Πανεπιστήμιο Δυτικής Μακεδονίας

Τμήμα Μηχανικών Πληροφορικής & Τηλεπικοινωνιών

Ανάπτυξη γεννήτριας κυκλωμάτων υλοποίησης αριθμητικών πράξεων σε γλώσσα περιγραφής υλικού

Ιωάννης Πετρουσόβ

Επιβλέπων Καθηγητής : Δρ. Μηνάς Δασυγένης

Εργαστήριο Ψηφιακών Συστημάτων και Αρχιτεκτονικής Υπολογιστών

Οκτώβριος 2016

Κοζάνη

Περίληψη

Τα εργαλεία σχεδίασης υλικού παίζουν καθοριστικό ρόλο σε όλη τη διαδικασία ανάπτυξης νέων συστημάτων. Ιδιαίτερα χρήσιμα είναι αυτά που εκτελούν και αυτόματη εξερεύνηση χώρου και προτείνουν μία βέλτιστη λύση για την εκάστοτε περίπτωση. Τα εργαλεία αυτά δεν απαιτούν συγγραφή κώδικα από τον σχεδιαστή (ή απαιτούν ελάχιστη). Αντίθετα, δέχονται τις απαιτήσεις και τους περιορισμούς του κυκλώματος και παράγουν τη βέλτιστη περιγραφή υλικού. Τέτοια εργαλεία αναπτύσσονται από μεγάλες εταιρείες του χώρου (Xilinx) και συνοδεύονται από υψηλό κόστος ενώ ελάχιστα είναι αυτά που προσφέρονται δωρεάν.

Οι επιταχυντές υλικού [31], είναι συστήματα που έχουν τη δυνατότητα να εκτελούν μία λειτουργία ή ένα σύνολο λειτουργιών σε υψηλότερες συχνότητες από ένα κύκλωμα γενικού σκοπού CPU (Central Processing Unit) και από το λογισμικό. Αποτελούν δηλαδή βοηθητικά κυκλώματα με υψηλή διεκπεραιωτική ικανότητα που μπορούν να ενσωματωθούν-χρησιμοποιηθούν ως μέρος ενός μεγαλύτερου συστήματος. Παραδείγματα επιταχυντών αποτελούν τα ASIC (κυκλώματα ειδικού σκοπού) όπως οι κάρτες γραφικών και οι μεταλλευτές ψηφιακών νομισμάτων, όπως το Jalapeño.

Λαμβάνοντας υπόψη το πρόβλημα έλλειψης δωρεάν διαδικτυακών εργαλείων σχεδίασης υλικού και τη σημαντικότητα των επιταχυντών, αναπτύξαμε ένα διαδικτυακό εργαλείο παραγωγής επιταχυντών αριθμητικών κυκλωμάτων. Το εργαλείο δέχεται ως είσοδο μία προσαρμοσμένη αριθμητική συνάρτηση με ανεξάρτητο αριθμό μεταβλητών, σταθερών και εύρους bits. Σαν έξοδο, παράγει την περιγραφή του κυκλώματος σε VHDL και μία βιβλιοθήκη με τα κυκλώματα και υποκυκλώματα που χρησιμοποιήθηκαν. Επίσης ο χρήστης μπορεί να επιλέξει να παραχθεί ένα αρχείο testbench (δοκιμαστικών εισόδων) του επιταχυντή και μία εικόνα με το σχηματικό του κυκλώματος. Όλα τα αρχεία VHDL είναι συνθέσιμα για ASIC ή FPGA ανεξαρτήτως κατασκευαστή.

Λέξεις κλειδιά: Python, FPGA, GHDL, ASIC, Cython

Abstract

Hardware design tools play a key role throughout the process of developing new systems. Particularly useful are those that perform automatic space exploration and propose an optimal solution for each case. These tools do not require writing code from the designer (or require minimal). Instead, they accept the requirements and limitations of the circuit and produce an optimal hardware description. Such tools are developed by large companies in the field (Xilinx) and are accompanied by high costs while few are those that are offered for free.

Hardware accelerators [31], are systems that have the ability to perform a function or a set of functions at higher frequencies than software or a general purpose CPU (Central Processing Unit) circuit. These auxiliary circuits have high throughput and may be incorporated-used as part of a larger system. Examples of accelerators are ASICs (Application-specific integrated circuits) such as the graphics cards and digital currency miners like the Jalapeno.

Considering the problem of lack of free web based hardware design tools and the importance of accelerators, we have developed a web based tool for generating accelerators of arithmetic circuits. The tool accepts a customized arithmetic function with an independent number of variables, constants and bitwidths. As output, it produces the description of the circuit in VHDL and a library with the circuits and subcircuits that were utilized Also, the user may selectively choose to produce a testbench file and the schematic of the accelerator circuit. All generated VHDL source files are synthesizable for ASIC or FPGA and are independant from manufacturers.

Keywords: Python, FPGA, GHDL, ASIC, Cython

Δήλωση Πνευματικών Δικαιωμάτων

Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν. 1599/1986 και τα άρθρα 2,4,6 παρ. 3 του Ν. 1256/1982, η παρούσα Διπλωματική Εργασία με τίτλο ''Ανάπτυξη γεννήτριας κυκλωμάτων υλοποίησης αριθμητικών πράξεων σε γλώσσα περιγραφής υλικού'' καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας και αναφέρονται ρητώς μέσα στο κείμενο που συνοδεύουν, και η οποία έχει εκπονηθεί στο Τμήμα Μηχανικών Πληροφορικής και Τηλεπικοινωνιών του Πανεπιστημίου Δυτικής Μακεδονίας, υπό την επίβλεψη του μέλους του Τμήματος κ. Μηνά Δασυγένη αποτελεί αποκλειστικά προϊόν προσωπικής εργασίας και δεν προσβάλλει κάθε μορφής πνευματικά δικαιώματα τρίτων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή / και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα. Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και μόνο.

Υπογραφή Φοιτητή

Περιεχόμενα

Κατάλογος σχημάτων

Κατάλογος αλγορίθμων

Κατάλογος πινάκων

Κατάλογος απεικονίσεων

Κεφάλαιο 1

Εισαγωγή

1.1 Ορισμός του προβλήματος

Αδιαμφισβήτητα, τα τελευταία χρόνια η κατανάλωση ενέργειας είναι ένας παράγοντας που παίζει καθοριστικό ρόλο στη σχεδίαση ηλεκτρικών κυκλωμάτων και ενσωματωμένων συστημάτων. Με τη σμίκρυνση του τρανζίστορ καταφέραμε να μειώσουμε την κατανάλωση και να αυξήσουμε τον αριθμό των στοιχείων που μπορούν να χωρέσουν στη ίδια ψηφίδα, ενώ παράλληλα αυξήθηκε και η συχνότητα λειτουργίας. Ωστόσο, καθώς πλησιάζουμε τις ατομικές διαστάσεις, η απλή κλιμάκωση τελικά σταματά και παραβιάζονται οι φυσικοί κανόνες λειτουργίας και η ορθότητα [6]. Ίσως το ανώτατο όριο σήμερα αποτελεί το τρανζίστορ με ένα μόνο ηλεκτρόνιο (single electron transistor-SET) [17].

Ένας άλλος καθοριστικός παράγοντας στη σχεδίαση νέων κυκλωμάτων είναι οι απαιτήσεις της αγοράς. Τεχνολογίες όπως η 4K και 4G, ενσωματώνονται ακόμα και σε μικρές συσκευές όπως τα έξυπνα κινητά τηλέφωνα (smartphones) μαζί με τις ήδη υπάρχουσες, όπως το σύστημα πλοήγησης και τους διάφορους αισθητήρες (φωτός, αξελερόμετρο). Επιπλέον, το σύνθετο λειτουργικό σύστημα που χρησιμοποιείται, αλλά και κάθε είδους λογισμικό (software), όπως οι εφαρμογές και τα παιχνίδια, αυξάνουν την πολυπλοκότητα του ήδη σύνθετου υλικού (hardware). Για τους λόγους αυτούς, τα νέα μοντέλα smartphones βγαίνουν με βελτιωμένες επιδόσεις συχνότητας και μνήμης ώστε να μπορούν να αντεπεξέλθουν στις ολοένα αυξανόμενες απαιτήσεις.

Ωστόσο, η διαδικασία παραγωγής νέων εκδόσεων κυκλωμάτων δεν είναι πάντα ομαλή και μπορεί να αποτύχει. Το 80% των τσιπ δεν μπορεί να ικανοποιήσει τις

αρχικές προθεσμίες διάθεσης στην αγορά [34]. Ο χρόνος ανάπτυξης είναι πολύ περιορισμένος με αποτέλεσμα να ασκείται μεγάλη πίεση στις ομάδες σχεδίασης. Αρκετές είναι και οι φορές που οι ημερομηνίες αναστέλλονται και δημιουργείται οικονομική ζημία στην εκάστοτε εταιρεία.

Στο χώρο της σχεδίασης συστημάτων, η εξερεύνηση χώρου (Digital Space Exploration-DSE) είναι η δραστηριότητα της διερεύνησης εναλλακτικών λύσεων σχεδιασμού πριν από την εφαρμογή. Αποτελεί μία χρονοβόρα διαδικασία καθώς απαιτεί να βρεθεί μία λύση που ικανοποιεί τους περισσότερους αν όχι όλους τους περιορισμούς. Εξαιτίας χρονικών περιορισμών, ασκείται μεγάλη πίεση στις υπεύθυνες ομάδες ώστε να ικανοποιηθούν και οι περιορισμοί του χρόνου διάθεσης στην αγορά (time to market).

Στην παρούσα διπλωματική αναλύεται η εφαρμογή που αναπτύξαμε για την αυτόματη δημιουργία προσαρμοσμένων αριθμητικών κυκλωμάτων επιταχυντών. Σε αντίθεση με άλλα παρόμοια εργαλεία, η εφαρμογή μας έχει λιγότερες απαιτήσεις από τη μεριά του σχεδιαστή-χρήστη και προσφέρει εξοικονόμηση χρόνου στη διαδικασία σχεδίαση κυκλωμάτων. Παρακάτω, δίνεται μία συνοπτική περιγραφή των εργαλείων και βιβλιοθηκών που χρησιμοποιήθηκαν, ενώ αναλύεται με λεπτομέρεια η λειτουργία και διαδικασία παραγωγής περιγραφής υλικού.

1.2 Κίνητρα και Στόχοι Υλοποίησης

Τα εργαλεία ηλεκτρονικής σχεδίασης και αυτοματοποίησης (Electronic Design Automation-EDA) των διάφορων IP (Intellectual Property) blocks αποτελούν τα θεμέλια της παραγωγικότητας και επιτρέπουν στους μηχανικούς να κάνουν σημαντικά άλματα στην προτυποποίηση. Ίσως το πιο αποτελεσματικό εργαλείο είναι εκείνο το οποίο δεχόμενο τους περιορισμούς που πρέπει να ικανοποιηθούν (ενέργειας, χώρου, αριθμού στοιχείων, κρίσιμου μονοπατιού, και άλλων) εκτελεί αυτόματα την εξερεύνηση και προτείνει την καλύτερη λύση. Παρατηρήσαμε ότι υπάρχει έλλειψη διαθεσιμότητας τέτοιων εργαλείων και αυτά που προσφέρονται, αναπτύσσονται από μεγάλες εταιρείες και συνοδεύονται από υψηλό κόστος.

Οι αριθμητικές λειτουργίες και υπολογισμοί, αποτελούν μία σημαντική λειτουργία ενός σύγχρονου ενσωματωμένου συστήματος. Χαρακτηριστικό παράδειγμα τέτοιου συστήματος αποτελεί η μονάδα ελέγχου μηχανής ενός αυτοκινήτου (engine

control unit-ECU) [35]. Η μονάδα αυτή ρυθμίζει την απόδοση της μηχανής ελέγχοντας μία σειρά από ενεργοποιητές. Τα παλαιότερα χρόνια, το σύστημα αυτό είχε προκαθορισμένη συμπεριφορά. Διάβαζε την είσοδο από τον αισθητήρα και ρύθμιζε τη λειτουργία σύμφωνα με έναν προκαθορισμένο πίνακα αναζήτησης (lookup table). Τα νέα αυτοκίνητα ωστόσο χρησιμοποιούν εγκεφάλους που ρυθμίζουν τη λειτουργία εκτελώντας αριθμητικούς υπολογισμούς και μάλιστα μπορούν να προγραμματιστούν μέσω μία σειριακής θύρας, όπως η USB. Χαρακτηριστικό παράδειγμα επίσης αποτελούν οι επιταχυντές υλικού (hardware accelerators) [33], που αποτελούν εξειδικευμένα κυκλώματα που εκτελούν μία συγκεκριμένη λειτουργία ή ένα σύνολο λειτουργιών και συνήθως ενσωματώνονται σε μεγαλύτερα συστήματα. Η ιδιαιτερότητα τους είναι ότι εκτελούν συγκεκριμένες πράξεις σε πολύ μεγαλύτερες συχνότητες από ότι θα μπορούσε ένα γενικό κύκλωμα. Χαρακτηριστικά παραδείγματα αποτελούν οι μονάδες αριθμών κινητής υποδιαστολής (FPU), οι κάρτες γραφικών, ήχου και πολλές άλλες συσκευές.

Λαμβάνοντας υπόψη τους παράγοντες σχεδίασης, τις δυσκολίες προτυποποίησης και τη σημαντικότητα των επιταχυντών, στην παρούσα διπλωματική αναπτύχθηκε ένα εργαλείο το οποίο μπορεί να παράγει αυτόματα, κυκλώματα επιταχυντών προσαρμοσμένων αριθμητικών πράξεων. Ο σχεδιαστής δίνει ως είσοδο την αριθμητική συνάρτηση με ανεξάρτητο αριθμό μεταβλητών και σταθερών και ανεξάρτητου εύρους bit. Ως έξοδο, λαμβάνει το κύκλωμα που εκτελεί τη συνάρτηση σε γλώσσα περιγραφής υλικού VHDL και μία βιβλιοθήκη με τις πύλες και τα υποκυκλώματα που χρησιμοποιήθηκαν. Η βιβλιοθήκη μπορεί να επαναχρησιμοποιηθεί σε άλλα κυκλώματα. Επιπλέον ο χρήστης μπορεί να επιλέξει να δημιουργηθεί μία εικόνα του σχηματικού του κυκλώματος και η περιγραφή της εικόνας σε γλώσσα dot. Για την επιβεβαίωση της ορθότητας λειτουργίας του κυκλώματος, ο χρήστης μπορεί να επιλέξει να παραχθεί αυτόματα ένα αρχείο testbench με τυχαίες δοκιμαστικές εισόδους και υπολογισμένες εξόδους το οποίο προσομοιώνει τη λειτουργία της γεννήτριας και επιβεβαιώνει την ορθότητα των πράξεων. Ο κώδικας είναι συνθέσιμος για FPGA (προγραμματιζόμενες συστοιχίες πυλών στο πεδίο) και για ASIC (ολοκληρωμένα κυκλώματα ειδικού σκοπού) ενώ είναι ανεξάρτητος από τον κατασκευαστή του υλικού. Το εργαλείο είναι προσβάσιμο μέσω internet[1], δεν απαιτεί τοπική εγκατά-

[1] `http://arch.icte.uowm.gr/hdl/equationparser_json.php`

σταση και είναι διαθέσιμο για χρήση προς όλους.

1.3 Περιπτώσεις παρόμοιων εργαλείων

Στο χώρο της ηλεκτρονικής σχεδίασης υπάρχουν ορισμένα παρόμοια εργαλεία που ξεχωρίζουν για τη χρηστικότητα και την ανάπτυξη που έχουν. Βήματα προς την αυτοματοποίηση γίνονται φυσικά από τις μεγάλες εταιρίες όπως είναι η Altera, η Xilinx η Synopsis, η Mathworks και άλλες. Ωστόσο, τα εργαλεία που προσφέρουν αυτές οι εταιρείες είναι εμπορικά, έχουν υψηλό κόστος και δεν είναι δωρεάν διαθέσιμα προς όλους. Επίσης τα εργαλεία αυτά δημιουργούν επιταχυντές που προορίζονται για σύνθεση σε πλακέτες συγκεκριμένου κατασκευαστή. Καθώς η γεννήτρια μας είναι ελεύθερη, θα γίνει μεγαλύτερη ανάλυση των επίσης ελεύθερων προς χρήση εργαλείων.

Μία κατηγορία εργαλείων είναι αυτά που παράγουν HDL κώδικα από μια γλώσσα προγραμματισμού υψηλού επιπέδου, όπως είναι η Ruby, ή μία γλώσσα συγκεκριμένου τομέα (Domain Specific Language-DSL). Στην περίπτωση αυτή οι δομές και οι συνδέσεις περιγράφονται με συγκεκριμένο συντακτικό που έχει καθοριστεί για το εργαλείο. Ένα παράδειγμα μιας δομής μπορεί να αποτελεί ένας καταχωρητής DFF (D flip flop). Προϋπόθεση για να χρησιμοποιήσει κάποιος ένα τέτοιο εργαλείο είναι να γνωρίζει την εκάστοτε γλώσσα προγραμματισμού και τη σύνταξη των δομών του κυκλώματος που θέλει να φτιάξει.

Ένα τέτοιο εργαλείο είναι το MyHDL [11]. Ουσιαστικά αποτελεί ένα πακέτο του οικοσυστήματος της Python, είναι ελεύθερο και ανοιχτού κώδικα και μπορεί να εγκατασταθεί χρησιμοποιώντας τον διαχειριστή πακέτων pip. Η σχεδίαση γίνεται δηλώνοντας μια σειρά από διακοσμητές (Python decorators) που περιγράφουν συμπεριφορά υλικού. Η γεννήτρια αυτή έχει τη δυνατότητα να παράγει την περιγραφή σε Verilog είτε σε VHDL. Επίσης, μπορεί να χρησιμοποιηθεί ως γλώσσα επιβεβαίωσης λειτουργίας κυκλωμάτων με τη δημιουργία testbench.

Στην ίδια κατηγορία εργαλείων ανήκει και το Bambu, ένα framework το οποίο δέχεται ως είσοδο τη συμπεριφοριστική περιγραφή (behavioral description) των προδιαγραφών, γραμμένο σε γλώσσα C, και δημιουργεί την περιγραφή HDL της αντίστοιχης εφαρμογής ως έξοδο [26]. Το έργο αυτό έχει δεχτεί χορηγίες από τις μεγαλύτερες εταιρίες του χώρου, όπως η Xilinx και η Altera, αλλά και από την Ευ-

ρωπαϊκή Ένωση, ενώ έχει συνεχώς ανανεούμενη λίστα δημοσιεύσεων. Τέλος, έχει τη δυνατότητα να παράγει κώδικα επιβεβαίωσης λειτουργίας (testbench).

Σχεδίαση ηλεκτρονικών κυκλωμάτων υποστηρίζει και η γλώσσα προγραμματισμού Haskell μέσω των εργαλείων Lava [4] και ClaSH [20]. Το Lava εκμεταλλεύεται τα χαρακτηριστικά της συναρτησιακής προσέγγισης (της Haskell) και προσφέρει πολλαπλές ερμηνείες μιας περιγραφής κυκλώματος. Η περιγραφή των κυκλωμάτων σε Lava γίνεται με τη δημιουργία μίας netlist στο επίπεδο των πυλών (gate level) στην οποία περιγράφονται λειτουργίες που αντιπροσωπεύουν βασικά στοιχεία της βιβλιοθήκης ή σύνθετα κυκλώματα. Η δομή αυτή μπορεί να προσομοιωθεί και να μετατραπεί σε VHDL χρησιμοποιώντας το Xilinx Synthesis Tool (XST). Μία παραλλαγή του Lava δημιουργήθηκε από το πανεπιστήμιο του Kansas και ονομάζεται Kansas Lava [14]. Η έκδοση αυτή χρησιμοποιεί πιο μοντέρνες τεχνικές των συναρτησιακών γλωσσών (applicative functors) για να αυξήσει την περιγραφικότητα του υλικού. Το ClaSH είναι μία συναρτησιακή DSL εμπνευσμένη από τη Haskell. Στη γλώσσα αυτή κάθε συνάρτηση δηλώνει ένα κυκλωματικό στοιχείο και η κλήση της δηλώνει ένα στιγμιότυπο του. Η δομές της ClaSH μεταγλωττίζονται από το ομώνυμο πρόγραμμα και παράγονται περιγραφές υλικού. Η περιγραφή μπορεί να προσομοιωθεί και να παραχθεί αντίστοιχος κώδικας σε VHDL, Verilog, ή SystemVerilog.

Ένα τελευταίο εργαλείο αυτής της κατηγορίας που χρησιμοποιεί μία DSL για να περιγράψει ένα σύστημα και είναι διαδικτυακό, είναι το έργο PSHDL [2]. Η γλώσσα αυτή επιτρέπει στον προγραμματιστή να εκφράσει σύνθετες δομές με πιο απλό τρόπο αποφεύγοντας την αυστηρότητα της VHDL. Το εργαλείο είναι γραμμένο σε Java, είναι ανοιχτού κώδικα και έχει συνεχής ανάπτυξη.

Μία άλλη κατηγορία εργαλείων είναι εκείνα που δεν απαιτούν από τον χρήστη γνώσεις προγραμματισμού, DSL και εσωτερικής λειτουργίας. Αντίθετα, η περιγραφή του κυκλώματος δίνεται μέσω μιας απλής διεπαφής όπου εισάγονται οι παράμετροι που καθορίζουν τις ιδιότητες του κυκλώματος που θα παραχθεί. Ορισμένες τέτοιες παράμετροι αποτελούν το εύρος bit των εισόδων και η διασωλήνωση. Έτσι ο χρήστης μπορεί να εκτελέσει ταχεία εξερεύνηση του χώρου αλλάζοντας απλά τις παραμέτρους του κυκλώματος.

Σε αυτή τη κατηγορία, που ανήκει και το δικό μας εργαλείο, εντάσσεται και το έργο SPIRAL [27]. Η γεννήτρια που παρέχεται είναι προσβάσιμη διαδικτυακά και

η παραμετροποίηση των κυκλωμάτων γίνεται μέσω HTML φορμών. Η βιβλιοθήκη επιταχυντών που διατίθεται ωστόσο είναι περιορισμένη και πολύ εξειδικευμένης εφαρμογής. Ένα παράδειγμα ενός επιταχυντή είναι ο Διακριτός Μετασχηματισμός Fourier (DFT). Τα κυκλώματα περιγράφονται μόνο σε Verilog ενώ δεν παρέχεται κώδικας επιβεβαίωσης λειτουργίας.

Το FloPoCo [10] είναι ένα εργαλείο γραμμής εντολών που μπορεί να δημιουργεί κυκλώματα αριθμητικών πυρήνων. Η γεννήτρια είναι γραμμένη σε C++, είναι ανοιχτού κώδικα και έχει μία συνεχής ανάπτυξη με την προσθήκη νέων πυρήνων κυκλωμάτων. Η γλώσσα περιγραφής που παράγει είναι VHDL ενώ μπορεί να δημιουργήσει κυκλώματα που υποστηρίζουν αριθμούς κινητής υποδιαστολής και διασωλήνωση (pipeline).

Τα περισσότερα από τα προηγουμένως αναφερθέντα εργαλεία απαιτούν εγκατάσταση ή και μεταγλώττιση (compile) του κώδικα για το εκάστοτε λειτουργικό σύστημα, μια διαδικασία που απαιτεί ικανοποίηση περιορισμών εκδόσεων απαραίτητων βιβλιοθηκών και εργαλείων που απαιτούνται (π.χ. gcc 4.4.4). Η διαδικασία αυτή απαιτεί δικαιώματα υπερχρήστη και γνώσεις διαχείρισης λειτουργικών συστημάτων. Ακόμα και μετά την εγκατάσταση, ο σχεδιαστής πρέπει να διαβάσει τα βοηθητικά έγγραφα (documentation) για τις οδηγίες χρήσης και πρέπει να εξοικειωθεί με τη γλώσσα προγραμματισμού η οποία μπορεί να είναι υψηλού επιπέδου ή DSL. Η διαδικασία αυτή είναι χρονοβόρα και αποσπαστική από τον κύριο σκοπό, που είναι η παραγωγή κυκλωμάτων.

Σε αντίθεση με τα προαναφερθέντα εργαλεία, η γεννήτρια μας προσφέρεται σαν διαδικτυακή υπηρεσία και δεν απαιτεί εγκατάσταση και γνώσεις προγραμματισμού python ή εσωτερική λειτουργίας. Η εισαγωγή παραμέτρων γίνεται μέσω απλών HTML φορμών που καθορίζουν τη συμπεριφορά του παραγόμενου κυκλώματος και παραμετροποιούν την έξοδο. Τέλος, σε αντίθεση με τα εμπορικά εργαλεία της Xilinx και της Altera, ο κώδικας είναι συνθέσιμος για οποιοδήποτε μοντέλο FPGA ή ASIC ανεξαρτήτως κατασκευαστή.

1.4 Διάρθρωση κειμένου

Τα υπόλοιπα κεφάλαια οργανώνονται ως εξής. Στο κεφάλαιο 2 δίνεται το θεωρητικό υπόβαθρο των εννοιών που χρησιμοποιούνται, ακολουθεί η περιγραφή του

λογισμικό μέρος του συστήματος στο κεφάλαιο 3. Στο κεφάλαιο 4 δίνονται ορισμένα πειραματικά αποτελέσματα που προέκυψαν από την ανάλυση κυκλωμάτων που παράχθηκαν. Στο κεφάλαιο 5 παρουσιάζεται μία προοπτική ανάπτυξης μίας startup βασισμένη στο εργαλείο και δίνεται η ανάλυση SWOT αυτής. Τέλος, το κεφάλαιο 6 κλείνει με συμπεράσματα και μελλοντικές επεκτάσεις.

Κεφάλαιο 2

Θεωρητικό υπόβαθρο

Το κεφάλαιο παρουσιάζει τη θεωρία στην οποία βασίζεται η παρούσα διπλωματική εργασία. Εξηγούνται οι λέξεις κλειδιά, παρουσιάζονται οι γλώσσες προγραμματισμού και δίνονται τα εργαλεία και οι τεχνολογίες που αξιοποιήθηκαν. Παράλληλα, τεκμηριώνεται η επιλογή και το σημείο στο οποίο χρησιμοποιήθηκε η κάθε τεχνολογία.

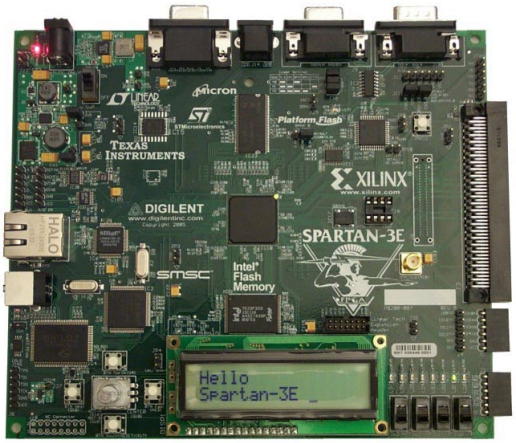
2.1 FPGA

Οι FPGAs [5] (Field Programmable Gate Array) είναι συσκευές ημιαγωγών που βασίζονται γύρω από μια μήτρα διαμορφώσιμων λογικών μπλοκ με προγραμματιζόμενες συνδέσεις. Αποτελούν κυκλώματα γενικού σκοπού και μπορούν να επαναπρογραμματιστούν ανάλογα με την εφαρμογή και τη λειτουργία που πρέπει να εκτελέσουν. Αποτελούν κατάλληλες συσκευές για τις περιπτώσεις όπου οι παράμετροι λειτουργίας αλλάζουν και πρέπει να προσαρμόζεται το κύκλωμα και οι δυνατότητες του. Στο σχήμα 2.1 δίνεται ως παράδειγμα μία πλακέτα FPGA της εταιρείας Xilinx. Στην αγορά υπάρχουν πολλές πλακέτες που έχουν περισσότερα οι λιγότερα χαρακτηριστικά. Ένα χαρακτηριστικό αποτελεί η συχνότητα λειτουργίας ρολογιού, η ύπαρξη θύρας κάρτας μνήμης, η ύπαρξη ελεγκτή δικτύου και άλλα. Η τιμή μιας τέτοιας πλακέτας διαμορφώνεται ανάλογα με τα χαρακτηριστικά που έχει.

2.2 ASIC

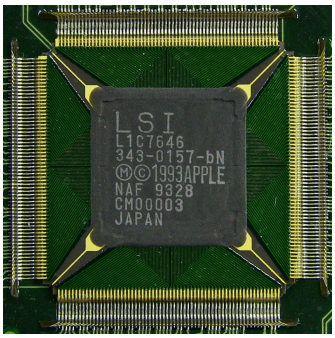
Το ASIC [13] (application-specific integrated circuit) αποτελεί ένα κύκλωμα μικροεπεξεργαστή συγκεκριμένου σκοπού και σε αντίθεση με τις FPGA, μπορεί να

Σχήμα 2.1: Παράδειγμα μιας πλακέτα FPGA.



εκτελεί μία συγκεκριμένη λειτουργία ή ένα σύνολο λειτουργιών και δεν είναι επαναπρογραμματιζόμενο. Οι επεξεργαστές αυτοί έχουν το πλεονεκτήματα τις πολλαπλάσιας συχνότητας λειτουργίας, τις μικρής κατανάλωσης ενέργειας και τις μικρότερες διαστάσεις. Επίσης το κόστος ενός ASIC είναι πολύ χαμηλότερο από αυτό μιας FPGA. Παράδειγμα ενός τέτοιου ολοκληρωμένου κυκλώματος δίνεται στο σχήμα 2.2. Όταν ένας μικροεπεξεργαστής συμπεριλαμβάνει περισσότερους από έναν πυρήνες, αναφέρεται σαν system on a chip (SoC). Εξαιτίας των παραπάνω πλεονεκτημάτων, αποτελεί ιδανικό κύκλωμα για εγκατάσταση στην τελική συσκευή, όπου οι λειτουργίες έχουν καθοριστεί και προγραμματιστεί.

Σχήμα 2.2: Παράδειγμα ενός ASIC chip.



2.3 VHDL

Η VHDL [1] (VHSIC Hardware Description Language) είναι μια γλώσσα περιγραφής της δομής και της συμπεριφοράς των ψηφιακών ηλεκτρονικών κυκλωμάτων και συστημάτων. Δημιουργήθηκε στις αρχές της δεκαετίας του 1980 ως υποπροϊόν

από ένα πρόγραμμα που χρηματοδοτούνταν από το υπουργείο άμυνας της Αμερικής. Με το πέρασμα του χρόνου η γλώσσα προτυποποιήθηκε από την IEEE ύστερα από σημαντικές βελτιώσεις και τροποποιήσεις. Σήμερα χρησιμοποιείται ως γλώσσα προγραμματισμού και προσομοίωσης ψηφιακών συστημάτων.

Για να χρησιμοποιηθεί μία περιγραφή VHDL σε ένα ψηφιακό σύστημα, πρέπει να προηγηθεί μία διαδικασία που ονομάζεται σύνθεση. Η διαδικασία αυτή εκτελείται από εργαλεία που μεταφράζουν την περιγραφή υψηλού επιπέδου σε περιγραφή στο επίπεδο των πυλών (gate level). Εξαιτίας των πολλών δομημάτων που προσφέρει η VHDL, ορισμένες περιγραφές δεν μπορούν να περάσουν από σύνθεση και να χρησιμοποιηθούν σε πραγματικό υλικό. Παράδειγμα μη συνθέσιμης περιγραφής αποτελούν οι καθυστερήσεις συγκεκριμένου χρόνου (wait, after). Αυτές οι δομές μπορούν να χρησιμοποιηθούν μόνο στην προσομοίωση του κυκλώματος από κατάλληλα προγράμματα. Όπως θα περιγραφεί αργότερα, η γεννήτρια μας χρησιμοποιεί τέτοιες δομές για να παράξει κώδικα που επιβεβαιώνει την ορθότητα λειτουργίας των επιταχυντών μας.

2.4 Python

Η Python είναι μια γενικού σκοπού υψηλού επιπέδου γλώσσα προγραμματισμού [32]. Η σύλληψη της έγινε τη δεκαετία του 1980 από τον Ολλανδό μαθηματικό Guido Van Rossum και πρωτοεμφανίστηκε το 1991. Υποστηρίζει πολλαπλά στυλ προγραμματισμού, όπως αντικειμενοστραφή, συναρτησιακό και ακολουθιακό. Παρά το γεγονός ότι είναι διερμηνευόμενη (interpreted), ο κώδικας της μπορεί να μεταφραστή (compiled) και να παραχθεί εκτελέσιμο αρχείο (*.exe). Ορισμένα από τα μεγαλύτερα πακέτα της αποτελούν το Django και το Flask που χρησιμοποιούνται στη δημιουργία διαδικτυακών εφαρμογών, ενώ ακόμα πιο πολλά πακέτα υπάρχουν για τον επιστημονικό τομέα (NumPy, matplotlib, SimPy, ...).

Πλεονέκτημα σε σχέση με άλλες γλώσσες αποτελεί το γεγονός ότι επιτρέπει την έκφραση εννοιών και τη δημιουργία συναρτήσεων με λιγότερες γραμμές κώδικα. Το συντακτικό της βασίζεται στις εσοχές και υποχρεώνει τον προγραμματιστή να δίνει όμορφη δόμηση στον κώδικα του. Η ευρεία χρήση της φαίνεται στο μεγάλο αριθμό βιβλιοθηκών που διατίθεται προς εγκατάσταση η οποία γίνεται πολύ απλά χρησιμοποιώντας το πρόγραμμα διαχείρισης πακέτων της, pip (python package index).

Στην παρούσα διπλωματική χρησιμοποιήθηκε η έκδοση 2.7.6 της CPython.

2.5 Cython

Η Cython είναι μία επέκταση της Python που επιτρέπει τη ρητή δήλωση μεταβλητών και συναρτήσεων [3] όπως στη C. Ο κώδικας γραμμένος σε Cython μεταφράζεται απευθείας σε C χρησιμοποιώντας ένα αρχείο εγκατάστασης (setup.py) και μεταγλωττίζεται σε κώδικα μηχανής (object file) για το εκάστοτε λειτουργικό σύστημα. Η χρήση των συναρτήσεων που περικλείονται μπορεί να γίνει πολύ απλά με την εισαγωγή (import) του αρχείου στο κώδικα της python.

Η τεχνική του συνδυασμού της Python και της Cython αυξάνει κατά πολύ την ταχύτητα εκτέλεσης των προγραμμάτων. Ιδιαίτερα αποτελεσματική είναι η χρήση της σε πολλαπλούς ένθετους βρόχους. Στη γεννήτρια μας καταφέραμε να βελτιώσουμε την ταχύτητα εκτέλεσης κατά 35% γράφοντας δύο συναρτήσεις σε Cython.

Ωστόσο η γλώσσα αυτή έχει και ορισμένα μειονεκτήματα. Ένα σημαντικό αποτελεί η έλλειψη βιβλιοθηκών που υποστηρίζει η Cython. Για παράδειγμα δεν μπορούμε να χρησιμοποιήσουμε τη βιβλιοθήκη Django, που χρησιμοποιείται για προγραμματισμό διαδικτυακών εφαρμογών, και να κάνουμε στατικές δηλώσεις. Ίσως το πιο σημαντικό μειονέκτημα αποτελεί το γεγονός ότι δεν μπορούμε να κάνουμε debug τη Cython με τον αποσφαλματωτή (debugger) pudb που έχουμε επιλέξει. Ωστόσο, ο κλασσικός απασφαλματωτής της Python, pdb, έχει περισσότερες δυνατότητες και μπορεί να καταλάβει κώδικα μηχανής και να τον εκτελέσει κατά βήμα.

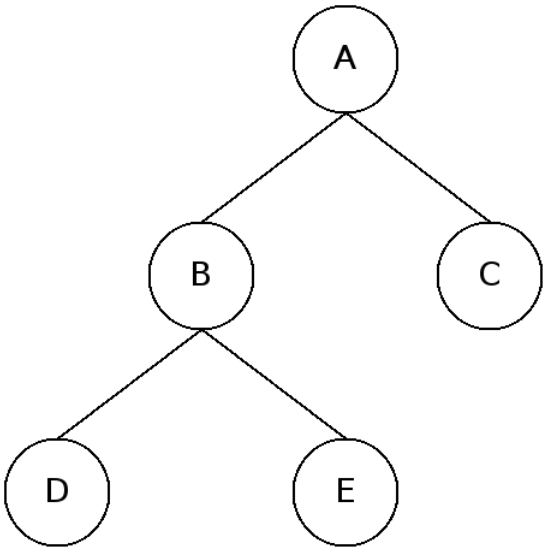
2.6 Δέντρα έκφρασης

Δέντρο, είναι μία δομή δεδομένων που αποτελείται από ένα σύνολο κόμβων που συνδέονται με ακμές [12]. Κάθε δέντρο διαθέτει έναν αρχικό κόμβο από τον οποίο ξεκινάει η διακλάδωση, αυτός ο κόμβος ονομάζεται ρίζα του δέντρου. Ο κόμβος που δεν έχει άλλη διακλάδωση, ονομάζεται φύλλο. Όταν υπάρχει ακμή που συνδέει έναν κόμβο x με έναν κόμβο y, και ο κόμβος y είναι σε χαμηλότερο επίπεδο από τον x, τότε ο x ονομάζεται πατέρας και ο y ονομάζεται παιδί. Οι κόμβοι που δεν διαθέτουν παιδιά, δεν έχουν άλλη διακλάδωση, ονομάζονται φύλλα του δέντρου. Ένα παράδειγμα εφαρμογής των δέντρων αποτελεί το σύστημα αρχείων του Unix,

όπου οι κατάλογοι εντάσσονται σε κόμβους και τα μονοπάτια τους σε ακμές.

Μία ειδική κατηγορία δέντρων στα οποία η διακλάδωση είναι πάντα διπλή, δηλαδή κάθε κόμβος έχει 2 παιδιά, ονομάζονται δυαδικά δέντρα. Στο σχήμα 2.3 δίνεται παράδειγμα ενός δυαδικού δέντρου. Ο κόμβος A αποτελεί τη ρίζα, ενώ οι κόμβοι B και C είναι απόγονοι του. Ο κόμβος B είναι πατέρας των κόμβων D και E οι οποίοι μαζί με τον C αποτελούν τα φύλλα του δέντρου.

Σχήμα 2.3: Παράδειγμα δυαδικού δέντρου.



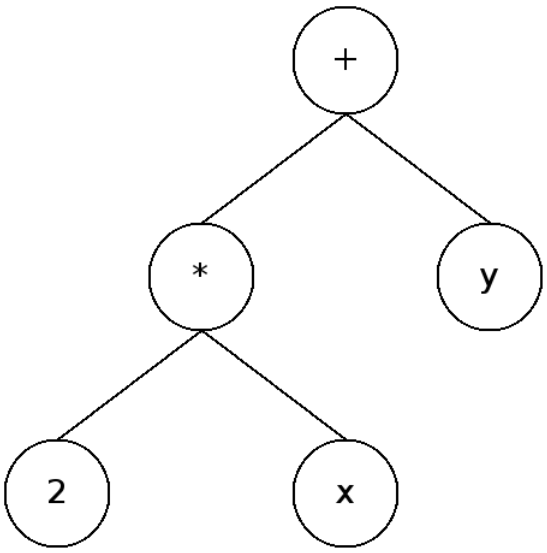
Τα δυαδικά δέντρα που χρησιμοποιούνται για να αναπαραστήσουν μία αριθμητική έκφραση, ονομάζονται δέντρα αριθμητικών εκφράσεων (expression trees). Σε αυτή την κατηγορία δέντρων, κάθε τελεστής που δηλώνει μία αριθμητική πράξη, είναι πάντα εσωτερικός κόμβος (πατέρας) με δύο παιδιά. Οι κόμβοι που είναι φύλλα, είναι πάντα μεταβλητές και αριθμοί. Ένα παράδειγμα δέντρου έκφρασης για τη συνάρτηση $f(x, y) = ((2 * x) + y)$ φαίνεται στο σχήμα 2.4.

2.7 Εργαλεία που χρησιμοποιήθηκαν

2.7.1 FreeBSD

Το FreeBSD [21] είναι το λειτουργικό σύστημα που επιλέχθηκε για να φιλοξενήσει την εφαρμογή. Αποτελεί τον άμεσο απόγονο του Unix και φημίζεται για τη σταθερότητα και την ασφάλεια του. Σε αντίθεση με τα λειτουργικά συστήματα που βασίζονται στο Linux, στο FreeBSD η ανάπτυξη του πυρήνα και των βοηθητικών προγραμμάτων (world) γίνονται στο ίδιο αποθετήριο (repository) και σε ένα ένα

Σχήμα 2.4: Παράδειγμα δέντρου έκφρασης της συνάρτησης $f(x, y) = ((2 * x) + y)$.



κλαδί (branch), ακολουθείται δηλαδή το σταδιακό (incremental) μοντέλο ανάπτυξης [16]. Επίσης το FreeBSD είναι λειτουργικό σύστημα που βασίζεται σε κώδικα (source based), δηλαδή τα επιπλέον πακέτα (ports) διατίθεται σε μορφή μορφή πηγαίου κώδικα που πρέπει να μεταγλωττιστεί (compile). Επιλέχθηκε το συγκεκριμένο λειτουργικό λόγο μεγαλύτερης εξοικείωσης στη χρήση, σταθερότητας και ασφάλειας που προσφέρει.

2.7.2 vim

Για τη συγγραφή του κώδικα χρησιμοποιήθηκε αποκλειστικά το πρόγραμμα σύνταξης vim [22]. Προτιμήθηκε το συγκεκριμένο πρόγραμμα λόγο μεγαλύτερη εξοικείωσης και ύπαρξης πολλών επεκτάσεων για τη γλώσσα προγραμματισμού Python. Σημαντικό παράγοντα έπαιξε και το γεγονός ότι ο προγραμματισμός έγινε σε τερματικό καθώς πρόκειται για πρόγραμμα backend.

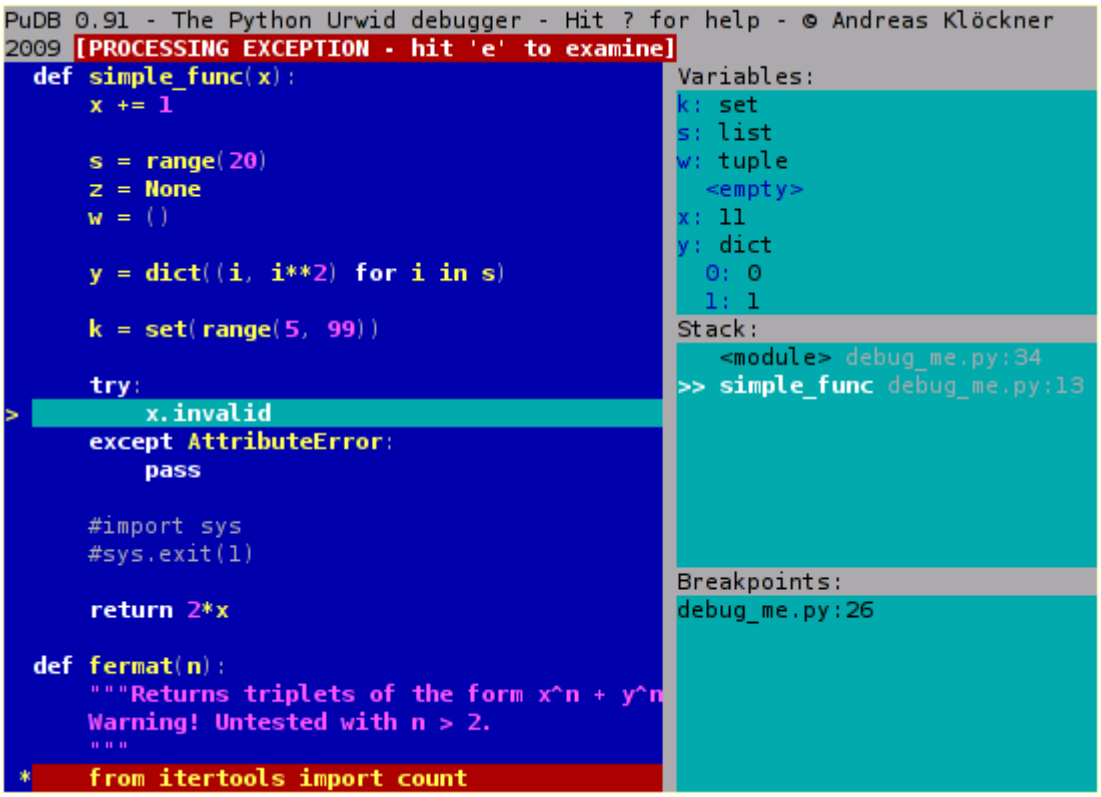
2.7.3 Nginx

Το Nginx είναι ένας διακομιστής υπερκειμένου (web server) και αντίστροφος διαμεσολαβητής (reverse proxy) [29]. Διακρίνεται για την υψηλή του απόδοση, σταθερότητα, απλότητα στη χρήση και χαμηλή κατανάλωση πόρων. Επίσης βασίζεται σε γεγονοδημούμενη-ασύγχρονη (event-driven) αρχιτεκτονική.

2.7.4 Pudb

Ο απασφαλματωτής που χρησιμοποιήθηκε ονομάζεται PuDB [19] (Python Debugger). Στόχος του είναι να ενσωματώνει όλα τα χαρακτηριστικά των μοντέρνων απασφαλματωτών και να τα προσφέρει σε ένα ελαφρύ περιβάλλον που εκτελείται σε τερματικό. Στην παρούσα διπλωματική χρησιμοποιήθηκε εκτενώς για την εύρεση δύσκολων λογικών λαθών. Το εργαλείο αυτό έχει πολλές δυνατότητες μερικές από τις οποίες είναι η εκτέλεση του κώδικα κατά βήμα, η εισαγωγή σημείων διακοπής (breakpoints) και η δημιουργία προσαρμοσμένων μεταβλητών. Ίσως η πιο δυνατή λειτουργία, είναι η δυνατότητα διεπαφής με τον κώδικα του προγράμματος καθώς αυτό εκτελείται. Η τελευταία επιτυγχάνεται με το άνοιγμα ενός τερματικού Python ή στην περίπτωση μας Ipython. Μέσω του τερματικού, ο χρήστης έχει πρόσβαση σε οποιαδήποτε μεταβλητή και παράμετρο που βρίσκεται στο namespace. Η προβολή του namespace γίνεται με τη μεταβλητή dir(). Στο σχήμα 2.5 δίνεται ένα παράδειγμα της διεπαφής του PuDB.

Σχήμα 2.5: Διεπαφή PuDB.



2.7.5 Virtualenv

Υπήρξαν περιπτώσεις όπου χρειάστηκε πειραματισμός με διαφορετικές εκδόσεις διερμηνευτών και πρόσθετων πακέτων της Python. Για το σκοπό αυτό χρησιμοποιήθηκε το εργαλείο virtualenv [30]. Το πακέτο αυτό δίνει τη δυνατότητα δημιουργίας απομονωμένων Python περιβαλλόντων. Το κάθε περιβάλλον μπορεί να χρησιμοποιεί διαφορετικής έκδοσης διερμηνέα (interpreter) python και διαφορετικής έκδοσης πακέτα.

Τα περιβάλλοντα αυτά έχουν τη δυνατότητα να μεταφέρονται (–relocatable) από ένα υπολογιστή σε ένα άλλο αρκεί να έχει το ίδιο λειτουργικό σύστημα. Η δυνατότητα αυτή επιτρέπει να μεταφέρουμε το περιβάλλον ανάπτυξης αλλά και λειτουργίας της γεννήτριας όπου κρίνεται απαραίτητο. Επίσης δεν απαιτείται επανεγκατάσταση εργαλείων και βιβλιοθηκών.

2.7.6 Ipython

Η IPython είναι ένα διαδραστικό περιβάλλον του διερμηνέα της Python [28]. Παρέχει μία περιεκτική βιβλιοθήκη πάνω στην οποία μπορεί να δομηθούν πιο εξελιγμένα συστήματα. Επίσης δίνει τη δυνατότητα εικονοποίησης δεδομένων (εικόνες, γραφήματα, κλπ) και παράλληλης επεξεργασίας. Στην παρούσα διπλωματική χρησιμοποιήθηκε για προτυποποίηση και έλεγχο αποσπασμάτων κώδικα.

2.7.7 line profiler

Για τη χρονομέτρηση των συναρτήσεων της γεννήτριας, αλλά και για την εξαγωγή μετρικών εκτέλεσης κώδικα, χρησιμοποιήθηκε το line_profiler [18]. Πρόκειται για ένα εργαλείο γραμμένο σε Cython, που εμφανίζει τους χρόνους και τον αριθμό εκτελέσεων μίας συνάρτησης γραμμή προς γραμμή. Στην υλοποίηση, δόθηκε μεγαλύτερη έμφαση στη βελτίωση της χρονικής απόδοση του εργαλείου, παρά στην κατανάλωση μνήμης και επεξεργαστικής ισχύος.

2.7.8 Xilinx ISE

Για την επιβεβαίωση λειτουργίας των παραγόμενων κυκλωμάτων αλλά και για τον υπολογισμό μετρικών, όπως του κρίσιμου μονοπατιού, χρησιμοποιήθηκε η σου-

ίτα εργαλείων της Xilinx[1]. Πρόκειται για ένα περιβάλλον εργασίας και προτυποποίησης ενσωματωμένων συστημάτων και ανάπτυξης VHDL.

2.7.9 GHDL

Το GHDL [15] είναι ένας ανοιχτού κώδικα προσομοιωτής για τη γλώσσα VHDL. Στην παρούσα διπλωματική χρησιμοποιήθηκε για τον έλεγχο της ορθής λειτουργίας και την προσομοίωση των παραγόμενων από τη γεννήτρια κυκλωμάτων.

2.7.10 Git

Για την εύκολη διαχείριση του έργου, χρησιμοποιήθηκε το git [7]. Πρόκειται για ένα ανοιχτού κώδικα πρόγραμμα διαχείρισης εκδόσεων. Σχεδιάστηκε από τον Linus Torvalds για τη διαχείριση του πυρήνα Linux. Σήμερα αποτελεί το πιο διάσημο αποκεντρωμένο πρόγραμμα διαχείρισης εκδόσεων πάνω στο οποίο στηρίζονται υπηρεσίες όπως το github και gitlab.

2.8 Σύνοψη Κεφαλαίου

Στο κεφάλαιο αυτό έγινε αναφορά στο θεωρητικό υπόβαθρο των λέξεων κλειδιών και των εργαλείων που χρησιμοποιήθηκαν σε αυτή τη διπλωματική. Επίσης δόθηκε ο τομέας στον οποίο χρησιμοποιήθηκε το κάθε βοηθητικό πρόγραμμα. Στο κεφάλαιο που ακολουθεί, περιγράφεται με λεπτομέρεια ο διαχωρισμός του λογισμικού μέρους του εργαλείου και δίνεται η ανάλυση κάθε του μέρους.

[1] https://www.xilinx.com/products/design-tools/ise-design-suite.html

Κεφάλαιο 3

Η υλοποίηση του λογισμικού μέρους

Στο κεφάλαιο αυτό παρουσιάζεται ο διαχωρισμός του εργαλείου σε frontend (προ-επεξεργασίας) και backend (τελικής επεξεργασίας). Γίνεται μεγαλύτερη ανάλυση στο backend και δίνονται οι αλγόριθμοι πάνω στους οποίους στηρίζεται η παραγωγή κυκλωμάτων.

3.1 Γενική επισκόπηση της εφαρμογής

Όπως κάθε διαδικτυακή εφαρμογή, το λογισμικό απαρτίζεται από δύο διακριτά μέρη, το frontend και το backend όπως φαίνεται στο σχήμα 3.1. Η πρόσβαση είναι εφικτή από οποιαδήποτε συσκευή που υποστηρίζει πλοήγηση στο διαδίκτυο. Τα δύο μέρη (frontend και backend) ανταλλάσσουν πληροφορίες μεταξύ του σε μορφή JSON [8]. Για την παραγωγή μιας περιγραφής ενός κυκλώματος, ακολουθούνται τα βήματα που φαίνονται στο σχήμα 3.2.

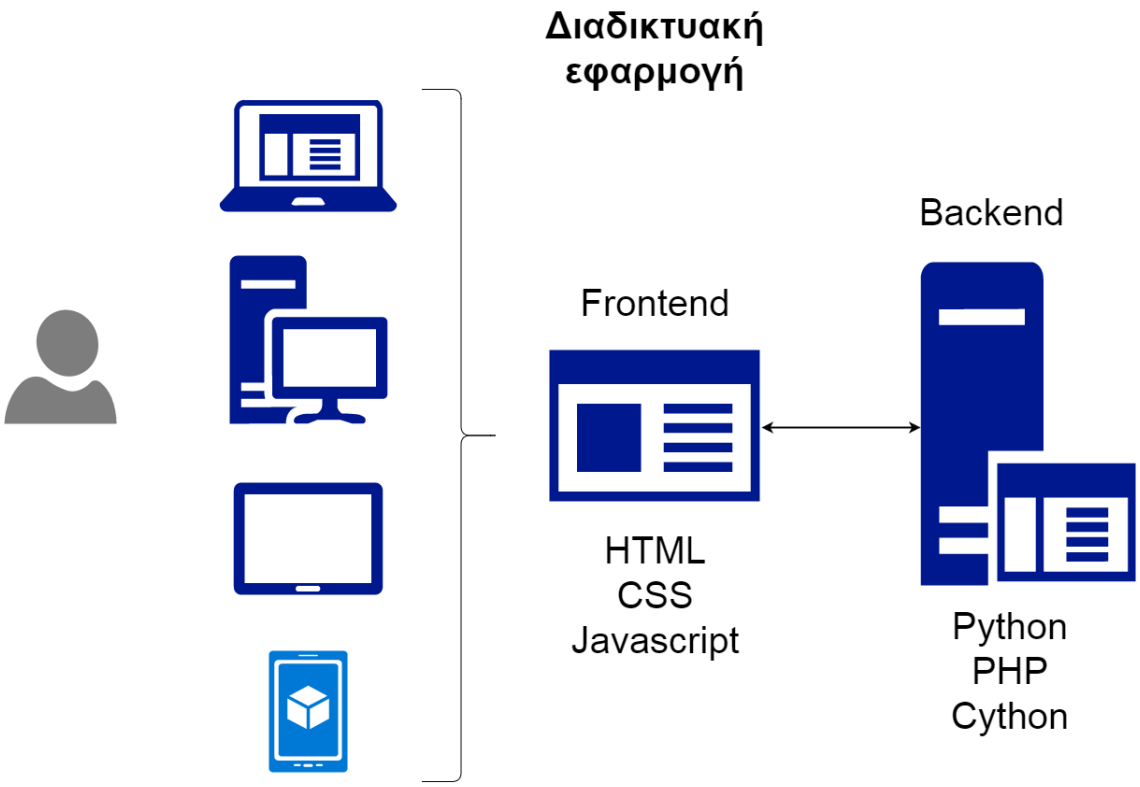
3.2 Ανάλυση του frontend

Για την ανάπτυξη του frontend, χρησιμοποιήθηκαν οι τεχνολογίες, HTML, CSS και Jacascript. Η είσοδος του χρήστη δίνεται μέσω μιας HTML φόρμας και με το πάτημα της υποβολής πραγματοποιείται επικύρωση των στοιχείων που εισήχθηκαν όπως φαίνεται στο σχήμα 3.3.

3.3 Αρχείο εισόδου

Η μονάδα σχεδίασης κυκλώματος είναι η μονάδα του backend που καλείται πρώτη. Ως είσοδο, δέχεται ένα αρχείο σε μορφή JSON που ονομάζεται input_definitions.

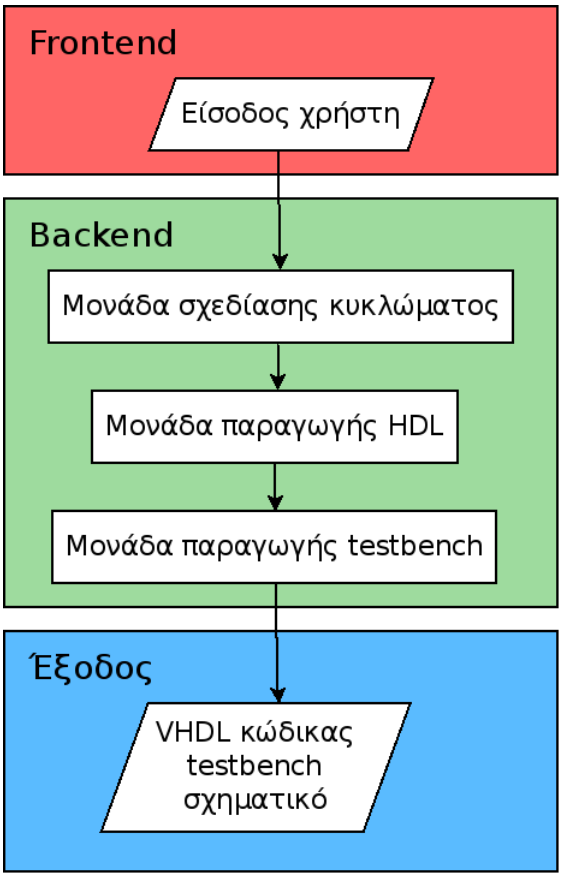
Σχήμα 3.1: Απεικόνιση της διαδικτυακής εφαρμογής.



Στο αρχείο αυτό δηλώνεται η συνάρτηση και τα εύρη bit των εισόδων (είσοδος, αποτελεί κάθε μεταβλητή). Ένα ενδεικτικό παράδειγμα τέτοιου αρχείου εισόδου δίνεται στην απεικόνιση 3.1. Στο παράδειγμα περιγράφεται μία συνάρτηση με 4 μεταβλητές και 2 πράξεις, πρόσθεση και πολλαπλασιασμός. Οι 4 γραμμές μετά τη συνάρτηση δηλώνουν τα εύρη bit της κάθε εισόδου, συγκεκριμένα οι μεταβλητές in0 και in1 θα είναι 2 bits, ενώ οι in2 και in3 θα είναι 5. Στο παράδειγμα μας ο πολλαπλασιασμός έπεται των δύο προσθέσεων και τα εύρη bit δεν συμπίπτουν. Σε αυτή την περίπτωση θα εισαχθούν ειδικά μηδενικά στοιχεία που θα προσαρμόσουν κατάλληλα τα εύρη.

Το αρχείο εισόδου φέρει δύο περιορισμούς σύνταξης τους οποίους πρέπει να τηρεί ο χρήστης. Ο πρώτος περιορισμός είναι ότι οι μεταβλητές πρέπει να φέρουν τη μορφή ονομασίας που δόθηκε στο παραπάνω παράδειγμα, δηλαδή ”inX”, όπου X είναι ακέραιος αριθμός. Ο δεύτερος περιορισμός είναι ότι οι μεταβλητές πρέπει να δηλώνονται κατά προτεραιότητα εισόδου. Όπως φαίνεται στην απεικόνιση 3.1, η μεταβλητή in0, αποτελεί την πρώτη είσοδο της συνάρτησης, για το λόγο αυτό δηλώνεται πρώτη μετά τη γραμμή της συνάρτησης. Ομοίως, η δεύτερη μεταβλητή

Σχήμα 3.2: Γενική ροή εργαλείου.



εισόδου είναι η in1 η οποία δηλώνεται δεύτερη μετά την in0, και ούτω καθεξής. Οι δύο αυτοί περιορισμοί αναλύονται λεπτομερώς στο παράρτημα Α΄.

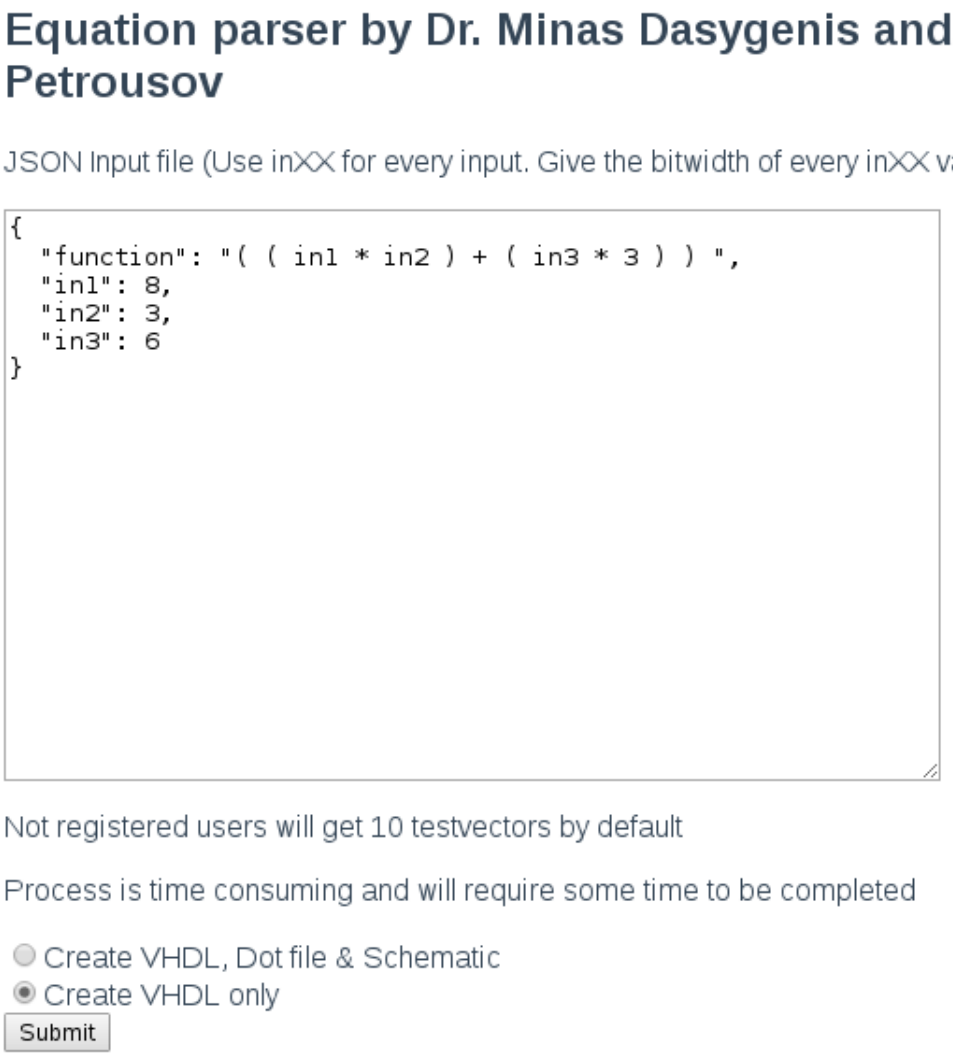
Απεικόνιση 3.1: Αρχείο εισόδου της $f(in0, in1, in2, in3) = ((in0 + in1) * (in2 + in3))$.

```
{
        "function": ( (in0 + in1 ) * ( in2 + in3 ) )",
        "in0": 2,
        "in1": 2,
        "in2": 5,
        "in3": 5
}
```

3.4 Η α-HDL netlist

Η σχεδίαση ενός αριθμητικού επιταχυντή στηρίζεται σε μία λίστα που ονομάζεται α-HDL (abstract HDL) [9]. Η λίστα αυτή διαμορφώνεται είτε από τρεις διαφορετικές δομές, ή συνενώνεται σε μία ενιαία δομή δεδομένων στη μορφή JSON.

Σχήμα 3.3: Frontend της ιστοσελίδας του εργαλείου.

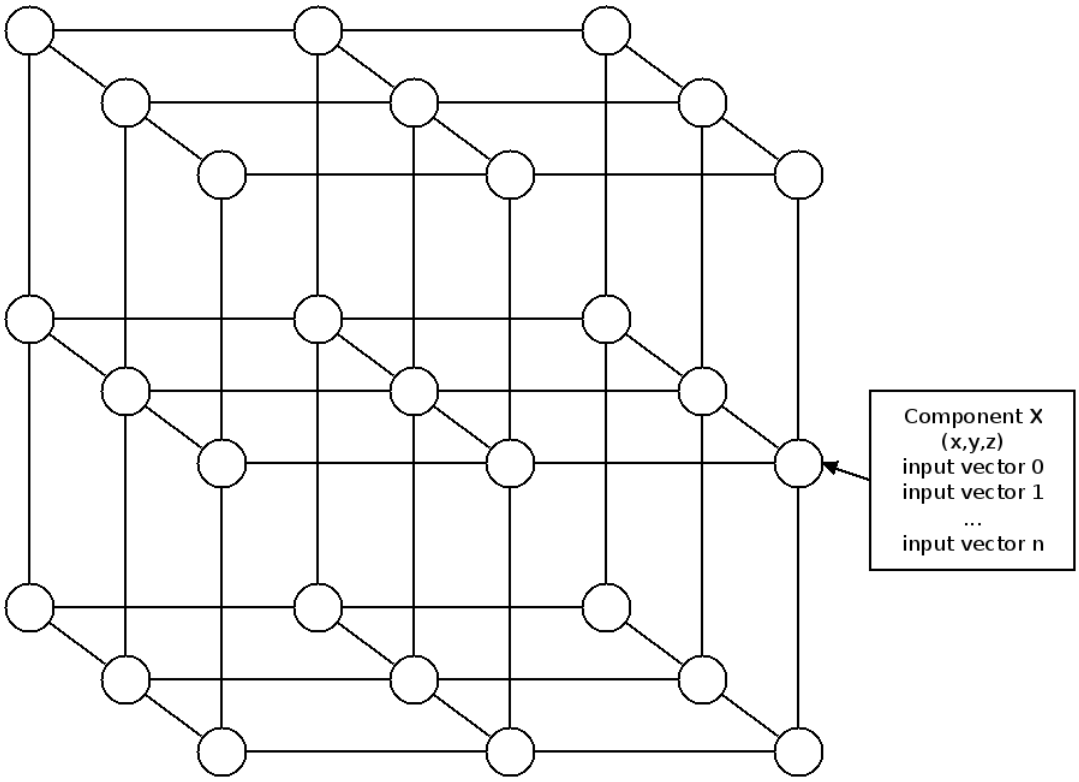


Η δημιουργία της λίστας γίνεται αυτόματα από τη γεννήτρια καθώς διαβάζει την είσοδο του χρήστη. Στο παράρτημα Β΄ δίνεται ένα πλήρες παράδειγμα της netlist που δημιουργείται για τη συνάρτηση 2.4.

Η πρώτη δομή δεδομένων που απαρτίζει την α-HDL, ονομάζεται components και μπορεί να απεικονιστεί ως υπερκύβος. Πρόκειται για έναν τρισδιάστατο πίνακα όπου σημειώνεται σε ποια συντεταγμένη (x,y,z) τοποθετείται κάποιο στοιχείο όπως φαίνεται στο σχήμα 3.4. Τα στοιχεία είναι κωδικοποιημένα με έναν μοναδικό αριθμό ο οποίος αντιστοιχεί σε όνομα στοιχείου βάση της επόμενης δομής. Ένα ολοκληρωμένο παράδειγμα της α-HDL αναλύεται στο παράρτημα Β΄.

Η δεύτερη δομή ονομάζεται componentlist. Αποτελεί έναν πίνακα αναζήτησης σε μορφή λεξιλογίου όπου κάθε αριθμός από τη δομή components αντιστοιχεί σε όνομα στοιχείου. Το όνομα μπορεί να είναι αντιπροσωπευτικό, όπως μία πύλη (AND,

Σχήμα 3.4: Ο υπερκύβος τη δομής components.



NAND) που υπάρχει στη βιβλιοθήκη, αλλά μπορεί και να είναι ειδική ονομασία μίας περίπλοκης συνδυαστικής λογικής, όπως ένας συγκριτής, που αποτελείται από άλλες απλούστερες δομές.

Η τρίτη δομή ονομάζεται interconnections. Πρόκειται για έναν τρισδιάστατο πίνακα όπου κάθε συντεταγμένη (x,y,z) περιέχει μία πληροφορία για τη συνδεσιμότητα του στοιχείου που βρίσκεται στην αντίστοιχη συντεταγμένη στον πίνακα components. Μία πληροφορία έχει δύο πιθανές μορφές ανάλογα με το στοιχείο που περιγράφει, όπως φαίνεται στην απεικόνιση 3.2. Η σημασία της κάθε μεταβλητής δίνεται στον πίνακα 3.1. Το στοιχείο της εισόδου και ο σταθερός αριθμός έχουν ειδική σημασία και δηλώνονται διαφορετικά από τα υπόλοιπα. Στην περίπτωση της εισόδου, οι μεταβλητές x,y,z έχουν τιμή -1, ενώ στην περίπτωση της σταθεράς έχουν τιμή -3.

Πίνακας 3.1: Επεξήγηση μεταβλητών πίνακα interconnections.

(x,y,z)	:	συντεταγμένες στοιχείου εισόδου
outputport	:	θύρα εξόδου προηγούμενου στοιχείου
(lsb,msb)	:	εύρος bit εισόδου
signaltype	:	τύπος σήματος (bit, stream)
output_port	:	αριθμός θύρας εξόδου

Απεικόνιση 3.2: Δομή interconnections

```
1. [x,y,z,outputport,lsb,msb,signaltype]
2. [x,y,z,outputport,lsb,msb,signaltype,output_port,lsb,msb]
```

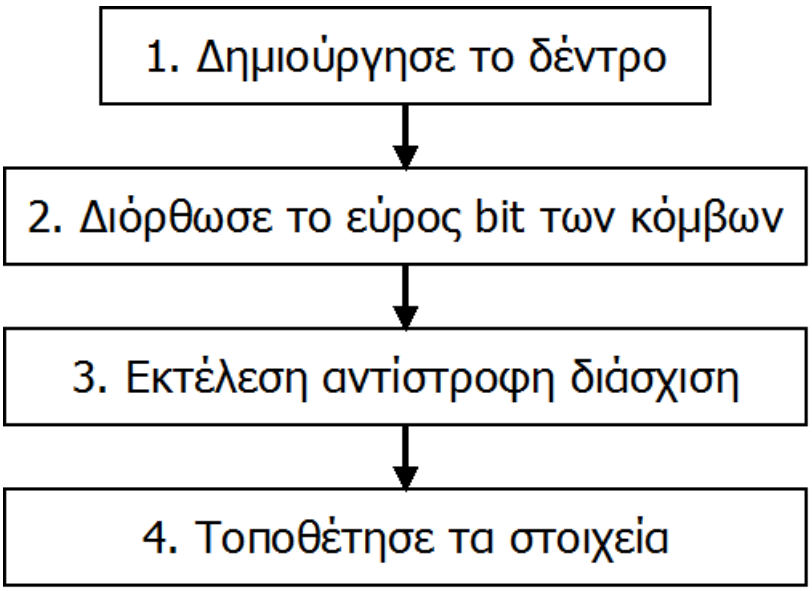
3.5 Ανάλυση του backend

Το μέρος τελικής επεξεργασίας περιέχει τις συναρτήσεις ανάλυσης και κατασκευής των επιταχυντών. Αποτελείται από τρεις κύριες μονάδες, τη μονάδα σχεδίασης κυκλώματος, τη μονάδα παραγωγής HDL και τη μονάδα παραγωγής testbench. Η σειρά κλήσης των μονάδων αυτών φαίνεται στο σχήμα 3.2. Η επικοινωνία μεταξύ τους, γίνεται μέσω της ειδικής netlist (α-HDL) που περιγράφτηκε παραπάνω. Δηλαδή, μία μονάδα καλεί την επόμενη περνώντας σαν παράμετρο την κατάλληλα διαμορφωμένη α-HDL. Το μεγαλύτερο μέρος του backend έχει γραφτεί σε Python με ορισμένες πολύ απαιτητικές συναρτήσεις να είναι γραμμένες σε Cython για λόγους ταχύτητας. Οι συναρτήσεις αυτές έχουν πολλούς ένθετους βρόγχους που ελέγχουν τη συνδεσιμότητα των στοιχείων του κυκλώματος που σχεδιάζεται για παρατυπίες.

3.5.1 Μονάδα σχεδίασης κυκλώματος

Μετά την επαλήθευση, η είσοδος του χρήστη περνιέται σαν παράμετρος στη μονάδα σχεδίασης. Στόχος της είναι να δημιουργήσει την α-HDL λίστα που περιγράφει τον επιταχυντή. Τα βήματα που ακολουθεί δίνονται στο σχήμα 3.5.

Σχήμα 3.5: Βήματα δημιουργίας α-HDL.



Δημιούργησε το δέντρο Στο πρώτο βήμα δημιουργείται το δέντρο έκφρασης που περιγράφει τη δοσμένη συνάρτηση του χρήστη. Η δημιουργία του δέντρου βασίζεται

σε μία κλάση που περιγράφει τους κόμβους και τις συνδέσεις που έχουν μεταξύ τους. Έτσι, δοσμένου του αντικειμένου της ρίζας του δέντρου μπορούμε να εκτελέσουμε διερεύνηση προς οποιοδήποτε άλλο κόμβο. Η δημιουργία του δέντρου γίνεται βάσει του αλγόριθμου 1 ο οποίος επιστρέφει τη ρίζα του δέντρου ως αντικείμενο.

Result: expression tree
create_empty_stack();
create_root_node();
current_node = *root_node*;
for *character in expression* **do**
  if *character* == *(* **then**
    `// case 1`
    insert_node_left();
    push_current_node_to_stack();
    current_node = *get_left_node*();
  else if *character not in [+, -, *, /,)]* **then**
    `// case 2`
    current_node_value = *character*;
    current_node = *stack.pop*();
  else if *character in [+, -, *, /]* **then**
    `// case 3`
    current_node_value = *character*;
    insert_node_right();
    push_current_node_to_stack();
    current_node = *get_right_node*();
  else if *character* == *)* **then**
    `// case 4`
    current_node = *stack.pop*();
end

Αλγόριθμος 1: Δημιουργία δέντρου έκφρασης.

Όπως βλέπουμε ο αλγόριθμος έχει ένα κύριο βρόχο επανάληψης όπου διαβάζεται ο επόμενος χαρακτήρας της συνάρτησης. Στη συνέχεια, χωρίζεται σε 4 περιπτώσεις. Στην πρώτη περίπτωση, αν ο χαρακτήρας είναι η αριστερή παρένθεση, τότε δημιουργείται το αριστερό παιδί στον κόμβο που βρισκόμαστε και γίνεται αυτός ο τωρινός κόμβος, ενώ ο προηγούμενος σπρώχνεται στη στοίβα. Στη δεύτερη περίπτωση, αν ο χαρακτήρας δεν είναι τελεστής ή δεξιά παρένθεση, τότε σημαίνει ότι ο χαρακτήρας είναι μεταβλητή ή σταθερά. Στην περίπτωση αυτή, αφού δώσουμε τιμή στον κόμβο αυτό, θέτουμε ως τωρινό κόμβο αυτόν που βρίσκεται στην κορυφή της στοίβας. Στην τρίτη περίπτωση, αν ο χαρακτήρας είναι τελεστής, θέτουμε την τιμή του τωρινού κόμβου, δημιουργούμε το δεξί παιδί, σπρώχνουμε τον τωρινό κόμβο στη στοίβα και

θέτουμε το δεξί παιδί ως τωρινό κόμβο. Στην τελευταία περίπτωση εξετάζεται αν ο χαρακτήρας είναι η αριστερή παρένθεση και θέτουμε ως τωρινό κόμβο αυτόν που βρίσκεται στην κορυφή της στοίβας.

Διόρθωσε το εύρος bit των κόμβων Αφού δημιουργηθεί το δέντρο, στο δεύτερο βήμα της μονάδας εκτελείται μία αντίστροφη διερεύνηση του δέντρου, δηλαδή από τα φύλλα προς τη ρίζα και υπολογίζονται και συμπληρώνονται τα εύρη bits των κόμβων. Ο υπολογισμός του εύρους γίνεται με το να δώσουμε το μεγαλύτερο δυνατό δεκαδικό αριθμό στις εισόδους και να εκτελέσουμε την πράξη του κόμβου τελεστή. Μετατρέποντας το δεκαδικό αποτέλεσμα της πράξης σε δυαδικό, υπολογίζουμε τον αριθμό των bits στην έξοδο. Η διαδικασία αυτή επαναλαμβάνεται αναδρομικά μέχρι τη ρίζα του δέντρου. Ο υπολογισμός αυτός δεν μπορεί να γίνει στο προηγούμενο βήμα, καθώς δεν είναι γνωστό το δέντρο εξαρχής.

Εκτέλεσε αντίστροφη διάσχιση Στο τρίτο βήμα εκτελείται η αντίστροφη διάσχιση του δέντρου και επιστρέφεται μίας στοίβα με τους κόμβους τελεστές κατά σειρά προτεραιότητας εκτέλεσης των πράξεων. Η δημιουργία της στοίβας βασίζεται στον αλγόριθμο 2.

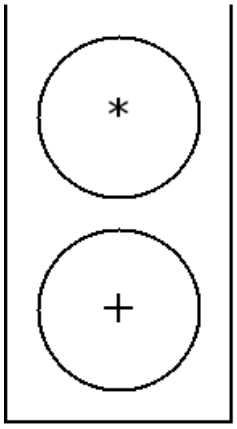
Result: stack
$theStack = create\_empty\_stack()$;
$theQueue = create\_empty\_queue()$;
$theQueue.enqueue(root\_node)$;
while *theQueue is not empty* **do**
  $root = theQueue.dequeue()$;
  if *root.RightChild == +,-,*,/,)* **then**
    `// case 1`
    $theQueue.enqueue(root.RightChild)$;
  if *root.LeftChild == +,-,*,/,)* **then**
    `// case 2`
    $theQueue.enqueue(root.LeftChild)$;
  theStack.push(root)
end

Αλγόριθμος 2: Επιστροφή στοίβας κόμβων δέντρου.

Η συνάρτηση δέχεται ως είσοδο τη ρίζα του δέντρου έκφρασης και επιστρέφει τα αντικείμενα των κόμβων που είναι τελεστές ξεκινώντας από τα φύλλα προς την κορυφή (ρίζα) σε μία στοίβα. Δηλαδή, δοσμένου του δέντρου έκφρασης του σχήματος 2.4 για τη συνάρτηση $f(x, y) = ((2 * x) + y)$, θα λάβουμε τη στοίβα που

φαίνεται στο σχήμα 3.6. Όπως βλέπουμε στην κορυφή της στοίβας βρίσκεται ο κόμβος του πολλαπλασιασμού και από κάτω ο κόμβος της πρόσθεσης όπως ορίζεται στη συνάρτηση. Η στοίβα με τους κόμβους των τελεστών τροφοδοτείται ως είσοδος στο επόμενο βήμα.

Σχήμα 3.6: Παράδειγμα στοίβας για τη συνάρτηση $f(x, y) = ((2 * x) + y)$.



Τοποθέτησε τα στοιχεία Στο τέταρτο βήμα γίνεται η τοποθέτηση των στοιχείων, δηλαδή η δημιουργία της λίστας α-HDL. Ο αλγόριθμος της ρουτίνας αυτής δίνεται στο 3. Η διαδικασία αυτή περιλαμβάνει πολλές συνθήκες που κάνουν αναγνώριση της πράξης που απαιτείται και χρησιμοποιούν το κατάλληλο στοιχείο που υπάρχει στη βιβλιοθήκη της γεννήτριας μας ώστε να ικανοποιούνται οι περιορισμοί που έχει δώσει ο χρήστης. Υπάρχουν βέβαια και στοιχεία που δεν έχουν υλοποιηθεί και δεν περιλαμβάνονται στη βιβλιοθήκη, τέτοιο παράδειγμα αποτελεί το στοιχείο που εκτελεί διαίρεση. Στην περίπτωση αυτή, η μονάδα σχεδίασης απλά τοποθετεί ψεύτικα στοιχεία που δηλώνουν την απουσία της ύπαρξης του κυκλώματος αυτού στη βιβλιοθήκη. Η προσθήκη νέων πράξεων αποτελεί εύκολη διαδικασία, αρκεί να προστεθεί μία κατάλληλη συνθήκη if με το σύμβολο της πράξης. Αξίζει να σημειωθεί ότι η πράξη μπορεί να αποτελεί μία σύνθετη συνάρτηση στην οποία έχουμε δώσει ένα αντιπροσωπευτικό όνομα και δεν χρειάζεται να είναι αυστηρά αριθμητικό σύμβολο. Για παράδειγμα, θα μπορούσαμε να έχουμε ένα στοιχείο που αυξάνει τον αριθμό εισόδου κατά 1 και να το έχουμε ονομάσει ”plusone”. Συνεπώς, αρκεί να προσθέσουμε τη συνθήκη if που θα αναγνωρίζει το ”plusone” και θα κάνει την τοποθέτηση. Ωστόσο πριν προχωρήσουμε στην προσθήκη κάποιου στοιχείου, απαιτείται να έχουμε δημιουργήσει τη γεννήτρια που παράγει το στοιχείο αυτό

και καλύπτει όλες τις περιπτώσεις εισόδων. Το κύκλωμα του νέου στοιχείου δηλαδή, πρέπει να μπορεί να περιγραφεί με έναν αλγόριθμο. Κατά την ενασχόληση μας έχουμε παράξει μία σειρά τέτοιων κυκλωμάτων και τα έχουμε προσθέσει στη βιβλιοθήκη [23, 24, 25]. Η υλοποίηση των κυκλωμάτων βιβλιοθήκης δεν αποτελεί μέρος της παρούσας εργασίας και δεν θα αναλυθεί περαιτέρω.

```
Result: α-HDL netlist
for sizeof(theStack) do
    node = theStack.pop();
    if node == + then
        create_input_wires();
        get_left_input();
        get_right_input();
        place_adder_component();
    else if node == - then
        create_input_wires();
        get_left_input();
        get_right_input();
        place_subtractor_component();
    else if node == * then
        create_input_wires();
        get_left_input();
        get_right_input();
        place_multiplier_component();
    else if node == /) then
        create_input_wires();
        get_left_input();
        get_right_input();
        place_divider_component();
end
```

Αλγόριθμος 3: Δημιουργία της α-HDL netlist.

3.5.2 Μονάδα παραγωγής HDL

Η λίστα α-HDL τροφοδοτείται ως είσοδος στη μονάδα παραγωγής HDL. Πρόκειται για μία γενικού σκοπού γεννήτρια που μπορεί εύκολα να προσαρμοστεί και σε άλλα εργαλεία. Αποτελείται από τα πέντε στάδια [24] που φαίνονται στο σχήμα 3.7.

Στο πρώτο στάδιο, η μονάδα εκτελεί επιβεβαίωση ορθότητας της α-HDL. Η διαδικασία αυτή περιλαμβάνει την επιβεβαίωση της συμβατότητας του πίνακα components με τον interconnections, επιβεβαίωση της ορθότητας των ονομάτων των

στοιχείων που χρησιμοποιήθηκαν και την επιβεβαίωση των συνδέσεων (interconnections). Ορισμένα παράδειγμα συνθηκών παραβίασης αποτελούν οι ασύνδετες θύρες, τα μη υπάρχοντα ονόματα στοιχείων στη βιβλιοθήκη και ονόματα στοιχείων που αρχίζουν με αριθμό.

Στο δεύτερο στάδιο η μονάδα πραγματοποιεί ανάλυση εισόδου και εξόδου. Εξετάζει τους πίνακες και καταγράφει τα διανύσματα εισόδου και εξόδου που δείχνουν σε κάθε θύρα. Οι θύρες συνοδεύονται από τρεις μεταβλητές που δηλώνουν τον αριθμό τους, το εύρος bit, τον τύπο σήματος και τον τύπο της θύρας. Στην έξοδο αυτού του βήματος παράγονται δύο δομές που καθορίζουν πλήρως τις θύρες εισόδου και εξόδου του επιταχυντή.

Στο τρίτο στάδιο πραγματοποιείται η διαδικασία της χαρτογράφησης των σημάτων στις θύρες (portmapping). Για κάθε στοιχείο εξετάζονται οι αντίστοιχοι πίνακες από τους οποίους προκύπτει η συνδεσιμότητα των θυρών τους.

Στο τέταρτο στάδιο πραγματοποιείται η παραγωγή κώδικα VHDL. Αρχικά παράγεται η βιβλιοθήκη με τα πρωτεύοντα στοιχεία που χρησιμοποιούνται στο κύκλωμα και ύστερα η αρχιτεκτονική που περιγράφει τη λειτουργία του επιταχυντή. Στην έξοδο λοιπόν, δημιουργούνται δύο αρχεία, μία βιβλιοθήκη και ένα αρχείο λειτουργίας.

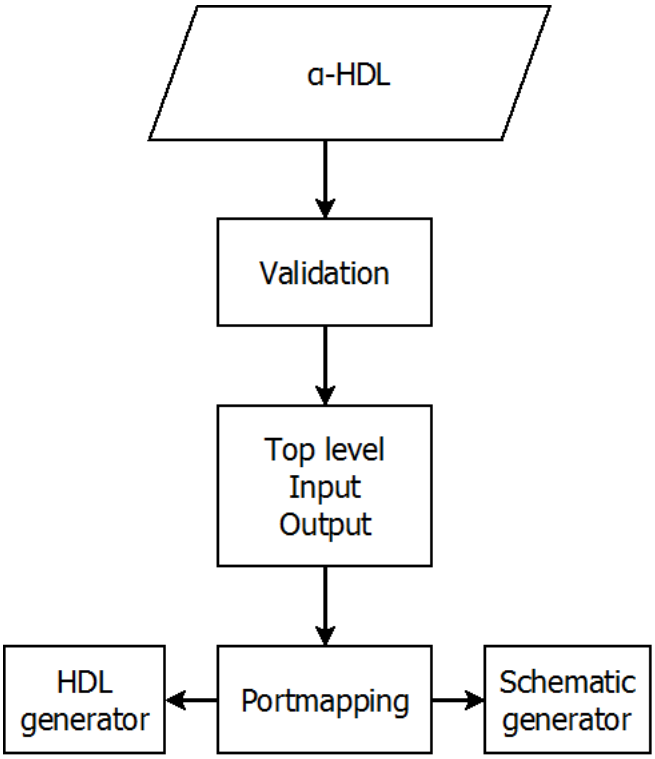
Στο πέμπτο στάδιο δημιουργείται προαιρετικά μία εικόνα του σχηματικού του κυκλώματος. Η εικόνα είναι σε μορφή PNG και συνοδεύεται από ένα αρχείο σε γλώσσα DOT. Το τελευταίο αρχείο προσφέρεται για υπολογιστή επεξεργασία.

3.5.3 Μονάδα παραγωγής testbench (κώδικας επιβεβαίωσης κυκλώματος)

Αυτή η μονάδα είναι ύψιστης σημαντικότητας καθώς παράγει το αρχείο (testbench) που επιβεβαιώνει την ορθότητα λειτουργίας του κυκλώματος του επιταχυντή. Η επιβεβαίωση λειτουργίας πραγματοποιείται μέσω της προσομοίωσης του κυκλώματος με την προσθήκη τυχαίων διανυσμάτων εισόδου και τη σύγκριση της εξόδου με το σωστά υπολογισμένο αποτέλεσμα. Ο αριθμός των τυχαίων εισόδων που θα παραχθούν καθορίζεται από τον χρήστη και το testbench παράγεται σε VHDL και είναι συνθέσιμο. Ο αλγόριθμος της μονάδας αυτής δίνεται στο 4.

Η μονάδα αυτή δέχεται το αλφαριθμητικό της συνάρτησης του χρήστη και τον αριθμό των τεστ που πρέπει να δημιουργηθούν. Ο πρώτος βρόχος μετράει των

Σχήμα 3.7: Βήματα της μονάδας HDL generator.



αριθμό των τυχαίων αριθμών, ενώ ο δεύτερος διαβάζει το αλφαριθμητικό της συνάρτησης χαρακτήρα προς χαρακτήρα. Στόχος του δεύτερου βρόχου είναι να βρίσκει κάθε φορά την πράξη που έχει προτεραιότητα, να υπολογίζει το αποτέλεσμα και να το σπρώχνει στη στοίβα. Όταν τελειώσει ο βρόχος, δηλαδή έχει διαβαστεί όλη η συνάρτηση, το αποτέλεσμα θα βρίσκεται στην κορυφή της στοίβας. Η δύο συνθήκες στον ένθετο βρόχο ερμηνεύονται ως εξής, αν ο χαρακτήρας δεν είναι η αριστερή παρένθεση, τότε εισάγεται στη στοίβα, αλλιώς αν ο χαρακτήρας είναι η αριστερή παρένθεση, τότε έχουμε βρει την κατά προτεραιότητα πράξη. Σε αυτή την περίπτωση οι χαρακτήρες εισόδου και ο τελεστής της πράξης βρίσκονται στη στοίβα και το μόνο που κάνουμε είναι απλά να τους βγάλουμε από την κορυφή. Πρώτα βγάζουμε την αριστερή είσοδο, μετά τον τελεστή και τέλος τη δεξιά είσοδο. Η τελευταία εντολή βγάζει την αριστερή παρένθεση από την κορυφή της στοίβας. Στη συνέχεια δημιουργούνται οι τυχαίοι πίνακες εισόδου και οδηγούνται στη σωστή θύρα του επιταχυντή. Μετά την ανάθεση τιμής στην είσοδο, τοποθετείται μία πρόταση αναμονής. Η διαδικασία αυτή επαναλαμβάνεται για όλες τις εισόδου. Ως είσοδος θεωρείται κάθε μεταβλητή στη προσαρμοσμένη συνάρτηση. Αφού δοθούν όλες οι είσοδοι, υπολογίζεται το σωστό αποτέλεσμα και δημιουργείται μία εντολή

(assert) που εκτελεί τη σύγκριση του σωστού αποτελέσματος (από τη στοίβα) και της εξόδου του κυκλώματος.

Αξίζει να σημειωθεί ότι η μονάδα αυτή όπως και η μονάδα σχεδίασης κυκλώματος είναι εύκολα επεκτάσιμη. Δηλαδή, υποστηρίζει την εισαγωγή νέων αριθμητικών πράξεων αλλά και προσαρμοσμένων συναρτήσεων. Εδώ, δεν απαιτείται η δημιουργία της αντίστοιχης γεννήτριας του στοιχείου, αλλά μόνο η δημιουργία τυχαίων αριθμών εισόδου και ο υπολογισμός του σωστού αποτελέσματος.

3.6 Παρουσίαση μετρικών κώδικα

Το έργο της παρούσας διπλωματικής, αποτελεί μία επέκταση της κύριας λειτουργίας του πυρήνα που αναπτύχθηκε τα προηγούμενα χρόνια. Συνολικά κατά την ενασχόληση μας, έχουν αναπτυχθεί πολλές γεννήτριες και βοηθητικές βιβλιοθήκες που δεν παρουσιάστηκαν εδώ, αλλά αποτέλεσαν μέρος άλλων ερευνητικών έργων. Με το πέρασμα του χρόνου, αυτές οι συναρτήσεις και βιβλιοθήκες διαφοροποιήθηκαν ανάλογα, για να υποστηρίξουν τις νέες λειτουργίες και επεκτάσεις που εισήχθηκαν. Συνεπώς, ο ακριβής υπολογισμός των γραμμών κώδικα που εισήχθηκαν στην παρούσα διπλωματική δεν μπορεί να υπολογιστεί με ακρίβεια, αλλά μπορεί μόνο να παρατηρηθεί σαν αλλαγή στο αποθετήριο μας.

Παρ' όλα αυτά, στον πίνακα 3.2 δίνονται ορισμένες μετρικές που βγήκαν χρησιμοποιώντας το εργαλείο cloc[1]. Η ακριβής εντολή που χρησιμοποιήθηκε για την εξαγωγή του πίνακα, δίνεται στην απεικόνιση 3.3. Πρόκειται για μία εντολή φλοιού στην οποία το αποτέλεσμα της εντολής ls-files του git, δίνεται σαν είσοδος στο cloc. Η εντολή του git, επιστρέφει μία λίστα με όλα τα αρχεία με αρχικό κατάλογο αυτόν στον οποίο βρισκόμαστε. Επομένως, είναι σημαντικό να βρισκόμαστε στον αρχικό κατάλογο (στον ίδιο κατάλογο που βρίσκεται και ο κατάλογος .git) του αποθετηρίου. Πρέπει να σημειώσουμε ότι ο κατάλογος vhdl_generated παραλήφθηκε (grep -v vhdl_generated) από τις μετρήσεις καθώς περιέχει κώδικα Python ο οποίος δημιουργήθηκε αυτόματα από ειδικές γεννήτριες που δημιουργούν κυκλωματικά στοιχεία βιβλιοθήκης. Αναφορικά, πρόκειται για 393399 γραμμές κώδικα Python που δεν αποτελούν μέρος των μετρήσεων.

[1] `https://github.com/AlDanial/cloc`

```
Result: αρχείο testbench
theStack = initialize();
for number_of_tests do
    for character in function do
        if character != ) then
            theStack.append(character);
        else if character == ) then
            operand0 = theStack.pop();
            operator = theStack.pop();
            operand1 = theStack.pop();
            theStack.pop();
            if operator == + then
                create_random_number;
                convert_random_to_binary;
                find_input_port_number();
                create_signal_assignment();
            else if character == - then
                create_random_number;
                convert_random_to_binary;
                find_input_port_number();
                create_signal_assignment();
            else if character == * then
                create_random_number;
                convert_random_to_binary;
                find_input_port_number();
                create_signal_assignment();
            else if character == / then
                create_random_number;
                convert_random_to_binary;
                find_input_port_number();
                create_signal_assignment();
            calculate_correct_result();
            theStack.append(result);
    end
    insert_wait_clause();
    create_signal_assert();
end
```

Αλγόριθμος 4: Δημιουργία testbench.

Απεικόνιση 3.3: Εντολή εξαγωγής μετρικών αρχείων κώδικα.

```
$ cloc $(git ls-files | grep -v vhdl_generated)
```

Πίνακας 3.2: Μετρικές αρχείων κώδικα του αποθετηρίου.

Γλώσσα προγραμματισμού	**Αριθμός αρχείων**	**Κενές γραμμές**	**Γραμμές σχολίων**	**Γραμμές κώδικα**
Python	59	6414	8300	14980
C	3	295	766	5850
Bourne Shell	13	195	154	559
Cython	2	44	46	119
Σύνολο	77	6948	9266	21508

Το σύνολο των αρχείων στο αποθετήριο ανέρχεται στα 107 και το μέγεθος τους είναι 9.4MB. Στο σύνολο αυτό περιλαμβάνονται οι βιβλιοθήκες, οι γεννήτριες, ο πυρήνας αλλά και διάφορα βοηθητικά προγράμματα, όπως συναρτήσεις ελέγχου ορθότητας, profilers, αυτόματοι χτίστες κυκλωμάτων και άλλα, που έχουν αναπτυχθεί κατά τον καιρό.

Αξίζει να παρατηρήσουμε ότι υπάρχουν 8300 γραμμές σχολίων στα αρχεία με κώδικα Python. Τα σχόλια παίζουν καθοριστικό ρόλο στην κατανόηση μίας συνάρτησης από έναν προγραμματιστή χωρίς να χρειάζεται να διαβάσει και να αναλύσει τον κώδικα της. Στον κώδικα μας προσπαθούμε να γράφουμε σχόλια σύμφωνα με ένα πρότυπο περιγραφής μαύρου κουτιού. Σύμφωνα με το πρότυπο, ένα σχόλιο αποτελείται από δύο μέρη. Στο πρώτο μέρος περιγράφεται σε μερικές προτάσεις η λειτουργία της συνάρτησης. Οι προτάσεις πρέπει να είναι αρκετές ώστε να δώσουν τη γενική λειτουργία χωρίς να αναλύονται σε βάθος και λεπτομέρειες. Το δεύτερο μέρος περιγράφει τις εισόδους και τις εξόδους, καθώς και τον τύπο που θα έχουν. Ένα παράδειγμα σχολίου από τη γεννήτρια μας που συμβαδίζει με το πρότυπο, δίνεται στην απεικόνιση 3.4. Διαβάζοντας μόνο την περιγραφή, μπορούμε να καταλάβουμε ότι πρόκειται για ένα πρόγραμμα που φορτώνει δυναμικά βιβλιοθήκες Python. Δέχεται την πλήρη διαδρομή προς το αρχείο και το όνομα της συνάρτησης που θέλουμε να φορτώσουμε και επιστρέφει τον χειριστή της συνάρτησης.

Αξίζει επίσης να παρατηρήσουμε ότι έχουμε μόλις 119 γραμμές κώδικα Cython μοιρασμένες σε 2 αρχεία. Αντίθετα τα αρχεία που είναι γραμμένα σε C αποτελούνται από 5850 γραμμές κώδικα. Αυτό συμβαίνει διότι η Cython εφαρμόζει ορισμένες

τεχνικές βελτιστοποίησης κώδικα κατά τη μετάφραση ενός αρχείου, έτσι μία απλή συνάρτηση 20 γραμμών Python μπορεί να μεταφραστεί σε εκατοντάδες γραμμές κώδικα C. Οι βελτιώσεις αυτές είναι πολύ σημαντικές και θα έπαιρναν πάρα πολύ χρόνο αν γραφόταν με το χέρι. Φυσικά, αυτές οι παράμετροι βελτίωσης καθορίζονται στο εκτάστοτε αρχείο εγκατάστασης (setup.py) που γράφεται για κάθε αρχείο που χρειάζεται μετάφραση.

Για την υλοποίηση της νέας λειτουργίας που παρουσιάστηκε σε αυτή τη διπλωματική, δημιουργήθηκε ένας ξεχωριστός κατάλογος στον οποίο προστέθηκαν τα νέα αρχεία πηγαίου κώδικα. Το μέγεθος του καταλόγου είναι 717KB και ο αριθμός των αρχείων είναι 5, όπως φαίνεται στον πίνακα 3.3. Τα αρχεία binaryTree.py, queue.py και stack.py, περιέχουν τις κλάσεις του δυαδικού δέντρου, της ουράς και της στοίβας αντίστοιχα. Οι δομές αυτές έχουν παρθεί από το πακέτο pythonds[2] (Data Structures) και έχουν υποστεί κατάλληλες τροποποιήσεις για να ανταποκρίνονται στις ανάγκες του προγράμματος μας. Το αρχείο equation_parser.py αποτελεί τη μονάδα σχεδίασης του κυκλώματος και το input_definitions.json είναι το αρχείο εισόδου.

Απεικόνιση 3.4: Παράδειγμα περιγραφής μαύρου κουτιού.

```
"""
This function will try to dynamically import the requested
   module item from the module's path.
If the module item does not exist, it will call the function
   that creates it and import it after
it has been created.

INPUT:
    module_path [str] : full path of the module you want to
       import [a pathname with __init__.py inside]
    module_item  [str] : a *.py file inside the module_path

OUTPUT:
    returns the handler for the requested module item.
    You can call the module_item.function()
"""
```

[2] https://github.com/bnmnetp/pythonds

Πίνακας 3.3: Αρχεία που προστέθηκαν για την υλοποίηση της γεννήτριας.

binaryTree.py
equation_parser.py
input_definitions.json
queue.py
stack.py

3.7 Σύνοψη κεφαλαίου

Στο κεφάλαιο αυτό περιγράφτηκαν τα μέρη που συγκροτούν το εργαλείο και η λειτουργία του καθενός από αυτά. Η περιγραφή συμπληρώθηκε με την αναπαράσταση των αλγορίθμων, σε ψευδοκώδικα, όπου κρίθηκε απαραίτητο. Δόθηκαν επίσης ορισμένες μετρικές κώδικα όπως αριθμός γραμμών και μέγεθος αρχείων. Η παραγωγή σωστών και αλάνθαστων περιγραφών κυκλωμάτων είναι μία περίπλοκη διαδικασία που απαιτεί πολλούς ελέγχους ορθότητας. Στο επόμενο κεφάλαιο δημιουργήσαμε έναν αριθμό περιγραφών κυκλωμάτων ώστε να βγάλουμε ορισμένα μετρικά αποτελέσματα για τα παραγόμενα κυκλώματα.

Κεφάλαιο 4

Πειραματικές μετρήσεις

Στο κεφάλαιο αυτό παρουσιάζεται με λεπτομέρεια η δημιουργία δύο αριθμητικών κυκλωμάτων και η εξαγωγή μετρικών αποτελεσμάτων από αυτές προκειμένου να αξιολογηθεί η αποτελεσματικότητα του εργαλείου. Ο έλεγχος και η σύνθεση του πηγαίου κώδικα, πραγματοποιήθηκαν με το Leonardo Spectrum, Xilinx Vivado 2013 για την οικογένεια Virtex6 (xc6vlx240t, speed -1) FPGA. Ο υπολογισμός την ενέργειας έγινε με το XPower Analyzer, ενώ ο αριθμός των τρανζίστορ είναι υπολογισμός του εργαλείου μας.

4.1 Παράδειγμα συνάρτησης δύο εισόδων

Στο παράδειγμα αυτό παρουσιάζεται η συνάρτηση $f(x, y) = ((2 * x) + y)$ από το κεφάλαιο 2, της οποίας το δέντρο έκφρασης δόθηκε στο σχήμα 2.4. Αρχικά, το εύρος bit και των δύο μεταβλητών x και y, επιλέχθηκε να είναι 2 για λόγους απλότητας. Συνεπώς, έχοντας αυτές τις πληροφορίες, έχουμε το αρχείο εισόδου (input_definitions) που δίνεται στην απεικόνιση 4.1.

Απεικόνιση 4.1: Αρχείο εισόδου συνάρτησης $f(x, y) = ((2 * x) + y)$.

```
{
    "function": "( ( 2 * x ) + y )",
    "x": 2,
    "y": 2
}
```

Η συνάρτηση από δύο αριθμητικούς τελεστές που σημαίνει ότι χρησιμοποιούνται δύο διαφορετικές γεννήτριες για την υλοποίηση του κυκλώματος. Συγκεκρι-

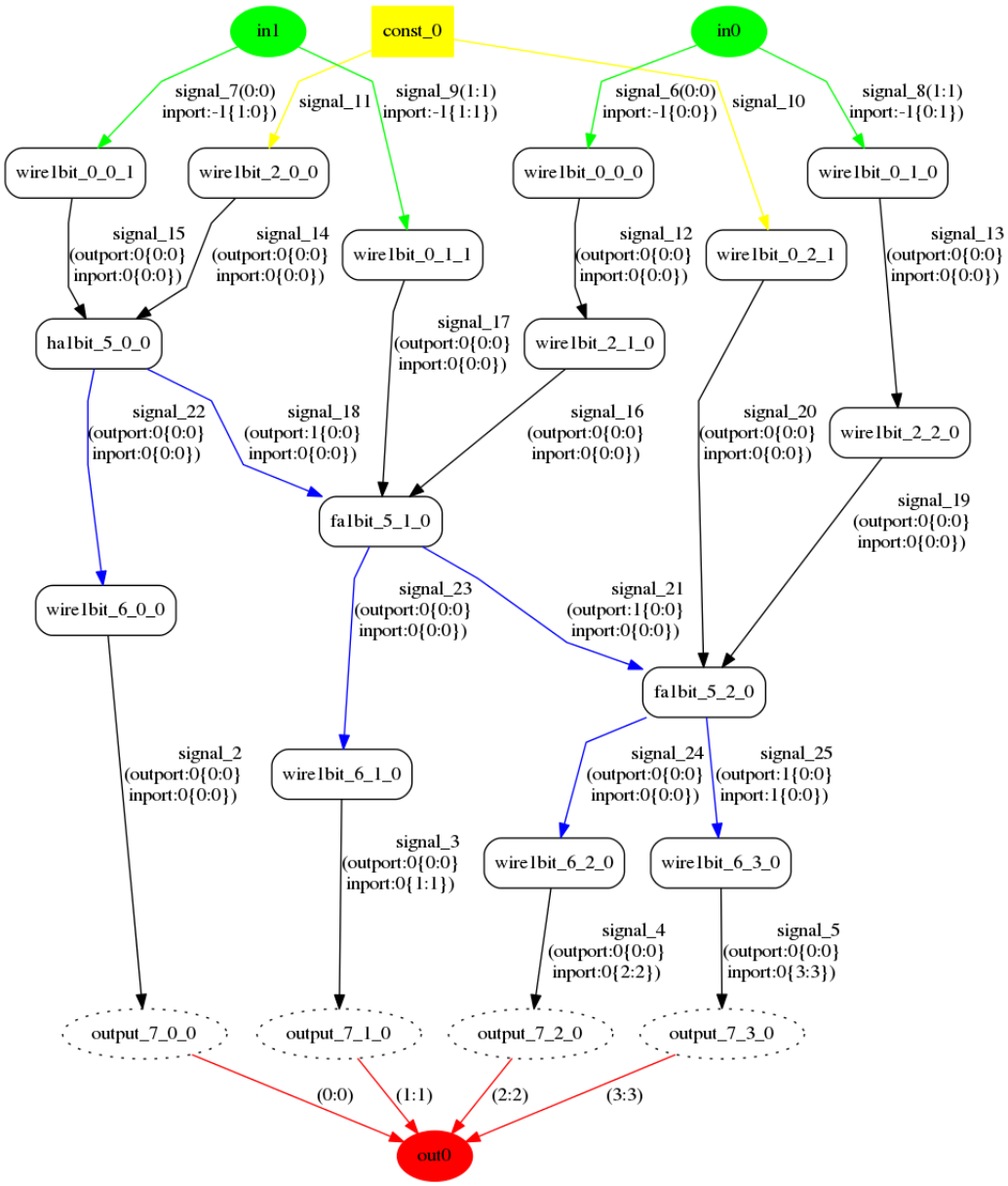
μένα, η πράξη $(2 * x)$ χρησιμοποιεί τη γεννήτρια του σταθερού πολλαπλασιασμού (constant_multiplication). Η γεννήτρια αυτή παράγει κυκλώματα που υλοποιούν αποκλειστικά πράξεις πολλαπλασιασμού μίας μεταβλητής με μία σταθερά. Αν συμβολίσουμε το αποτέλεσμα της πρώτης πράξης με μία μεταβλητή z, τότε η δεύτερη πράξη είναι η $z + y$. Η πράξη αυτή περιγράφει την πρόσθεση δύο μεταβλητών. Για την παραγωγή αυτού του κυκλώματος, χρησιμοποιείται η γεννήτρια του κανονικού αθροιστή (standard_adder). Η γεννήτρια αυτή παράγει αποκλειστικά κυκλώματα πρόσθεσης δύο μεταβλητών.

Παρά το γεγονός ότι το εύρος bit και των δύο μεταβλητών είναι ίδιο, η πρώτη πράξη (z), έχει στην έξοδο 3 bits, διότι ο μέγιστος αριθμός εισόδου είναι το 3, συνεπώς, $3 * 2 = 6_{10} = 110_2$. Προκειμένου να συνδεθεί το αποτέλεσμα της πράξης αυτής με την είσοδο y που είναι 2 bits, η γεννήτρια δημιουργεί κύκλωμα αθροιστή με εισόδους τριών bits και προσθέτει τη σταθερά 0 (μηδέν) στο ποιο σημαντικό ψηφίο (Most Significant Bit-MSB) του μεγαλύτερου αριθμού. Έτσι δεν εμφανίζονται μηνύματα ασύνδετων καλωδίων από το GHDL. Η διαδικασία αυτή επεκτείνεται και για αριθμούς με μεγαλύτερες διαφορές.

Τρέχοντας τη γεννήτρια με το παραπάνω αρχείο εισόδου, παράγεται το σχηματικό του κυκλώματος που φαίνεται στο σχήμα 4.1. Στο σχηματικό αυτό φαίνονται ποια στοιχεία της βιβλιοθήκης χρησιμοποιήθηκαν, η τοποθέτηση τους στο χώρο και οι συνδέσεις μεταξύ τους. Για παράδειγμα, ο συμβολισμός $ha1bit\_5\_0\_0$, σημαίνει ότι στη θέση (5,0,0) τοποθετήθηκε το στοιχείο $ha1bit$ που με τη σειρά του συμβολίζει τον ημιαθροιστή (half adder) ενός bit. Επίσης, οι ετικέτες πάνω στις ακμές, φέρουν το όνομα του σήματος, το εύρος bit, καθώς και τη θύρα εισόδου ενός στοιχείου και τη θύρα εισόδου στο επόμενο. Τα ονόματα αυτά είναι αντιπροσωπευτικά και μπορούν να εντοπιστούν στα αντίστοιχα αρχεία VHDL του κυκλώματος.

Μαζί με το σχηματικό, δημιουργήθηκε και το testbench ένα μέρος του οποίου δίνεται στην απεικόνιση 4.2. Στο τελευταίο, φαίνεται ότι δίνονται οι δοκιμαστικές τιμές 1 και 2 στις εισόδους x και y και αναμένεται η έξοδος 4. Χρησιμοποιώντας το εργαλείο GHDL και εκτελώντας την ανάλυση και προσομοίωση του κώδικα, λαμβάνουμε τα μηνύματα "TESTBENCH3–N OK" για την αντίστοιχη περίπτωση εισόδου όπως φαίνεται στο σχήμα 4.2. Για μεγαλύτερη λεπτομέρεια και παρατήρηση των κυματομορφών των σημάτων, μπορεί να χρησιμοποιηθεί το εργαλείο Modelsim.

Σχήμα 4.1: Σχηματικό συνάρτησης $f(x, y) = ((2 * x) + y)$.



Απεικόνιση 4.2: Αρχείο δοκιμαστικών εισόδων συνάρτησης $f(x, y) = ((2 * x) + y)$.

```
signal0<="01" AFTER  8 ns ;-- input vector: 1
wait for waittime * 1 ns ;
signal1<="10" AFTER  8 ns ;-- input vector: 2
wait for waittime * 1 ns ;
-- output: 4
assert (vec2int(signal2) = 4  ) report "TESTBENCH3-N Output:"&
   integer'image(vec2int(signal2))&" Expected:"&integer'image(4
   ) severity error ;
assert (vec2int(signal2) /= 4  ) report "TESTBENCH3-N OK"
   severity note ;
```

Σχήμα 4.2: Πλήρες παράδειγμα ελέγχου και προσομοίωσης κυκλώματος που παράχθηκε.

```
ictest00343@pleiades:~/aHDL/equation_parser$ ghdl -a func2.vhd
ictest00343@pleiades:~/aHDL/equation_parser$ ghdl -a library.func2.vhd
ictest00343@pleiades:~/aHDL/equation_parser$ ghdl -a func2_tb.vhd
ictest00343@pleiades:~/aHDL/equation_parser$ ghdl -e func2_tb
ictest00343@pleiades:~/aHDL/equation_parser$ ghdl -r func2_tb
func2_tb.vhd:112:1:@0ms:(report note): Activating Clock
func2_tb.vhd:125:8:@0ms:(report note): Testbench Activated
func2_tb.vhd:141:1:@318ns:(assertion note): TESTBENCH3-N OK
func2_tb.vhd:150:1:@636ns:(assertion note): TESTBENCH3-N OK
func2_tb.vhd:159:1:@954ns:(assertion note): TESTBENCH3-N OK
func2_tb.vhd:168:1:@1272ns:(assertion note): TESTBENCH3-N OK
func2_tb.vhd:177:1:@1590ns:(assertion note): TESTBENCH3-N OK
func2_tb.vhd:186:1:@1908ns:(assertion note): TESTBENCH3-N OK
func2_tb.vhd:195:1:@2226ns:(assertion note): TESTBENCH3-N OK
func2_tb.vhd:204:1:@2544ns:(assertion note): TESTBENCH3-N OK
func2_tb.vhd:213:1:@2862ns:(assertion note): TESTBENCH3-N OK
func2_tb.vhd:222:1:@3180ns:(assertion note): TESTBENCH3-N OK
ictest00343@pleiades:~/aHDL/equation_parser$
```

Τα πειραματικά αποτελέσματα συνοψίζονται στον πίνακα 4.1. Οι πρώτες δύο στήλες δίνουν το εύρος bit των δύο μεταβλητών x και y, ενώ η στήλη 3 υπολογίστηκε από το εργαλείο μας και δίνει τον αριθμό των τρανζίστορ του κυκλώματος. Οι υπόλοιπες στήλες υπολογίστηκαν από τα αντίστοιχα εργαλεία της Xilinx, συγκεκριμένα η τέταρτη στήλη δίνει τον αριθμό τομών (slices) που χρησιμοποιήθηκαν, η στήλη πέντε έχει υπολογισμένο το κρίσιμο μονοπάτι (critical path) σε nanoseconds, η έκτη στήλη υπολογίζει τη μέγιστη συχνότητα λειτουργίας του κυκλώματος και τέλος η κατανάλωση ενέργειας δίνεται στην τελευταία στήλη.

Πίνακας 4.1: Πειραματικές μετρήσεις συνάρτησης $f(x, y) = ((2 * x) + y)$.

in0 bits	in1 bits	Αριθμός τρανζίστορ	Αριθμός τομών	Κρίσιμο μονοπάτι (ns)	Συχνότητα (MHz)	Ενέργεια (W)
2	2	70	1	1.060	943,39	3.422
4	4	126	3	1.690	591.71	3.422
8	8	238	5	2.409	415.11	3.422
16	16	462	9	4.701	212.72	3.422
24	24	686	14	6.993	143	3.422
12	22	602	13	5.218	191.644	3.422

4.2 Παράδειγμα συνάρτησης τριών εισόδων

Στο παράδειγμα αυτό παρουσιάζεται μία πιο σύνθετη συνάρτηση το αρχείο εισόδου της οποίας δίνεται στην απεικόνιση 4.3. Η συγκεκριμένη συνάρτηση χρησιμοποιεί όλες τις διαθέσιμες αριθμητικές γεννήτριες που έχουμε δημιουργήσει. Συγκεκριμένα, η πράξη $z = (in0 * in1)$ χρησιμοποιεί τη γεννήτρια κανονικού πολλαπλασιαστή, η $x = (in2 * 3)$ τη γεννήτρια σταθερού πολλαπλασιαστή και η $(z + x)$ τη γεννήτρια κανονικού αθροιστή. Τα εύρη bit των μεταβλητών επιλέχθηκαν τυχαία. Όπως και πριν, το πρόγραμμα έκανε προσαρμογές εισόδων όπου κρίθηκε απαραίτητο και πρόσθεσε τη σταθερά μηδέν. Το σχηματικό του κυκλώματος που δημιουργείται είναι πολύ μεγάλο, για το λόγο αυτό δεν θα παρουσιαστεί εδώ.

Απεικόνιση 4.3: Αρχείο εισόδου της $f(in0, in1, in2) = ((in0 * in1) + (in2 * 3))$.

```
{
    "function": "( ( in0 * in1 ) + ( in2 * 3 ) ) ",
    "in0": 2,
    "in1": 2,
    "in2": 2
}
```

Τα πειραματικά αποτελέσματα της συνάρτησης συνοψίζονται στον πίνακα 4.2. Η συνάρτηση αυτή χρησιμοποιεί περισσότερα στοιχεία κυκλωμάτων από την προηγούμενη, για το λόγο αυτό η συχνότητα λειτουργίας της είναι πολύ μικρότερη σε σχέση με την προηγούμενη. Επίσης ο γεννήτρια που υλοποιεί την πρόσθεση, χρησιμοποιεί αθροιστή κυματικού κρατουμένου (Ripple Carry Adder) ο οποίος υπολογίζει το αποτέλεσμα σειριακά και όχι παράλληλα.

Πίνακας 4.2: Πειραματικές μετρήσεις της $f(in0, in1, in2) = ((in0 * in1) + (in2 * 3))$.

in0 bits	**in1 bits**	**in2 bits**	**Αριθμός τρανζίστορ**	**Αριθμός τομών**	**Κρίσιμο μονοπάτι (ns)**	**Συχνότητα (MHz)**
2	2	2	178	3	2.788	358.68
8	8	8	2470	31	8.121	123.13
16	16	16	9334	192	15.935	62.75
16	12	14	7046	147	14.629	68.35

4.3 Πειραματικές μετρήσεις γεννήτριας

Οι χρόνοι που απαιτήθηκαν από τη γεννήτρια για να παράγει τις περιπτώσεις του πίνακα 4.1 για τη συνάρτηση $f(x, y) = ((2 * x) + y)$, δίνονται στον πίνακα 4.3. Παρατηρούμε ότι η παραγωγή σχηματικού είναι πολύ απαιτητική διαδικασία και απαιτεί όλο και αυξανόμενο χρόνο καθώς αυξάνεται το εύρος bit των εισόδων και χρησιμοποιούνται όλο και περισσότερα στοιχεία.

Πίνακας 4.3: Χρόνοι παραγωγής πρώτης συνάρτησης.

f=((2*x(+y))		Χρόνος (seconds) Σχηματικό	
in0 bits	**in1 bits**	**Ναι**	**Όχι**
2	2	0.632	0.212
4	4	0.904	0.313
8	8	1.651	0.212
16	16	3.440	0.252
24	24	7.767	0.293
12	22	5.764	0.280

Αντίστοιχα, για τη δεύτερη συνάρτηση που αναλύθηκε πιο πάνω, τα αποτελέσματα παρουσιάζονται στον πίνακα 4.4. Όπως και πριν, παρατηρούμε μεγάλη διαφορά στη δημιουργία μιας περιγραφής με σχηματικό και χωρίς. Αξίζει να σημειώσουμε ότι οι εικόνες που παράγονται για περιγραφές με πολλά στοιχεία, γίνονται πολύ μεγάλες για ανάλυση. Για παράδειγμα, η εικόνα του σχηματικού της συνάρτησης $f = ((in0 * in1) + (in2 * 3))$ για την τελευταία περίπτωση του πίνακα 4.4, είναι 32767x3365 pixels.

Πίνακας 4.4: Χρόνοι παραγωγής δεύτερης συνάρτησης.

f=((in0*in1)+(in2*3))			Χρόνος (seconds)	
			Σχηματικό	
in0 bits	**in1 bits**	**in2 bits**	**Ναι**	**Όχι**
2	2	2	1.061	0.218
8	8	8	12.404	0.696
16	16	16	50.132	4.695
16	12	14	33.974	2.747

4.4 Σύνοψη κεφαλαίου

Στο κεφάλαιο αυτό παρουσιάστηκαν ορισμένα πειραματικά αποτελέσματα που αφορούν τα κυκλώματα που παράγονται από το εργαλείο μας. Επιλέχθηκαν δύο χαρακτηριστικές συναρτήσεις με δύο και τρεις εισόδους και με διαφοροποιήσεις στο εύρος bit των μεταβλητών εισόδου, εξάχθηκαν χρήσιμα αποτελέσματα με τη σουίτα εργαλείων του Xilinx ISE. Επίσης, δόθηκαν και οι χρόνοι εκτέλεσης της γεννήτριας για όλες τις περιπτώσεις των συναρτήσεων. Το επόμενο κεφάλαιο παρουσιάζει την προοπτική δημιουργίας μίας startup που βασίζεται στο εργαλείο που δημιουργήσαμε.

Κεφάλαιο 5

Προοπτική δημιουργίας επιχείρησης

Το παρόν κεφάλαιο εξετάζει την περίπτωση δημιουργίας μίας πιθανής επιχείρησης που βασίζεται στη δημιουργία κυκλωμάτων με αυτόματο τρόπο χρησιμοποιώντας το εργαλείο που αναπτύχθηκε. Εξετάζονται οι πιθανές μορφές με τις οποίες μπορεί να προσφερθεί η υπηρεσία και γίνεται μία SWOT ανάλυση.

5.1 Μοντέλο επιχείρησης

Το SaaS[1] (Software-as-a-Service) περιγράφει ένα μοντέλο στο οποίο ο πελάτης δεν αγοράζει το λογισμικό, από μία επιχείρηση, αλλά την παροχή υπηρεσιών του λογισμικού. Πλεονέκτημα αυτού του μοντέλου αποτελεί το γεγονός ότι ο πελάτης δεν χρειάζεται να ανησυχεί για την εγκατάσταση και τη συντήρηση του προγράμματος και του υλικού στο οποίο φιλοξενείται. Επίσης η υπηρεσία είναι προσβάσιμη ανά πάσα στιγμή μέσω του διαδικτύου. Καθώς το εργαλείο που αναπτύχθηκε εδώ είναι διαδικτυακό, θα μπορούσε δημιουργηθεί μία επιχείρηση η οποία βασίζεται στην παροχή υπηρεσίας δημιουργίας κυκλωμάτων. Ένα σημαντικό στοιχείο σε αυτή την περίπτωση, θα αποτελούσε η ύπαρξη ενός web API (Application Programming Interface), δηλαδή ενός συνόλου λειτουργιών που μπορούν να εκτελούνται χωρίς την ανάγκη αλληλεπίδρασης με φυσικό πρόσωπο αλλά από άλλο πρόγραμμα.

Ένα άλλο πιθανό μοντέλο παράδοσης λογισμικού (delivery model) είναι το αδειοδοτημένο μοντέλο (licensed model). Στο μοντέλο αυτό, η επιχείρηση πουλάει το λογισμικό σε δυαδική-εκτελέσιμη μορφή. Στην περίπτωση αυτή, απαιτείται η συνοδεία του λογισμικού από κατάλληλα έγγραφα που περιγράφουν τη λειτουργία αλλά και τον τρόπο επέκτασης της λειτουργίας με νέες γεννήτριες κυκλωμάτων.

[1] `https://www.salesforce.com/saas/`

5.2 Ανάλυση SWOT

Η ανάλυση SWOT[2] (Strengths, Weaknesses, Opportunities, Threats) είναι το βασικό συστατικό του πλάνου μάρκετινγκ μιας επιχείρησης. Για την περίπτωση σύστασης μιας επιχείρησης με βάση το εργαλείο που παρουσιάστηκε, δίνεται η ανάλυση SWOT στον πίνακα 5.1.

Πίνακας 5.1: SWOT ανάλυση για τη σύσταση επιχείρησης με βάση το εργαλείο.

Δυνατά σημεία	**Αδύνατα σημεία**
Cloud εφαρμογή Ευκολία στη χρήση Εύκολη επέκταση Καλύτερη πρόσβαση	Έλλειψη πολλών γεννητριών Ταχύτητα παραγωγής κυλώματος Έλλειψη API
Ευκαιρίες	**Απειλές**
Αυτοματοποίηση παραγωγής κυκλωμάτων Τεχνολογική τάση εφαρμογών νέφους	Μεγάλες εταιρείες στο χώρο

5.3 Σύνοψη κεφαλαίου

Στο κεφάλαιο αυτό παρουσιάστηκαν δύο μοντέλα επιχειρήσεων που μπορούν να χρησιμοποιηθούν για τη δημιουργία μιας πιθανής επιχείρησης που βασίζεται στο εργαλείο που παρουσιάστηκε. Τέλος, δόθηκε και η ανάλυση SWOT. Η παρούσα διπλωματικής τελειώνει με την παρουσίαση συμπερασμάτων και μελλοντικών βελτιώσεων στο επόμενο κεφάλαιο.

[2] http://epixeirein.gr/2009/07/31/swot-analysis-efarmogi/

Κεφάλαιο 6

Συμπεράσματα και Μελλοντικές βελτιώσεις

Στην παρούσα εργασία αναπτύχθηκε μία εφαρμογή για τη δημιουργία επιταχυντών υλικού προσαρμοσμένων αριθμητικών συναρτήσεων. Το σύστημα αποτελείται από τον ιστότοπο, μέσω του οποίου αλληλεπιδρά ο χρήστης, και τη γεννήτρια που παράγει τις περιγραφές κυκλωμάτων.

6.1 Συμπεράσματα

Στην παρούσα διπλωματική αναπτύχθηκε μία γεννήτρια που είναι ικανή να παράγει περιγραφές υλικού προσαρμοσμένων αριθμητικών συναρτήσεων. Οι περιγραφές που παράγονται είναι σε γλώσσα VHDL και είναι συντακτικά ορθές και δεν περιέχουν λογικά λάθη. Επίσης, είναι συνθέσιμες είτε σε FPGA είτε σε ASIC ανεξαρτήτως κατασκευαστή. Εκτός από τις περιγραφές, η γεννήτρια παράγει και αρχείο δοκιμαστικών εισόδων (testbench) και σχηματικό κυκλώματος. Το testbench είναι συνθέσιμο και μπορεί να χρησιμοποιηθεί από τον τελικό χρήστη για να επιβεβαιώσει την ορθότητα λειτουργίας του παραγόμενου κυκλώματος.

Ένας στόχος που επετεύχθη, είναι η προσθήκη ακόμα ενός εργαλείου στη φαρέτρα των μηχανικών και σχεδιαστών κυκλωμάτων που διευκολύνει την προτυποποίηση και την εξερεύνηση του χώρου, ενώ παράλληλα εξοικονομεί πολύτιμο χρόνο. Σε σύγκριση με άλλα παρόμοια προγράμματα, η πρόσβαση στο εργαλείο μας είναι διαδικτυακή και δεν απαιτεί τοπική εγκατάσταση. Επίσης, η υπηρεσία μας είναι δωρεάν και δεν απαιτεί πληρωμή και συνδρομές. Το γεγονός αυτό, ίσως παροτρύνει και άλλους φορείς στην ανάπτυξη ελεύθερων εργαλείων στο συγκεκριμένο τομέα.

Άλλος ένας στόχος που επετεύχθη, ήταν αυτός της προώθησης ενδιαφέροντος προς άλλους φοιτητές στο να ασχοληθούν με την περαιτέρω ανάπτυξη της παρούσας γεννήτριας, αλλά και με τη δημιουργία άλλων γεννητριών αριθμητικών και λογικών πράξεων που χρησιμοποιούν τον ίδιο πυρήνα. Το μέρος τελικής επεξεργασίας στηρίζεται στη Python, η οποία αποτελεί μία εύκολη στην εκμάθηση και ευχάριστη στη χρήση γλώσσα προγραμματισμού που επιτρέπει τη γρήγορη συγγραφή προγραμμάτων με λιγότερες γραμμές κώδικα.

Η γεννήτρια που παρουσιάστηκε, αναπτύχθηκε με τρόπο που διευκολύνει την επέκταση της λειτουργικότητας της με την προσθήκη νέων λειτουργιών με ελάχιστο κόπο. Οι πιθανές επεκτάσεις, μπορούν να αποτελέσουν μέρος άλλης διπλωματικής ή πιθανής ερευνητικής εργασίας. Ορισμένες από αυτές τις επεκτάσεις αναλύονται στο επόμενο μέρος.

6.2 Μελλοντικές επεκτάσεις

Με την ανάπτυξη του παρόντος εργαλείου, τέθηκαν οι βάσεις για μία σειρά βελτιώσεων και επεκτάσεων που μπορούν να γίνουν ώστε να αυξηθεί η λειτουργικότητα και να συμπληρωθεί η χρηστικότητα. Οι βελτιώσεις αυτές είναι χρήσιμες όχι μόνο για τον τελικό χρήστη, αλλά και για τους ανθρώπους που θα επιλέξουν να συνεχίσουν την ανάπτυξη του παρόντος έργου. Ορισμένες προτάσεις έχουν επίσης ως στόχο τον εκσυγχρονισμό του κώδικα ώστε να προχωράει παράλληλα με την τεχνολογική τάση και να είναι σε θέση να ανταγωνίζεται παρόμοια εργαλεία που χρησιμοποιούν διαφορετικές τεχνολογίες.

Μία κατηγορία μελλοντικών επεκτάσεων αποτελεί η προσθήκη νέων γεννητριών αριθμητικών πυρήνων στη βιβλιοθήκη. Ένα παράδειγμα μιας γεννήτριας ενός πυρήνα αποτελεί το κύκλωμα της διαίρεσης. Η δημιουργία μιας γεννήτριας αποτελεί μία χρονοβόρα και δύσκολη διαδικασία. Ωστόσο, η προσθήκη της στη βιβλιοθήκη απαιτεί σχεδόν μηδαμινό χρόνο. Αυτό που κάνει τη δημιουργία ενός πυρήνα χρονοβόρα διαδικασία, είναι το γεγονός ότι το κύκλωμα του νέου πυρήνα πρέπει να μπορεί να εκφραστεί μέσω ενός αλγορίθμου που θα μπορεί να χτίζει το κατάλληλο κύκλωμα ανεξαρτήτως εισόδου και εύρους bit. Επίσης το παραγόμενο κύκλωμα θα πρέπει να είναι εύκολα συνδέσιμο με άλλα κυκλώματα.

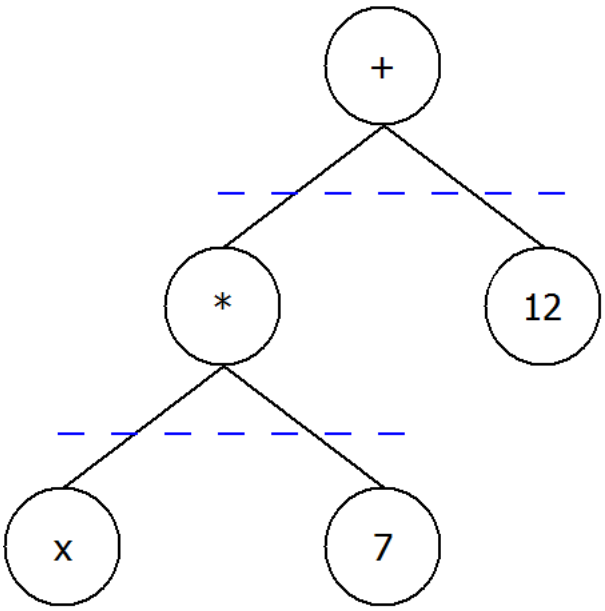
Στην ίδια κατηγορία βελτιώσεων ανήκει και η προσθήκη πυρήνων που υποστηρί-

ζουν αριθμούς κινητής υποδιαστολής. Η σημαντικότητα της προσθήκης αυτής γίνεται αντιληπτή από το γεγονός ότι η ροή δεδομένων δεν μπορεί να είναι πάντα σε ακέραια μορφή. Επίσης ο κώδικας που εκτελεί αριθμητικές πράξεις μπορεί να παράγει μη ακέραιο αποτέλεσμα. Με την προσθήκη αυτή, θα μπορούν να δημιουργούνται κυκλώματα συναρτήσεων που υποστηρίζουν πράξεις αριθμών κινητής υποδιαστολής. Όπως αναφέρθηκε και προηγουμένως, η διαδικασία δημιουργίας τέτοιων γεννητριών δεν είναι απλή διαδικασία.

Μία σημαντική βελτίωση αποτελεί η προσθήκη της επιλογής για διασωλήνωση (pipeline) των παραγόμενων κυκλωμάτων. Η διασωλήνωση επιτρέπει στα κυκλώματα να εκτελούν περισσότερες πράξεις στην ίδια χρονική μονάδα (ακμή ρολογιού) και επιτυγχάνεται με την τοποθέτηση καταχωρητών DFF (D flip-flop) στα κατάλληλα σημεία. Η προσθήκη καταχωρητών μπορεί να είναι ελεγχόμενη και να γίνεται βάση κάποιας στρατηγικής που θα επιλέγεται από τον χρήστη. Μία στρατηγική θα ήταν να τοποθετούνται οι καταχωρητές σε κάθε ακμή του δέντρου. Ωστόσο η επιλογή αυτή μπορεί να δημιουργήσει προβλήματα συγχρονισμού σε ορισμένες περιπτώσεις και θα αυξήσει την πολυπλοκότητα της γεννήτριας που θα πρέπει να είναι σε θέση να τα λύνει. Οι περιπτώσεις στις οποίες θα προκύψουν προβλήματα συγχρονισμού είναι τα μη ομοιόμορφα δέντρα έκφρασης. Παράδειγμα ενός τέτοιου δέντρου, για τη συνάρτηση $f(x) = ((x * 7) + 12)$ δίνεται στο σχήμα 6.1. Στην περίπτωση αυτή τα μονοπάτια που ακολουθούνται από τη ρίζα προς στα φύλλα δεν έχουν το ίδιο μήκος. Έτσι αν εφαρμοστεί η παραπάνω στρατηγική και οι καταχωρητές εισαχθούν στα σημεία που τέμνονται οι ακμές με τις οριζόντιες (διακεκομμένες) γραμμές, το αριστερό κλαδί από τη ρίζα θα έχει μεγαλύτερη καθυστέρηση από το δεξί και θα παράγεται σωστό αποτέλεσμα στην έξοδο σε κάθε 2 ακμές ρολογιού. Συνεπώς, το πρόγραμμα θα πρέπει να είναι σε θέση να προβλέψει αυτή την περίπτωση και να υπολογίσει το σωστό αριθμών των καταχωρητών που πρέπει να εισαχθούν σε κάθε κλαδί, όπως φαίνεται στο σχήμα 6.2.

Η αύξηση της απόδοσης και της ποιότητας υπηρεσίας της εφαρμογής, αποτελεί μία βελτίωση που θα συμβάλει στην προώθηση της χρήσης του εργαλείου μας. Η προσαρμογή των μονάδων της γεννήτριας ώστε να λειτουργούν ανεξάρτητα η μία από την άλλη, θα οδηγήσει στην παραλληλοποίηση τους και θα μειώσει το χρόνο παραγωγής κυκλωμάτων. Η βελτίωση αυτή δεν μπορεί να υλοποιηθεί παντού, καθώς

Σχήμα 6.1: Λανθασμένη διασωλήνωση μη ομοιόμορφου δέντρου έκφρασης.



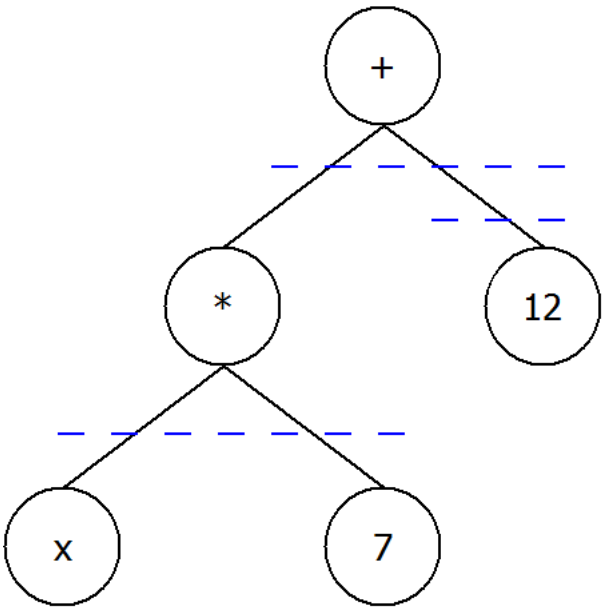
υπάρχουν μονάδες των οποίων η είσοδος εξαρτάται από τις προηγούμενες. Σε αυτές τις περιπτώσεις, η βελτίωση μπορεί να γίνει με τη μετατροπή των εξαρτώμενων ρουτινών σε γλώσσες χαμηλότερου επιπέδου, όπως η C με η χρήση της Cython.

Καθώς το εργαλείο αναπτύσσεται και μεγαλώνει σε γραμμές κώδικας (Lines Of Code-LOC), γίνεται όλο και πιο δύσκολος ο έλεγχος και προσδιορισμός ενός σφάλματος. Επομένως, θεωρείται καλή πρακτική να γράφεται κώδικας δοκιμής που εξετάζει την ορθότητα λειτουργίας μιας συνάρτησης. Η Python προσφέρει πολλά εργαλεία[1] που βοηθάνε στη συγγραφή προγραμμάτων δοκιμής. Με τη χρήση κατάλληλων βιβλιοθηκών όπως Unittest, Doctest και άλλων, θα είμαστε σε θέση να προσδιορίζουμε το πρόβλημα χωρίς να χρειάζεται να εκτελούμε αποσφαλμάτωση κατά βήμα.

Όπως αναφέρθηκε προηγουμένως, η εφαρμογή στηρίζεται στην δεύτερη έκδοση της γλώσσας προγραμματισμού Python, συγκεκριμένα στη Python 2.7.6 που κυκλοφόρησε τον Ιούνιο του 2015. Το 2008 ύστερα από μακρά περίοδο δοκιμών, κυκλοφόρησε η τρίτη έκδοση της, η οποία ήταν μη συμβατή με την προηγούμενη. Σήμερα, οι δύο εκδόσεις αναπτύσσονται παράλληλα και υπάρχει διαπληκτισμός ανάμεσα στους προγραμματιστές που υποστηρίζουν διαφορετικές εκδόσεις. Ωστόσο, σε αυτό που φαίνεται συμφωνούν όλοι είναι ότι κάποια στιγμή η προηγούμενη έκδοση θα σταματήσει να υποστηρίζεται και επίσημα θα χρησιμοποιείται μόνο η τρίτη. Επί-

[1] http://docs.python-guide.org/en/latest/writing/tests/

Σχήμα 6.2: Σωστή διασωλήνωση μη ομοιόμορφου δέντρου έκφρασης.



σης, η πρόταση των παλαιότερων προς τους νέους προγραμματιστές που θέλουν να μάθουν Python, είναι να ξεκινήσουν απευθείας με τη νεότερη έκδοση της. Έτσι, για να εξασφαλιστεί μακρόχρονη υποστήριξη και να αυξηθεί η προσέλκυση από νέους προγραμματιστές να ασχοληθούν με την ανάπτυξη του παρόντος εργαλείου, η αλλαγή έκδοσης της Python αποτελεί μία στέρεη βελτίωση.

Η μεταφορά του αποθετηρίου (repository) στο cloud (github, gitlab, bitbucket) αποτελεί μία βελτίωση που θα αυξήσει την εξωστρέφεια της ανάπτυξης της εφαρμογής και θα διευκολύνει την εισαγωγή νέων προγραμματιστών που θέλουν να προσθέσουν βελτιώσεις.

Όπως αναφέρθηκε προηγουμένως, ο αριθμός γραμμών κώδικα των δομικών μερών της εφαρμογής αυξάνεται συνεχώς με τη προσθήκη νέων λειτουργιών και τη διόρθωση σφαλμάτων. Επίσης, κατά καιρούς έγιναν ορισμένες ενέργειες διαχωρισμού των συναρτήσεων που εκτελούν μία κατηγορία λειτουργιών σε ξεχωριστά αρχεία βιβλιοθηκών. Ύστερα από αυτές τις αλλαγές έγινε και μία προσπάθεια για συγγραφή ενός οδηγού ανάπτυξης (developer's guide) ο οποίος εξηγεί σε λεπτομέρεια τη λειτουργία και το αποτέλεσμα των δομικών συναρτήσεων. Ωστόσο, σήμερα ο οδηγός αυτός δεν περιέχει τις απαραίτητες οδηγίες ανάπτυξης και εισαγωγής νέων προγραμματιστών στη ροή και στον τρόπο επέκτασης του εργαλείου. Συνεπώς, η ανανέωση του οδηγού ανάπτυξης και ο διαχωρισμός σε θεματικές ενότητες αποτελεί επίσης μία καλή βελτίωση για την εξασφάλιση της μελλοντικής ενασχόλησης

νέων φοιτητών.

Τέλος, μία σημαντική αλλά και μεγάλη επέκταση, αποτελεί η προσθήκη της επιλογής για παραγωγή περιγραφών κυκλωμάτων σε Verilog αντί για VHDL. Με την προσθήκη αυτή, ο κώδικας θα είναι κατανοητός από προγραμματιστές οικείους στην εκάστοτε γλώσσα προτίμησης.

Παραρτήματα

Παράρτημα Α΄

Κανόνες σύνταξης αρχείου εισόδου

Για τη σύνταξη του αρχείο εισόδου, πρέπει να ακολουθούνται δύο συντακτικοί κανόνες. Ο πρώτος κανόνας ορίζει ότι οι μεταβλητές της αριθμητικής συνάρτησης, πρέπει να δίνονται με το πρότυπο ονομασίας "inX", όπου X είναι ακέραιος αριθμός που ξεκινάει από το 0 (μηδέν). Ο περιορισμός αυτός οφείλεται στη λειτουργία της συνάρτησης $json.load()$ της βιβλιοθήκης json. Η συνάρτηση αυτή εκτελεί μετατροπές από αντικείμενο json σε αντικείμενο Python σύμφωνα με τον πίνακα Α΄.1. Στην περίπτωση μας, η συνάρτηση μετατρέπει το αντικείμενο JSON σε αντικείμενο dict της Python. Ο κώδικας Python, της συνάρτησης που διαβάζει το αρχείο εισόδου και εκτελεί τη μετατροπή αυτή, δίνεται στην απεικόνιση Α΄.1. Δυστυχώς, η κλήση στη συνάρτηση αυτή, δεν επιστρέφει την είσοδο αυτούσια όπως είναι γραμμένη στο αρχείο, αλλά εκτελεί αλλαγή στη σειρά των μεταβλητών. Η αλλαγή αυτή έχει σαν αποτέλεσμα να μπερδεύει τη γεννήτρια και να μη δηλώνονται σωστά τα εύρη εισόδων των μεταβλητών. Ύστερα από μελέτη της λειτουργίας της συγκεκριμένης συνάρτησης με το Pudb (2.7.4), δεν βρέθηκε η αιτία του προβλήματος. Ωστόσο, βρέθηκε τρόπος να παρακαμφθεί το πρόβλημα αυτό με τη δήλωση των μεταβλητών με την παραπάνω ονομασία. Μία πιθανή λύση στην περίπτωση αυτή, θα ήταν να προηγηθεί μία συνάρτηση μετονομασίας των μεταβλητών με οποιοδήποτε όνομα στο πρότυπο "inX".

Πίνακας Α΄.1: Αντιστοίχηση αντικειμένου JSON σε αντικείμενο Python.

JSON	Python
object	dict
array	list
string	unicode
number (int)	int, long
number (real)	float
true	True
false	False
null	None

Απεικόνιση Α΄.1: Συνάρτηση Python που διαβάζει το αρχείο εισόδου.

```
def open_input_definitions_and_return_data(input_filename):
    """
    Read JSON equation definition input file and return
    Python dict
    INPUT:
        name of the JSON file
    OUTPUT:
        Python dictionary
    """
    with open(input_filename) as jdata:
        json_data = json.load(jdata)
    return json_data
```

Ο δεύτερος κανόνας σύνταξης, ορίζει ότι η σειρά δήλωσης των μεταβλητών, κάτω από τη γραμμή της συνάρτησης, πρέπει να ακολουθεί τη σειρά που έχουν δοθεί στη συνάρτηση, από αριστερά προς τα δεξιά. Η ιδιαιτερότητα αυτή οφείλεται στη χρήση της ενσωματωμένης συνάρτησης $D.keys()$ της Python, η οποία επιστρέφει μία λίστα με τα κλειδιά του λεξιλογίου D. Η συνάρτηση αυτή, τείνει να ταξινομεί τα κλειδιά κατά αλφαβητική σειρά από προεπιλογή. Η χρήση της γίνεται στη γεννήτρια των testbench (3.5.3), όπου τα ονόματα των μεταβλητών εξάγονται ως κλειδιά και η σειρά τους δηλώνει τη θύρα εισόδου στο κύκλωμα. Αν λοιπόν οι μεταβλητές δεν δηλώνονται με σωστή σειρά ή ταξινομηθούν από τη συνάρτηση keys(), το testbench θα οδηγήσει εισόδους λανθασμένου εύρους σε λάθος θύρα εισόδου. Αξίζει να σημειωθεί ότι η περιγραφή του κυκλώματος, δεν επηρεάζεται από τον συγκεκριμένο κανόνα.

Παράρτημα Β΄

Δομή της α-HDL

Η α-HDL netlist είναι μία ειδική δομή που δημιουργήθηκε για να αποτελεί είσοδο της μονάδας παραγωγής HDL (3.5.2). Συγκροτείται από τρεις δομές που ονομάζονται componentlist, components και interconnections. Παρακάτω, αναλύεται η α-HDL netlist (3.4) που παράγεται για τη συνάρτηση του παραδείγματος 2.4 και έχει αρχείο εισόδου το 4.1.

Η πρώτη δομή ονομάζεται componentlist και δίνεται στην απεικόνιση Β΄.1. Πρόκειται για μία δομή dict της Python και περιέχει στοιχεία της βιβλιοθήκης που χρησιμοποιήθηκαν στη δημιουργία του κυκλώματος, κωδικοποιημένα με ένα τυχαίο κλειδί. Για παράδειγμα, το κλειδί '10' αντιστοιχεί στο στοιχείο βιβλιοθήκης fa1bit, το '11' στο ha1bit, κ.ο.κ. Η ονομασία των στοιχείων έχει καθοριστεί από τον εκάστοτε προγραμματιστή που έφτιαξε το αντίστοιχο στοιχείο και το πρόσθεσε στη βιβλιοθήκη. Η απόφαση για την προσθήκη ενός στοιχείου στη βιβλιοθήκη εξαρτάται από τη συχνότητα χρήσης του στοιχείου και κατ' επέκταση της χρησιμότητας του. Για παράδειγμα, έχουμε δημιουργήσει ένα στοιχείο που αποτελείται από ακριβώς τρεις πύλες NOT και ονομάζεται "trinot". Το στοιχείο αυτό, δέχεται είσοδο τριών bit και τα αντιστρέφει. Πρακτικά, το στοιχείο αυτό από μόνο του δεν είναι χρήσιμο, ωστόσο χρησιμοποιείται κατά κόρον στη γεννήτρια που παράγει συγκριτές αριθμών (magnitude comparators) [25]. Στο παράδειγμα μας, η σημασία της ονομασίας των στοιχείων δίνεται στον πίνακα Β΄.1.

Απεικόνιση Β΄.1: Δομή componentlist για τη συνάρτηση $f(x, y) = ((2 * x) + y)$.

```
{'10': 'fa1bit',
 '11': 'ha1bit',
 '128': 'wire1bit',
 '2': 'output',
 '20': 'fa1bitclock',
 '21': 'ha1bitclock',
 '7': 'dff1bit',
 '8': 'dff2bit'}
```

Πίνακας Β΄.1: Επεξήγηση ονομασίας στοιχείων του πίνακα componentlist.

fa1bit	:	πλήρης αθροιστής 1 bit
ha1bit	:	ημιαθροιστής 1 bit
wire1bit	:	καλώδιο 1 bit
output	:	κόμβος εξόδου
fa1bitclock	:	πλήρης αθροιστής 1 bit με ρολόι
ha1bitclock	:	ημιαθροιστής 1 bit με ρολόι
dff1bit	:	καταχωρητής D flip flop 1 bit
dff2bit	:	καταχωρητής D flip flop 2 bit

Η δεύτερη δομή ονομάζεται components και δίνεται στην απεικόνιση Β΄.2. Η δομή αυτή είναι μία τρισδιάστατη Python λίστα και δείχνει τις συντεταγμένες στις οποίες τοποθετούνται τα στοιχεία που δηλώθηκαν στη δομή componentlist. Στο παράδειγμα, βλέπουμε ότι στη συντεταγμένη (0,0,0), έχει τοποθετηθεί το στοιχείο με κλειδί '128' που αντιστοιχεί στο στοιχείο wire1bit που είναι το καλώδιο ενός bit. Αντίστοιχα, στη συντεταγμένη (5,1,0) έχει τοποθετηθεί το στοιχείο με κλειδί '10' που αντιστοιχεί στο στοιχείο fa1bit που είναι ο πλήρης αθροιστής ενός bit κ.ο.κ.

Απεικόνιση Β΄.2: Δομή components για τη συνάρτηση $f(x, y) = ((2 * x) + y)$.

```
[[[128, 128], [128, 128], [0, 128]],
 [],
 [[128], [128], [128]],
 [],
 [],
 [[11], [10], [10]],
 [[128], [128], [128], [128]],
 [[2], [2], [2], [2]]]
```

Η τελευταία δομή ονομάζεται interconnections και δίνεται στην απεικόνιση Β΄.2.

Όπως και η προηγούμενη, η δομή αυτή είναι μία τρισδιάστατη Python λίστα και δείχνει τις συνδέσεις των στοιχείων της λίστας components. Η εξήγηση της σημασίας της κάθε μεταβλητής στη λίστα, δόθηκε στον πίνακα 3.1. Το στοιχείο fa1bit στη συντεταγμένη (5,1,0), έχει τρεις εισόδους, όπως φαίνεται στην απεικόνιση Β΄.4. Η πρώτη είσοδος είναι από καλώδιο 1 bit που βρίσκεται στη συντεταγμένη (2,1,0), η δεύτερη είσοδος είναι από το καλώδιο που βρίσκεται στη συντεταγμένη (0,1,1) και η τρίτη είσοδος είναι από το στοιχείο ha1bit που βρίσκεται στη συντεταγμένη (5,0,0). Πρέπει να σημειωθεί ότι το χτίσιμο της netlist με το χέρι είναι μία επίπονη και επιρρεπή σε λάθη διαδικασία, για το λόγο αυτό, γίνεται αυτόματα από τη γεννήτρια με την κλήση κατάλληλων συναρτήσεων.

Απεικόνιση Β΄.3: Δομή interconnections για τη συνάρτηση $f(x, y) = ((2 * x) + y)$.

```
[[[[[-1, -1, -1, 0, 0, 0, 0]], [[-1, -1, -1, 1, 0, 0, 0]]],
  [[[-1, -1, -1, 0, 1, 1, 0]], [[-1, -1, -1, 1, 1, 1, 0]]],
  [[], [[-3, -3, -3, 0, 2, 2, 0]]]],
 [],
 [[[[-3, -3, -3, 0, 0, 0, 0]]],
  [[[0, 0, 0, 0, 0, 0, 0]]],
  [[[0, 1, 0, 0, 0, 0, 0]]]],
 [],
 [],
 [[[[2, 0, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0]]],
  [[[2, 1, 0, 0, 0, 0, 0], [0, 1, 1, 0, 0, 0, 0], [5, 0, 0, 1,
     0, 0, 0]]],
  [[[2, 2, 0, 0, 0, 0, 0], [0, 2, 1, 0, 0, 0, 0], [5, 1, 0, 1,
     0, 0, 0]]]],
 [[[[5, 0, 0, 0, 0, 0, 0]]],
  [[[5, 1, 0, 0, 0, 0, 0]]],
  [[[5, 2, 0, 0, 0, 0, 0]]],
  [[[5, 2, 0, 1, 0, 0, 0]]]],
 [[[6, 0, 0, 0, 0, 0, 0, 0, 0, 0]],
  [[6, 1, 0, 0, 0, 0, 0, 0, 1, 1]],
  [[6, 2, 0, 0, 0, 0, 0, 0, 2, 2]],
  [[6, 3, 0, 0, 0, 0, 0, 0, 3, 3]]]]
```

Απεικόνιση Β΄.4: Συνδέσεις στοιχείου fa1bit στη συντεταγμένη $(5, 1, 0)$.

```
[[2, 1, 0, 0, 0, 0, 0],
[0, 1, 1, 0, 0, 0, 0],
[5, 0, 0, 1, 0, 0, 0]]
```

Βιβλιογραφία

[1] Peter J. Ashenden. *The Designer's Guide to VHDL*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 1999.

[2] K. Becker. Plain & simple hardware description language. http://pshdl.org/.

[3] Stefan Behnel, Robert Bradshaw, Craig Citro, Lisandro Dalcin, Dag Sverre Seljebotn, and Kurt Smith. Cython: The best of both worlds. *Computing in Science and Engineering*, 13(2):31–39, 2011.

[4] Per Bjesse, Koen Claessen, Mary Sheeran, and Satnam Singh. Lava: hardware design in haskell. In *ACM SIGPLAN Notices*, volume 34, pages 174–184. ACM, 1998.

[5] Stephen Brown and Jonathan Rose. FPGA and CPLD architectures: A tutorial. *IEEE Design and Test of Computers*, 13(2):42–57, 1996. An approachable tutorial for a general audience on FPGA and CPLD architectures.

[6] Benton H. Calhoun, Xin Li Yu Cao, Ken Mai, Lawrence T. Pileggi, Rob A. Rutenbar, and Kenneth L. Shepard. Digital circuit design challenges and opportunities in the era of nanoscale cmos. *Proceedings of the IEEE (Special Issue on Integrated Electronics: Beyond Moore's Law)*, 96:343–365, 02/2008 2008.

[7] Scott Chacon and Ben Straub. *Pro Git*. Apress, 2014.

[8] D Crockford. The application/json media type for javascript object notation (json). *Internet RFC 4627*, July 2006.

[9] M. Dasygenis. A web eda tool for the automatic generation of synthesizable vhdl architectures for a rapid design space exploration. In *Design Technology of Integrated Systems In Nanoscale Era (DTIS), 2014 9th IEEE International Conference On*, pages 1–2, May 2014.

[10] Florent de Dinechin and Bogdan Pasca. Designing custom arithmetic data paths with FloPoCo. *IEEE Design & Test of Computers*, 28(4):18–27, July 2011.

[11] J. Decaluwe. Myhdl. http://www.myhdl.org/, 2015.

[12] Allen B. Downey, Jeffrey Elkner, and Chris Meyers. *How to Think Like a Computer Scientist: Learning with Python*. Green Tea Press, 2002.

[13] Norman G. Einspruch et al. Inside front cover. In Norman G. Einspruch and Jeffrey L. Hilbert, editors, *Application Specific Integrated Circuit (ASIC) Technology*, pages ii –. Academic Press, 1991.

[14] Andy Gill, Tristan Bull, Garrin Kimmell, Erik Perrins, Ed Komp, and Brett Werling. Introducing Kansas Lava. In *Proceedings of the Symposium on Implementation and Application of Functional Languages*, volume 6041 of *LNCS*. Springer-Verlag, Sep 2009.

[15] T. Gingold. Ghdl. `http://ghdl.free.fr/`.

[16] Niels Jørgensen. Putting it all in the trunk: incremental software development in the freebsd open source project. *Information Systems Journal*, 11(4):321–336, 2001.

[17] M. A. Kastner. The single-electron transistor. *Rev. Mod. Phys.*, 64:849–858, Jul 1992.

[18] R. Kern. line_profiler. `https://github.com/rkern/line_profiler`.

[19] A. Kloeckner. pudb. `https://github.com/inducer/pudb`, 2009.

[20] M. Kooijman. Haskell as a higher order structural hardware description language, December 2009.

[21] Greg Lehey and Marshall Kirk McKusick. *The Complete FreeBSD*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 4 edition, 2003.

[22] Steven Oualline. *Vi iMproved*. New Riders Publishing, 2001.

[23] Giannis Petrousov and Minas Dasygenis. Designing optimized forward residue number systems IP blocks converters from a network interface. In *18th Panhellenic Conference on Informatics, PCI '14, Athens, Greece, October 2-4, 2014*, pages 32:1–32:6, 2014.

[24] Giannis Petrousov and Minas Dasygenis. A unique network EDA tool to create optimized ad hoc binary to residue number system converters. In *24th International Workshop on Power and Timing Modeling, Optimization and Simulation, PATMOS), Palma de Mallorca, Spain, September 29 - Oct. 1, 2014*, pages 1–8, 2014.

[25] Ioannis Petrousov and Minas Dasygenis. A CAD tool for custom magnitude comparators. In *Proceedings of the 19th Panhellenic Conference on Informatics, PCI 2015, Athens, Greece, October 1-3, 2015*, pages 413–418, 2015.

[26] C. Pilato and F. Ferrandi. Bambu: A modular framework for the high level synthesis of memory-intensive applications. In *2013 23rd International Conference on Field programmable Logic and Applications*, pages 1–4, Sept 2013.

[27] Markus Püschel, José M. F. Moura, Jeremy Johnson, David Padua, Manuela Veloso, Bryan Singer, Jianxin Xiong, Franz Franchetti, Aca Gacic, Yevgen Voronenko, Kang Chen, Robert W. Johnson, and Nicholas Rizzolo. SPIRAL: Code generation for DSP transforms. *Proceedings of the IEEE, special issue on "Program Generation, Optimization, and Adaptation"*, 93(2):232– 275, 2005.

[28] Fernando Pérez and Brian E. Granger. Ipython: A system for interactive scientific computing. *Computing in Science & Engineering*, 9(3):21–29, 2007.

[29] Will Reese. Nginx: the high-performance web server and reverse proxy. *Linux Journal*, 2008(173):2, 2008.

[30] Kenneth Reitz and Tanya Schlusser. *The Hitchhiker's Guide to Python: Best Practices for Development*. O'Reilly Media, 2016.

[31] Arrvindh Shriraman, Michael F Spear, Hemayet Hossain, Virendra J Marathe, Sandhya Dwarkadas, and Michael L Scott. An integrated hardware-software approach to flexible transactional memory. In *ACM SIGARCH Computer Architecture News*, volume 35, pages 104–115. ACM, 2007.

[32] Guido Van Rossum et al. Python programming language. In *USENIX Annual Technical Conference*, volume 41, 2007.

[33] SM Vidanagamachchi, SD Dewasurendra, Roshan G Ragel, and M Niranjan. Tile optimization for area in fpga based hardware acceleration of peptide identification. In *2011 6th International Conference on Industrial and Information Systems*, pages 140–145. IEEE, 2011.

[34] K. Wakabayashi and T. Okamoto. C-based soc design flow and eda tools: An asic and system vendor perspective. *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, 19(12):1507–1522, November 2006.

[35] Wikipedia. Engine control unit — wikipedia, the free encyclopedia, 2016. [Online; accessed 5-September-2016].