	Σετ εντολών
Integration	Ενσωμάτωση
Logic Gates	Λογικές Πύλες
Machine Learning	Μηχανική Μάθηση
Sequential Logic	Ακολουθιακή Λογική
Static Timing Analysis	Στατική Ανάλυση Χρονισμού
Structural Description	Δομική Συμπεριφορά
Virtualisation	Εικονοποίηση
Wafer	Δισκίο πυριτίου

## ΣΥΝΤΜΗΣΕΙΣ – ΑΡΚΤΙΚΟΛΕΞΑ – ΑΚΡΩΝΥΜΙΑ

ASIC	Application Specific Integrated Circuit
CLB	Control Logic Block
CPU	Central Processing Unit
DSL	Domain Specific Languages
E.UV.L	Extreme ultraviolet lithography
EDA	Electronic Design Automation
FPGA	Field Programmable Gate Array
GPU	Graphics Processing Unit
HDL	Hardware Description Language
IO	Input\Output
IoT	Internet Of things
IP	Intellectual Property
ISA	Instruction Set Architecture
LUT	Look Up Table
MIPS	Microprocessor without Interlocked Pipelined Stages
RISC	Reduced Instruction Set Computer
SoC	System on a Chip
SOHO	Small Office House office
SPI	Serial Peripheral Interface
TPU	Tensor Processing Unit
TSC	Technical Steering Committee

Ο ENIAC χρησιμοποιήθηκε κατά τη διάρκεια του Β' παγκοσμίου πολέμου όπου χρησιμοποιώντας μαθηματικές μεταβλητές εισήγαγαν στον υπολογιστή παραμέτρους για να υπολογίζει γρήγορα μαθηματικούς πίνακες που χρειαζόταν το πυροβολικό για να αυξήσει την ακρίβεια στις βολές μεγάλου βεληνεκούς. Η εσωτερική τεχνολογία που χρησιμοποιήθηκε ήταν λυχνίες, δίοδοι και αντιστάσεις. Το 1954 κατασκευάζεται ο TRADIC (TRAnsistor DIgital Computer or TRansistorized Airborne DIgital Computer). Ο πρώτος Ηλεκτρονικός Υπολογιστής με τρανζίστορ. Είχε μόλις 684 τρανζίστορ και 10.358 διόδους γερμανίου.

## **1.2 Οι Τεχνολογικές εξελίξεις της τελευταίας δεκαετίας**

Οι σημερινοί επεξεργαστές των σύγχρονων Ηλεκτρονικών Υπολογιστών είναι ένα σύνθετο λογικό κύκλωμα δισεκατομμυρίων τρανζίστορ. Τα τελευταία 70 χρόνια έχει σημειωθεί πολλή μεγάλη πρόοδος (Εικόνα 1) σε όλα τα στάδια κατασκευής Η/Υ και κατά επέκταση ψηφιακών λογικών κυκλωμάτων. Ο σχεδιασμός λογικών κυκλωμάτων γίνεται μετά από συλλογή απαιτήσεων και μετατρέπει τις απαιτήσεις αυτές σε λογικά κυκλώματα μέσω προγραμμάτων EDA (Electronic Design Automation) όπως CADENCE και SYNOPSIS.

Τα λογισμικά αυτοματοποίησης ηλεκτρονικού σχεδιασμού εξασφαλίζουν τον σωστό, γρήγορο και επικυρωμένο σχεδιασμό ενός λογικού κυκλώματος χρησιμοποιώντας αφαιρετική λογική. Επίσης οι μηχανικοί υλικού έχουν τη δυνατότητα της επιτάχυνσης της κατασκευής ενός σχεδίου επεξεργαστή μέσω επαναχρησιμοποιούμενων μπλοκ κώδικα I.P. (Intellectual Property) για κοινές διαδικασίες όπως για IO (Input/Output). Η επαναχρησιμοποίηση της λογικής συνεισφέρει σημαντικά στον όλο και μικρότερο χρόνο που απαιτείται για τη σχεδίαση λογικών κυκλωμάτων [3]. Τα παραπάνω προτερήματα έχουν οδηγήσει των αριθμό των μηχανικών που απαιτείται για την κατασκευή ενός επεξεργαστή με μερικά δισεκατομμύρια τρανζίστορ να έχει μειωθεί σημαντικά [4] σε σχέση με το παρελθόν.

## **1.3 Οι σύγχρονες απαιτήσεις Υλικού \ Λογισμικού**

Τα τελευταία 70 χρόνια οι μηχανικοί προσπαθούσαν να χωρέσουν όσο περισσότερα τρανζίστορ μπορούσαν στον ίδιο χώρο. Το φαινόμενο αυτό εξασφάλιζε μεγαλύτερες επιδόσεις και μικρότερη κατανάλωση σε μικροτσίπ που είχαν το ίδιο ή και μικρότερο μέγεθος. Στο φαινόμενο αυτό αναφέρεται ο νόμος του Moore [5]. Ο αριθμός των τρανζίστορ στο εσωτερικό των μικροτσίπ διπλασιαζόταν κάθε δύο περίπου χρόνια (Εικόνα 1).

Δυστυχώς όμως ο σχεδιασμός και η κατασκευή (fabbing) όλο και μικρότερων μικροτσίπ οδηγεί σε ολοένα και μεγαλύτερη αύξηση του κόστους παραγωγής. Αυτό οφείλεται στην συνεχώς επιταχυνόμενη σμίκρυνση του τρανζίστορ σε σημείο που κοντεύει να αγγίξει ατομικά επίπεδα [6]. Όταν το μέγεθος του τρανζίστορ είναι τόσο μικρό τότε υπερισχύουν κβαντικά φαινόμενα που δυσχεραίνουν την περαιτέρω σμίκρυνση του τρανζίστορ.

Ένα ακόμη πρόβλημα που αντιμετωπίζουν οι κατασκευαστές μικροτσίπ είναι το φαινόμενο της επιβραδυνόμενης αύξησης επιδόσεων λόγω των ανωτέρω φαινομένων. Αυτό οφείλεται στην απόσυρση ενός ακόμα «νόμου» αυτού της κλιμάκωσης Dennard



(Dennard Scaling). Όσο μικραίνουν τα τρανζίστορ η πυκνότητα της ισχύος παραμένει περίπου σταθερή με την επιφάνεια του τσιπ αφού η τάση και το ρεύμα κλιμακώνονται με την επιφάνεια που καλύπτει το τσιπ. Το φαινόμενο αυτό που αναφέρεται και ως «Power Wall» στην βιβλιογραφία, οφείλεται στις ολοένα μικρότερες διαστάσεις των εσωτερικών αγωγών τροφοδοσίας στα εσωτερικά των μοντέρνων επεξεργαστών καθώς περιορίζεται η τροφοδοσία των τρανζίστορ και δυσχεραίνεται η απαγωγή θερμότητας [6].

Για να συνεχίσουν να είναι ανταγωνιστικοί οι επιλογές που έχουν οι κατασκευαστές είναι είτε να περάσουν σε λιθογραφική μέθοδο που να επιτρέπει περισσότερα τρανζίστορ στον ίδιο χώρο ή να μεγαλώσουν το μέγεθος του μικροτσίπ. Με την πρώτη επιλογή το κόστος αυτό που προκύπτει για την έρευνα και αγορά λύσεων όπως αυτής της E.UV.L (Extreme ultraviolet lithography) [7] μπορεί πλέον να δικαιολογηθεί μόνο με την παραγωγή εκατομμυρίων μικροτσίπ όπως αυτά των κινητών. Ούτε όμως η δεύτερη επιλογή είναι καλύτερη.

Μεγάλο μέγεθος μικροτσίπ σημαίνει από την μία περισσότερες ευκαιρίες για λάθη κατά την κατασκευή, όπου τα λάθη αυτά δεν βαραίνουν το εργοστάσιο κατασκευής του μικροτσίπ αλλά τον αγοραστή του, και από την άλλη υπάρχει ένα «ταβάνι» στο πόσο μεγάλες είναι οι λιθογραφικές μάσκες που επιβάλλετε από την επιλεγόμενη λιθογραφική μέθοδο. Ακόμα μεγάλο μέγεθος μικροτσίπ σημαίνει ότι υπάρχουν λιγότερα μικροτσίπ σε κάθε φέτα σιλικόνης (wafer). Αποδεικνύεται έτσι ότι είναι οικονομικότερο αν παράγονται μεγάλα τσιπ από μικρότερα τσιπ που συνδέονται μεταξύ τους.



**Εικόνα 2.** CPU AMD Epyc 7302P [8]

Η τάση αυτή της συνεχόμενης σμίκρυνσης των τρανζίστορ οδήγησε μερικούς κατασκευαστές στην κατασκευή SoCs (System on Chip) όπου σε ένα πακέτο χωρούν αρκετούς μικροεπεξεργαστές, όπως νευρωνικά δίκτυα και επεξεργαστές εικόνες για

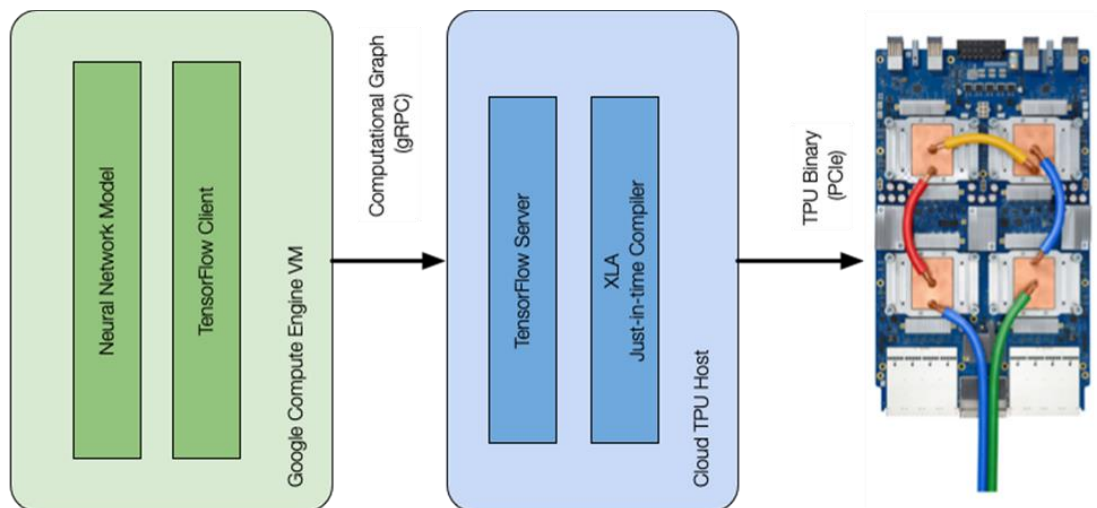
μεγαλύτερη ενσωμάτωση (Integration). Άλλοι πάλι κατασκευαστές όπως η AMD και η Intel σταμάτησαν να κυνηγάνε την τεχνολογία αιχμής σε διαδικασίες λιθογραφίας αλλά βρήκαν πρωτότυπες λύσεις όπως αυτή των chiplets (AMD) \ tiles (Intel) ή αλλιώς της ετερογενούς ενσωμάτωσης (Heterogeneous Integration).

Με την μέθοδο αυτή παίρνουν ένα μεγάλο μονολιθικό μικροσίπ και το χωρίζουν σε πολλά μικρότερα αυτοτελή τσιπ με αυτοτελή συσκευασία (packaging). Συνήθως επιλέγουν τα πιο κρίσιμα κομμάτια όπως του ελέγχου και τα πιο χρησιμοποιούμενα όπως αυτά που πραγματοποιούν μαθηματικές πράξεις και τα υλοποιούν σε μία σύγχρονη, οικονομικά ακριβότερη λιθογραφική μέθοδο όπως η E.UV.L ώστε με αυτό τον τρόπο να υπάρχει μεγαλύτερη ταχύτητα, μεγαλύτερη ενσωμάτωση και χαμηλότερη κατανάλωση.

Έπειτα τα υπόλοιπα κομμάτια μπορούν να κατασκευαστούν με μία οικονομικότερη λιθογραφική μέθοδο. Με αυτό τον τρόπο κερδίζουν διπλά. Από την μία την μία μικρότερα μικροσίπ (chiplets\tiles) μπορούν να συνδυαστούν σε διαφορετικά προϊόντα επιτρέποντας την κατασκευάστρια εταιρεία ευλυγισία ώστε να υπακούει στην επιταγές της αγοράς κατασκευάζοντας είτε περισσότερους επεξεργαστές για εξυπηρετητές (Σειρά Epyc για την AMD) ή SOHO (Small Office House office, σειρά Ryzen για την AMD). Από την άλλη μπορούν να δώσουν τα διαφορετικά κομμάτια σε διαφορετικούς κατασκευαστές κερδίζοντας σε ασφάλεια του φυσικού μικροσίπ και σε ασφάλεια της πνευματικής της ιδιοκτησίας.

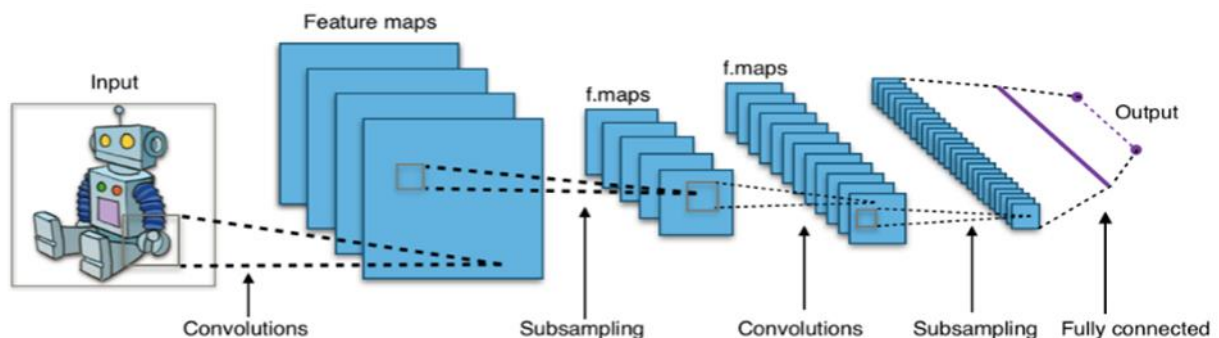
#### **1.4 Οι σύγχρονες απαιτήσεις Αρχιτεκτονικής και Οργάνωσης.**

Οι υπάρχουσες αρχιτεκτονικές ηλεκτρονικών υπολογιστών, όπως Harvard και Von Neumann, αν και χρησιμοποιούνται ευρέως σήμερα δεν είναι πλέον αρκετές για να ικανοποιήσουν τις σύγχρονες απαιτήσεις σε επεξεργαστική ισχύ. Εφαρμογές όπως τα σύγχρονα νευρωνικά δίκτυα ξεκίνησαν να υλοποιούνται από CPUs (Central Processing Unit) και GPUs (Graphics Processing Unit) με μεγάλη κατανάλωση για την επεξεργαστική ισχύ που προσέφεραν. Μεγάλες εταιρίες στον χώρο όπως η Google, που έχει και δικό της λογισμικό (TensorFlow) για την υλοποίηση και επεξεργασία δεδομένων νευρωνικών δικτύων, έφτιαξαν καινούργια αρχιτεκτονική υλικού, το TPU (Tensor Processing Unit) (Εικόνα 3). Μελέτες έχουν δείξει [9] ότι ειδικό υλικό σχεδιασμένο σε ειδικά σχεδιασμένο ASIC (Application Specific Integrated Circuit) ή FPGA (Field Programmable Gate Array) είναι δύο (2) τουλάχιστον τάξεις καλύτερο σε επιδόσεις και μικρότερο σε κατανάλωση ισχύος.



**Εικόνα 3.** Running TensorFlow on Cloud TPU [10]

Αναπαριστώντας αφαιρετικά ένα νευρωνικό δίκτυο (Εικόνα 4) τότε αυτό δεν είναι παρά μία τεράστια μαθηματική εξίσωση με πολλές εισόδους με πάρα πολλούς πολλαπλασιασμούς πινάκων. Ερευνητές κατάφεραν να υλοποιήσουν τα παραπάνω χρησιμοποιώντας ένα FPGA εκμεταλλευόμενοι τις δυνατότητες τις παράλληλης επεξεργασίας.



**Εικόνα 4.** Typical CNN architecture [11]

### 1.5 Γιατί χρειαζόμαστε λογισμικά όπως το LiteX

Η παρούσα πτυχιακή εργασία έχει σαν σκοπό να επιδείξει τρόπους επίλυσης των υπολογιστικών αναγκών του σήμερα προσφέροντας ταυτόχρονα έδαφος για έρευνα στις ανάγκες του αύριο. Όπως αναφέρθηκε παραπάνω οι ανάγκες της αγοράς πρόκειται να αλλάξουν απότομα και τώρα περισσότερο από ποτέ χρειάζονται μέθοδοι γρήγορης παραγωγής και γρήγορης προσομοίωσης λογικών σχεδίων.

Στα πρώτα ψηφιακά κυκλώματα η σχεδίαση λιθογραφικών μασκών γινόταν με το χέρι έπειτα αναπτύχθηκαν λογισμικά EDA για γρήγορη παραγωγή και έλεγχο λογικών κυκλωμάτων. Μετά δημιουργήθηκαν οι γλώσσες περιγραφής υλικού για ακόμα μικρότερο χρόνο παραγωγής και γρήγορης προσομοίωσης λογικών σχεδίων. Τώρα χρειάζονται λογισμικά που να υποστηρίζουν τις σύγχρονες μεθόδους σχεδίασης SoC και να

επεκτείνουν τη λειτουργικότητα τους και να μειώνουν το χρόνο που χρειάζεται μία ιδέα να βγει στην αγορά.

Ένα από τα γρήγορα αναπτυσσόμενα ανοιχτού κώδικα λογισμικά αυτής της κατηγορίας είναι το LiteX. Μέσω του LiteX Framework μπορεί ο μηχανικός υλικού να δημιουργεί γρήγορα ένα SoC που θα καλύπτει τις εξειδικευμένες ανάγκες σε υλικό και επεξεργαστική ισχύ του προς επίλυση προβλήματος μέσω της χρήσης ανοιχτών αρχιτεκτονικών όπως η RISC-V ISA και επαναπρογραμματιζόμενων FPGA. Με αυτό τον τρόπο μπορεί ανά πάσα στιγμή να επαναπροσδιοριστεί το υλικό και τα χαρακτηριστικά του στις ανάγκες τις δικές μας ή της αγοράς χωρίς κανένα επιπρόσθετο κόστος.

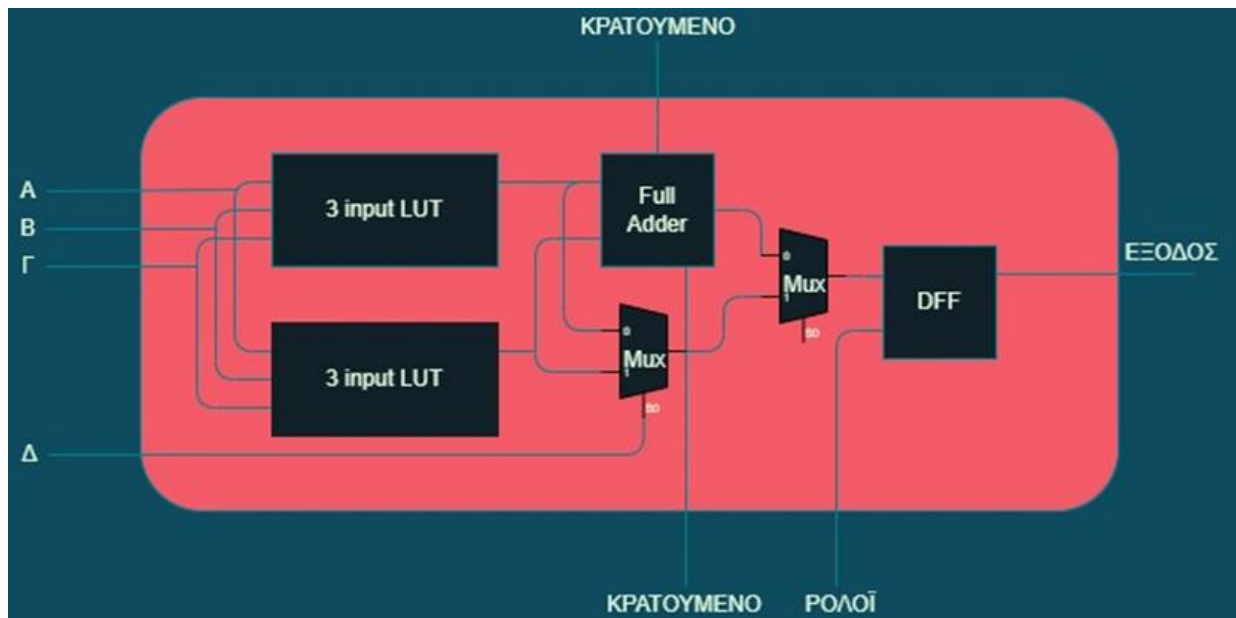
Ακόμα μπορεί να σχεδιαστεί ένα SoC με παράλληλους RISC-V πυρήνες που να εξυπηρετεί τις υπολογιστικές ανάγκες για ταυτόχρονη επεξεργασία δεδομένων που θα προκύπτουν από συστήματα προορισμένα εξειδικευμένη επεξεργαστική εργασίας όπως Machine Learning, Deep Learning ή για εξειδικευμένη αγορά όπως Infrastructure as a Service (IaaS), Cloud servers απλά προσθέτοντας ή αφαιρώντας υλικό όπως μνήμη και περιφερειακά όπως Physical Ethernet, PCIe, SPI interfaces ανάλογα με τις ανάγκες της έρευνας και της αγοράς.

Τέλος, υπάρχει η δυνατότητα να χρησιμοποιηθούν τα ενσωματωμένα εργαλεία ή άλλα εργαλεία ανοιχτού κώδικα για να μοντελοποιηθεί, προσομοιωθεί και να πιστοποιηθεί το ψηφιακό σύστημα σε οποιοδήποτε στάδιο.

## 2<sup>ο</sup> ΚΕΦΑΛΑΙΟ: ΤΕΧΝΟΛΟΓΙΑ FPGA

### 2.1 Τι είναι τα FPGA

Η Προγραμματιζόμενη μέσω Πεδίου Διάταξη Πυλών (FPGA – Field Programmable Gate Array) είναι μια επαναπρογραμματιζόμενη ψηφιακή λογική συσκευή. Η εσωτερική διαμόρφωση του FPGA αποτελείται από Control logic Blocks (CLBs), ένα δίκτυο συνδέσεων (Interconnects), μονάδες εισόδου-εξόδου (IO Blocks), μνήμη και κάποιου είδους διαχείριση των σημάτων του εσωτερικού ρολογιού. Τα Control logic Blocks με τη σειρά τους αποτελούνται από D flip-flops, λογικούς πίνακες αναζήτησης (Look UP Table - LUT) και πολυπλέκτες (Multiplexer - mux) και πλήρεις αθροιστές (full adder) που προγραμματίζονται για μία συγκεκριμένη λειτουργία (Εικόνα 5). Με αυτό τον τρόπο ένα λογικό κελί μπορεί να προγραμματιστεί να κάνει διαφορετικά συνδυαστικά και ακολουθιακά κυκλώματα.



Εικόνα 5. Αφαιρετική αναπαράσταση των περιεχομένων ενός CLB

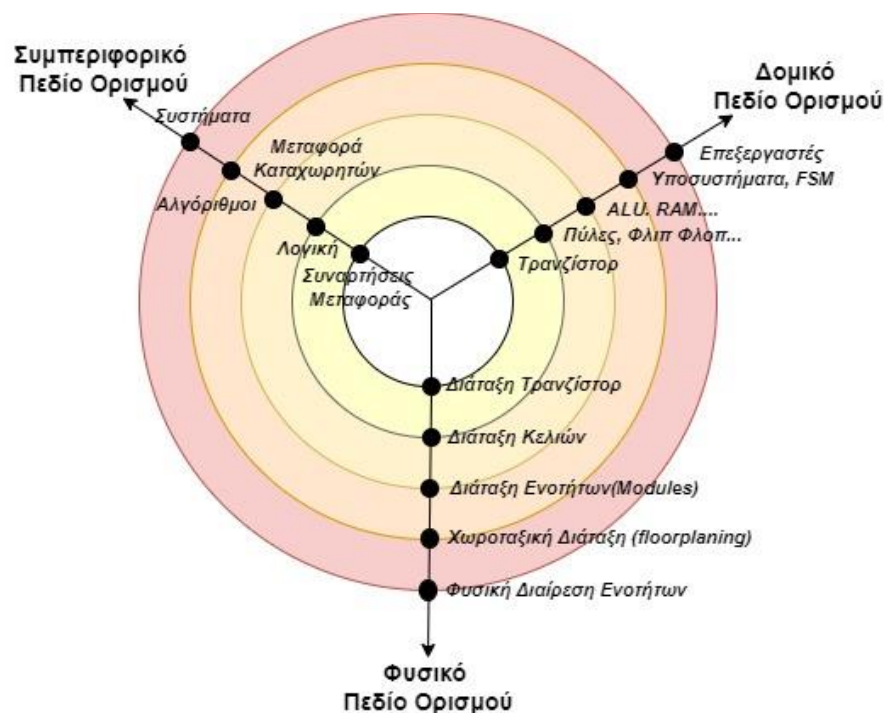
Το FPGA μπορεί να καθοριστεί από ένα συνδυασμό υλικού-λογισμικού, που ονομάζεται Gateware. Ο προγραμματισμός του FPGA με καινούργιο Gateware επιτρέπει την αλλαγή της λειτουργικότητας του.

Για την υλοποίηση ενός συγκεκριμένου λογικού κυκλώματος σε ένα FPGA μπορούν να χρησιμοποιηθούν διάφορες μεθοδολογίες. Η πιο ευρέως διαδεδομένη μέθοδος είναι η μετατροπή ειδικών γλωσσών περιγραφής υλικού (Hardware Description Languages) σε bitstream (Ροή Δεδομένων). Με αυτήν τη μέθοδο οι πόροι του FPGA μπορούν να επαναπρογραμματιστούν κατά βούληση. Οι πόροι αυτοί συνθέτουν ένα πολύπλοκο ιστό συνδυαστικών (combinational) και ακολουθιακών (sequential) κυκλωμάτων με σκοπό τη δημιουργία ψηφιακών λογικών σχεδίων.

Η απόδοση ενός λογικού κυκλώματος που προγραμματίζεται σε ένα FPGA εξαρτάται από την ποσότητα των πόρων που απαιτούνται καθώς και με τον τρόπο με το οποίο αυτά διανέμονται μέσα στο FPGA. Ακόμα κάθε FPGA κερδίζει τη θέση του στην αγορά σύμφωνα με χαρακτηριστικά του όπως ο αριθμός κυψελών μνήμης, οι μονάδες εισόδου/εξόδου και τα ενσωματωμένα σε αυτά δυνατοτήτων όπως mems, hard processing cores, buttons, seven segments κ.τ.λ.

## 2.2 Αφαιρετικά επίπεδα

Κατά τον σχεδιασμό λογικών ψηφιακών κυκλωμάτων οι μηχανικοί υλικού χρησιμοποιούν διαφορετικά αφαιρετικά επίπεδα τα οποία μπορούν να αντιπροσωπευτούν στο σύνολο τους από το διάγραμμα Gajski-Kuhn [13] (Εικόνα 6). Δημιουργήθηκε το 1983 από τους Robert Kuhn και Daniel Gajski και έπειτα βελτιώθηκε το 1985 από τους Robert Walker και Donald Thomas. Στο διάγραμμα χρησιμοποιούνται τρεις άξονες που αντιπροσωπεύουν τα αφαιρετικά επίπεδα του υλικού και οι τρεις άξονες μαζί δημιουργούν ένα Υ. Στο εσωτερικό του διαγράμματος υπάρχουν δακτύλιοι για να ενοποιούν τις αφαιρετικές όψεις των αξόνων. Οι εξωτερικοί δακτύλιοι έχουν το μεγαλύτερο επίπεδο αφαίρεσης και οι εσωτερικοί δακτύλιοι το μικρότερο επίπεδο αφαίρεσης.



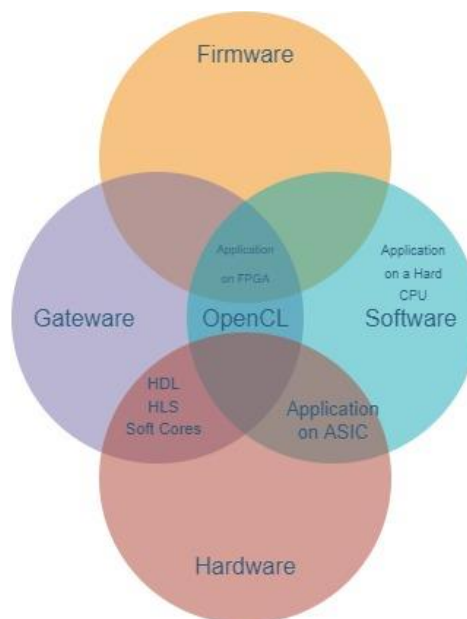
Εικόνα 6. Διάγραμμα Gajski-Kuhn

Κατά την ανάπτυξη ενός ψηφιακού λογικού κυκλώματος ένας μηχανικός υλικού συνήθως ξεκινάει από το μικρότερο αφαιρετικό επίπεδο και κινείται προς το μεγαλύτερο έχοντας την δυνατότητα να ορίζει το υλικό και τα κυκλώματα είτε σύμφωνα με το περιγραφή συμπεριφοράς ή με δομική περιγραφή.



## 2.3 Gateware

Το Gateware [14] είναι ένα υψηλό αφαιρετικό επίπεδο που δεν αναφέρεται στο διάγραμμα Gajski-Kuhn καθώς από αποτελεί μία υψηλή αναπαράσταση των λογικών ψηφιακών κυκλωμάτων που περιέχονται σε ένα FPGA. Στο Gateware περιέχεται η συμπεριφορά, η δομή και οι συνδέσεις (netlist) των ψηφιακών λογικών πυλών που μπορούν να επαναπρογραμματιστούν σε FPGA. Άρα ο κώδικας HDL είναι διαφορετικός από τα αρχεία προγραμματισμού ενός FPGA και αυτά με τη σειρά τους είναι διαφορετικά από μια υλοποίηση ενός λογικού σχεδίου γραμμένου σε HDL που λειτουργεί μέσα σε ένα FPGA (δηλ. το Gateware). Το Gateware δεν ανήκει στην κατηγορία του υλικολογισμικού αλλά περιέχει υλικολογισμικό. Η κατανόηση των επιπέδων του Gateware και των εννοιών που το περιβάλλουν βοηθά στην καλύτερη χρήση των διαφορετικών γλωσσών και εργαλείων που υπάρχουν για τη δημιουργία λογικών σχεδίων σε τεχνολογία FPGA/ASIC καθώς και στον σαφή διαχωρισμό των βημάτων που απαιτούνται από το design flow κατά την δημιουργία του Gateware. Συνοψίζοντας, το Gateware ορίζεται μέσα από τα επαναπρογραμματιζόμενα λογικά στοιχεία του FPGA μέσω των οποίων αποκτά διακριτά όρια.



Εικόνα 7. Τι είναι το Gateware;

## 2.4 Περιγραφή Ψηφιακών Κυκλωμάτων

Οι μηχανικοί υλικού μπορούν να χρησιμοποιήσουν τρεις διαφορετικού τρόπους για να περιγράψουν την υλοποίηση ενός ψηφιακού λογικού κυκλώματος [16]:

- Περιγραφή Συμπεριφοράς (Behavioral)

Με αυτό τον τρόπο περιγράφετε το κύκλωμα ως προς τη συμπεριφορά του. Περιγράφονται οι είσοδοι και οι έξοδοι καθώς και το τι πρέπει να γίνεται από την είσοδο στην έξοδο.

- Δομική Περιγραφή (Structural)

Με αυτό τον τρόπο περιγράφουμε δομικά ένα σχέδιο ως προς τα μέρη που το απαρτίζουν.

- Περιγραφή σε επίπεδο πυλών (Gate Level)

Σε αυτό το επίπεδο περιγράφουμε τις λογικές πύλες. Κανονικά ένας μηχανικός υλικού δεν θα πρέπει να αναπτύσσει λογικά κυκλώματα σε αυτό το επίπεδο λόγω του μεγάλου χρόνου που απαιτείται για την ανάπτυξη ενός κυκλώματος σε αυτό το επίπεδο και επειδή το έργο αυτό συνήθως το αναλαμβάνει ο μεταφραστής της γλώσσας περιγραφής υλικού.

Ο μηχανικός υλικού κατά τον σχεδιασμό λογικών ψηφιακών κυκλωμάτων έχει τη δυνατότητα να χρησιμοποιεί ένα τρόπο περιγραφής υλικού ή περισσότερους και ο τρόπος που το επιτυγχάνει αυτό είναι μέσω του αρθρωτού (modular) προγραμματισμού. Με αυτό τον τρόπο προγραμματισμού χρησιμοποιούνται κομμάτια κώδικα που υλοποιούν ψηφιακά κυκλώματα όπως αθροιστές, πολυπλέκτες κτλ.

Το SoC που δημιουργείται στο τέλος είναι ένα σύνθετο σύστημα ένθετων ψηφιακών λογικών κυκλωμάτων. Το σύστημα αυτό αποτελείται από ένθετα μπλοκ κώδικα που δημιουργούν άλλα ένθετα λογικά κυκλώματα.

## **2.5 Γλώσσες Περιγραφής Υλικού**

Οι γλώσσες περιγραφής υλικού υπάρχουν ως ιδέα από τη δεκαετία του '50. Πρακτικά όμως άρχισαν να χρησιμοποιούνται μετά το 1985 αφού αναπτύχθηκαν οι τεχνολογίες σχεδιασμού και κατασκευής λογικών κυκλωμάτων. Ιστορικά αναπτύχθηκαν διάφοροι τρόποι για να αναπαραστήσουν τη λογική συμπεριφορά των ψηφιακών κυκλωμάτων με την χρήση άλλοτε υψηλότερων και άλλοτε χαμηλότερων επιπέδων υλικού όπως οι εξισώσεις του Boole, διαγράμματα χρονισμών, πίνακες μετάβασης καταστάσεων και γλώσσες περιγραφής υλικού.

Μια γλώσσα περιγραφής υλικού είναι ένας τρόπος υψηλής αφαιρετικής αναπαράστασης της λογικής λειτουργίας των ψηφιακών κυκλωμάτων καθώς και των συστημάτων που αυτά ανήκουν. Συντακτικά μοιάζουν με τις ποιο γνωστές γλώσσες προγραμματισμού λογισμικού. Ο λόγος που χρησιμοποιούνται οι γλώσσες περιγραφής υλικού ως αφαιρετικό επίπεδο του υλικού είναι η σημαντική ελαχιστοποίηση του κύκλου σχεδίασης



με αποτέλεσμα να παράγονται πιο αξιόπιστες υλοποιήσεις λογικών σχεδίων σε σύγκριση με τις εναλλακτικές μεθόδους σχεδίασης λογικών κυκλωμάτων.

Μέσω των γλωσσών περιγραφής υλικού ο μηχανικός υλικού αναπτύσσει λογικά κατασκευάσματα που εμπίπτουν σε δύο βασικές κατηγορίες.

- Περιγραφικά Μοντέλα Ψηφιακών Λογικών Κυκλωμάτων
- Κώδικα δοκιμών (Testbenches) των παραπάνω λογικών κυκλωμάτων.

## **2.6 Οι Διαφορές των HDL μεταξύ τους.**

Οι γλώσσες περιγραφής υλικού χωρίζονται σε δύο (2) βασικές κατηγορίες [17].

- Γλώσσες Αμιγούς Περιγραφής Υλικού.

Οι γλώσσες αυτές περιγράφουν τη δομή και τη συμπεριφορά του υλικού από απλές λογικές πύλες μέχρι και σύνθετα περιγραφικά διαγράμματα ολοκληρωμένης λογικής μέσω τις σύνταξης και τις σημασιολογίας. Περιγράφουν τη λειτουργία λογικών κυκλωμάτων που μπορούν να κατασκευαστούν χρησιμοποιώντας διάφορα επίπεδα αφαίρεσης. Οι γλώσσες αυτές χρησιμοποιούνται ευρέως στη βιομηχανία και χαρακτηριστικά παραδείγματα τους είναι οι VHDL, Verilog, System Verilog.

- Γλώσσες λογισμικού ως Domain Specific Languages (DSL) περιγραφής υλικού

Οι γλώσσες αυτές μοιάζουν με τις αντίστοιχες του λογισμικού. Οι βασικές διαφορές τους συντακτικά και σημασιολογικά προέρχονται από τη χρήση τους ως γλώσσες περιγραφής υλικού. Από τη στιγμή που δεν περιγράφουν δεδομένα αλλά λογικά κυκλώματα πρέπει να υπάρχουν διασφαλίσεις και κανόνες που αφορούν τις ιδιαιτερότητες του υλικού. Ο τρόπος προγραμματισμού βάσει γεγονότων (event driven programming) αντικαθίσταται με έννοιες συνδυαστικών και ακολουθιακών δηλώσεων.

Με αυτό τον τρόπο γλώσσες, όπως η Python, μπορούν να δημιουργήσουν λογικά κυκλώματα. Με αυτό τον τρόπο οι μηχανικοί υλικού μπορούν να κάνουν χρήση των προγραμματιστικών δομών και σχεδιαστικών προτύπων (design patterns) της γλώσσας για να δημιουργήσουν καλά δομημένα και κυρίως επαναχρησιμοποιήσιμα λογικά κυκλώματα. Προσέχοντας τις ιδιαιτερότητες του υλικού, όπως η αναπαράσταση των ακεραίων μέσω της σωστής μαθηματικής αναπαράστασή τους. Άλλα προτερήματα που προκύπτουν από τη χρήση αυτής της μεθόδου περιγραφής υλικού είναι η χρήση του αντικειμενοστραφή προγραμματισμού, οι παράμετροι συναρτήσεων, οι γεννήτριες, η υπερφόρτωση χειριστή, οι βιβλιοθήκες κ.λπ.

Οι γλώσσες αυτές δε χρησιμοποιούνται στη βιομηχανία αλλά κυρίως σε ακαδημαϊκά ιδρύματα και λογικά σχέδια ανοιχτού κώδικα. Χαρακτηριστικά παραδείγματα τέτοιων

γλωσσών είναι οι amaranth, nMigen, migen βασισμένες στην Python και η Chisel, SpinalHDL που είναι βασισμένες στην Scala.

## 2.7 Migen

Η Migen [11] είναι γλώσσα περιγραφής υλικού για την κατασκευή πολύπλοκων ψηφιακών σχεδίων χρησιμοποιώντας την Python 3.5. Στη βιομηχανία αυτή τη στιγμή κυριαρχούν οι γλώσσες Verilog και VHDL. Η χρήση τους όμως είναι χρονοβόρα και αναποτελεσματική.

Οι σύγχρονες μορφές ψηφιακών σχεδίων, τα οποία αποτελούν την πλειονότητα των σύγχρονων αρχιτεκτονικών, χρησιμοποιούν τη προγραμματιστική μέθοδο που βασίζεται σε γεγονότα εισάγοντας με αυτό τον τρόπο περιπλοκές και ανθρώπινη κωδικοποίηση που δεν χρειάζονται. Αυτός ο τρόπος προγραμματισμού είναι δύσκολος στην εκμάθηση και χρειάζεται πολύς χρόνος να αποκτηθεί δεξιότητα. Επίσης οι προγραμματιστικές δομές τύπου «generate» είναι δύσχρηστες και ο κώδικας που αναπτύσσεται με αυτές δεν είναι φορητός ώστε να μεταφέρεται σε πολλαπλά λογικά σχέδια.

Τα παραπάνω προβλήματα λύνονται με τη χρήση γλωσσών που ενσωματώνουν τις έννοιες των συνδυαστικών και ακολουθιακών κυκλωμάτων και επιτρέπουν στους μηχανικούς υλικού να δημιουργούν οργανωμένα και κυρίως φορητά λογικά σχέδια χρησιμοποιώντας τις πλούσιες δομές του αντικειμενοστραφούς προγραμματισμού τις γλώσσας Python χρησιμοποιώντας τις κλάσεις και τις μεθόδους αυτών για να μετατρέψει τα αρχεία περιγραφής υλικού σε ένα υπερ-αντικείμενο που περιέχει όλη τη λογική.

Η βασική κλάση είναι αυτή του «Module». Η κλάση αυτή περιέχει πέντε βασικές μεθόδους μέσω των οποίων μπορεί να περιγραφεί οποιαδήποτε λογική ψηφιακού κυκλώματος. Οι μέθοδοι αυτοί είναι:

- **«Comb»** - Για τη δημιουργία Συνδυαστικής Λογικής.
- **«Sync»** - Για τη δημιουργία Ακολουθιακής Λογικής.
- **«Submodules»** - Προσθήκη Ιεραρχικού module λογικής.
- **«Specials»** - Για ειδικά κατασκευάσματα Λογικής που δεν μπορούν να περιγράψουν αφαιρετικά μέσω Comb/Sync. Κοινό παράδειγμα τέτοιας λογικής είναι η μνήμη και η κάθε αρχιτεκτονική την υλοποιεί διαφορετικά.
- **«Clock domains»** - Για ορισμό λογικών περιοχών που έχουν το ίδιο κύκλο ρολογιού.

## 2.8 Ανάπτυξη λογικών κυκλωμάτων σε VHDL, Verilog, Migen

Για την καλύτερη κατανόηση των διαφορών μεταξύ των γλωσσών παρακάτω γίνεται υλοποίηση ενός D flip flop σε VHDL, Verilog, Migen. Ένα D flip flop αποτελεί ένα από τα πρώτα κυκλώματα που μαθαίνει κάποιος σε μάθημα ψηφιακής λογικής σε προπτυχιακό επίπεδο. Πρόκειται για ένα ακολουθιακό κύκλωμα και η λειτουργία του είναι σχετικά απλή. Όταν το ρολόι του συστήματος γίνεται θετικό τότε το κύκλωμα flip flop γίνεται διάφανο και ότι δεδομένα υπάρχουν στην είσοδο καταλήγουν στην έξοδο. Παρακάτω υλοποιείται σε τρεις διαφορετικές γλώσσες:

### 1. VHDL

```
library ieee;
use ieee.std_logic_1164.all;
--ΟΝΤΟΤΗΤΑ(ENTITY) ΕΔΩ ΓΙΝΕΤΑΙ ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΥΛΙΚΟΥ
entity my_DFF is
    port(
        clk : in std_logic;
        output : out std_logic
    );
end entity;
-- ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΕΔΩ ΓΙΝΕΤΑΙ ΠΕΡΙΓΡΑΦΗ ΤΗΣ ΛΕΙΤΟΥΡΓΙΚΟΤΗΤΑΣ
architecture rtl of my_DFF is
    signal d : std_logic;
    signal q : std_logic;
begin
    --συνδιαστικό λογικό κύκλωμα
    output <= q;
    d <= not q;

    --ακολουθιακό λογικό κύκλωμα
    process(clk)
    begin
        if rising_edge(clk) then
            q <= d;
        end if;
    end process
end rtl;
```

**Κώδικας 1.** Υλοποίηση D-Flip Flop σε VHDL

### 2. Verilog HDL

```
//ΟΝΤΟΤΗΤΑ - ΠΕΡΙΓΡΑΦΗ ΥΛΙΚΟΥ
module my_Dff ( input d,clk,
                output reg q);
//ΑΡΧΙΤΕΚΤΟΝΙΚΗ - ΠΕΡΙΓΡΑΦΗ ΛΕΙΤΟΥΡΓΙΚΟΤΗΤΑΣ
//ΑΚΟΛΟΥΘΙΑΚΗ ΛΟΓΙΚΗ
always@(posedge clk)
begin
    q<=d;
end
endmodule
```

## Κώδικας 2. Υλοποίηση D-Flip Flop σε Verilog

### 3. Migen

```
from migen import *
class my_DFF(module):
    def __init__(self):
        self.output = Signal()
        d = Signal()
        q = Signal()
        self.comb += [self.output.eq(q), d.eq(~q)]
        self.sync += d.eq(q)
```

## Κώδικας 3. Υλοποίηση D-Flip Flop σε Migen

Με το παραπάνω παράδειγμα φαίνονται ξεκάθαρα οι διαφορές των γλωσσών ως προς τον τρόπο που υλοποιούν το ίδιο λογικό κύκλωμα.

## 2.9 Διαδικασία Ανάπτυξης Λογικών Σχεδίων σε FPGA

Κατά την ανάπτυξη ενός λογικού κυκλώματος σε ένα FPGA υπάρχουν δύο διακριτοί ρόλοι:

### 1. Υλοποίηση του Λογικού κυκλώματος (Implementation).

Οι μηχανικοί υλικού (digital design engineers) που αναλαμβάνουν την υλοποίηση του λογικού κυκλώματος μέσω υψηλών αφαιρετικών επιπέδων και κατεβαίνοντας επίπεδα μέχρι να υλοποιηθεί η λογική του κυκλώματος. Τα βήματα που απαιτούνται κατά τη διαδικασία σχεδίασης (design flow) είναι συνήθως:

- Συγγραφή κώδικα,
- Λογική σύνθεση,
- Περιορισμοί χρήστη από την υλοποίηση,
- Δρομολόγηση (Place and Route),
- Προγραμματισμός FPGA.

### 2. Πιστοποίηση του Λογικού κυκλώματος (Verification).

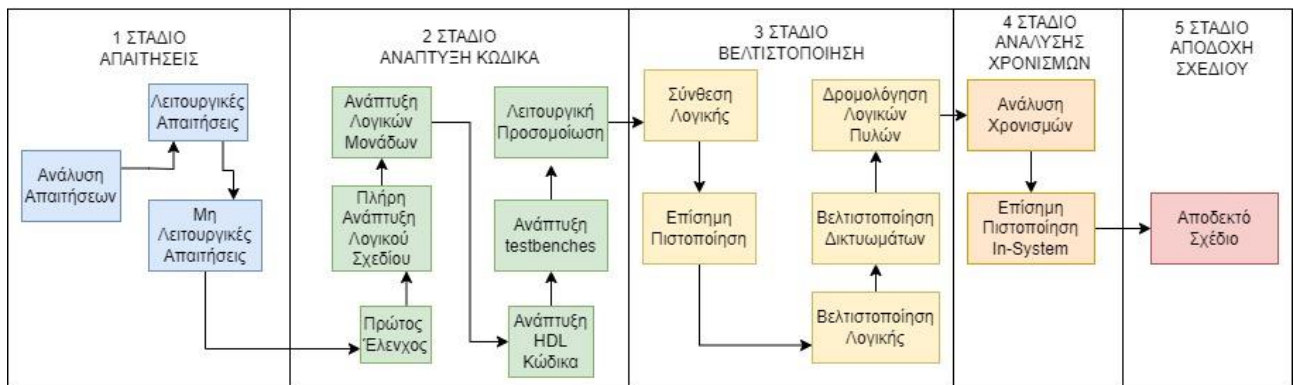
Οι μηχανικοί υλικού (digital verification engineers) που αναλαμβάνουν την πιστοποίηση του λογικού σχεδίου και των διαφόρων σταδίων της υλοποίησης αυτού χρησιμοποιούν διάφορες στρατηγικές για την επιτύχουν την σωστή και απρόσκοπτη πιστοποίηση των υπό δοκιμή ψηφιακών κυκλωμάτων. Παρακάτω αναλύονται περισσότερο κάποιες στρατηγικές όμως ένα ψηφιακό κύκλωμα πρέπει να είναι γενικά πιστοποιημένο ως προς:

- Τη συμπεριφορά του.
- Τη Λειτουργική Αξιοπιστία του.

- Τον χρονισμό του.
- Την εντός-κυκλώματος (in-circuit) συμπεριφορά του.

## 2.10 Στρατηγικές ελέγχου αξιοπιστίας λογικών σχεδίων

Το μέγεθος και η πολυπλοκότητα των προς υλοποίηση λογικών σχεδίων σε FPGA διαχρονικά αυξάνονται. Αυτή η τάση έχει οδηγήσει τις προσομοιώσεις λογικών κυκλωμάτων ανάλογα με την πολυπλοκότητα του λογικού σχεδίου, να χρειάζονται για να ολοκληρωθούν από ώρες μέχρι και μέρες [18-19].



**Εικόνα 8.** Στρατηγικές ελέγχου αξιοπιστίας λογικών σχεδίων σε FPGA

Η ανάπτυξη των λογικών σχεδίων σε FPGA ξεκινάει με την ανάλυση απαιτήσεων. Σε κάθε στάδιο (Εικόνα 8) υπάρχουν έλεγχοι και δοκιμές της παραγόμενης λογικής σε κάθε στάδιο. Η σύγχρονη αγορά οι μηχανικοί υλικού απαιτούν γρηγορότερες και πιο αξιόπιστες τεχνικές επαλήθευσης. Οι παραδοσιακές μέθοδοι μέσω των οποίων γίνεται ο έλεγχος της αξιοπιστίας είναι:

- Functional Simulation (Λειτουργική Προσομοίωση)

Η λειτουργική προσομοίωση είναι η πρώτη δοκιμή που περνάει ένα προς υλοποίηση σε FPGA λογικό σχέδιο. Σκοπός της προσομοίωσης αυτής είναι η δημιουργία ενός dut (design under test) σε ένα ένθετο μπλοκ κώδικα που ονομάζεται testbench και που ενσωματώνει το προς έλεγχο κύκλωμα καθώς και σήματα εισόδου όπου ελέγχεται η λειτουργικότητα του ψηφιακού κυκλώματος σε RTL (Register Transfer Logic) αφαιρετικό επίπεδο. Το μειονέκτημα αυτής της μεθόδου δοκιμής είναι ότι δε λαμβάνετε υπ' όψη η βελτιστοποίηση και υλοποίηση των πυλών μέσα στο FPGA κατά τη δοκιμή οπότε οι πληροφορίες χρονισμού που λαμβάνονται μέσω της δοκιμής δε θα πρέπει να λαμβάνονται υπ' όψη ως αξιόπιστες.

- Static Timing Analysis (Στατική Ανάλυση Χρονισμού)

Τα λογικά κυκλώματα που υλοποιούνται σε FPGA πρέπει να τηρούν συγκεκριμένους χρόνους καταστάσεων. Η στατική ανάλυση χρονισμού αποτελεί την de facto μέθοδο

δοκιμών λογικών σχεδίων για το έλεγχο του χρονισμού των λογικών κυκλωμάτων αν τηρούν τους χρόνους setup και hold.

- In-System Testing

Η τελική δοκιμή γίνεται με την υλοποίηση στο FPGA και έλεγχο του λογικού κυκλώματος αφού έχει υλοποιηθεί. Αποτελεί την πιο αξιόπιστη δοκιμή υλοποίησης και χρονισμού αλλά μπορεί να μην ανιχνεύσει όλα τα πιθανά προβλήματα όπως προβλήματα χρονισμού

- Formal Verification

Η επίσημη πιστοποίηση γίνεται σε ένα λογικό κύκλωμα συγκρίνοντας τα αποτελέσματα των δοκιμών με ένα μαθηματικό μοντέλο του υπό ελέγχου κυκλώματος. Κάθε ψηφιακό κύκλωμα για να χρησιμοποιηθεί στην αγορά θα πρέπει να έχει περάσει από επίσημη πιστοποίηση ώστε να μην υπάρχουν άσχημες εκπλήξεις ή δυσλειτουργίες του κατά τη χρήση του από τους τελικούς χρήστες.

Στις πιο σύγχρονες στρατηγικές ελέγχου ανήκουν που οι εξής:

- Assertion Based DUTs.

Δημιουργία testbenches με assertions. Τα asserts αποτελούν γλωσσικά κατασκευάσματα τις System Verilog και μπορούν να θέτουν ελέγχους και περιορισμούς στο υπό έλεγχο κύκλωμα.

- Universal Verification Methodology (UVM)

Αποτελεί μία σύγχρονη τυποποιημένη μέθοδο πιστοποίησης μέσω της γλώσσας System Verilog η οποία θέτει το λογικό κύκλωμα σε ένα σετ από περιβάλλοντα και δοκιμές.

- Open Source VHDL Verification Methodology (OSVVM)

Η παραπάνω μέθοδος χρησιμοποιεί ένα πλαίσιο VHDL για να υλοποιήσει τις δοκιμές ελέγχου.

- COCOTB (Coroutine Co-simulation Test Bench)

Η COCOTB είναι ένα πλαίσιο αυτοματισμού δοκιμών γραμμένο σε Python μέσω ελέγχου των προγραμμάτων που κάνουν την RTL προσομοίωση στα σύγχρονα HDL IDEs.

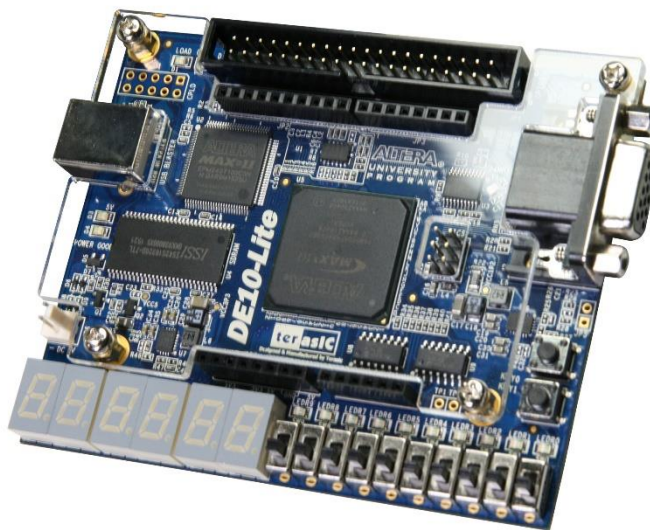
## 2.11 Οι μεγαλύτεροι κατασκευαστές FPGAs

Από τους μεγαλύτερους κατασκευαστές FPGA την πρώτη θέση [20] κατέχει η Xilinx με την μεγαλύτερη παραγωγή και πώληση FPGA σε διάφορα πακέτα και με πληθώρα διάφορων χαρακτηριστικών ικανών να ικανοποιήσουν διάφορες υπολογιστικές ανάγκες.

Επίσης έχει τη μεγαλύτερη κοινότητα συζητήσεων (forum) μηχανικών και δική της σειρά εργαλείων (toolchain) για την ανάπτυξη λογικών σχεδίων και τον προγραμματισμό τους σε FPGA. Στη δεύτερη θέση βρίσκεται η Intel όπου και αυτή έχει δική της σειρά εργαλείων (toolchain) για την ανάπτυξη λογικών σχεδίων και τον προγραμματισμό τους σε FPGA. Τις υπόλοιπες θέσεις διεκδικούν εταιρίες όπως η Infineon, Achronix, Microchip, Microsemi και Quicklogic. Άξια λόγου είναι η σειρά ICE40 της εταιρίας Lattice καθώς όλη η σειρά FPGA είναι ανοιχτού υλικού και η ανάπτυξη κώδικα καθώς και ο προγραμματισμός των FPGA γίνεται μέσω ανοιχτού λογισμικού.

## 2.12 Το De10lite της Intel

Η Intel έχει μία σειρά από γενιές FPGA που καλύπτουν τις ανάγκες της αγοράς. Η MAX 10 σειρά στηρίζεται στην NOR flash λιθογραφική μέθοδο των 55nm της TSMC. Διαθέτει 64mb RAM και 50.000 λογικά κελιά καθιστώντας το ανταγωνιστικό ως προς τις επιλογές χαρακτηριστικών\κόστους ειδικά για φοιτητές.



**Εικόνα 9.** Το DE10-Lite FPGA Board

Ο προγραμματισμός του γίνεται μέσω της ενσωματωμένης τεχνολογίας USB Blaster. Ακόμα διαθέτει επιταχυνσιόμετρο 3 αξόνων και δυνατότητες παραγωγής εικόνας μέσω του ενσωματωμένου VGA output, ενσωματωμένο διπλό ADC και τέσσερα PLL κυκλώματα για τη ρύθμιση του ρολογιού. Το DE10-Lite FPGA Board είναι αυτό που διαθέτουμε στο εργαστήριο και θα χρησιμοποιηθεί στην πτυχιακή αυτή.





### **3° ΚΕΦΑΛΑΙΟ: RISC-V ISA**

#### **3.1 Γιατί χρειαζόμαστε μία ανοιχτή αρχιτεκτονική**

Τη σημερινή εποχή οι RISC επεξεργαστές [21-26] χρησιμοποιούνται σε συσκευές που χρειάζονται χαμηλή κατανάλωση ισχύος όπως τα κινητά και οι IoT συσκευές.

Στο χώρο αυτό των επεξεργαστών RISC αρχιτεκτονικής κυριαρχεί η εταιρία ARM. Για χρήση επεξεργαστών ARM σε ολοκληρωμένα λογικά σχέδια όπως SoC ή για ολοκληρωμένα κυκλώματα για συγκεκριμένες εφαρμογές (ASIC), πρέπει πρώτα να ληφθεί άδεια. Ακόμα άδεια πολλές φορές χρειάζεται και το λογισμικό ανάπτυξης που συνοδεύει τους επεξεργαστές. Σε οικονομικούς όρους αυτό μπορεί να είναι εκατομμύρια ευρώ για τους πιο πρόσφατους πυρήνες και πρόσφατα συνοδευόμενα λογισμικά όπως μεταφραστές και περιβάλλοντα ανάπτυξης.

Επίσης η χρήση πυρήνων κλειστού κώδικα αποτελεί εμπορική συμφωνία. Μία εμπορική συμφωνία μπορεί επίσης να είναι αρκετά χρονοβόρα για να επιτευχθεί ειδικά σε διακρατικό επίπεδο και αυτό μεταφράζεται σε οικονομικό κόστος. Ακόμα για κάθε μικροσίπ που παράγεται με πνευματική ιδιοκτησία της ARM πρέπει ο κατασκευαστής να πληρώσει ένα τέλος στην ARM. Για μεγάλες πολυεθνικές όπως η NXP, η Freescale και η STM32 που παράγουν εκατομμύρια τέτοια μικροσίπ το κόστος αυτό αποτελεί ποσοστό του συνολικού κόστους ανάπτυξης ενός μικροσίπ.

Υπάρχει όμως εναλλακτική. Η χρήση ανοιχτών προδιαγραφών όπως η RISC-V αρχιτεκτονική. Επειδή το λογισμικό και άδειες που απαιτούνται είναι ανοιχτές αυτό σημαίνει πρακτικά σημαντικά μικρότερο κόστος ανάπτυξης.

#### **3.2 Η Ιστορία της RISC-V ISA**

Η ιστορία της αρχιτεκτονικής ξεκινά το 2010 στο UC Berkeley Parallel Computing Lab, του οποίου Διευθυντής ήταν ο David Patterson. Εκεί υπήρχε μία ομάδα που αποτελούνταν από τους Καθηγητή Krste Asanović και τους πτυχιούχους Yunsup Lee και Andrew Waterman όπου στα πλαίσια ενός καλοκαιρινού εργαστηρίου ξεκίνησαν την δημιουργία του πρώτου σετ εντολών (instruction set) του RISC-V συνόλου εντολών αρχιτεκτονικής (ISA). Η αρχιτεκτονική δεν είναι «ανοιχτού κώδικα» με τον ίδιο τρόπο που είναι ανοιχτό το λογισμικό. Η RISC-V αρχιτεκτονική αποτελείται από έγγραφα προδιαγραφών, όχι κώδικα. Περιέχει δηλαδή έγγραφα που περιγράφουν αφαιρετικά την λογική κάποιων υψηλού επιπέδου ψηφιακών κυκλωμάτων. Καθώς δεν περιέχει πηγαίο κώδικα η υλοποίηση των προδιαγραφών επαφίεται στον μηχανικό υλικού. Η

προδιαγραφές λειτουργούν ως σχεδιαγράμματα για το πως θα έπρεπε να λειτουργεί ένα ψηφιακό λογικό κύκλωμα.

Η αρχιτεκτονική, κυκλοφορεί και διανέμεται μέσω των αδειών Creative Commons. Αυτό σημαίνει πως η RISC-V είναι μία ανοιχτή προδιαγραφή. Δηλαδή πως το λογισμικό (όπως οι compilers, linkers κ.α.), οι τεχνικές προδιαγραφές, τα σχεδιαγράμματα και οι άδειες συμμόρφωσης, χρησιμοποιούν ανοιχτές άδειες χρήσης όπως BSD, MIT ώστε η RISC-V αρχιτεκτονική να είναι διαθέσιμη σε όλους χωρίς έξτρα κόστος.

Ο όρος RISC-V μπορεί να σημαίνει πολλά διαφορετικά πράγματα ανάλογα με το πλαίσιο αναφοράς και τα συμφραζόμενα, όπως:

- Η ίδια η Αρχιτεκτονική (ISA).
- Η κοινότητα χρηστών και προγραμματιστών
- Το λογισμικό που αναπτύσσετε για την αρχιτεκτονική.
- Η Διεθνής Ένωση RISC-V που είναι μία νομική ένωση προσώπων.
- Τα παραγόμενα από την ανοιχτή αρχιτεκτονική.

Όταν αναφέρεται ο όρος RISC-V πάντα νοείται η ανοιχτή αρχιτεκτονική καθώς και οι προδιαγραφές της. Όλες οι υπόλοιπες έννοιες ανήκουν στην RISC-V αφού περιβάλλουν νοητά την ανοιχτή αρχιτεκτονική.

### **3.3 Η Αρχιτεκτονική και Οργάνωση της RISC-V ISA**

Το πανεπιστήμιο του U.C. Berkley ήταν διαχρονικά πρωτοπόρο στον σχεδιασμό υπολογιστών με επεξεργαστές με Μειωμένο Σετ Εντολών (RISC). Καθώς η αρχιτεκτονική RISC-V μοιάζει με την MIPS (Microprocessor without Interlocked Pipelined Stages) αρχιτεκτονική καθώς αναπτυσσόταν ταυτόχρονα στο πανεπιστήμιο του Στάνφορντ την δεκαετία του 1980. Και οι δύο αρχιτεκτονικές επεξεργαστών βρήκαν τη θέση τους στην αγορά με τους επεξεργαστές με μειωμένο σετ εντολών (RISC) να κατέχουν ιδιαίτερη θέση λόγω της εξαιρετικά μειωμένης ενεργειακής κατανάλωσης που προσφέρουν. Η πρώτη φορά που παρουσιάζονται λογικά κυκλώματα επεξεργαστών που χρησιμοποιούν το μειωμένο σετ εντολών RISC είναι το 1980. Μερικά παραδείγματα επιτυχημένων επεξεργαστών με μειωμένο σετ εντολών (RISC) είναι η σειρά SPARC της Sun Microsystems, η σειρά επεξεργαστών Alpha της DEC, οι επεξεργαστές i860 και i960 της Intel καθώς και οι επεξεργαστές ARM.

Το γράμμα «V» στον όρο RISC-V είναι στα λατινικά το νούμερο πέντε καθώς ιστορικά κατά την ανάπτυξη αρχιτεκτονικής υπήρξαν πέντε σημαντικά σημεία που αποτελούν

ορόσημα: RISC-I, RISC-II, SOAR, SPUR και RISC-V. Επίσης φημολογείται πως αφού μία από τις προθέσεις του ιδρύματος RISC-V είναι να αναπτύξει τις μεθόδους που θα επιτρέψουν στην αρχιτεκτονική να κυριαρχήσει στον αγώνα της παράλληλης επεξεργασίας δεδομένων κάποιοι υποθέτουν ότι και το γράμμα «V» χρησιμοποιείται ως φωνητικό συνώνυμο της λέξης «Vectors».

### **3.4 RISC-V International**

Η «RISC-V International» ιδρύθηκε το 2015 ως μία εταιρεία με μη κερδοσκοπικό χαρακτήρα. Ο σκοπός της RISC-V International είναι να δημιουργήσουν μία ομάδα από επιστήμονες, μηχανικούς και προγραμματιστές, όπου αυτήν την στιγμή προέρχονται από περισσότερες από 40 χώρες, που θα εξελίσουν την ανοιχτή αρχιτεκτονική στις σύγχρονες ανάγκες έρευνας και αγοράς.

Αν και η RISC-V International μετέφερε τον Μάρτιο του 2020 την έδρα της στην Ελβετία στην πραγματικότητα δεν έχει βάση καμία χώρα. Το κάθε μέλος της έχει το δικαίωμα της συμμετοχής κατά τη δημιουργία και εξέλιξη της ανοιχτής αρχιτεκτονικής καθώς και των επεκτάσεων αυτής. Ακόμα το κάθε μέλος του ιδρύματος έχει δικαίωμα να εκλέξει και να εκλεγεί ως μέλος του Διοικητικού Συμβουλίου. Μπορεί δηλαδή το κάθε μέλος να συμμετέχει ενεργητικά στη διοίκηση της RISC-V International. Εκτός αυτού κάθε μέλος έχει δικαίωμα να οριστεί ως επικεφαλής ομάδας εργασίας. Το δικαίωμα αυτό του δίνει την δυνατότητα να οριστεί μελλοντικά και ως μέλος Τεχνικής Επιτροπής.

Τα πνευματικά δικαιώματα που παράγονται από το ίδρυμα μέσω του υλικού και του λογισμικού που αναπτύσσονται από τους επιστήμονες και τους μηχανικούς έχουν κατά βάση άδειες ανοιχτού κώδικα ώστε να μην υπάρχει κάποιος νομικός ή γεωγραφικός περιορισμός κατά τη χρήση τους.

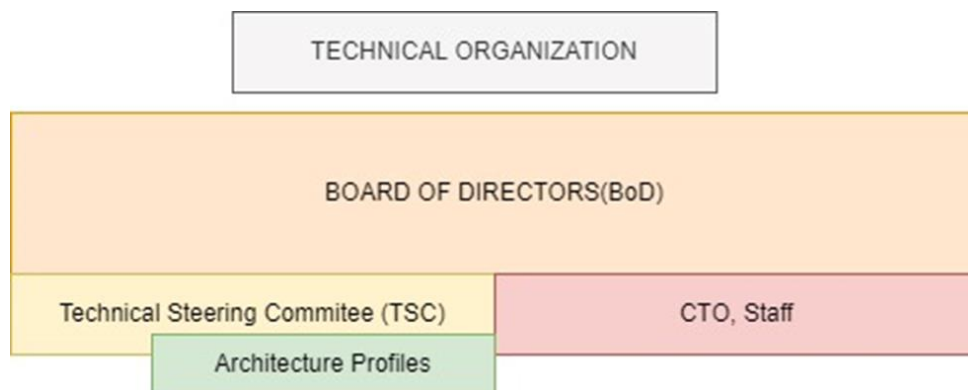
### **3.5 RISC-V Community**

Η Κοινότητα της ανοιχτής Αρχιτεκτονικής αποτελείται από τα μέλη του ιδρύματος, που προέρχονται από όλο τον κόσμο. Τα μέλη είναι υπεύθυνα για την περαιτέρω ανάπτυξη και εξέλιξη της ανοιχτής αρχιτεκτονικής RISC-V. Η πολυπολιτισμική αυτή ομάδα που αποτελείται πάνω από 2.000 άτομα σε πάνω από 70 χώρες, όπου ανάμεσά τους βρίσκεται και ο συγγραφέων. Πολλά μέλη της κοινότητας εκπροσωπούν πολλούς διαφορετικούς οργανισμούς και εταιρίες ανάμεσα τους η Google, Intel, Huawei, Alibaba Group που βρίσκουν πως η RISC-V αρχιτεκτονική αποτελεί το μέλλον.

Το έργο της ανάπτυξης της αρχιτεκτονικής παρότι η RISC-V International έχει υπαλλήλους επαφίεται στα μέλη της. Για να μπορέσει να γίνει κάποιος μέλος της Κοινότητας θα πρέπει για αρχή να συμφωνήσει ότι οι στόχοι του και η εργασία που θα προσφέρει εφάπτονται με αυτούς της RISC-V International και πως κατανοεί τις ανοιχτές άδειες που διέπουν την RISC-V αρχιτεκτονική. Αυτό επιτυγχάνεται μέσω της σύμβασης που όλα τα μέλη είναι υποχρεωμένα να υπογράψουν.

### 3.6 RISC-V Technical Organization

Η οργάνωση στην RISC-V International έχει κάθετα και οριζόντια επίπεδα και προκύπτει από τους επικεφαλής του εκάστοτε επιπέδου. Η οργάνωση αυτή έχει άμεση σχέση με την ανάπτυξη και υλοποίηση των προδιαγραφών της αρχιτεκτονικής.



**Εικόνα 10.** Τεχνική Οργάνωση RISC-V

Η κάθε προδιαγραφή της RISC-V αρχιτεκτονικής αρχίζει όταν η Διευθύνουσα Τεχνική Επιτροπή (TSC) δημιουργεί μία Ομάδα Εργασίας.

Από τη στιγμή που η ομάδα Εργασίας πάρει επίσημη έγκριση ξεκινάει την ανάπτυξη της προδιαγραφής καθώς δημιουργούν πρώτα ένα δημόσιο αποθετήριο (repository) στο GitHub. Η διαδικασία της ανάπτυξης της προδιαγραφής γίνεται σε αυτό το αποθετήριο. Κοινοποιούν τη δουλειά τους μέσω εγγράφων και τα έγγραφα τους τα αποθηκεύουν σε μορφή AsciiDoc στο αποθετήριο. Αυτά τα αποθετήρια στο GitHub μπορούν να λαμβάνουν μόνο αιτήματα έλξης (pull repository) μόνο από μέλη της κοινότητας. Επειδή όμως τα δημόσια αποθετήρια στο GitHub είναι δημόσια για όλους για αυτό μπορεί ο καθένας να δει δημόσια την ανάπτυξη της προδιαγραφής. Όσες ομάδες κάνουν συναντήσεις και αυτές οι συναντήσεις παράγουν έργο τότε καταγράφουν το περιεχόμενο των συναντήσεων αυτών και διανέμουν το περιεχόμενο των συναντήσεων ώστε να διατίθεται ελεύθερα. Και πάλι επειδή τα δημόσια αποθετήρια στο GitHub είναι δημόσια για όλους μπορεί ο καθένας χρησιμοποιώντας την πλατφόρμα του GitHub να εγείρει ερωτήσεις και σχόλια γύρω από την προδιαγραφή.

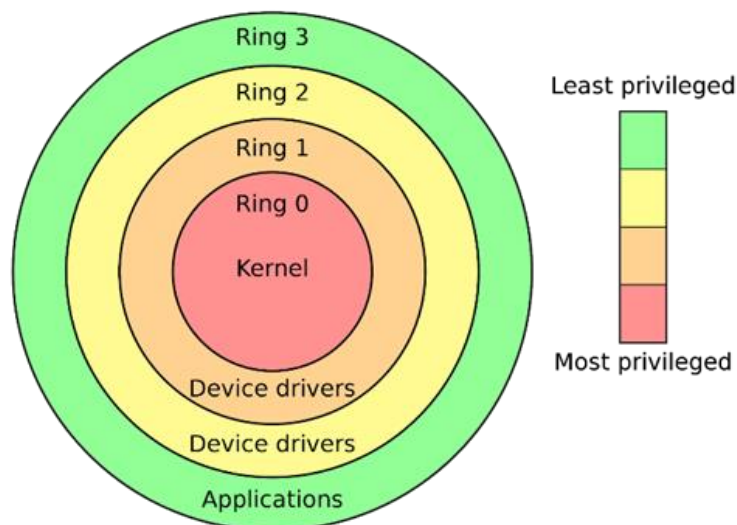
### 3.7 RISC-V CPUs

Έχουν αναπτυχθεί πολυάριθμοι πυρήνες RISC-V με διάφορες υλοποιήσεις των προδιαγραφών σε παγκόσμιο επίπεδο κυρίως από πανεπιστημιακά ιδρύματα. Ο κώδικας που έχουν αναπτυχθεί είναι γραμμένος σε πολλές διαφορετικές γλώσσες περιγραφής υλικού (HDL) όπως VHDL, Verilog, System Verilog. Οι αποδόσεις αυτών των επεξεργαστών είναι πλέον εφάμιλλες αυτών τις ARM. Στα πλαίσια έρευνας κατά την συγγραφή της παρούσας διαπιστώθηκε ότι η πιο σύγχρονη λίστα με νέους RISC-V επεξεργαστές μπορεί να βρεθεί στην ιστοσελίδα της RISC-V International [28]. Ταυτόχρονα στο GitHub βρίσκεται από την RISC-V International ένα αρχειοθετημένο αποθετήριο που περιέχει τους legacy [29] πυρήνες.

### 3.8 RISC-V Specifications

Η ανοιχτή αρχιτεκτονική RISC-V χωρίζεται σε δύο προδιαγραφές:

- Τη Μη προνομιακή (Unprivileged)
- Την Προνομιακή (Privileged)



**Εικόνα 11.** Δακτύλιοι δικαιωμάτων για αρχιτεκτονικές x86 [30]

Για να γίνει κατανοητό γιατί η αρχιτεκτονική έχει δύο προδιαγραφές πρέπει πρώτα να γίνει κατανοητό πως η ασφάλεια του επεξεργαστή ξεκινάει από την αρχιτεκτονική και στον τρόπο που αυτή υλοποιείται.

Διαχρονικά ο σχεδιασμός επεξεργαστών γίνεται με πολλούς τομείς ιεραρχικής προστασίας, που ονομάζονται δακτύλιοι προστασίας. Οι δακτύλιοι αυτοί υπάρχουν για να προστατεύουν τα δεδομένα και την ορθή εκτέλεση του κώδικά τους από κακόβουλους χρήστες.

Ο επεξεργαστής καθορίζει τα δικαιώματα που θα εκχωρηθούν στον εκτελούμενο κώδικα με βάση το επίπεδο άδειας. Ο πιο προνομιακός κώδικας εκτελείται στο "ring 0" και έχει πρόσβαση σε όλο το σύστημα του επεξεργαστή. Αυτό γίνεται ώστε να μπορεί να περιοριστεί η πρόσβαση κακόβουλου κώδικα στη μνήμη μέσω μιας φυσικής διεύθυνσης στο "ring 0". Σε αυτό το επίπεδο κανονικά δουλεύει ο bootloader και ο πυρήνας του λειτουργικού. Ο κώδικας που εκτελείται με λιγότερα προνόμια εκτελείται σε άλλους δακτυλίους που δεν βλέπουν τη φυσική μνήμη αλλά μία ανακατεμένη έκδοση της στον εικονικό χώρο διευθύνσεων. Στην Εικόνα 11, φαίνεται μία πως μία τυπική αρχιτεκτονική χρησιμοποιεί τους δακτυλίους. Κανονικά, ένας επεξεργαστής μπορεί να εκτελείται μόνο σε μία λειτουργία κάθε φορά και υπάρχουν ειδικές οδηγίες και κανόνες για την εναλλαγή των λειτουργιών και πως ο κώδικας εκτελείται σε κάθε επίπεδο. Όλες αυτές οι λεπτομέρειες για τη λειτουργία των δακτυλίων μπορεί να διαφέρουν από σύστημα σε σύστημα και από αρχιτεκτονική σε αρχιτεκτονική. Για την ομαλή εκτέλεση του κώδικα πρέπει να τηρούνται οι κανόνες που ορίζονται στις προδιαγραφές για την σωστή εκτέλεση του κώδικα σε κάθε δακτύλιο σε κάθε αρχιτεκτονική.

**Πίνακας 1.** Επίπεδα προνομίων

Ring	Κωδιση		Όνομα	Συντλφία
0	00		USER	U
1	01		SUPERVISOR	S
2	10		RESERVED	
3	11		MACHINE	M

Η RISC-V αρχιτεκτονική συγκεκριμένα, έχει τρία επίπεδα προνομίων:

- Τη λειτουργία χρήστη (U),
- Τη λειτουργία επόπτη (S) και
- Τη λειτουργία μηχανής (M).

Οι λειτουργίες με την κωδικοποίηση τους φαίνονται στον Πίνακα 1. Η λειτουργία hypervisor (λειτουργία H) δεν έχει υλοποιηθεί ακόμα παρότι προβλέπεται και αυτήν τη στιγμή βρίσκεται υπό ανάπτυξη.

Για να διατηρηθεί η σωστή εκτέλεση του κώδικα υπάρχουν ειδικές θέσεις μνήμης που ονομάζονται καταχωρητές κατάστασης (Control Status Registers - CSRs). Κάθε επίπεδο μπορεί να έχει πρόσβαση σε ένα συγκεκριμένο σετ καταχωρητών CSR. Τα υψηλότερα επίπεδα προνομίων έχουν πρόσβαση στους καταχωρητές CSR χαμηλότερου επιπέδου προνομίων. Το αντίθετο όμως δεν ισχύει.

### 3.8.1 Unprivileged Specification

Η μη προνομιακή προδιαγραφή ανήκει στη βασική αρχιτεκτονική και έχει τα λιγότερα προνόμια από όλες τις προδιαγραφές και περιγράφει στοιχεία που δε σχετίζονται με τη λειτουργία μηχανής (M) και επόπτη (S). Η προδιαγραφή αυτή περιέχει την επέκταση B που περιέχει και τις επεκτάσεις του Ακέραιου (I), κινητής υποδιαστολής (F), αριθμού κινητής υποδιαστολής διπλής ακρίβειας (D).

**Πίνακας 2.** Βασική Αρχιτεκτονική RISC-V

Όνομα	Λειτουργία	GCC Macro
RV32I	Σετ εντολών ακεραίων με 32bit & 32 καταχωρητές	__riscv __riscv_xlen=32
RV32E	Σετ εντολών ακεραίων για ενσωματωμένες συσκευές με 32bit & 16 καταχωρητές	__riscv __riscv_32e __riscv_xlen=32
RV64I	Σετ εντολών ακεραίων με 64bit & 32 καταχωρητές	__riscv __riscv_xlen=64
RV128I	Σετ εντολών ακεραίων με 128 bit & 32 καταχωρητές	

Το βασικό σύνολο εντολών (Base Instruction Set ) είναι αυτό που περιγράφει την ελάχιστη δυνατή λειτουργικότητα για την υλοποίηση ενός επεξεργαστή RISC-V. Ανάλογα με το μέγεθος των εντολών του επεξεργαστή της ανοιχτής αρχιτεκτονικής RISC-V χωρίζουμε και τις επεκτάσεις άλλες βασικές λεπτομέρειες όπως οι καταχωρητές. σύμφωνα με τον Πίνακα 2. Η πλειοψηφία των σύγχρονων λειτουργικών γενικής χρήσης χρειάζονται το βασικό σύνολο εντολών ISA και αυτό σημαίνει ότι χρειάζονται τις επεκτάσεις mafd για να λειτουργήσουν.

Ωστόσο, οι πυρήνες των λειτουργικών που προορίζονται για ενσωματωμένα συστήματα και οι πυρήνες λειτουργικών ειδικού σκοπού δεν απαιτούν τεχνικά αυτές τις επεκτάσεις. Οι καταχωρητές CSR και η εικονική μνήμη θα απαιτηθούν από ένα λειτουργικό σύστημα γενικής χρήσης. Δεν αποτελούν μέρος του βασικού ISA, επιτρέποντας στους πυρήνες με το μικρότερο δυνατό αποτύπωμα να είναι συμβατοί με τα ενσωματωμένα συστήματα.

Για την καλύτερη κατανόηση του διαχωρισμού έστω ότι όλες οι επεκτάσεις που περιέχονται στην βασική αρχιτεκτονική (Base ISA) είτε μειώνουν είτε επεκτείνουν το σετ εντολών βάσης RV32I. Ομοίως, το σετ εντολών RV64I διευρύνει τους καταχωρητές ακεραίων και τον υποστηριζόμενο χώρο διευθύνσεων χρήστη σε 64 bit αυτό έχει ως αποτέλεσμα κάποιες από τα μνημονικά εντολών Assembly όπως οι LOAD και STORE εντολές να λειτουργούν λίγο διαφορετικά από ό,τι στο RV32I.

Η μη προνομιακή προδιαγραφή αναλύει τις διαφορές του βασικού συνόλου εντολών με περισσότερη ακρίβεια.

### 3.8.2 Base ISA Extensions

Οι περιγραφές των επεκτάσεων της βασικής αρχιτεκτονικής (Πίνακας 2) περιέχονται στην μη προνομιακή προδιαγραφή. Παρακάτω παρατίθενται επικυρωμένες επεκτάσεις που υλοποιούνται σε επεξεργαστές σύμφωνα με τον Πίνακα 3:

- Τυπική επέκταση "M".

Η μη προνομιακή προδιαγραφή περιγράφει τον τρόπο με τον οποίο θα πρέπει να επιτυγχάνεται ο πολλαπλασιασμός και η διαίρεση ακεραίων. Στην επέκταση αυτή ανήκουν οι εντολές πολλαπλασιασμού (MUL, MULH, MULHU, MULHU, MULW). Σε αυτή την επέκταση ορίζεται ο πολλαπλασιασμός μεταξύ αριθμών στον επεξεργαστή. Ο ορισμός του πολλαπλασιασμού σημαίνει και ορισμός της διαίρεσης καθώς είναι αντίθετη μαθηματική πράξη. Η επέκταση αυτή δεν είναι απαραίτητη καθώς μπορεί να υλοποιηθεί σε ανώτερο επίπεδο όπως του λογισμικού.

**Πίνακας 3.** Standard ISA Extensions.

	Γράμμα	Λειτουργία
G	M	Integer multiplication and division
	A	Atomic Instructions
	F	Single precision floating point
	D	Double precision floating point
	Q	Quad precision
	L	Decimal floating point
	C	Compressed instructions (16 bit instructions)
	B	Bit manipulation
	J	Dynamically translated languages
	T	Transactional memory
	P	Packed SIMD instructions
	V	Vector operations
	N	User level interrupts
	H	Hypervisor
	G	Additional standard extensions present
	ZAM	Standard Extension for Misaligned Atomics



	Γράμμα	Λειτουργία
	Zicsr	Control and Status Register (CSR) Instructions
	Zifencei	Instruction-Fetch Fence
	Ztso	Standard Extension for Total Store Ordering

- Τυπική επέκταση "F".

Όπως και στην προηγούμενη επέκταση η επέκταση F δεν είναι απαραίτητο να υλοποιηθεί. Στην επέκταση αυτή ορίζονται οι εντολές κινητής υποδιαστολής μονής ακρίβειας όπως αυτές ορίζονται στο πρότυπο 754-2008 της IEEE. Επίσης η τυπική επέκταση Q ορίζει εντολές για αριθμούς κινητής υποδιαστολής τετραπλής ακρίβειας 128 bit.

- Τυπική επέκταση "C".

Στην επέκταση αυτή ορίζεται η συμπιεσμένη μορφή των επεκτάσεων συνόλου εντολών. Η συμπιεσμένη μορφή επεκτάσεων γίνεται για να υπάρχει μεγαλύτερη πυκνότητα κώδικα (code density) καθώς με αυτό τον τρόπο αυξάνουμε την ταχύτητα επεξεργασίας. Μετατρέπουμε δηλαδή τις εντολές που χρησιμοποιούνται ποιο συχνά σε εντολές μεγέθους 16 bit.

### 3.8.3 Privileged Specification

Η προνομιακή προδιαγραφή είναι αυτή που περιγράφει την λογική εκείνη που είναι απαραίτητη για να εκτελεστεί κώδικας με αυξημένα προνόμια σε λειτουργία Machine Mode (M-mode) ή Supervisor Mode (S-mode). Στη λογική που περιγράφετε στην προνομιούχα προδιαγραφή εμπεριέχονται οι καταχωρητές CSR του επιπέδου αυτού που είναι απαραίτητοι για να καλυφθούν οι προδιαγραφές για εκτέλεση πυρήνων λειτουργικών όπως το Linux.

### 3.8.4 Machine-Level (M-Mode) ISA, Version 1.11

Η επέκταση αυτή περιλαμβάνει τη λογική εκείνη που έχει τα ποιο μεγάλα προνόμια σε σχέση με τα υπόλοιπα επίπεδα. Ο κώδικας που θα έπρεπε να εκτελείτε σε αυτό το επίπεδο είναι ο κώδικας αρχικοποίησης του επεξεργαστή (bootstrapping). Η εύρυθμη λειτουργία του επεξεργαστή εξαρτάται από χρονοδιακόπτες (watchdog counters) και συστήματα λογικής που βρίσκονται σε αυτό το επίπεδο όπου σε περίπτωση κρίσιμου λάθους προκαλεί είτε την επανεκκίνηση (reset) του επεξεργαστή είτε την εκτέλεση επεμβατικού κώδικα.

Η λογική που περιγράφεται σε αυτή την προδιαγραφή έχει τρεις ιδιαίτερα σημαντικές λειτουργίες για την ασφάλεια του εκτελούμενου κώδικα:

- Διακοπές χωρίς μάσκα (Non-Maskable Interrupts - NMIs),
- Χαρακτηριστικά φυσικής μνήμης (Physical Memory Attributes - PMA), και
- Προστασία φυσικής μνήμης (Physical Memory Protection - PMP).

#### *3.8.5 Non-Maskable Interrupts (NMIs)*

Κατά την εκτέλεση κώδικα σε ένα επεξεργαστή πολλές φορές προκαλούνται λάθη που μπορούν να οδηγήσουν από λανθασμένη εκτέλεση προγραμμάτων μέχρι και την ολική διακοπή της ροής εκτέλεσης (halt). Ένας τρόπος αποφυγής της κατάστασης αυτής είναι μέσω των διακοπών χωρίς μάσκα (NMI). Όταν προκληθεί ένα λάθος που ενεργοποιεί την διακοπή τότε προκαλείται εκτέλεση κώδικα για τον χειρισμό του λάθους ώστε να συνεχίσει η κανονική ροή εκτέλεσης. Ο κώδικας που εκτελείται δεν μπορεί να αλλάξει την κατάσταση που βρίσκονται οι καταχωρητές του επεξεργαστή επιτρέποντας με αυτόν τον τρόπο την εκσφαλμάτωση. Η σωστή διάγνωση και αναφορά των λαθών σε συνδυασμό με της διακοπές υλικού οδηγούν στον περιορισμό των σφαλμάτων υλικού.

#### *3.8.6 Physical Memory Attributes (PMA)*

Ο χάρτης φυσικής μνήμης σε ένα λογικό κύκλωμα επεξεργαστή αποτελείται από κομμάτια μνήμης (memory ranges) που περιλαμβάνουν ελεύθερες και δεσμευμένες περιοχές μνήμης και καταχωρητές. Στην ενότητα αυτή της προδιαγραφής ορίζονται και οι ιδιότητες των κρυφών μνημών, τις συνοχής μνήμης καθώς και της ασφάλειας και του ελέγχου. Καθότι υπάρχουν πολλοί διαφορετικοί τρόποι οργάνωσης της μνήμης και πολλοί τρόποι επικοινωνίας των περιφερειακών με αυτή χρησιμοποιείται η προδιαγραφή αυτή για να ορίσει τις ιδιότητες της μνήμης στο λογικό κύκλωμα του επεξεργαστή.

Οι λεπτομέρειες των PMA θα μπορούσαν εύκολα να λάβουν ένα ολόκληρο κεφάλαιο αυτού του μαθήματος. Δε θα καλύψουμε PMA σειράς μνήμης, PMA αδυναμίας, PMA συνοχής ή PMA κρυφής μνήμης. Οι λεπτομέρειες των PMA περιγράφονται λεπτομερώς στην Προνομιακή Προδιαγραφή [31]. Οι προχωρημένοι χρήστες μπορεί να θέλουν να ελέγξουν αυτήν την ενότητα.

#### *3.8.7 Physical Memory Protection (PMP)*

Ο τρόπος πρόσβασης στη μνήμη, όπως αναφέρθηκε και προηγουμένως, αποτελεί διαχρονικά την τρόπο μέσω του οποίου διασφαλίζεται η ασφαλής εκτέλεση κώδικα. Όλες

οι σύγχρονες αρχιτεκτονικές έχουν ένα τρόπο εκτέλεσης ασφαλούς υπολογισμού. Ο τεχνικός όρος είναι «αξιόπιστο περιβάλλον εκτέλεσης (Trusted Execution Environment - TEE)». Όλοι οι μεγάλοι κατασκευαστές επεξεργαστές υλοποιούν ένα τέτοιο αξιόπιστο περιβάλλον με τον δικό τους τρόπο. Η Intel έχει τις Software Guard Extensions (SGX), η AMD έχει το Secure Encrypted Virtualization (SEV) και η Arm έχει την Trust Zone. Η ανοιχτή αρχιτεκτονική παρότι δεν έχει δική της υλοποίηση προσφέρει μέσω του μηχανισμού προστασίας μνήμης (Physical Memory Protection - PMP) στους μηχανικούς υλικού να υλοποιήσουν το δικό τους ασφαλές περιβάλλον. Αυτό επιτυγχάνεται μέσω του ελέγχου των φυσικών διευθύνσεων καθώς και την πρόσβαση που θα έχει ο κώδικας στις διευθύνσεις μνήμης.

#### *3.8.8 Supervisor-Level (S-Mode) ISA, Version 1.11*

Τέλος περιγράφεται η επέκταση που ορίζει τη λειτουργία επόπτη στην ανοιχτή αρχιτεκτονική. Σκοπός αυτού του επιπέδου είναι η δημιουργία λογικών σχημάτων που θα επιτρέπουν την μετάφραση των φυσικών θέσεων μνήμης σε εικονικές. Η σωστή υλοποίηση αυτού του επιπέδου υποθέτει ότι το υλικό δεν έρχεται σε συχνή επαφή με αυτό το επίπεδο με σκοπό να περιορίζονται οι κακόβουλες προσπάθειες εκμείευσης των φυσικών διευθύνσεων

#### *3.8.9 Non-ISA Specifications*

Οι ομάδες εργασίας εκτός από τα παραπάνω καθήκοντα τους μπορούν να έχουν καθήκοντα που αφορούν στην αφομοίωση της αρχιτεκτονικής από την αγορά ή σε εργασίας υποστήριξης και δημιουργίας ενός συνόλου κανόνων συνεργασίας μηχανικών και προγραμματιστών γύρω από την ανοιχτή αρχιτεκτονική. Μία τέτοια ομάδα είναι αυτή που σχετίζεται με τον εντοπισμό σφαλμάτων. Μία άλλη ομάδα αναπτύσσει δοκιμές και κανόνες όπου ο τεχνικό όρος είναι «πλαίσια συμμόρφωσης». Τέλος είναι η Ομάδα Εργασίας Δομής Διαμόρφωσης που είναι υπεύθυνοι για την οργάνωση και αναπαράστασης της δομής διαμόρφωσης τόσο σε μορφή που μπορεί να γίνει κατανοητή από τον άνθρωπο όσο και από τον υπολογιστή.

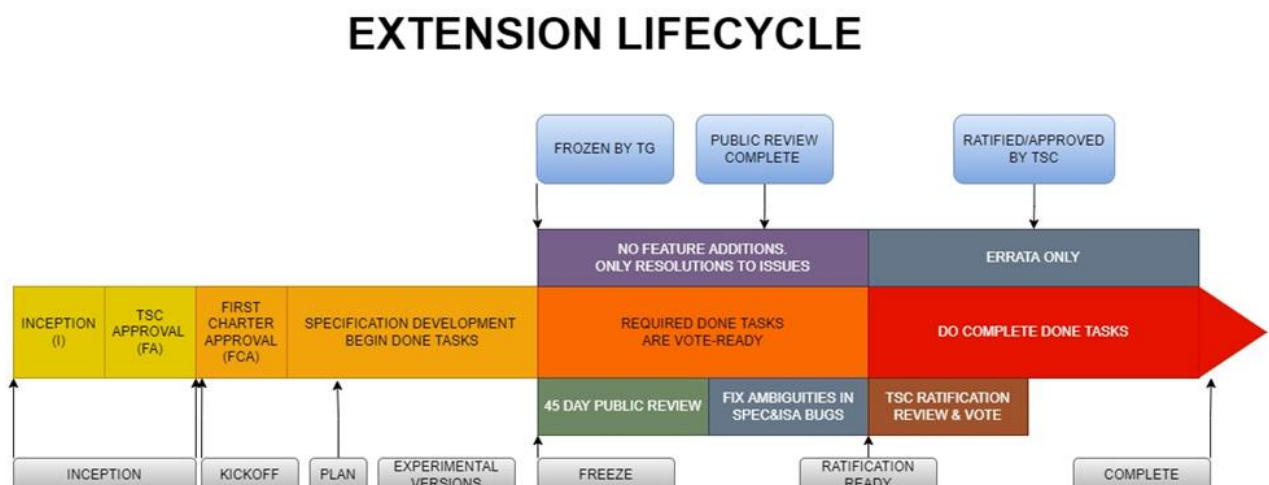
Όπως αναφέρθηκε στην ενότητα της τεχνικής οργάνωσης κάθε Ομάδα Εργασίας είναι υπεύθυνη για την οργάνωση μίας επέκτασης της ανοιχτής αρχιτεκτονικής. Μερικές από τις ομάδες που έχουν ιδιαίτερο ενδιαφέρον καθώς αναπτύσσουν επεκτάσεις που δεν έχει επικυρωθεί ακόμη είναι:

- Το Crypto Task Group που αναπτύσσει τις επεκτάσεις που μπορούν να υλοποιηθούν γρήγορα και αξιόπιστα τους κυριότερους κρυπτογραφικούς αλγόριθμους τις αγορές σε υλικό
- B Extension Task Group είναι υπεύθυνο για την ανάπτυξη και εξέλιξη των επεκτάσεων των μαθηματικών πράξεων και κατ' επέκταση των επεκτάσεων που χειραγωγούν δυαδικά ψηφία (bit manipulation).
- Vector Extension Task Group (V) είναι υπεύθυνο για την ανάπτυξη και εξέλιξη των επεκτάσεων που υπολογίζουν πράξεις μεταξύ διανυσμάτων. Οι ανάπτυξη αυτών των επεκτάσεων είναι μείζονος σημασίας καθώς είναι απαραίτητες για την Τεχνητή Νοημοσύνη και τα γραφικά.

Από τη στιγμή που θα περάσουν το στάδιο της επικύρωσης θα προστεθούν στις μη προνομιούχες προδιαγραφές.

### 3.9 RISC-V Extension Lifecycle

Κάθε επέκταση RISC-V από τη στιγμή της σύλληψης της ως ιδέας έως την τελική αποδοχής της περνά από διάφορα στάδια ανάπτυξης. Το τελικό βήμα είναι αυτό της επικύρωσης και επιτρέπει στους μηχανικούς υλικού να την χρησιμοποιούν για την παραγωγή SoC που απευθύνονται στην αγορά. Τα στάδια αυτά είναι αρκετά σημαντικά για τους μηχανικούς υλικούς και για τους προγραμματιστές και για αυτό είναι γνωστά ως «ορόσημα».



**Εικόνα 12.** Ο κύκλος επέκτασης ζωής για την RISC-V [32]

#### 1. Σύλληψη

Μετά από σχετική έρευνα η τεχνική ηγεσία της RISC-V International εγκρίνει την ανάπτυξη της επέκτασης. Βρισκόμαστε στο στάδιο της σύλληψης.

## 2. Έναρξη

Δημιουργείται μία ομάδα που ονομάζεται «έδρα ενέργειας» που στο καθήκοντα της είναι να οδηγήσει στη δημιουργία μίας άλλης ομάδας με όνομα «ομάδα εργασιών» με συγκεκριμένα καθήκοντα ανάπτυξης. Έπειτα ο Πρόεδρος του ιδρύματος δημιουργεί ένα περιγραφικό όνομα για την ομάδα εργασιών και ορίζει τις εργασίες ανάπτυξης που θα πρέπει να ολοκληρωθούν από την συγκεκριμένη ομάδα.

## 3. Σχέδιο

Στο στάδιο του Σχέδιου η ομάδα ορίζει το τρόπο και βήματα που χρειάζεται να πάρει για να ολοκληρωθούν τα καθήκοντα που τις ανατέθηκαν.

## 4. Πειραματικές Εκδόσεις

Κατά το επόμενο στάδιο παράγονται διάφορες προδιαγραφές που έχουν σαν την πειραματική ετικέτα. Αυτές οι εκδόσεις των προδιαγραφών δεν είναι σταθερές πιθανώς να έχουν σημαντικά λάθη και προβλήματα και δεν θα πρέπει να χρησιμοποιούνται σε προς παραγωγή σχέδια.

## 5. Πάγωμα

Όταν η ομάδα φτάσει στο σημείο ορόσημο όπου η ανάπτυξη της ιδέας σε προδιαγραφή έχει ωριμάσει και πλέον δεν υπάρχουν άγνωστοι παράγοντες και δεν χρειάζονται επιπλέον αλλαγές στην προδιαγραφή τότε η προδιαγραφή φτάνει στο στάδιο του παγώματος. Σε αυτό το στάδιο επιτρέπονται αλλαγές μόνο αν βρεθούν λάθη.

## 6. Έτοιμη η επικύρωση

Το επόμενο στάδιο είναι αυτό της επικύρωσης. Για να περάσει από αυτό το στάδιο μία προδιαγραφή πρέπει αρχικά να μην επισημαίνονται λάθη στα δημόσια αποθετήρια ώστε να χρειάζεται η ανάκληση της προδιαγραφής.

Εφόσον περάσει και αυτό το βήμα η Τεχνική Συντονιστική Επιτροπή που καλείται να ψηφίσει για την τελική επικύρωση.

## 7. Πλήρης

Μετά την επικύρωση η προδιαγραφή είναι πλέον είναι κομμάτι της αρχιτεκτονικής RISC-V και τοποθετείται είτε στην μη προνομιακή ή στην προνομιακή προδιαγραφή.

### 3.10 Τα Format Εντολών του βασικού RISC-V

Κάθε επέκταση της RISC-V αρχιτεκτονικής περιλαμβάνει ένα σετ εντολών. Οι εντολές αυτές βρίσκονται αφαιρετικά ένα επίπεδο πάνω από τον δυαδικό κώδικά αφού για τον επεξεργαστή οι εντολές και τα δεδομένα είναι πληροφορία κωδικοποιημένη στο δυαδικό σύστημα. Καθότι η λέξη (word) στην RV32I αρχιτεκτονική είναι 32bit [31:0] και η κάθε

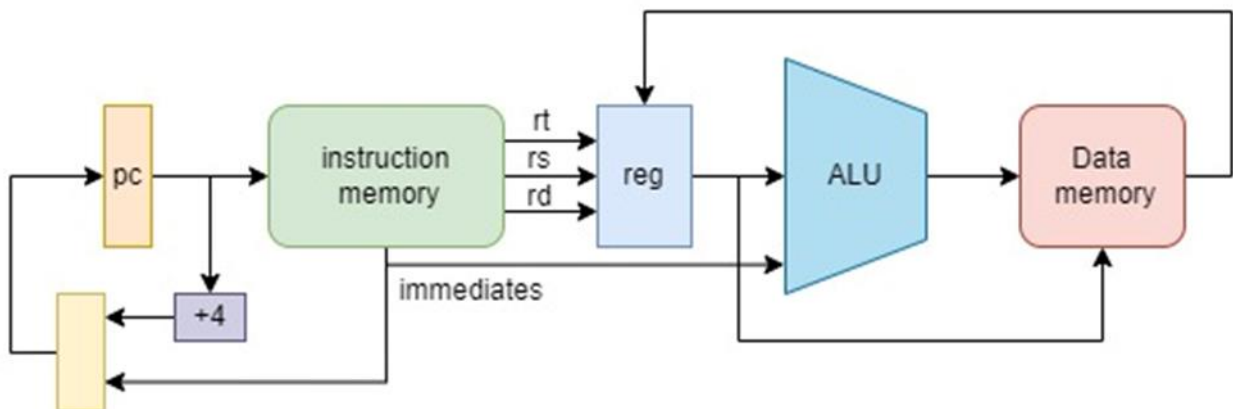
επέκταση έχει τις δικές της ανάγκες χωρίζετε η λέξη σε νοητά πεδία. Υπάρχουν διαφορετικές διατάξεις (formats) που έχουν διαφορετικό μέγεθος εσωτερικών πεδίων ώστε να καλύπτονται οι ανάγκες πρόσβασης σε μνήμη κυρίως της κάθε επέκτασης. Ο Πίνακας 4 προσδιορίζει τις διατάξεις και τα πεδία.

**Πίνακας 4. RISC-V Instruction formats**

	31	27	25	20	14	12	7	0
R	funct7		rs2	rs1	funct3	rd	Opcode	
I	imm[11:0]			rs1	funct3	rd	Opcode	
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	Opcode	
SB	imm[12 10:5]		rs2	rs1	funct3	imm[4:1 11]	Opcode	
U	imm[31:12]					rd	Opcode	
UJ	imm[20 10:1 11 19:12]					rd	Opcode	

### 3.11 Γενικός κύκλος Φόρτωσης/Εκτέλεσης (Fetch/Execute cycle)

Στην αρχιτεκτονική RISC-V τυπικά οι εντολές assembly εκτελούνται στον επεξεργαστή σε πέντε στάδια, όπως παρουσιάζεται και στην Εικόνα 13.



**Εικόνα 13. Κύκλος εκτέλεσης**

Τα στάδια αυτά είναι:

1. Φόρτωση οδηγίας από τη μνήμη,
2. Αποκωδικοποίηση Οδηγίας,
3. Μαθηματική Πράξη στην ALU,
4. Πρόσβαση Μνήμης,
5. Εγγραφή αποτελέσματος πίσω στον καταχωρητή.

### 3.12 Περιβάλλοντα εκτέλεσης λογισμικού RISC-V

Η συμπεριφορά κατά την εκτέλεση ενός προγράμματος που μεταγλωττίζεται στην RISC-V αρχιτεκτονική εξαρτάται από το περιβάλλον στο οποίο εκτελείται. Η αρχική κατάσταση του προγράμματος, ο αριθμός και ο τύπος των νημάτων υλικού (Hardware threads) στο SoC, ποιες από τις λειτουργίες προνομιών υποστηρίζονται από τα νήματα υλικού, ο τρόπος πρόσβασης της μνήμης, τα χαρακτηριστικά της μνήμης αυτής καθώς και των περιοχών εισόδου/εξόδου και ο τρόπος πρόσβασης σε αυτά. Επιπλέον η συμπεριφορά όλων των οδηγιών που εκτελούνται σε κάθε νήμα εκτέλεσης, ο τρόπος που χειρίζονται οι διακοπές και οι εξαιρέσεις που δημιουργούνται κατά την εκτέλεση ενός προγράμματος, συμπεριλαμβανομένων σε αυτό των κλήσεων περιβάλλοντος, ορίζονται όλες από μια διεπαφή περιβάλλοντος εκτέλεσης RISC-V (Execution Environment Interface - EEI).

Η δυαδική διεπαφή εφαρμογών Linux (Application Binary Interface - ABI) και η δυαδική διεπαφή επόπτη RISC-V είναι δύο παραδείγματα EEI (SBI). Ένα περιβάλλον εκτέλεσης RISC-V μπορεί να υλοποιηθεί χρησιμοποιώντας μόνο υλικό, μόνο λογισμικό ή ένα υβρίδιο από τα δύο.

### 3.13 RISC-V vs ARM vs x64

Η επιλογή της ARM αρχιτεκτονικής για ένα επεξεργαστή αποτελεί για να ένα μηχανικό υλικού μια ρεαλιστική επιλογή. Η ανάπτυξη του υλικού καθώς και οι αποφάσεις που βασίζονται γύρω από την αρχιτεκτονική ARM στηρίζονται στο παρελθόν και στο μέλλον. Νέες σύνθετες οδηγίες προστίθενται μόνο αν μπορούν να αυξήσουν τη συνολική απόδοση της αρχιτεκτονικής. Το σετ εντολών που περιλαμβάνεται στην αρχιτεκτονική ARM εκτελεί σημαντικό όγκο εργασίας χωρίς να καταναλώνει μεγάλη ισχύ και αυτό γίνεται γιατί η αρχιτεκτονική επιτρέπει την εκτέλεση υπό όρους με σύνθετες μεθόδους διευθυνσιοδότησης ώστε με αυτό τον τρόπο να αποφεύγεται η άσκοπη διακλάδωση μέσω των καταχωρητών CSR του επεξεργαστή.

Η RISC-V αρχιτεκτονική από την άλλη παρέχει συμπιεσμένες οδηγίες. Με αυτόν τον τρόπο, μια λέξη 32-bit μπορεί να φιλοξενήσει δύο από τις πιο δημοφιλείς εντολές. Το μεγαλύτερο προτέρημα είναι ότι η συμπίεση δεν προσθέτει καθυστέρηση καθώς αποτελεί κομμάτι της κανονικής διαδικασίας αποκωδικοποίησης εντολών.

Η x64 είναι χωρίς αμφιβολία η αποδοτικότερη αρχιτεκτονική. Έχει το απίστευτο νούμερο των 981 μνημονικών και 3684 παραλλαγές των μνημονικών αυτών. Η λογική είναι πάρα πολλή σύνθετη και απαιτεί μεγάλη ισχύ. Ακόμη δεν έχει την ίδια πυκνότητα δεδομένων (data density) που έχουν οι άλλες αρχιτεκτονικές.





## 4<sup>ο</sup> ΚΕΦΑΛΑΙΟ: LiteX & SoC DESIGN FLOW

### 4.1 LiteX: SoC builder framework

Το LiteX [33] είναι ένα προγραμματιστικό πλαίσιο (framework) μέσω του οποίου μπορούν να δημιουργηθούν πλήρη υπολογιστικά συστήματα σε FPGA. Τα συστήματα αυτά μπορούν να έχουν πληθώρα προσαρμόσιμων χαρακτηριστικών μεταξύ άλλων το μέγεθος μνήμης, το είδος (32bit/64bit) και η ταχύτητα επεξεργαστή, το είδος καθώς και το πλάτος του διαύλου επικοινωνίας κ.α. καθώς και πληθώρα προσαρμόσιμων περιφερειακών όπως PCIe, SPI, Ethernet, LiteScope (debugging module).

### 4.2 Τί είναι το SoC

Το System-on-Chip ή αλλιώς SoC είναι ένα ψηφιακό σύστημα που μέσα του ενσωματώνονται τα περιφερειακά ενός ηλεκτρονικού υπολογιστή καθώς και οι απαραίτητοι μεταξύ τους επικοινωνία δίαυλοι. Ένα SoC μπορεί να περιέχει, χωρίς να είναι απαραίτητο, επεξεργαστές, επεξεργαστές γραφικών, ελεγκτές μνήμης και περιφερειακά που επικοινωνούν με διάφορους τρόπους όπως SPI, I2C, PWM και Onewire.

### 4.3 Ορολογία: Library, Framework, toolkit, API, SDK

- Βιβλιοθήκη (Library) ονομάζονται τα κομμάτια κώδικα όπου μπορούν να κληθούν μέσα από τον δικό μας κώδικα και που χρησιμοποιούνται για την υλοποίηση μιας διεργασίας. Η στάνταρντ βιβλιοθήκη της γλώσσας προγραμματισμού C είναι ένα καλό παράδειγμα.
- Μία Διεπαφή Προγραμματισμού Εφαρμογών (Application Programming Interface - API) είναι ένας όρος που περιγράφει τις συναρτήσεις και τις μεθόδους μιας βιβλιοθήκης που μπορούν να κληθούν για να κάνουν αυτό για το οποίο έχουν προγραμματιστεί. Τα R.E.S.T. API είναι ένα καλό παράδειγμα.
- Ένα κιτ ανάπτυξης λογισμικού (Software Development Kit - SDK) είναι ένα σύνολο από βιβλιοθήκες κώδικα, εφαρμογές, αρχεία δεδομένων και έτοιμος κώδικας με σκοπό την ανάπτυξη κώδικα σε ένα συγκεκριμένο σύστημα. Το .Net είναι ένα καλό παράδειγμα.
- Ένα πλαίσιο (Framework) είναι μία μεγάλη βιβλιοθήκη ή ένα σύνολο από βιβλιοθήκες με σκοπό την προσφορά πολλών λειτουργιών για την ανάπτυξη ενός λογισμικού. Το LiteX είναι ένα καλό παράδειγμα.

#### 4.4 Εγκατάσταση του LiteX

Για να παρουσιαστούν οι ελάχιστες προδιαγραφές για την ομαλή εκτέλεση του LiteX αρχικά πρέπει να ξεκαθαριστεί πως αν και είναι γραμμένο σε Python το LiteX δεν είναι cross platform. Για τη λειτουργία του χρειάζεται βιβλιοθήκες και προγράμματα που υπάρχουν σε Linux\Unix συστήματα. Μπορεί να τρέξει σε Linux\MacOs αλλά για να τρέξει σε περιβάλλον windows χρειάζεται είτε κάποιο επίπεδο εικονοποίησης (virtualization) όπως VM ή docker.

Τα λογικά σχέδια που θα παρουσιαστούν παρακάτω υλοποιήθηκαν σε Linux περιβάλλον και συγκεκριμένα στην 22.04 LTS (Jammy Jellyfish) έκδοση του Ubuntu x64 με τα εξής βήματα:

1. Αρχικά πρέπει να κατεβεί το σενάριο εντολών (script) αρχικής εγκατάστασης από το αποθετήριο [enjoy-digital/litex](https://github.com/enjoy-digital/litex) από το GitHub:

Αυτό γίνεται πληκτρολογώντας σε ένα τερματικό:

```
wget https://raw.githubusercontent.com/enjoy-digital/litex/master/litex_setup.py
```

2. Γίνεται παραχώρηση στο σενάριο εντολών δικαιωμάτων εκτέλεσης:

```
chmod +x litex_setup.py
```

3. Έπειτα πρέπει να κληθεί το σενάριο εντολών να εκτελεστεί, δίνοντας τις παραμέτρους --init για να γίνει αρχικοποίηση, --install για να γίνει εγκατάσταση και --user για τον χρήστη:

```
./litex_setup.py --init --install --user
```

4. Προτείνεται να ελέγχουμε ότι έχουμε τον τελευταία ενημερωμένο κώδικα με το --update:

```
./litex_setup.py --update
```

5. Σε αυτό το βήμα γίνεται εγκατάσταση των υπόλοιπων εργαλείων\βιβλιοθηκών που θα χρειαστούμε:

```
sudo apt-get install libevent-dev libjson-c-dev
```

6. Έπειτα πρέπει να γίνει εγκατάσταση του λογισμικού ανοιχτού κώδικα Verilator. Το Verilator είναι ένα πρόγραμμα ανοιχτού λογισμικού\ανοιχτού κώδικα που μετατρέπει τη γλώσσα περιγραφής υλικού Verilog σε ένα μοντέλο συμπεριφοράς C++. Περιορίζεται για την μοντελοποίηση των τμημάτων της Verilog που μπορούν να συντεθούν σε υλικό.

```
sudo apt-get install verilator
```

7. Τέλος για να δουλέψουν σωστά οι προσομοιώσεις μας πρέπει να οριστεί στο LiteX ο επεξεργαστής που θα ενσωματωθεί στα σχέδια μας. Σε αυτό το σημείο τονίζεται πως αυτό το βήμα δεν είναι απαραίτητο αν δε θα χρησιμοποιηθεί επεξεργαστής στο SoC μας αλλά κάποιο συνδυασμός διαύλου επικοινωνίας με κάποια συνδυαστική λογική.

```
litex_sim --cpu-type=vexriscv
```

#### 4.5 Πυρήνες Λειτουργικότητάς

Το LiteX έχει ενσωματωμένες μονάδες λειτουργικότητας με τη μορφή παραμετροποίησης βιβλιοθηκών κώδικα HDL για την τέλεση κοινών εργασιών, όπως:

- **Litescope** - Το LiteScope είναι μια ελαφριά και προσαρμόσιμη υλοποίηση ενός ενσωματωμένου αναλυτή λογικής.
- **LiteHyperBus** - Είναι μία ελεύθερη υλοποίηση διαύλου HyperBus.
- **LiteSPI** - Είναι μία ελεύθερη υλοποίηση διαύλου SPI.
- **LiteICLink** - Προσφέρει διαύλους επικοινωνίας μεταξύ διαφορετικών πυρήνων.
- **LiteSDCard** - Ο πυρήνας αυτός περιέχει τα κυκλώματα που είναι απαραίτητα για την λειτουργία μιας κάρτας SD.
- **LiteSATA** - Είναι μία ελεύθερη υλοποίηση πρωτοκόλλου SATA.
- **LiteEth** – Περιέχει την λογική για επικοινωνία κυκλωμάτων μέσω Ethernet
- **LiteDRAM** – Περιέχει τη λογική για δημιουργία μνήμης Τυχαίας Προσπέλασης.

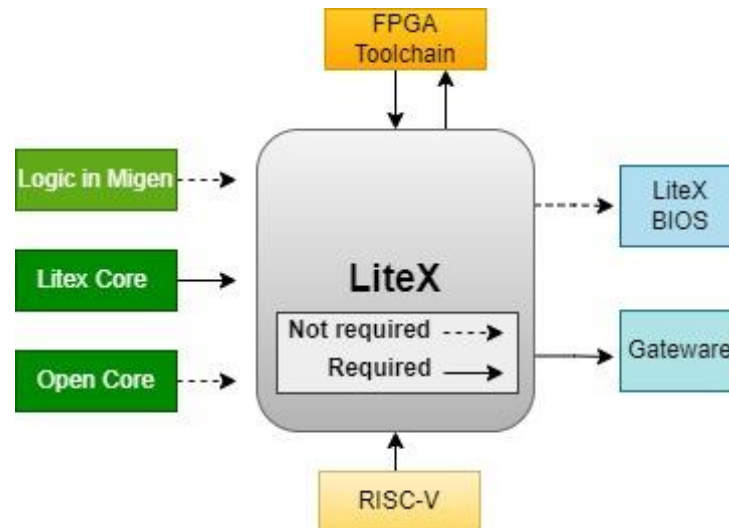
#### 4.6 Δίαυλοι Επικοινωνίας

Οι μέθοδοι μέσω των οποίων συνδέονται δύο ή περισσότερες ψηφιακές συσκευές, ώστε να μπορούν να ανταλλάσσουν δεδομένα μεταξύ τους ονομάζονται δίαυλοι (communication buses). Στα λογικά σχέδια που υλοποιούνται σε FPGA υπάρχουν οι κλειστού κώδικα υλοποιήσεις όπως η AXI της Xilinx και η Avalon της Intel. Ακόμα υπάρχει ο δίαυλος wishbone που είναι μία ανοιχτή υλοποίηση ενός διαύλου επικοινωνίας για λογικά σχέδια σε FPGA με στόχο τη σύνδεση και επικοινωνία πολλών πυρήνων μέσα σε ένα τσιπ μεταξύ τους.

Το LiteX υποστηρίζει όλους τους ανωτέρους τρόπους επικοινωνίας και ταυτόχρονα προσθέτει και έναν ακόμη τον δίαυλο CSR. Με αυτό τον τρόπο η επικοινωνία επιτυγχάνεται μέσω καταχωρητών με είσοδο και έξοδο αντιστοιχισμένη σε μνήμη (memory mapped input output - MMIO) και είναι προσβάσιμοι από τον επεξεργαστή με διαφορετικές κατευθύνσεις που προσδιορίζονται από το λογισμικό.

#### 4.7 Ένα τυπικό SoC Design flow με το LiteX

Σε μία τυπική σχεδίαση ενός SoC χρησιμοποιούμε το LiteX ως το front-end του Intel Quartus IDE για την μετατροπή των λογικών σχεδίων μας σε bitstream. Μέσω του LiteX γίνεται μετατροπή της Python σε Verilog HDL.



Εικόνα 14. Τυπικό SoC Design flow με το LiteX [34]

Η σχεδίαση γίνεται με την περιγραφή του λογικού κυκλώματος σε Migen. Μετά κάνουμε build το περιβάλλον μας τροποποιώντας το αρχείο της αρχιτεκτονικής. Στο αρχείο αυτό περιέχονται τα εργαλεία που είναι απαραίτητα για από τον κατασκευαστή του FPGA για τον προγραμματισμό του καθώς περιέχεται το IDE και ο προγραμματιστής.

Έπειτα καλούμε τη συνάρτηση build() η οποία μετατρέπει τη λογική από Migen σε Verilog HDL σε ένα top.v αρχείο. Με την ίδια συνάρτηση μεταγλωττίζεται και το LiteX BIOS. Αμέσως μετά μετατρέπουμε την Verilog HDL σε bitstream. Για να ολοκληρωθεί η διαδικασία σχεδίασης θα χρησιμοποιηθεί η συνάρτηση upload() για να ανεβεί το bitstream στο FPGA για τον προγραμματισμό του.

#### 4.8 Κατασκευάζοντας συνδυαστικά κυκλώματα με το LiteX

Χρησιμοποιώντας την Migen και το LiteX για να υλοποιηθεί ένα συνδυαστικό κύκλωμα στο De10 lite της Intel. Το κύκλωμα είναι ένας απλός μετρητής που ανάβει τα led στο board με τέτοιο τρόπο που να θυμίζει τη γνωστή σειρά του ιππότη της ασφάλτου (Knight Rider). Ο κώδικας σε Migen όπως θα υλοποιηθεί στο DE10Lite.

```
# Η κλάση υλοποιεί ένα κύκλωμα που αναβοσβήνει τα led στο board
class LedChaser(Module, AutoCSR):
    def __init__(self, pads, sys_clk_freq, period=1e0, polarity=0): #Αρχικοποίηση
        self.pads = pads
        self.polarity = polarity
        self.n = len(pads)
        self._out = CSRStorage(len(pads), description="Led Output(s) Control.")

        ###

        chaser = Signal(self.n)#Σήματα
        mode = Signal(reset=_CHASER_MODE)
        timer = WaitTimer(int(period*sys_clk_freq/(2*self.n)))
        leds = Signal(self.n)
        self.submodules += timer #Υλοποίηση συνδιαστικού κυκλώματος μέσω self.comb
        self.comb += timer.wait.eq(~timer.done)
        self.sync += If(timer.done, chaser.eq(Cat(~chaser[-1], chaser)))
        self.sync += If(self._out.re, mode.eq(_CONTROL_MODE))
        self.comb += [
            If(mode == _CONTROL_MODE,
                leds.eq(self._out.storage)
            ).Else(
                leds.eq(chaser)
            )
        ]
        self.comb += pads.eq(leds ^ (self.polarity*(2**self.n-1)))

    def add_pwm(self, default_width=512, default_period=1024, with_csr=True):#Υλοποίηση κυκλώματος
        μέσω PWM
        from litex.soc.cores.pwm import PWM
        self.submodules.pwm = PWM(
            with_csr = with_csr,
            default_enable = 1,
            default_width = default_width,
            default_period = default_period
        )
        #Έλεγχος των LED μέσω PWM
        self.comb += If(~self.pwm.pwm, self.pads.eq(self.polarity*(2**self.n-1)))
```

#### Κώδικας 4. Led Chaser Συνδιαστικό κύκλωμα

## 4.9 Διαμόρφωση Ubuntu 22.04

Για να διαμορφωθεί σωστά το περιβάλλον λειτουργίας του LiteX στο Ubuntu 22.04 θα πρέπει να γίνουν κάποιες παραμετροποιήσεις.

Για αρχή θα πρέπει να διαμορφωθούν οι μεταβλητές περιβάλλοντος (Environment Variables) που αφορούν το \$PATH. Με αυτό τον τρόπο συμπληρώνονται οι τοποθεσίες που αναζητεί το λειτουργικό σύστημα τα εκτελέσιμα αρχεία. Σε αντίθεση με τα υπόλοιπα Linux συστήματα η ρύθμιση αυτή γίνεται στο αρχείο .profile στον «~» φάκελο του χρήστη. Ένας ακόμη τρόπος είναι η προσθήκη των μεταβλητών στο αρχείο που αρχικοποιεί το

shell του χρήστη (π.χ. `.bashrc`, `.zshrc`). Συγκεκριμένα πρέπει να προστεθούν οι εξής γραμμές:

```
# ΠΡΟΣΘΗΚΗ CUSTOM PATH
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi

if [ -d "$HOME/intelFPGA_lite/21.1/quartus/bin" ] ; then
    PATH="$HOME/intelFPGA_lite/21.1/quartus/bin:$PATH"
fi

if [ -d "$HOME/riscv64-unknown-elf-gcc-10.1.0-2020.08.2-x86_64-linux-ubuntu14/bin" ] ; then
    PATH="$HOME/riscv64-unknown-elf-gcc-10.1.0-2020.08.2-x86_64-linux-ubuntu14/bin:$PATH"
fi

if [ -d "$HOME/.local/bin" ] ; then
    PATH="$HOME/.local/bin:$PATH"
fi

export QSYS_ROOTDIR="/home/iamme/intelFPGA_lite/21.1/quartus/sopc_builder/bin"
```

#### Κώδικας 5. Ρύθμιση Μεταβλητών Περιβάλλοντος

Ακόμα η 22.04 έκδοση του Ubuntu περιέχει ένα λανθασμένα ρυθμισμένο πρόγραμμα το «brltty». Το πρόγραμμα αυτό προκαλεί δυσλειτουργία κατά τη φόρτωση του οδηγού της CH340 σειριακής συσκευής μας προκαλώντας την άμεση αποσύνδεση της. Μία εύκολη λύση είναι να απεγκατασταθεί το συγκεκριμένο λογισμικό μέσω της παρακάτω εντολής σε τερματικό:

```
sudo apt-get purge -y brltty
```

Για τη σωστή λειτουργία της σειριακής πρέπει επίσης ο χρήστης να είναι μέλος του group «dialout» και η σειριακή ως αρχείο να έχει δικαιώματα 777. Αυτό επιτυγχάνεται της μέσω της παρακάτω εντολής σε τερματικό:

```
sudo adduser $USER dialout && sudo chmod 777 /dev/ttyUSB0
```

## 4.10 Integrated Development Environments (IDEs)

Το προεπιλεγμένο Integrated Development Environment (IDE) για την ανάπτυξη κώδικα σε γλώσσα Migen και τα unit tests επιλέχθηκε το Visual Studio Code καθότι:

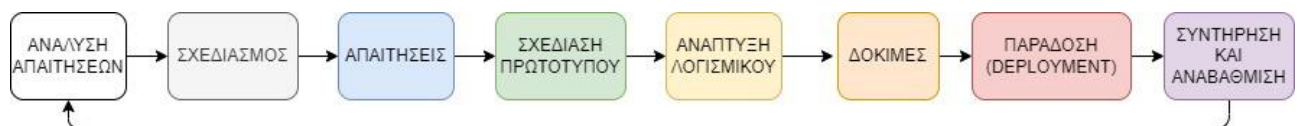
- είναι cross platform,
- είναι ανοιχτού κώδικα,
- είναι ελαφρύ ως προς τις ελάχιστες προδιαγραφές του,
- αυξάνεται σημαντικά η λειτουργικότητα του μέσω εγκατάστασης plugins, και
- είναι εύκολη η ενσωμάτωση του σε CI\CD pipelines.

Το προεπιλεγμένο Integrated Development Environment (IDE) για την ανάπτυξη κώδικα σε γλώσσα Verilog HDL \ System Verilog HDL είναι το Intel Quartus λόγω των δυνατοτήτων προσομοίωσης λογικών κυκλωμάτων και στην αντιμετώπιση προβλημάτων με τον προγραμματισμό του board.

Για την προσομοίωση του SoC χρησιμοποιήθηκε το ανοιχτού κώδικα πρόγραμμα Verilator λόγω της στενής σχέσης του με το LiteX.

#### 4.11 Κύκλος ζωής ανάπτυξης λογισμικού (Software Development Life Cycle - SDLC)

Τα τελευταία χρόνια έχουν αναπτυχθεί αρκετά μοντέλα ενός πλαισίου διαδικασιών με κατευθυντήριες γραμμές μέσω του οποίου κινείται η ανάπτυξη λογισμικού. Το μοντέλο αυτό έχει πολλά πλεονεκτήματα όπως ξεκάθαρους στόχους και ευθύνες μεταξύ των συμμετεχόντων, κοινό λεξιλόγιο, τη δυνατότητα να ελέγχεται η ποιότητα του παραγόμενου κώδικα, γρηγορότερα αναπτυσσόμενος κώδικας.



Εικόνα 15. Ο κύκλος ανάπτυξης λογισμικού

Τα κυριότερα μοντέλα είναι:

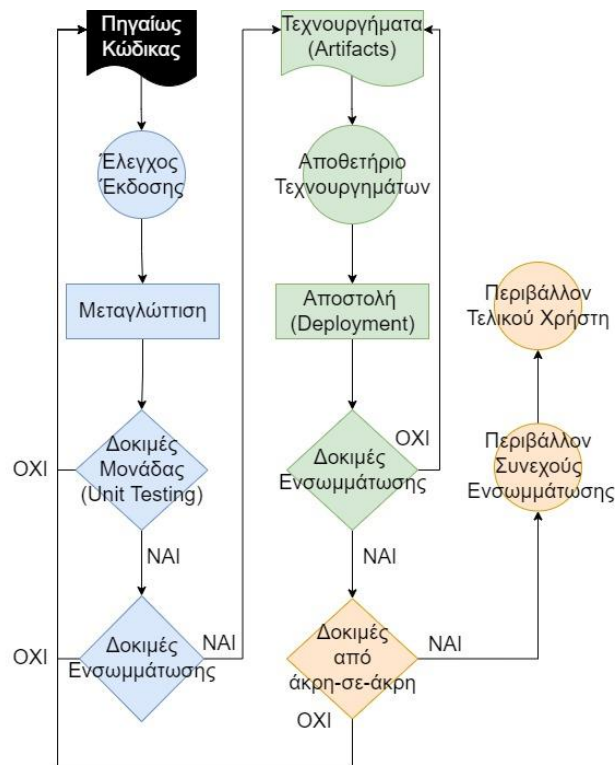
- Το μοντέλο του καταρράκτη (Waterfall)
- Το προσθετικό μοντέλο (Incremental)
- Το επαναληπτικό μοντέλο (Iterative)
- Το V-Μοντέλο
- Το Σπειροειδές Μοντέλο
- Το Ευκίνητο Μοντέλο (Agile Development)
- Το μοντέλο ανάπτυξης μέσω Δοκιμών (Test Driven Development)

Η ανάπτυξη λογικών ψηφιακών κυκλωμάτων μέσω γλωσσών περιγραφής υλικού χρησιμοποιεί αρχές και λογισμικά παρεμφερή με αυτά που χρησιμοποιούν οι γλώσσες ανάπτυξης λογισμικού. Κατά την ανάπτυξη της παρούσας εργασίας χρησιμοποιήθηκαν οι αρχές και μέθοδοι που προσδιορίζονται στο ευκίνητο μοντέλο (Agile Development). Κατά τη διάρκεια ανάπτυξης του SoC και κώδικα για την ανάπτυξη προγραμμάτων για τη λειτουργία επίδειξης του SoC η εύρεση λύσεων και η ανάπτυξη κώδικα έγινε μέσω του

πλαίσιου, των μεθόδων και των αρχών της επαναληπτικής και της συνεχούς βελτίωσης που προκύπτει από τη χρήση του μοντέλου της Agile Development.

#### 4.12 Η ανάγκη για Continuous Integration \ Continuous Development pipelines

Στη σύγχρονη βιομηχανία τα τελευταία χρόνια επικρατεί μία μέθοδος ανάπτυξης και ελέγχου κώδικα [35].



Εικόνα 16. CI/CD pipeline Template [35]

Η Συνεχής Ενσωμάτωση (Continuous Integration) είναι ένα σετ πρακτικών\μεθόδων σύμφωνα με τις οποίες μια εφαρμογή μεταγλωττίζεται και ελέγχεται αρκετά συχνά. Ιδανικά σε κάθε μεταγλώττιση ή σκηνοθέτηση που κάνει ο προγραμματιστής.

Η Συνεχής Παράδοση (Continuous Development) είναι ένα σετ πρακτικών\μεθόδων σύμφωνα με τις οποίες μια εφαρμογή αφού μεταγλωττιστεί πρέπει να εκτελείται σε ένα περιβάλλον όπως αυτό του τελικού χρήστη ώστε να γίνονται δοκιμές ενσωμάτωσης και δοκιμές που να διασφαλίζουν την τελική αποδοχή του μεταγλωττισμένου προγράμματος από τους χρήστες στο εργασιακό τους περιβάλλον.

##### 4.12.1 ZSH + ZSH plugins

Στην παρούσα εργασία χρησιμοποιήθηκε το Z shell (zsh) ως προεπιλεγμένο shell. Η λειτουργικότητα του Z Shell επεκτάθηκε με τα plugins p10k, oh-my-zsh και zsh-autosuggestions. Οι λόγοι που χρησιμοποιήθηκε το shell αυτό αντί του bash είναι:

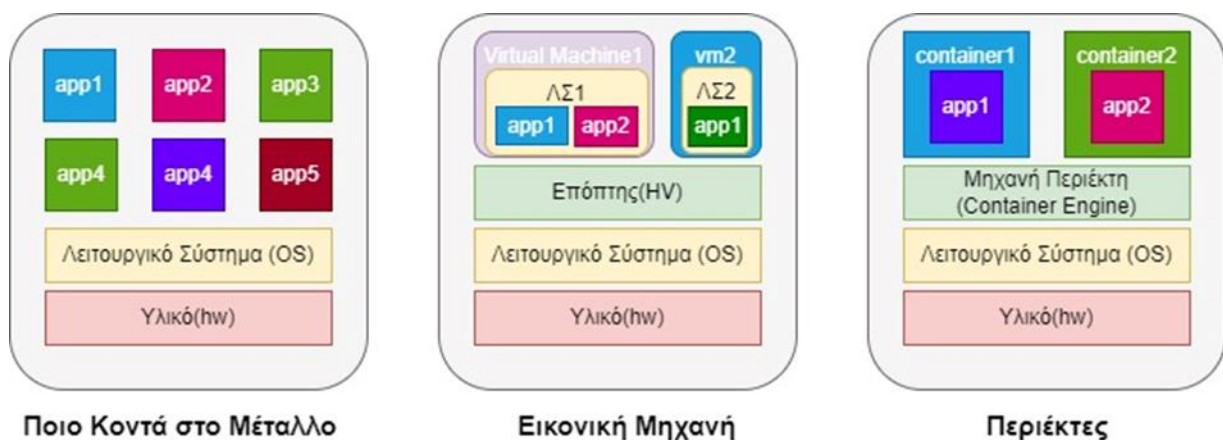
- Καλύτερη διαλειτουργικότητα με το Λειτουργικό Σύστημα.



- Επέκταση της λειτουργικότητας του shell με τις CI \ CD pipelines.
- Αυτόματη διόρθωση εντολών.
- Αυτόματη διόρθωση path.
- Αυτόματη Διόρθωση Λαθών κατά την πληκτρολόγηση εντολών.

#### 4.12.2 Docker

Το μοντέλο εκτέλεσης των προγραμμάτων σε ένα υπολογιστή με και χωρίς εικονοποίηση φαίνεται στην Εικόνα 17. Σε έναν Η\Υ ή εξυπηρετητή έχουμε το υλικό που πάνω του τρέχει το λειτουργικό και τις εφαρμογές που επικοινωνούν αφαιρετικά με το υλικό μέσω του Λειτουργικού Συστήματος [36].



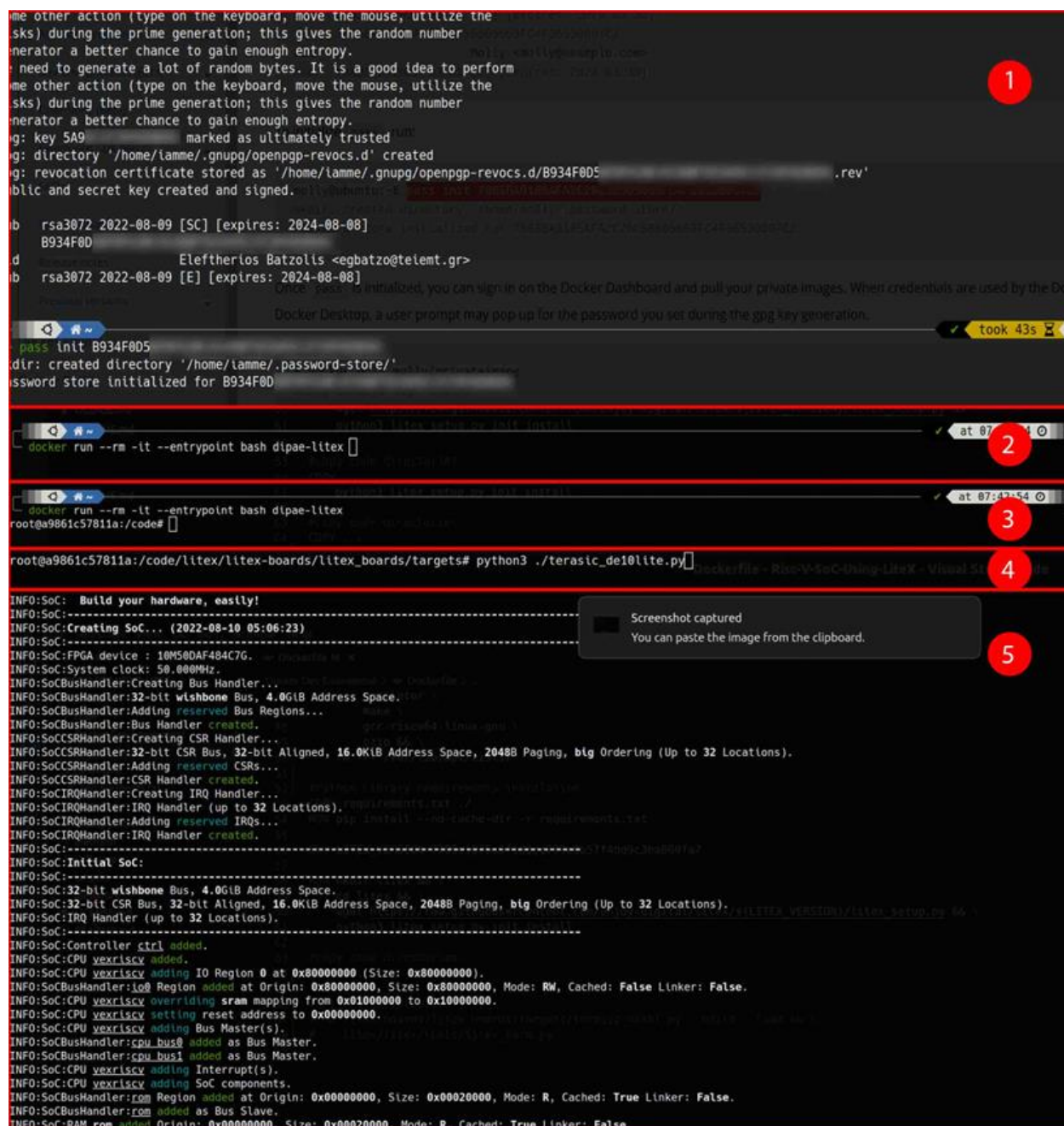
**Εικόνα 17.** Τα είδη της Εικονοποίησης [37]

Σε μία εικονική μηχανή, έχουμε το πρόγραμμα επόπτη που λειτουργεί για τις εικονικές μηχανές ως αφαιρετικό επίπεδο του λειτουργικού και του υλικού. Η κάθε εικονική μηχανή είναι αυτοτελής με το δικό της λειτουργικό. Εάν προσβληθεί από κακόβουλους χρήστες μία εικονική μηχανή ή το Host OS δεν θα επηρεαστούν οι άλλες εικονικές μηχανές καθώς έχουν τους δικούς τους ξεχωριστούς και αυτοτελείς πόρους .

Στην περίπτωση των Container (Περιέκτες) έχουμε το αφαιρετικό επίπεδο του Container Engine (Μηχανή Περιέκτη) όπου λειτουργεί αφαιρετικά ως προς το υλικό μέσω του λειτουργικού. Εάν προσβληθεί από κακόβουλους χρήστες ένα container ή το Host OS μπορούν να επηρεαστούν και τα υπόλοιπα container. Η container engine καθώς και τα ίδια τα containers ζητούν για την εκτέλεση τους πολλή λιγότερους πόρους από μία εικονική μηχανή.

Ακόμα τα containers έχουν την ιδιότητα να είναι stateless (Χωρίς κατάσταση). Η ιδιότητα επιτρέπει να ξεκινούν να κάνουν τη δουλειά για την οποία προγραμματίστηκαν και να τερματίζουν χωρίς να αλλάζει η κατάσταση τους. Αυτή η ιδιότητα είναι πάρα πολλή

χρήσιμη σε περιβάλλοντα όπως αυτά των datacenter, τα περιβάλλοντα ανάπτυξης εφαρμογών κ.α. Φυσικά υπάρχει η δυνατότητα σε ένα container να ενσωματωθεί χώρος μόνιμης αποθήκευσης. Η συνιστώμενη μέθοδος για την μόνιμων αποθήκευση δεδομένων που παράγονται και καταναλώνονται από Docker containers είναι μέσω docker volumes (τόμων).



```
me other action (type on the keyboard, move the mouse, utilize the
sks) during the prime generation; this gives the random number
erator a better chance to gain enough entropy.
need to generate a lot of random bytes. It is a good idea to perform
me other action (type on the keyboard, move the mouse, utilize the
sks) during the prime generation; this gives the random number
erator a better chance to gain enough entropy.
g: key 5A9 marked as ultimately trusted
g: directory '/home/iamme/.gnupg/openpgp-revocs.d' created
g: revocation certificate stored as '/home/iamme/.gnupg/openpgp-revocs.d/B934F805
blic and secret key created and signed.

b rsa3072 2022-08-09 [SC] [expires: 2024-08-08]
B934F80D
d Eleftherios Batzolis <egbatzo@telemt.gr>
b rsa3072 2022-08-09 [E] [expires: 2024-08-08]

Once pass is initialized, you can sign in on the Docker Dashboard and pull your private images. When credentials are used by the Do
Docker Desktop, a user prompt may pop up for the password you set during the gpg key generation.

pass init B934F805
dir: created directory '/home/iamme/.password-store/'
password store initialized for B934F80D

docker run --rm -it --entrypoint bash dipae-litex
root@a9861c57811a:/code#
root@a9861c57811a:/code/litex/litex-boards/litex_boards/targets# python3 ../terasic_de10lite.py
INFO:SoC: Build your hardware, easily!
INFO:SoC:Creating SoC... (2022-08-10 05:06:23)
INFO:SoC:INFO:SoC:FPGA device : 10M500AF484C7G.
INFO:SoC:System clock: 50.000MHz.
INFO:SoCBusHandler:Creating Bus Handler...
INFO:SoCBusHandler:32-bit wishbone Bus, 4.0GiB Address Space.
INFO:SoCBusHandler:Adding reserved Bus Regions...
INFO:SoCBusHandler:Bus Handler created.
INFO:SoCCSRHandler:Creating CSR Handler...
INFO:SoCCSRHandler:32-bit CSR Bus, 32-bit Aligned, 16.0KiB Address Space, 2048B Paging, big Ordering (Up to 32 Locations).
INFO:SoCCSRHandler:Adding reserved CSRs...
INFO:SoCCSRHandler:CSR Handler created.
INFO:SoCIRQHandler:Creating IRQ Handler...
INFO:SoCIRQHandler:IRQ Handler (up to 32 Locations).
INFO:SoCIRQHandler:Adding reserved IRQs...
INFO:SoCIRQHandler:IRQ Handler created.
INFO:SoC:Initial SoC:
INFO:SoC:32-bit wishbone Bus, 4.0GiB Address Space.
INFO:SoC:32-bit CSR Bus, 32-bit Aligned, 16.0KiB Address Space, 2048B Paging, big Ordering (Up to 32 Locations).
INFO:SoC:IRQ Handler (up to 32 Locations).
INFO:SoC:Controller ctrl added.
INFO:SoC:CPU vexriscv added.
INFO:SoC:CPU vexriscv adding IO Region 0 at 0x80000000 (Size: 0x80000000).
INFO:SoC:CPU vexriscv adding IO Region 1 at 0x80000000 (Size: 0x80000000, Mode: RW, Cached: False Linker: False).
INFO:SoC:CPU vexriscv overriding sram mapping from 0x01000000 to 0x10000000.
INFO:SoC:CPU vexriscv setting reset address to 0x00000000.
INFO:SoC:CPU vexriscv adding Bus Master(s).
INFO:SoC:CPU vexriscv adding Bus Master.
INFO:SoC:CPU vexriscv adding Interrupt(s).
INFO:SoC:CPU vexriscv adding SoC components.
INFO:SoC:CPU vexriscv adding Region 0 at Origin: 0x00000000, Size: 0x00020000, Mode: R, Cached: True Linker: False.
INFO:SoC:CPU vexriscv adding Region 1 at Origin: 0x00020000, Size: 0x00020000, Mode: R, Cached: True Linker: False.
```

Εικόνα 18. Επίδειξη Docker Development Environment

Στην παρούσα εργασία χρησιμοποιήθηκε Docker ως λογισμικό για container ώστε να επιτρέπει την stateless εκτέλεση των εργαλείων που χρειάζεται το LiteX και να υλοποιεί γρήγορα ένα έτοιμο προγραμματιστικό περιβάλλον σε καινούργιους υπολογιστές.

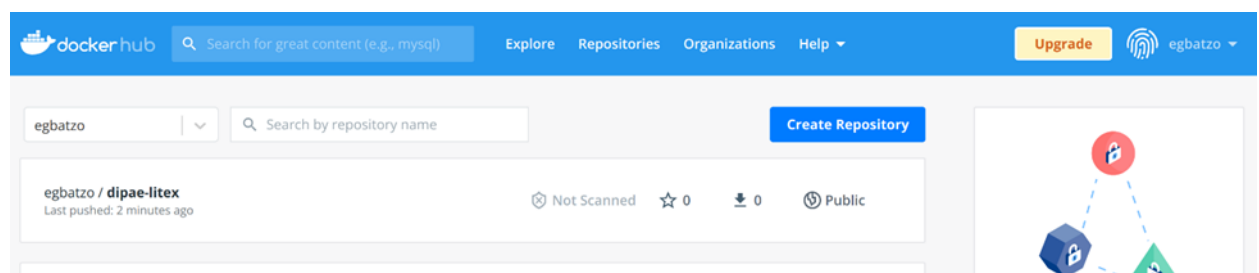
Καθότι δεν είναι όλα τα εργαλεία ανοιχτού κώδικά, συγκεκριμένα η toolchain της Intel απαιτεί ειδική άδεια για τον διαμοιρασμό το λογισμικού της, δεν μπορούσε να ενοποιηθεί

```

cd /Risc-V-SoC-Using-LiteX/Docker Dev Environment
ls
Dockerfile Makefile requirements.txt
docker build dipae-litex
unable to prepare context: path "dipae-litex" not found
docker build
[+] Building 24.6s (7/12)
Name and optionally a tag in the 'name:tag' format
[internal] load build definition from Dockerfile
=> transferring dockerfile: 1.68kB
[internal] load .dockerignore
=> transferring context: 2B
[internal] load metadata for docker.io/library/ubuntu:22.04
[auth] library/ubuntu:pull token for registry-1.docker.io
[1/7] FROM docker.io/library/ubuntu:22.04@sha256:34fea4f31bf187bc915536831fd0afc9d214755bf700b5c0b1336c82516d154e
=> resolve docker.io/library/ubuntu:22.04@sha256:34fea4f31bf187bc915536831fd0afc9d214755bf700b5c0b1336c82516d154e
=> sha256:34fea4f31bf187bc915536831fd0afc9d214755bf700b5c0b1336c82516d154e 1.42kB / 1.42kB
=> sha256:42ba2dfce475de113d5560269915415997167d47c2045ec766d9746ff148f 529B / 529B
=> sha256:d15de72bd3b3711ab04eca685b1f42c72ccba1837ed3fdb548a9282af2d836d 1.46kB / 1.46kB
=> sha256:d19f32bd9e4106d487f1a703fc2f09c8edd92db4405d477978e8e466ab29bd 30.43MB / 30.43MB
=> extracting sha256:d19f32bd9e4106d487f1a703fc2f09c8edd92db4405d477978e8e466ab29bd
[internal] load build context
=> transferring context: 2.65kB
[2/7] WORKDIR /code
[3/7] RUN apt-get -y update && apt-get install -y git python3 python3-setuputils autotools-dev curl lldm 10.1s
=> # Get:98 http://archive.ubuntu.com/ubuntu jammy/main amd64 libfuse3-3 amd64 3.10.5-1build1 [81.2 kB]
=> # Get:99 http://archive.ubuntu.com/ubuntu jammy/main amd64 fuse3 amd64 3.10.5-1build1 [24.7 kB]
=> # Get:100 http://archive.ubuntu.com/ubuntu jammy/main amd64 libcbores0.8 amd64 0.8.0-2ubuntu1 [24.6 kB]
=> # Get:101 http://archive.ubuntu.com/ubuntu jammy/main amd64 libbedt2 amd64 3.1-20210918-1build1 [96.8 kB]
=> # Get:102 http://archive.ubuntu.com/ubuntu jammy/main amd64 libfido2-1 amd64 1.10.0-1 [82.8 kB]
=> # Get:103 http://archive.ubuntu.com/ubuntu jammy/main amd64 libnghttp2-14 amd64 1.43.0-1build3 [76.3 kB]
Running in 909e81802aeb
2023.04
docker ps
docker ps --all
docker exec
docker exec
docker exec
docker exec
docker exec

```

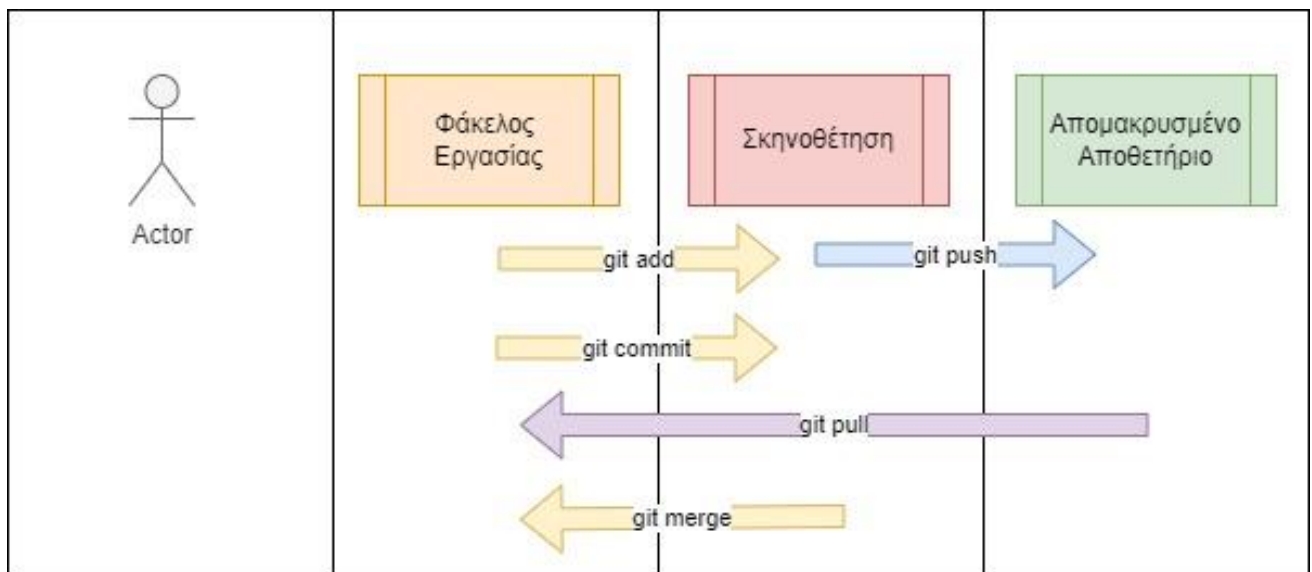
Ταυτόχρονα ενώθηκε με το αποθετήριο στο GitHub για την υλοποίηση των C.I.\C.D. Pipelines. Κάθε φορά που ο προγραμματιστής κάνει ένα git push αυτόματα ενημερώνεται το πρόγραμμα Jenkins ώστε να ξεκινήσει μία job. Το image είναι σηκωμένο στο δημόσιο Docker Hub με όνομα: **egbatzo\dipae-litex**



### 4.12.3 *GitHub*

Το GitHub [38] είναι μία πλατφόρμα μέσω της οποίας οι προγραμματιστές μπορούν να συνεργάζονται κατά την ανάπτυξη πηγαίου κώδικα μέσω του ελέγχου της έκδοσης του πηγαίου κώδικα. Μέσω του λογισμικού αυτού οι προγραμματιστές μπορούν να δουλεύουν σε μία ομάδα ανάπτυξης ενός λογισμικού απομακρυσμένα. Στηρίζεται στην

Λειτουργία του git που είναι ένα πρόγραμμα που αναπτύχθηκε από τον δημιουργό του πυρήνα του kernel Linus Torvalds για τον έλεγχο του πηγαίου κώδικα του kernel.



Εικόνα 21. Git workflow [39]

Οι βασικές λειτουργίες είναι:

- **Push** - Για την ώθηση κώδικα στον εξυπηρετητή.
- **Pull** - Για το κατέβασμα κώδικα στον τοπικό Η/Υ.
- **Commit** - Αποθηκεύει τις σκηνοθετημένες (staged) αλλαγές του τρέχοντα κώδικα.
- **Branch** - Δημιουργία παρακλαδίου χαρακτηριστικών κατά την ανάπτυξη του λογισμικού.
- **Merge** – Ένωση παρακλαδίου με το master.

Αρχικά ο προγραμματιστής κατεβάζει το αποθετήριο του κώδικα μέσω git pull. Έπειτα αναπτύσσει τα χαρακτηριστικά του προς ανάπτυξη προγράμματος και μόλις φτάσει σε ένα ικανοποιητικό για αυτόν στάδιο σκηνοθετεί των κώδικα του μέσω git commit με ένα σχόλιο για τις αλλαγές που έκανε. Έπειτα μπορεί να κάνει git push για να ωθήσει τις αλλαγές του στο κεντρικό αποθετήριο.

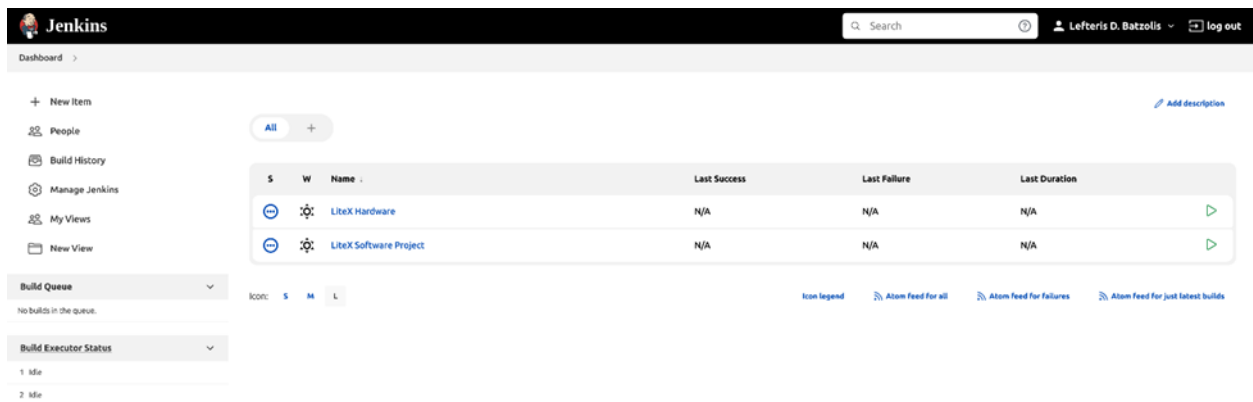
Το GitHub στην παρούσα εργασία χρησιμοποιήθηκε ως αποθετήριο του πηγαίου κώδικα και ταυτόχρονα το μέρος που γινόταν τα πρώτα unit tests μέσω των GitHub Actions. Η διεύθυνση του αποθετηρίου που μπορεί να βρεθεί ο κώδικας και τα αρχεία της παρούσας πτυχιακής είναι η εξής: <https://github.com/Lefteris-B/RISC-V-SoC-Using-LiteX>

#### 4.12.4 Jenkins

Το λογισμικό ανοιχτού κώδικα Jenkins [40] χρησιμοποιείται για την αυτοματοποίηση διαδικασιών συνεχούς ενσωμάτωσης και συνεχούς παράδοσης. Μπορεί να χρησιμοποιηθεί για την αυτοματοποίηση πολλών και διαφορετικών λειτουργιών

ανάπτυξης λογισμικού όπως υλοποίηση δοκιμών του κώδικα και δοκιμών μεταγλώττισης και παράδοσης.

Από μόνο του έχει περιορισμένη λειτουργικότητα ειδικά για το design flow ψηφιακών κυκλωμάτων είναι εξαιρετικά περιορισμένη. Αυτό το πρόβλημα ξεπεράστηκε μέσω της εγκατάστασης plugins. Η πρόσθετη αυτή λειτουργικότητα επιτρέπει στο Jenkins να κάνει ελέγχους στον κώδικα της Python και των γλωσσών περιγραφής υλικού ώστε να ανεβαίνει στο GitHub αυτόματα.



**Εικόνα 22.** Στιγμιότυπο του Jenkins που δείχνει τα Jobs

#### 4.12.5 Renode.io

Το Renode.io [41] είναι ένα πλαίσιο διαφορετικών προγραμμάτων ανάπτυξης με σκοπό την επιτάχυνση της δημιουργίας λογικών σχεδίων ενσωματωμένων συστημάτων και λογικών σχεδίων συστημάτων IoT. Μέσα από το λογισμικό του Renode.io μπορεί να προσομοιωθεί ένα λογικό σχέδιο με κυκλώματα όπως η CPU, τα περιφερειακά, οι αισθητήρες μέσω του LiteX.

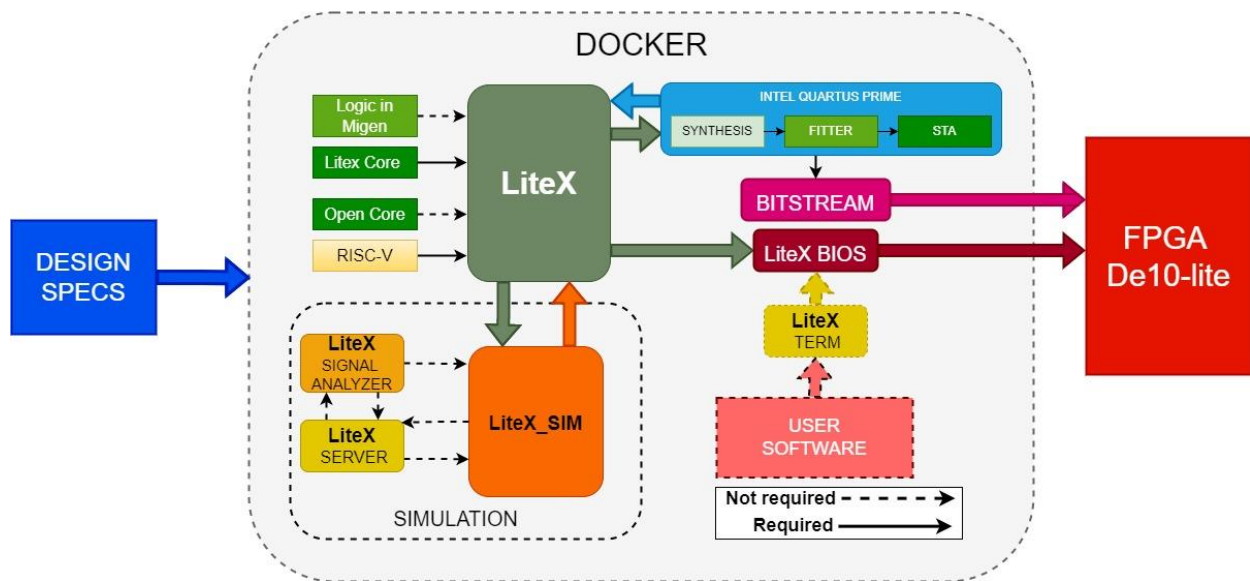
Επιπλέον οι προγραμματιστές μπορούν να δουλεύουν και να αναπτύσσουν εφαρμογές λογισμικού στο υπό ανάπτυξη λογικό σχέδιο του κυκλώματος ενώ αυτό αναπτύσσεται σε ένα προσομοιωμένο περιβάλλον. Με αυτό τον τρόπο το υπό ανάπτυξη λογισμικό μπορεί να εκτελεστεί, να διορθωθεί και να δοκιμαστεί με πλαίσιο αυτό προγραμμάτων που αποτελούν το πρόγραμμα Renode.io. Μπορεί να προσομοιωθούν με αυτό τον τρόπο από βασικά SoC έως πλήρη συστήματα πολλαπλών κόμβων.

#### 4.13 LiteX SoC Design Flow

Η ροή σχεδίασης του SoC, όπως παρουσιάζεται στην Εικόνα 23, ξεκινάει με την ανάλυση των απαιτήσεων. Οι απαιτήσεις χωρίζονται σε λειτουργικές και μη λειτουργικές. Έπειτα ο μηχανικός υλικού μετατρέπει τις απαιτήσεις σε κομμάτια (modules) κώδικα. Η



λειτουργικότητα του SoC που απορρέει από τα LiteX modules ενσωματώνεται σε αυτό το στάδιο.



Εικόνα 23. LiteX SoC Design Flow

Έπειτα ξεκινά η ενσωμάτωση των modules και η μεταγλώττιση τους σε Verilog HDL σε ένα ενιαίο αρχείο λειτουργικότητας. Αυτό γίνεται μέσω της παραμετροποίησης της υπάρχουσας λειτουργικότητας (εάν υπάρχει για το FPGA). Υπάρχουν δύο αρχεία που περιγράφουν τη λειτουργικότητα του FPGA το **architecture file** (altera.py) και το **board file** (terasic\_de10lite.py). Τα δυο αυτά αρχεία επεκτείνουν την κλάση **soc** και **modules** της migen.

Το επόμενο βήμα είναι η προσομοίωση της λειτουργικότητας του SoC. Υπάρχουν πολλές επιλογές σε αυτό το σημείο ανάλογα με τις ανάγκες της προσομοίωσης που προκύπτουν από την ανάλυση απαιτήσεων. Στην παρούσα επιλέχθηκε το script `litex_sim.py` που είναι υπεύθυνο για τη μεταγλώττιση του gateway και την προσομοίωση του αφού προστεθεί το LiteScope module. Το script `litex_sim.py` σε συνδυασμό με το script `litex_server` είναι επίσης υπεύθυνα για τη συλλογή σημάτων (Verilator/gtkwave) για περαιτέρω ανάλυση των απαιτήσεων χρονισμού που προκύπτουν από τις απαιτήσεις.

Έπειτα το Quartus IDE αναλαμβάνει την μετατροπή της Verilog HDL σε Bitstream για τον προγραμματισμό του FPGA. Για να γίνει αυτό με επιτυχία, θα πρέπει το Quartus IDE να περάσει τα στάδια της δημιουργίας netlist, layout, και STA με επιτυχία.

Το τελικό στάδιο είναι ο προγραμματισμός του FPGA μέσω του USB Blaster interface από το Quartus. Όταν ολοκληρωθεί χρησιμοποιούμε τα εργαλεία του LiteX για να ανεβάσουμε το πρόγραμμα μας στον soft επεξεργαστή.

#### 4.14 Δημιουργία RISC-V SoC ME LiteX

Για να ξεκινήσει η διαδικασία σχεδιασμού και υλοποίησης ενός SoC [42] πρέπει πρώτα να αναλυθούν οι απαιτήσεις του συστήματος. Από την ανάλυση των απαιτήσεων προκύπτουν οι Λειτουργικές και Μη Λειτουργικές απαιτήσεις.

##### 4.14.1 Λειτουργικές απαιτήσεις (*Functional Requirements*) SoC

- Υλοποίηση ενός 32bit RISC-V επεξεργαστή.
- Να μπορεί να τρέξει κάποιο πρόγραμμα επίδειξης.
- Να υλοποιείται μέσω του LiteX.
- Να υπάρχει UART για την επικοινωνία μέσω σειριακού τερματικού.
- Να περνάει με επιτυχία την προσομοίωση του Verilator.
- Να υλοποιείται με λιγότερα από 50000 logic cells.

##### 4.14.2 Μη Λειτουργικές Απαιτήσεις (*Non Functional Requirements*) SoC

- Να είναι ανοιχτής Αρχιτεκτονικής και να διατίθεται με ανοιχτές άδειες.
- Να υλοποιείται με ανοιχτό Λογισμικό.

##### 4.14.3 Ανάλυση Συστήματος

Στο SoC ενσωματώνεται ο Vexriscv [43] που είναι ένας πέντε σταδίων 32bit soft παραμετρικός επεξεργαστής με αρχιτεκτονική RV32I[MAFDC] RISC-V γραμμένος σε SpinalHDL που είναι μία παραλλαγή της Scala. Τα ενσωματωμένα περιφερειακά είναι η RAM και μία θύρα UART (115000-8-N-1) για την επικοινωνία του SoC με υπολογιστή. Τα περιφερειακά ενώνονται με τον επεξεργαστή μέσω του διαύλου επικοινωνίας Wishbone.

##### 4.14.4 Παραμετροποίηση *Architecture file*

Η ανάπτυξη του κώδικα ξεκινάει από το αρχείο που περιγράφει την αρχιτεκτονική. Από τον φάκελο `Litex-boards\platforms` βρίσκουμε το αρχείο `altera`. Πρέπει να παραμετροποιηθεί με τέτοιο τρόπο το αρχείο ώστε να περιγράφει σωστά τα περιφερειακά και πως ορίζονται αυτά μέσα στο datasheet του Altera DE10-Lite. Ακόμα να περιγράφεται σωστά η toolchain και ο προγραμματιστής του DE10-Lite.

##### 4.14.5 Παραμετροποίηση *Board file*

Αμέσως μετά παραμετροποιείται το αρχείο `terrasicde10lite.py` που βρίσκεται μέσα στον φάκελο `Litex-boards/targets`. Το αρχείο αυτό περιγράφει την υλοποίηση του SoC. Στα περιεχόμενα του πρέπει να βρίσκονται τα περιφερειακά καθώς και οι δίαυλοι επικοινωνίας. Καλώντας το σενάριο εντολών με την παράμετρο **-- build** μετατρέπουμε την migen σε Verilog HDL και μέσω του Intel Quartus μετατρέπεται σε bitstream.

#### 4.14.6 Προσομοίωση SoC

Η προσομοίωση του SoC έγινε μέσω του βοηθητικού προγράμματος *litex\_sim*. Μέσω του λογισμικού αυτού μπορεί να προσομοιωθεί το gateway και το software του SoC. Η προσομοίωση έγινε μέσω του τερματικού με τα παρακάτω flags:

- **Integrated-main-ram-size**= Ορισμός μεγέθους RAM
- **Sdram-module** = Δήλωση της RAM που έχει το de10lite board
- **Bus-standard** = Ορισμός του διαύλου επικοινωνίας
- **Bus-datawidth, bus-address-width** = Ορισμός σωστών μεγεθών
- **Integrated-rom-size** = Ορισμός μεγέθους ROM
- **With-analyzer,sim-debug,gtkwave-file** = Debug options που ορίζουν στον προσομοιωτή να σώσει αρχεία για περαιτέρω ανάλυση με Verilator + ModelSim

Δόθηκε δηλαδή στο τερματικό η παρακάτω εντολή:

```
litex_sim --integrated-main-ram=65536 --sdram-module=ISA42S16320 --integrated-rom-size=8000 --bus-standard=wishbone \
--bus-data-width=32 --bus-address-width=32 --cpu-type=vexrisc 0 --with-gpio -gtkwave-file --with-analyzer --sim-debug
```

Αφού έχουμε κάνει build το SoC έπειτα καλείτε το λογισμικό *litex\_sim* που στην πραγματικότητα είναι ένα python script που έχει σαν σκοπό την μεταγλώττιση του λογισμικού και την προσομοίωση του στο hardware που έχουμε υλοποιήσει. Μέσω του flag **--gtkwave-file** αποθηκεύουμε μία προσομοίωση του Verilator για περαιτέρω ανάλυση των σημάτων του SoC. Η Εικόνα 24 είναι από την προσομοίωση του υπό υλοποίηση σχεδίου εκτελέστηκε με επιτυχία. Το SoC δουλεύει όπως ήταν αναμενόμενο και όλα τα τεστ ολοκληρώθηκαν με επιτυχία.



```
[serial2console] loaded (0x563add257090)
[xgmi_ethernet] loaded (0x563add257090)
[serial2tcp] loaded (0x563add257090)
[spdeeprom] loaded (addr = 0x0)
[clocker] sys_clk: freq_hz=1000000, phase_deg=0

RISC-V SoC Using LiteX
Build your hardware, easily!

(c) Copyright 2012-2022 Enjoy-Digital
(c) Copyright 2007-2015 M-Labs

BIOS built on Aug 15 2022 10:04:27
BIOS CRC passed (1ade06ac)

LiteX git sha1: 552d7bdb

===== SoC =====
CPU: VexRiscv @ 1MHz
BUS: WISHBONE 32-bit @ 4GiB
CSR: 32-bit data
ROM: 78KiB
SRAM: 8KiB
MAIN-RAM: 64KiB

===== Initialization =====
Memtest at 0x40000000 (64.0KiB)...
Write: 0x40000000-0x40010000 64.0KiB
Read: 0x40000000-0x40010000 64.0KiB
Memtest OK
Memspeed at 0x40000000 (Sequential, 64.0KiB)...
Write speed: 1.6MiB/s
Read speed: 918.1KiB/s

===== Boot =====
Booting from serial...
Press Q or ESC to abort boot completely.
sL5DdSMmkekro
Timeout
No boot medium found

===== Console =====
litex> 
```

Εικόνα 24. Αποτέλεσμα προσομοίωσης

#### 4.14.7 GitHub

Χρησιμοποιήθηκε το github ως δημόσιο αποθετήριο για την αποθήκευση των αρχείων και βρίσκεται στην ηλεκτρονική διεύθυνση <https://github.com/Lefteris-B/RISC-V-SoC-Using-LiteX> [44].

Στον φάκελο code βρίσκεται ο κώδικας του demo. Στον φάκελο firmware βρίσκονται τα δυαδικά αρχεία για τον προγραμματισμό του SoC. Στον φάκελο sim τα αρχεία για την περαιτέρω προσομοίωση.

### 4.15 Προγραμματισμός του RISC-V SoC

#### 4.15.1 Προγραμματισμός του SoC

Καλώντας το σενάριο εντολών terrasicde10lite.py με την παράμετρο --load προγραμματίζουμε το Gateway στο FPGA μέσω του ενσωματωμένου σε αυτό USB Blaster. Μετά πρέπει να μεταγλωττιστεί το demo που βρίσκεται στον φάκελο

/litex/litex/soc/software/demo. Έπειτα συνδέουμε τη σειριακή CH340 και μέσω εφαρμογής τερματικού ανεβάζουμε το εκτελέσιμο στον επεξεργαστή. Εκτελέστηκαν οι παρακάτω εντολές:

```
/home/iamme/litex/litex/soc/software/demo/demo.py --build-  
path=/home/build/terasic_de10light  
litex_term /dev/ttyUSB0 --baud-rate=115200 --kernel=demo.bin
```

#### 4.15.2 Προγραμματισμός του επεξεργαστή με Demo εφαρμογή Χρήστη

Ο επεξεργαστής του SoC έχει σε αυτό το σημείο προγραμματιστεί. Μέσω οποιασδήποτε εφαρμογής τερματικού γίνεται επικοινωνία με το LiteX BIOS και καλούνται τα demo προγράμματα.

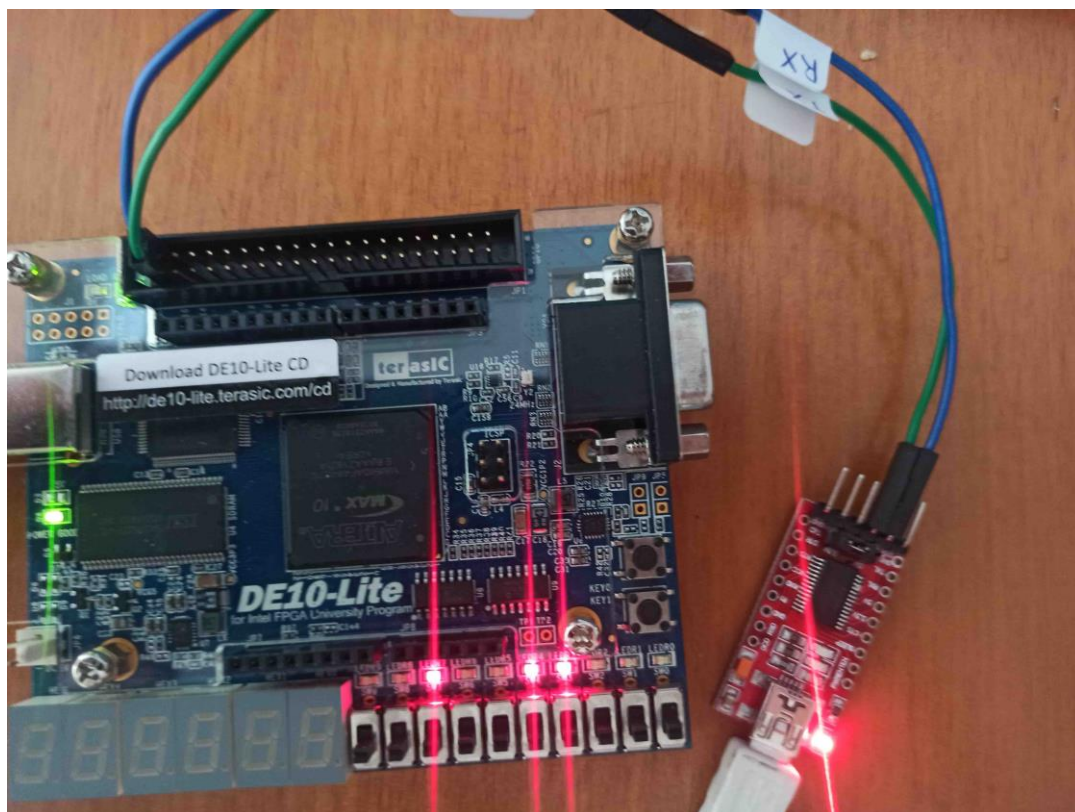
Το BIOS έχει τα εξής demo προγράμματα:

- **leds** = Εισάγουμε ένα αριθμό από το πληκτρολόγιο αυτός μετατρέπεται στο δυαδικό σύστημα και ανάβουν τα led στο FPGA για να υποδείξουν το αποτέλεσμα τις μετατροπής.
- **mem\_write** = Γράφεται ένα 32bit word σε μία θέση μνήμης.
- **mem\_read** = Διαβάζεται ένα 32bit word σε μία θέση μνήμης
- **mem\_speed** = Διαβάζονται τα πρώτα 2mb μνήμης και υπολογίζεται ο χρόνος που χρειάστηκε για να διαβαστούν.
- **boot\_serial** = bootstrapping code για την φόρτωση λογισμικού χρήστη. Φορτώνεται το λογισμικό στη μνήμη μέσω σειριακής και έπειτα καλείται το πρόγραμμα για εκτέλεση.
- **help** = Εμφανίζονται οι διαθέσιμες εντολές εκτέλεσης.

Το firmware έχει τα εξής demo προγράμματα:

- **led** = Το πρόγραμμα αυτό χρησιμοποιεί τα led του FPGA για δείξει ένα ακολουθιακό κύκλωμα μετρητή (counter) στο δυαδικό σύστημα. Το πρόγραμμα ξεκινάει από το μηδέν και συνεχίζει μέχρι το τριάντα δύο.
- **donut** = Το πρόγραμμα αυτό αποτελείται από βρόγχους που εναλλάσσουν σύμβολα στην οθόνη με σκοπό να δημιουργήσουν τη ψευδαίσθηση ενός τρισδιάστατου donut που γυρίζει γύρω από τον εαυτό του στην οθόνη.
- **hello** = Εμφανίζει ένα απλό μήνυμα "hello world" για να επιδείξει την ικανότητα εκτέλεσης προγραμμάτων της γλώσσας προγραμματισμού C.

Οι παρακάτω εικόνες παρουσιάζουν τα αποτελέσματα από την εκτέλεση διάφορων τύπων δοκιμαστικών προγραμμάτων.



Εικόνα 25 LED Demo με κώδικα Migen



```
litex_term /dev/ttyUSB0 --kernel=demo.bin

(c) Copyright 2007-2015 M-Labs

BIOS built on Jul 31 2022 19:49:34
BIOS CRC passed (205ec000)

LiteX git sha1: 7789e187

----- SoC -----
CPU:      VexRiscv @ 50MHz
BUS:      WISHBONE 32-bit @ 4GiB
CSR:      32-bit data
ROM:      128KiB
SRAM:     8KiB
MAIN-RAM: 32KiB

----- Initialization -----
Memtest at 0x40000000 (32.0KiB)...
  Write: 0x40000000-0x40008000 32.0KiB
  Read: 0x40000000-0x40008000 32.0KiB
Memtest OK
Memspeed at 0x40000000 (Sequential, 32.0KiB)...
  Write speed: 81.9MiB/s
  Read speed: 42.0MiB/s

----- Boot -----
Booting from serial...
Press Q or ESC to abort boot completely.
sL5DdSMmkekro
[LITEX-TERM] Received firmware download request from the device.
[LITEX-TERM] Uploading demo.bin to 0x40000000 (7260 bytes)...
[LITEX-TERM] Upload calibration... (inter-frame: 10.00us, length: 64)
[LITEX-TERM] Upload complete (9.7KB/s).
[LITEX-TERM] Booting the device.
[LITEX-TERM] Done.
Executing booted program at 0x40000000

----- Liftoff! -----

LiteX minimal demo app built Jul 31 2022 19:27:29

Available commands:
help      - Show this command
reboot    - Reboot CPU
led       - Led demo
donut     - Spinning Donut demo
helloC    - Hello C
litex-demo-app> 
```

Εικόνα 27. LiteX BIOS με demo προγράμματα για επίδειξη λειτουργίας SoC



#### 4.16 ΣΥΜΠΕΡΑΣΜΑΤΑ

Η δυνατότητα του LiteX μέσω λίγων γραμμών Python να επιτρέπει την εύκολη δημιουργία και παραμετροποίηση ενός SoC φαντάζει ασύλληπτη. Επιτρέπει την κατασκευή ενός πάρα πολλή σύνθετου κυκλώματος σε λίγα μόνο λεπτά.

Η μεγαλύτερη δύναμη του είναι η παραμετροποίηση του υλικού κατά την ανάπτυξη του λογισμικού. Επιτρέπει στους μηχανικούς υλικού τη δυνατότητα να αλλάζουν τις λειτουργικές απαιτήσεις ενός σχεδίου δυναμικά κατά την ανάπτυξη του υλικού και του λογισμικού δίνοντας ένα τεράστιο πλεονέκτημα της σχεδίασης αυτής στην αγορά. Ακόμα ένας έμπειρος μηχανικός υλικού μπορεί να αναπτύξει το δικό του περιφερειακό σε Migen και να το ενσωματώσει στο firmware του επεξεργαστή πάρα πολλή γρήγορα.

Μία ακόμη σύγκριση που πρέπει να γίνει είναι ανάμεσα στους αυτοματοποιημένους τρόπους κατασκευής SoC που προσφέρουν τα σύγχρονα IDE όπως το XILINX ISE και το INTEL Quartus. Τα λογισμικά αυτά δίνουν τη δυνατότητα μέσω μίας μεγάλης βιβλιοθήκης με IP blocks που μπορούν να πραγματοποιήσουν όλες τις κοινές διεργασίες μέσω ενός απλού GUI. Υπάρχουν IP blocks που είναι δωρεάν και IP blocks που χρειάζονται ειδική άδεια για να χρησιμοποιηθούν. Το αρνητικό είναι πως τα IP blocks δε μπορούν να χρησιμοποιηθούν σε άλλη πλατφόρμα. Επίσης σύμφωνα με την άδεια τους δε θα πρέπει να τροποποιούνται. Αυτό οδηγεί σε μη χρησιμοποιούμενα χαρακτηριστικά και ποιο σύνθετη λογική χωρίς να υπάρχει λόγος.

Το LiteX δεν έχει την ποικιλία που έχουν οι παραπάνω βιβλιοθήκες IP αλλά μπορούν να ενσωματωθούν σε αυτό περιφερειακά και IP blocks χωρίς κανένα πρόβλημα. Ακόμα μπορεί ο μηχανικός υλικού να παραμετροποιήσει των κώδικα άρα και τα λογικά κυκλώματα σε οποιοδήποτε στάδιο χωρίς κανένα πρόβλημα λόγω των ελεύθερων αδειών που το συνοδεύουν.

Στα μειονεκτήματα κατά τη χρήση του LiteX είναι κάποια προβλήματα που προκύπτουν από την χρήση της γλώσσας Migen και έπειτα από τις δυνατότητες των περιφερειακών. Καταρχάς η γλώσσα Python χρησιμοποιείται στα περισσότερα Linux συστήματα για τον αυτοματισμό διαδικασιών εντός του λειτουργικού. Η εγκατάσταση διαφορετικών εκδόσεων της Python πρέπει να γίνει με προσοχή λόγω του αντίκτυπου που μπορεί να έχει στην εύρυθμη λειτουργία του Λειτουργικού Συστήματος.

Η Python3 είναι εγγενώς μη ντετερμινιστική. Αυτό σημαίνει πως για τις ίδιες εισόδους μπορεί να έχει διαφορετικά αποτελέσματα και αρχικά θεωρείται ως χαρακτηριστικό που βοηθούσε στην ασφάλεια του κώδικα αλλά στην περίπτωση της περιγραφής υλικού κάτι τέτοιο δεν ισχύει. Το χαρακτηριστικό αυτό έχει να κάνει με το τρόπο που η Python

υλοποιεί τα λεξικά (dictionaries) και τους επαναλήπτες κατακερματισμού (hash iterators). Με τον τρόπο αυτό λειτουργίας αλλάζει δυναμικά με κάθε εκτέλεση του σεναρίου εντολών η Verilog netlist και ο χώρος διευθυνσιοδότησης των καταχωρητών. Αυτό το πρόβλημα μπορεί να διορθωθεί μέσω παραμετροποίησης του περιβάλλοντος της Python αλλάζοντας τη μεταβλητή PYTHONHASHSEED.

Τα ενσωματωμένα στο LiteX περιφερειακά έχουν μόνο πολλή βασικές δυνατότητες σε σχέση με τις δυνατότητες των περιφερειακών που υπάρχουν στην αγορά από εταιρείες όπως η STM32 και η NXP. Παρόλα αυτά, το LiteX framework είναι σημαντικά ταχύτερο κατά την υλοποίηση ενός SoC. Ακόμα είναι ταχύτερο κατά την προσομοίωση του SoC μέσω του πολυνηματικού μοντέλου προσομοίωσης του Verilator. Επίσης έχει τη δυνατότητα να χρησιμοποιεί εργαλεία και υλικά από διάφορους κατασκευαστές. Ακόμα το αποθετήριο του στο GitHub έχει 150 συνεισφέροντες προγραμματιστές και 1500 αστέρια και αυτό σημαίνει πως οι δυνατότητες του εμπλουτίζονται καθημερινά.

Μέσω εξερεύνησης και ανάλυση των τάσεων [45] που υπάρχουν αυτήν την στιγμή στο χώρο ανάπτυξης λογισμικού και υλικού για FPGA παρατηρείται η εκκίνηση πολλών startups που χρησιμοποιούν ανοιχτές αρχιτεκτονικές και λογισμικά για την ανάπτυξη SoC. Χαρακτηριστικό παράδειγμα αποτελεί η εταιρεία Rapid Silicon με τη σειρά εργαλείων ανοιχτού κώδικα Raptor [46] για την κατασκευή SoC και ASIC. Πρόκειται για μια toolchain όπου όλα τα λογισμικά που χρησιμοποιούνται είναι ανοιχτού κώδικα, με κυρίαρχο ρόλο να κατέχει το LiteX framework, και η αυτοματοποίηση των εργασιών μεταξύ εργαλείων πραγματοποιείται μέσω TCL scripts.

Μία ακόμη προσπάθεια αυτοματοποίησης της παραγωγής SoC είναι αυτό του ChipFlow project μίας Αγγλικής εταιρείας με PaaS (Platform as a Service) επιχειρηματικό μοντέλο [47]. Χρησιμοποιεί το LiteX και την Python για να αυτοματοποιήσει το σύνολο των εργασιών που είναι απαραίτητες από τη σύλληψη μίας επιχειρηματικής ιδέας μέχρι την παραγωγή ενός SoC.

Συνοψίζοντας το LiteX framework έχει τη δυνατότητα να διεκδικήσει σημαντικό μερίδιο στην αυτοματοποίηση κατασκευής SoC στην αγορά προσφέροντας μια σύγχρονη ροή σχεδίασης SoC που θα βασίζεται σε ανοικτά λογισμικά και θα αξιοποιεί τη νέα τάση στο υλικό υπολογιστών που βασίζεται στις ανοικτές αρχιτεκτονικές RISC-V.



## ΑΝΑΦΟΡΕΣ

- [1] Timeline of computer history | Computer history Museum. (n.d.). CHM. <https://www.computerhistory.org/timeline/>, Retrieved July 24, 2022
- [2] Το πλήθος των τρανζίστορ της CPU ανά έτος από το 1970 - 2020, Max Rosser, Hannah Ritchie. (2020). <https://commons.wikimedia.org/>, CC-BY-CA 4.0
- [3] H. D. Foster, "Why the design productivity gap never happened," 2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2013, pp. 581-584, doi: 10.1109/ICCAD.2013.6691175.
- [5] Foster, H. (2021, June 10). Beyond the water cooler: 2020 report on IC/ASIC design and verification trends. Semiconductor Engineering. <https://semiengineering.com/beyond-the-water-cooler-2020-report-on-ic-asic-design-and-verification-trends/>
- [6] Moore, Gordon E. (1965-04-19). "Cramming more components onto integrated circuits" (PDF). intel.com. Electronics Magazine. Retrieved April 1, 2022.
- [7] Interactive, W. L. (n.d.). Photomasks. Toppan Photomasks Inc. - The World's Premier Photomask Company. <https://www.photomask.com/products/euv-masks>
- [8] Smial. (2020, March 16). CPU AMD Epyc 7302P. <https://commons.wikimedia.com/>, GNU FDL 1.2
- [9] Field-programmable gate array (FPGA) market size, analysis & forecast. (2022, May 8). Verified Market Research. <https://www.verifiedmarketresearch.com/product/field-programmable-gate-array-market/>
- [10] Google LLC. (2020, March 16). Running TensorFlow on Cloud TPU. <https://cloud.google.com/>, CC-BY-CA 4.0
- [11] Aphex34. (2015, September 16). Typical CNN architecture. Πηγή <https://commons.wikimedia.org/>. CC-BY-CA 4.0
- [12] Brown, S., & Rose, J. (1996). FPGA and CPLD architectures: A tutorial. IEEE design & test of computers, 13(2), 42-57.
- [13] Hemani, A. (2004). Charting the EDA roadmap. IEEE Circuits and Devices Magazine, 20(6), 5-10.
- [Parsons, A., Backer, D., Siemion, A., Chen, H., Werthimer, D., Droz, P., ... & Wright, M. (2008). A scalable correlator architecture based on modular FPGA hardware, reusable gateway, and data packetization. Publications of the Astronomical Society of the Pacific, 120(873), 1207.
- [14] C. Tiri, K., & Verbaauwhede, I. (2006). A digital design flow for secure integrated circuits. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 25(7), 1197-1208. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello and Y. LeCun, "NeuFlow: A runtime reconfigurable dataflow processor for vision," CVPR 2011 WORKSHOPS, 2011, pp. 109-116, doi: 10.1109/CVPRW.2011.5981829.
- [15] Τι είναι το Gateway Πηγή: <https://tinyurl.com/2p8c2way> Retrieved April 1, 2022.
- [16] Villar, J. I., Juan, J., Bellido, M. J., Viejo, J., Guerrero, D., & Decaluwe, J. (2011, April). Python as a hardware description language: A case study. In 2011 VII Southern Conference on Programmable Logic (SPL) (pp. 117-122). IEEE.
- [17] Jou, J. Y., & Liu, C. N. J. (1999). Coverage analysis techniques for HDL design validation. Proc. Asia Pacific CHIP Design Languages, 48-55, Retrieved July 24, 2022
- [18] Waldrop, M. Mitchell (2016-02-09). "The chips are down for Moore's law". Nature. 530 (7589): 144–147. Bibcode:2016Natur.530..144W. doi:10.1038/530144a. ISSN 0028-0836. PMID 26863965.
- [19] Agrawal A, Mukhopadhyay S, Raychowdhury A, Roy K, Kim CH (2006) Leakage power analysis and reduction in nanoscale circuits. IEEE Micro 26(2):68–80
- [20] Alfke, P. (2005). FPGAs in 2005 and Beyond.
- [21] Jeffrey "Jefro" Osier-Mixon, & Stephano Cetola. (n.d.). Introduction to RISC-V. edx.org. <https://learning.edx.org/course/course-v1:LinuxFoundationX+LFD110x+1T2021>
- [16] History - RISC-V international. RISC. (2020, October 17). Retrieved March 26, 2022, from <https://riscv.org/about/history/>
- [17] RISC-V Genealogy | EECS at UC Berkeley (UCB/EECS-2016-6). (2016). UCBERKLEY. <https://www.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-6.html>, Retrieved July 24, 2022

- [22] Krste Asanović, & David A. Patterson. (2014). Instruction Sets Should Be Free: The Case For RISC-V [Doctoral dissertation]. <https://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-146.pdf>. Retrieved August 04, 2022
- [23] David A. Patterson and David R. Ditzel. 1980. The case for the reduced instruction set computer. SIGARCH Comput. Archit. News 8, 6 (October 1980), 25–33. DOI:<https://doi.org/10.1145/641914.641917>
- [24] Waterman, A., Lee, Y., Patterson, D. A., and Asanović, K. (2011). The RISC-V Instruction Set Manual, Volume I: Base User-Level ISA (EECS-2011-62).
- [25] Steve Hoover, Redwood EDA. (n.d.). Building a RISC-V CPU Core. edx.org. <https://learning.edx.org/course/course-v1:LinuxFoundationX+LFD111x+1T2021>. Retrieved August 04, 2022
- [26] Patterson, D., Hennessy, J. L. (2021). Computer Organization and Design RISC-V. Morgan Kaufmann.
- [27] Τεχνική Οργάνωση RISC-V, Πηγή: RISC-V technical organization. (n.d.). Google Docs. <https://tinyurl.com/msf2xf9n>, Retrieved August 04, 2022
- [28] [https://riscv.org/exchange/?\\_sft\\_exchange](https://riscv.org/exchange/?_sft_exchange), Retrieved August 04, 2022
- [29] Riscvarchive/riscv-cores-list. (n.d.). GitHub. <https://github.com/riscvarchive/riscv-cores-list>, Retrieved August 04, 2022
- [30] Hertzprung. (n.d.), Privilege rings for the x86,. <https://commons.wikimedia.org/> CC-BY-SA 3.0
- [31] RISC-V Priviledge Instruction Set. (n.d.). RISC-V International. <https://riscv.org/wp-content/uploads/2017/05/riscv-privileged-v1.10.pdf>, Retrieved August 04, 2022
- [32] Ο κύκλος επέκτασης ζωής για την RISC-V, Πηγή: RISC-V International wiki. <https://wiki.riscv.org/>
- [33] Enjoy-digital. (n.d.). Enjoy-digital/litex. GitHub. Retrieved July 24, 2022, from <https://github.com/enjoy-digital/litex>, Retrieved August 04, 2022
- [34] Τυπικό SoC Design flow με το LiteX Πηγή: <https://tinyurl.com/mrx8kpns>
- [35] Arachchi, S. A. I. B. S., & Perera, I. (2018, May). Continuous integration and continuous delivery pipeline automation for agile software project management. In 2018 Moratuwa Engineering Research Conference (MERCon) (pp. 156-161). IEEE.
- [36] Bui, T., 2015. Analysis of docker security. arXiv preprint arXiv:1501.02967.
- [37] Bui, T., 2015. Analysis of docker security. arXiv preprint arXiv:1501.02967.
- [38] Spinellis, D. (2012). Git. IEEE software, 29(3), 100-101.
- [39] Git workflow Πηγή: <https://tinyurl.com/2a6bj3f8>
- [40] Armenise, V. (2015, May). Continuous delivery with Jenkins: Jenkins solutions to implement continuous delivery. In 2015 IEEE/ACM 3rd International Workshop on Release Engineering (pp. 24-27). IEEE.
- [41] Herdt, V., & Drechsler, R. (2022). Advanced virtual prototyping for cyber-physical systems using RISC-V: Implementation, verification and challenges. Science China Information Sciences, 65(1), 1-17.
- [42] Goossens, K., Dielissen, J., Gangwal, O. P., Pestana, S. G., Radulescu, A., & Rijpkema, E. (2005, March). A design flow for application-specific networks on chip with guaranteed performance to accelerate SOC design and verification. In Design, Automation and Test in Europe (pp. 1182-1187). IEEE.
- [43] SpinalHDL/VexRiscv. (n.d.). GitHub. Retrieved August 4, 2022, from <https://github.com/SpinalHDL/VexRiscv>
- [44] Lefteris-B/Risc-V-SoC-Using-LiteX. (2022). GitHub. <https://github.com/Lefteris-B/RISC-V-SoC-Using-LiteX>, Retrieved August 04, 2022
- [45] Kuon, I., Tessier, R., & Rose, J. (2008). FPGA architecture: Survey and challenges. *Foundations and Trends® in Electronic Design Automation*, 2(2), 135-253.
- [46] (2022). Raptor toolchain. <https://rapidsilicon.com/raptor/> [Προσπελάστηκε 15/1/08]
- [47] ChipFlow UK. (2022). ChipFlow. Retrieved August 20, 2022, from <https://www.chipflow.io/> [Προσπελάστηκε 15/1/08]

## **ΠΑΡΑΡΤΗΜΑ Ι**

Η διεύθυνση του αποθετηρίου Github που μπορεί να βρεθεί ο κώδικας και τα αρχεία της παρούσας πτυχιακής είναι η εξής:

<https://github.com/Lefteris-B/RISC-V-SoC-Using-LiteX>